Technische Universität München
TUM School of Computation, Information and Technology

Securing Networked Systems by Identifying and Monitoring Multi-Layer Dependencies

Lars Wüstrich

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz: Prof. Dr. Jens Großklags

Prüfende der Dissertation:

1. Prof. Dr.-Ing. Georg Carle
2. Prof. Dr. Marc-Oliver Pahl

Die Dissertation wurde am 20.09.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 10.02.2025 angenommen.

Acknowledgements

Persevering to finish this project was only possible thanks to the support of many people. Many have provided indispensable support throughout the years, making it impossible to name them all here. Without all of you, I would have never been able to make it this far. The following paragraphs acknowledge an incomplete subset of them.

Thank you, Prof. Georg Carle, for welcoming me to your chair and allowing me to pursue a PhD. Your support and advice throughout the years shaped my approach to research and my ability to contribute to the community. Thank you for giving me the freedom to explore less conventional ideas and for our valuable discussions throughout the years.

I am equally grateful to you, Prof. Marc-Oliver Pahl, for believing in me and putting your trust in my abilities. Without your unwavering support, I would never have had the opportunity to start this endeavor in the first place, and without your continuous advice and guidance, I would never have made it across the finish line. Even after your departure to France, you made sure to maintain our weekly exchanges to discuss novel ideas, shape my academic progress and enable the contributions that made it into this thesis, thank you! By integrating me into the research community and other activities early on, you showed me how to be a good scientist. Your approach to embracing challenges and finding positives even in the most difficult circumstances inspires me forever and will shape my approach to tackle problems for the rest of my life.

Thank you also to Prof. Jens Großklags for chairing the examination committee.

I am also grateful to all of my co-authors (in no particular order): Sebastian, Stefan E., Markus S., Markus B., Dominik, Lukas, Stephan, Michel, Joaqin, Christian, Satu, Jussi K., Roope, Hanne, Olli, Arpit, Markus D., Josef, Jussi N., and Thomas. Thank you also to all the students I had the pleasure of working with over the years.

Furthermore, I would like to thank my colleagues at the chair, who were always open to discussions and fooling around, creating an excellent research environment. In particular, I want to mention my colleagues Dominik, Florian, and Eric, with whom I spent weeks organizing the iLab1 course and conducting a lot of oral attestations. In addition, thank you, Stefan G., for taking me on an unexpected but fascinating detour into the German eID and trust services; this work will improve many people's lives.

Thank you also to the people outside of academia who prevented me from living in the office and encouraged me to enjoy life. Particularly, thank you, Benedikt Graswald, for being a great friend, motivating me to push my limits beyond what I thought was possible at the ZHS, and keeping my spirits up in difficult times.

Lastly, I would like to thank my family, who shaped me into the person I am, particularly those who always believed in me.

Thank you,

Lars

Abstract

Network activity always happens in order to provide a service to other processes. Examples are applications that exchange information with remote services or central control units that instruct remote devices to execute actions. This creates a relationship between network and device activity. We refer to such phenomena as multi-layer relationships, connecting events across multiple layers, e.g., network events with application or physical layer events. The network is a prime vantage point to monitor devices, e. g., allowing to identify security incidents by monitoring multi-layer relationships. However, most monitoring tools, such as network intrusion detection systems, focus purely on information from individual layers. By lacking the ability to correlate events from multiple layers, the monitoring may fail to detect abnormal system states.

This thesis explores mechanisms to secure networked systems by correlating events from the network with device activity. We introduce novel methods to identify abnormal system behavior via multi-layer event correlation. First, we investigate which multi-layer relationships exist in computer networks. We analyze how applications influence network activity, and how control traffic implicates physical changes in the environment due to device activity. Afterward, we introduce methods to monitor these multi-layer relationships to identify abnormal device behavior. We use this approach to build accurate multi-layer models of device behavior in computer networks, describing expected system behavior.

We focus on two multi-layer relationships linking network activity to events on other layers. First, we investigate the relationship between application and network activity. Typically, operating systems handle transmitting and receiving network traffic for applications. This separation of handling network traffic impedes the attribution of packets observed on a network interface to its causing application. Existing methods to create this association cause significant overhead or yield an incomplete data collection. We develop a novel method that efficiently associates packets to applications based on the extended Berkeley Packet Filter to address this issue. Based on this data collection, we introduce network application profiles that capture the typical network behavior of applications. We show that these profiles allow us to identify unusual application behavior causing abnormal communication activity.

We also investigate the link between control traffic and physical activity of devices in network-controlled environments. We use this relationship to create a cyber-physical model that predicts physical changes in the environment based on observed control traffic. This allows anomaly detection in signals of composite side-channels in which multiple devices simultaneously influence physical measurements. Our approach addresses limitations of current side-channel monitoring methods, which often aim at monitoring isolated devices.

The results from this thesis show that setting events from multiple layers into context enables advanced anomaly detection techniques. In particular, it allows the identification of attacks and faults that cause inconsistent behavior on different layers. An example is covert attacks in which adversaries concurrently manipulate the input and output of devices to mislead the remote monitoring. The methods presented in this thesis enable the development of advanced multi-layer anomaly detection systems. Our approach allows us to detect anomalies, which anomaly detection systems that only use one source of information fail to identify.

Zusammenfassung

Netzwerkaktivität hat stets den Zweck, einen Dienst für andere Prozesse bereitzustellen. Beispiele dafür sind Applikationen, welche Informationen mit entfernten Diensten austauschen oder zentrale Steuereinheiten, die entfernte Geräte zur Ausführung von Aktivität instruieren. Dies stellt einen direkten Zusammenhang zwischen Ereignissen im Netzwerk und Geräteaktivität auf anderen Schichten her, wie zum Beispiel der Veränderung des physischen Gerätezustands. Wir nennen diese Verknüpfung eine mehrschichtige Beziehung. Daher ist das Netzwerk ein wichtiger Knotenpunkt zur Überwachung von verteilten System, z. B. zur Erkennung von Sicherheitszwischenfällen. Jedoch fokussieren sich die meisten Überwachungsinstrumente, wie Systeme zur Erkennung von Netzwerk-Penetrationsversuchen, ausschließlich auf einzelne Schichten. Durch die fehlende Fähigkeit, Ereignisse von unterschiedlichen Schichten zueinander in Kontext zu setzen, ist es möglich, dass Angriffe oder kritische Systemzustände in der Systemüberwachung unerkannt bleiben.

In dieser Arbeit werden Mechanismen zur Absicherung von vernetzten Systemen erforscht, welche Ereignisse im Netzwerk zu denen in andern Schichten in Kontext setzten. Es werden neue Methoden zur Absicherung von verteilten Systemen mithilfe von Mehr-Schicht Ereignis-Korrelation vorgestellt. Zuerst wird erarbeitet, welche mehrschichtigen Beziehungen in Rechnernetzwerken existieren. Insbesondere analysieren wir Beziehungen, in welchen Applikationen das Netzwerk nutzen und wie Kontrollverkehr mit physischen Veränderungen durch Geräteaktivität in einer Umgebung zusammenhängt. Danach stellen wir Methoden vor, um diese Beziehungen zu überwachen, um abnormales Verhalten zu identifizieren. Wir nutzen dann diesen Ansatz, um akkurate, mehrschichtige Modelle von Geräteverhalten zu entwickeln, welche zu erwartendes Systemverhalten beschreiben.

Wir fokussieren uns auf zwei Beziehungen, die Ereignisse im Netzwerk mit Ereignissen auf anderen Schichten verknüpfen. Zuerst wird die Beziehung zwischen Applikationen und Netzwerkaktivität untersucht. Üblicherweise übernehmen Betriebssysteme die Aufgabe, für Applikationen Netzwerkdaten zu senden und zu empfangen. Durch diese Separation der Aufgaben ist es herausfordernd, effizient Pakete, die an einem Netzwerkinterface beobachtet werden können, einem verursachendem Prozess zuzuschreiben. Existierende Methoden, die eine solche Verknüpfung ermöglichen, verursachen entweder signifikaten Overhead auf Endgeräten oder führen zu einer unvollständigen Datensammlung. Wir stellen eine neue Methode zur Verknüpfung von Netzwerkpaketen und Prozessen vor, basierend auf dem extended Berkeley Paket Filter. Mithilfe dieser Methode zur Datensammlung schlagen wir Applikations-Netzwerkprofile vor, welche das typische Netzwerkverhalten von Applikationen festhalten. Wir zeigen, dass Applikations-Netzwerkprofile es uns erlauben, unübliches Applikationsverhalten zu erkennen.

Zusätzlich untersuchen wir die Beziehung zwischen Kontrollverkehr und physischer Geräteaktivität in netzwerkgesteuerten Umgebungen. Wir nutzen diesen Zusammenhang, um ein Cyber-physisches Modell zu erstellen, welches physikalische Änderungen in der Umgebung mithilfe von beobachtetem Kontrollverkehr vorhersagt. Die vorgeschlagene Methode zur Vorhersage der physikalischen Änderungen ermöglicht Anomalieerkennung in gemischen Seitenkanalsignalen, welche gleichzeitig von mehreren Geräten beeinflusst werden. Dadurch werden die Limitierungen

aktueller Seitenkanalüberwachungsmethoden adressiert, welche oft nur isolierte Geräte überwachen können.

Die in dieser Arbeit präsentierten Erbnisse zeigen, dass das zueinander in Kontext Setzen von Ereignissen auf unterschiedlichen Schichten erweiterte Anomalieerkennungsmethoden ermöglicht. Insbesondere erlaubt es die Erkennung von Angriffen und Fehlern, die inkonsistentes Verhalten auf unterschiedlichen Schichten verursachen. Ein Beispiel dafür sind verdeckte Angriffe, in welchen Angreifende gleichzeitig die Eingaben und Ausgaben eines Geräts manipulieren, um so die entfernte Geräteüberwachung irrezuführen. Die präsentierten Methoden können dabei helfen, fortgeschrittene mehrschichtige Anomalieerkennungssysteme zu entwickeln. Durch unseren Ansatz können Anomalien erkannt werden, welche Anomalieerkennungssysteme, die nur Informationen einer einzelnen Schicht verarbeiten, nicht erkennen können.

Contents

Chapter 1

Introduction

Computer networks enable applications to fetch or send data from and to remote sources, or to instruct remote-controlled devices to execute actions. This allows the creation of distributed systems in which multiple devices coordinate via a network to fulfill a task. The device coordination via network creates a relationship between network and device activity. Device activity specifies how a device behaves and there are various influences that affect it. We consider various types of device activity, depending where it occurs. Application activity is caused by local applications that consume local resources for computations and interact with other local applications. Some applications require a connection to remote services to provide their functionality, this causes network activity for the communication. Another type of device activity physical device activity, in which devices change their physical state, e. g. actuators in Cyber-Physical Systems (CPSs) and the Internet of Things (IoT). Due to the change of the physical device state, network-instructed device activity can affect the physical environment. Safety is particularly important in cyber-physical setups to ensure that device activity does not cause harm to humans. Thus, devices must operate as intended to provide a safe, efficient and uninterrupted service [1]. It is, therefore, essential to monitor such systems to reliably and quickly identify abnormal device activity.

This thesis investigates how it is possible to secure networked systems by identifying and monitoring multi-layer relationships. Multi-layer relationships link events from different layers caused by device activity, e. g., an application to the emission of network packets on a host. The relationships we investigate set events from the network layer into the context of applications, i. e. the application layer, and physical device activity, i. e. the cyber-physical layer. Applications like browsers are usually controlled by humans and, therefore, typically show a different and more unpredictable behavior than a fully automated process such as automation in smart homes [2]. Since human behavior is less predictable than automated processes, which follow predefined algorithms and mechanisms and, therefore, allow for a better understanding of monitored systems. Thus, we limit our investigation to highly automated environments with low human interaction. Examples of such environments are Industrial Control Systems (ICSs) or Data Centers (DCs).

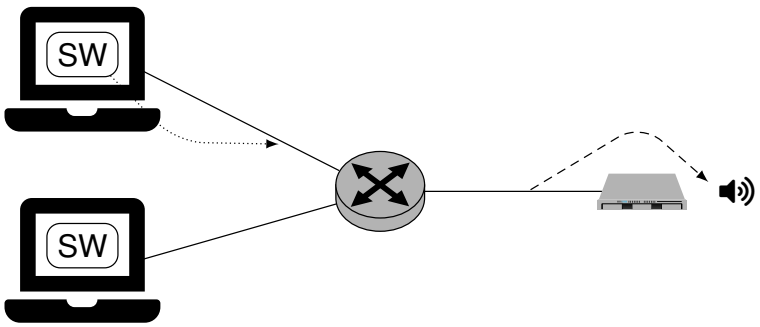

Figure 1.1: The considered scenario of this thesis. Hosts on the left run software (SW) causing network communication. The dotted arrows indicate this relationship. On the right side, a remote controlled server receives instructions which cause it to execute a physical activity. The dashed arrows highlight this relationship. This thesis investigates the described multi-layer relationships shown by the dotted and dashed arrows.

Figure 1.1 depicts the scenario and the multi-layer relationships that we investigate throughout the thesis. Applications running on separate hosts usually cause network activity because they use services from remote services [3]. This causes them to exhibit application-specific packet sequences [4]. The dotted arrow on the left of Figure 1.1 implicates this multi-layer relationship between a devices local applications and network activity. On the right hand side of the figure, devices receive instructions via the network e. g., a power-on command for a server [5], [6], causing physical device activity. Physical device activity often influences the environment, accelerating cooling fans on a server emit noise which affects the soundscape of the environment. This creates a relationship between control traffic and detectable physical changes of the environment due to physical device activity, indicated by the dashed arrow on the right of the figure.

Identifying and understanding multi-layer relationships has various benefits. It allows operators to optimize the network resource depending on how services in distributed systems interact [7], identify the effects of service maintenances [3] or identify abnormal system behavior, also referred to as anomalies [8], [9]. Monitoring that only considers one layer of a system to collect information can be tricked into displaying a wrong system state by attackers, exemplified by the Stuxnet attack [10]. During the Stuxnet attack, adversaries simultaneously manipulated the input and output of remotely managed centrifuges. By altering the physical behavior of the centrifuges to enter a detrimental state while reporting a healthy device state, the adversaries were able to hide abnormal and device damaging activity from the remote monitoring. Such attacks can only be identified by a system which sets events from multiple layers into context, as this context allows to identify mismatching behavior in reported and executed device activity. Therefore, it is essential to understand multi-layer relationships in order to build advanced systems for the detection of anomalies in distributed systems.

We aim at enhancing the security of distributed systems by monitoring device activity to detect abnormal behavior. Various types of relationships link the different types of device activity. In this thesis, we focus on two multi-layer relationships in particular: First (1), between applications and network activity, capturing how applications use the network. One cause of this relationship can be the dependency of applications on remote services [3]. Secondly (2), we investigate

how network activity allows to predict the physical behavior of devices. An example for this relationship is a device changing its physical state on command, such as a remote controlled server startup. Relationships between network activity events, i.e. communication flows, also exist, such as the link between the HyperText Transport Protocol (HTTP) and Domain Name System (DNS) [11] which occurs during web-browsing. However, in this thesis we limit our investigation to multi-layer relationships.

1.1 Research Questions

This thesis answers the main Research Question (RQ)

> *RQ1: How can we improve the security in distributed systems by identifying and monitoring multi-layer relationships?*

We apply our research to networks with low human interaction, such as ICSs, IoT, and DC networks. Thus, we analyze properties of those environments first, including a differentiation to traditional computer networks. Monitoring multi-layer relationships to enhance a system's security requires first understanding those relationships and their characteristics. Recognizing known relationships can be done using formal methods, such as rule sets [11]. Inferring new relationships is more challenging, in particular, when they span across multiple layers and sources [8]. We, therefore, first introduce the multi-layer relationships we consider in this thesis. Since we are interested in monitoring the multi-layer relationships to secure distributed systems, we also analyze how they influence the impact of attacks on the security of a system. We formalize this in the following two sub RQs:

> *RQ1.1: What distinguishes an industrial network from a traditional network?*

> *RQ1.2: Which relationships and types of dependencies exist in networks?*

The answer to these questions defines the relationships which the methods proposed in this thesis aim to identify and monitor.

The thesis then analyzes two different multi-layer relationships identified by the answer to RQ1.2 to answer two more main RQs: The second RQ focuses on the relationship between application and network activity (see left side of Figure 1.1):

> *RQ2: What relationships exist between software running on a host and the host's network activity?*

Operating Systems (OSs) handle the network operation for the applications running on the host. Thus, applications pass the data to the OS, which assembles the packets and sends traffic with applications data or delivers the content of ingress traffic to the correct process. This separates the networking functionality from the application. Due to this separation it is a non-trivial task to efficiently attribute all packets observed at a Network Interface Card (NIC) to applications. While well-known ports and protocols help to identify the type of application, they fail to identify the specific application running on a host. We propose a method to reliably attribute a packet to its associated application. This ability is necessary to be able to analyze

the relationship between an application and its network activity. Based on this method we present a new approach to characterize how applications typically use a network, and identify anomalous behavior. Our work answers RQ2.1 and RQ2.2.

RQ2.1: How can we reliably attribute network activity to applications?

RQ2.2: How can we determine typical application behavior and identify deviations?

The third main RQ of the thesis focuses on the relationship between network activity, i. e. control traffic containing instructions, and physical device activity. Our work also studies the effects of physical device activity on the environment (see right side of Figure 1.1) We formulate RQ3 as:

RQ3: What is the relationship between control traffic in networked systems and physical changes?

To answer this RQ we investigate physical device activity and characterize their unique physical properties. Since devices rarely operate in isolated environments, we analyze how we can identify individual device activity in composite measurements which are affected by multiple devices at once. We then study the relationship between networked control and physical device activity. Networked control allows operators to send control messages via the network to instruct devices to execute commands. We investigate how we can use the relationship between networked control and physical device activity to validate it in composite Side Channels (SCs) and identify anomalies. With our work we answer RQ3.1, RQ3.2 and RQ3.3:

RQ3.1 How can we characterize and identify physical device activity in composite signals?

RQ3.2 What is the relation between network control traffic and physical device activity?

RQ3.3 How can we use network control traffic to validate physical device activity in composite signals?

1.2 Methodology

This thesis has a three part structure. Part I analyzes the necessary background and defines the investigated multi-layer relationships in networked systems. Chapter 2 introduces the architecture, components, and protocols that are typically used in CPSs and DCs, and which are the environments where the proposed methods are applied effectively. The description is the result of a survey on related work. In addition to the environment, it presents Anomaly Detection (AD) as a use case for monitoring relationships to identify outages and attacks. Based on this analysis, we derive how highly automated networks differ from traditional ones (RQ1.1). We analyze to existing work with respect to the goals of this thesis in Chapter 3. We identify shortcomings in the state of the art which motivate the contributions of this thesis.

Afterward, the thesis focuses on the impact of attacks on a system due to multi-layer relationships. Therefore, Chapter 4 introduces a classification scheme of attacks based on the effects have on the security of a system. By creating a taxonomy of how threats affect a system, we further motive to monitor the relationships identified in the answer to (RQ1.2). Based on

these insights we create models that describe multi-layer relationships to detect deviations from expected system behavior in the following parts.

Part II focuses on the relationship between activity of applications and network traffic (RQ2). Chapter 5 designs and implements a method to associate network traffic to processes (RQ2.1). The presented approach enables the required data collection in order to understand the network usage of individual applications on a host. In particular it allows to link network activity to individual applications. The result is a framework that attributes network traffic to applications on the process level.

Chapter 6 uses the framework presented in Chapter 5 to introduce network application-profiles. Network application-profiles characterize the behavior and capture how individual applications use the network. This work results in a method to capture and formalize application behavior, allowing the validation of legitimate application activity. At the end of this chapter, we show how knowledge about the typical network usage of an application can help to identify malicious applications (RQ2.2).

Part III investigates the multi-layer relationship between network activity, i. e. control traffic, and physical device behavior (RQ3). Chapter 7 first evaluates the possibility of identifying physical device behavior in composite signals. Characterizing such behavior enables methods that define and identify the physical characteristics of the tasks a device executes and identify events that make up physical device activity (RQ3.1).

In Chapter 8, we investigate the link between control traffic and the previously identified physical changes. The result is a model that combines the activity in the network and changes in the physical environment (RQ3.2).

Finally, Chapter 9 applies this model to build a cyber-physical Anomaly Detection System (ADS). It leverages the previously identified link to enable reasoning in composite physical measurements, which are influenced by multiple devices at once (RQ3.3). The presented approach demonstrates improved capabilities to detect anomalies in networked systems. This addresses limitations of existing approaches that analyze physical measurements, but are often limited to monitoring isolated devices.

Chapter 10 concludes the thesis by answering the previously stated RQs and explaining possible future work.

1.3 Key Contributions

The thesis has two core contributions. The first contribution is a framework to reliably associate packets and processes. Contrary to previously existing methods, the presented approach allows extensive data collection while being less resource-intensive than other approaches [12], [13]. This thesis is the first to develop and implement network application-profiles based on this data collection, which can accurately describe and monitor the network activity of individual applications, allowing the detection of abnormal behavior.

The second contribution is a cyber-physical ADS, which combines network control traffic with physical measurements. The approach enables the validation of individual device behavior in composite physical measurements. An automatic validation of physical device activity enables operators to identify issues where the device behavior differs from the reported activity of a device, e. g. due to attacks [10].

The presented contributions led to various of publications listed below. While each chapter has a summary on which contribution it builds upon, the following list provides an overview of them:

Ch. 4: [RQ1.2]: L. Wüstrich, M. Pahl, and S. Liebald, "Towards an Extensible IoT Security Taxonomy", in *IEEE Symposium on Computers and Communications, ISCC 2020, Rennes, France, July 7-10, 2020*, IEEE, 2020, pp. 1–6. DOI: 10.1109/ISCC50000.2020.9219584

Ch 5 and Ch. 6: [RQ2.1] and [RQ2.2]: L. Wüstrich, M. Schacherbauer, M. Budeus, D. F. von Künßberg, S. Gallenmüller, M. Pahl, and G. Carle, "Network Profiles for Detecting Application-Characteristic Behavior Using Linux eBPF", in *Proceedings of the 1st Workshop on eBPF and Kernel Extensions, eBPF 2023, New York, NY, USA, 10 September 2023*, ACM, 2023, pp. 8–14. DOI: 10.1145/3609021.3609294

Ch. 7: [RQ3.1]: L. Wüstrich, S. Gallenmüller, M. Pahl, and G. Carle, "AC/DCIM: Acoustic Channels for Data Center Infrastructure Monitoring", in *2022 IEEE/IFIP Network Operations and Management Symposium, NOMS 2022, Budapest, Hungary, April 25-29, 2022*, IEEE, 2022, pp. 1–5. DOI: 10.1109/NOMS54207.2022.9789839

Ch. 8: [RQ3.2] and [RQ3.3]: L. Wüstrich, L. Schröder, and M. Pahl, "Cyber-Physical Anomaly Detection for ICS", in *17th IFIP/IEEE International Symposium on Integrated Network Management, IM 2021, Bordeaux, France, May 17-21, 2021*, T. Ahmed, O. Festor, Y. Ghamri-Doudane, J. Kang, A. E. S. Filho, A. Lahmadi, and E. R. M. Madeira, Eds., IEEE, 2021, pp. 950–955

Ch. 9: [RQ3.2] and [RQ3.3]: L. Wüstrich, S. Gallenmüller, S. M. Günther, G. Carle, and M. Pahl, "Shells Bells: Cyber-Physical Anomaly Detection in Data Centers", in *NOMS 2024 IEEE Network Operations and Management Symposium, Seoul, Republic of Korea, May 6-10, 2024*, IEEE, 2024, pp. 1–10. DOI: 10.1109/NOMS59830.2024.10575124

The author further contributed to the following publications on the security of cyber-physical environments and network controlled systems:

M. Barbeau, J. García-Alfaro, C. Lübben, M. Pahl, and L. Wüstrich, "Resilience via Blackbox Self-Piloting Plants", in *Proceedings of the 29th Computer & Electronics Security Application Rendezvous co-located with the 7th European Cyber Week (ECW 2022), Rennes, France, November 15-16, 2022*, G. le Guernic, Ed., ser. CEUR Workshop Proceedings, vol. 3329, CEUR-WS.org, 2022, pp. 35–46

A. Piccoli, M. Pahl, and L. Wüstrich, "Group Key Management in Constrained IoT Settings", in *IEEE Symposium on Computers and Communications, ISCC 2020, Rennes, France, July 7-10, 2020*, IEEE, 2020, pp. 1–6. DOI: 10.1109/ISCC50000.2020.9219619

S. Paiho, J. Kiljander, R. Sarala, H. Siikavirta, O. Kilkki, A. Bajpai, M. Duchon, M.-O. Pahl, L. Wüstrich, C. Lübben, *et al.*, "Towards Cross-Commodity Energy-Sharing Communities – A Review of the Market, Regulatory, and Technical Situation", *Renewable and Sustainable Energy Reviews*, vol. 151, p. 111 568, 2021

Part I

Background and Analysis of the Studied Environment

Chapter 2

Background

This chapter introduces the environments we consider in this thesis. Since human behavior when interacting with compting resource is often less predictable than pre-defined algorithms, we focus on environments with high automation. Examples of such environments are Industrial Control Systems (ICSs), the Internet of Things (IoT), and Data Centers (DCs). Devices in those environments operate most of the time without human interaction and are controlled via a network based on pre-defined algorithms. This allows us to study multi-layer relationships in environments with predictable behavior. On one hand we analyze how applications for device control use the network. On the other hand we also investigate relationship between network traffic and physical device behavior. For this relationship, the cyber-physical nature of ICSs and IoT devices allows us to analyze how networked control is correlated with physical device activity. We further apply our research to DC environments where physical changes that occur when devices execute instructions are a byproduct of device activity. This shows that the proposed methods are also applicable to other remote controlled systems in which devices do not have the primary task of physically interacting with the environment.

While we focus on highly automated environments due to their predictability, the methods presented in the thesis can also be applied to environments with humans in the loop. However, the proposed models are less accurate in such environments due to the unpredictability of human behavior. The following sections introduce common components, protocols, and the typical network architecture of the considered environments. We further introduce Anomaly Detection (AD) as a use case for improving the security of systems by monitoring the studied multi-layer relationships.

2.1 Networks with High Automation

Examples of highly automated networks are ICS, the IoT, and DC. While those networked environments seem to be different on a first look, they share various characteristics. Most of the traffic in highly automated networks is the result of device-to-device communication,

e. g. polling [22] or publish-subscribe patterns [23]. We refer to this traffic as machine-to-machine (M2M) communication. This automation is one example of autonomous application behavior that causes network activity without human interaction, which we study in Part II of the thesis. Further, operators typically manage devices in those setups remotely [1], [5], allowing to control devices via the network. The focus of ICS and IoT devices is to manage physical processes, setting them apart from DCs. However, DC operators also need to manage the physical DC environment [24]. Some management activities, such as instructing devices to turn on and off on demand also influence the DC environment as a side-effect, similar to the physical activity of ICS and IoT devices. Combined with the remote management of devices this allows to study the relationship between control traffic and physical changes in Part III of this thesis. We introduce the properties of ICSs, IoT networks, and DCs in the following sections.

2.1.1 Industrial Control Systems

ICSs are typically networked control systems, consisting of distributed sensors and actuators interacting with the physical world [25], [26]. Devices of networked control systems communicate via a network and are often remote controlled [27]. The goal of an ICS is typically to manage a physical process, such as water treatment facilities, the smart grid or production facilities in factories [26], [28]. Therefore, Industrial Control System (ICS) are often a type of Cyber-Physical System (CPS) [26].

The main goals when operating an ICS is safety and availability [26]. To ensure the correct functioning of ICS devices and minimize human risk, ICSs are highly regulated [1]. As a result, ICS need to fulfill several requirements, such as real-time communication between devices and permanent monitoring. This makes ICS different from other network types in less regulated environments, which do not need to meet such requirements. The methods we study in this thesis enhance the monitoring capabilities of systems like ICSs by setting events from multiple layers into the context of each other.

Most devices in ICSs are distributed sensors and actuators. This device composition makes ICS fundamentally different from traditional IT networks. While ICS devices are often remote-controlled, traditional IT networks typically connect devices such as laptops, smart phones and desktop computers which are influenced by local user input. ICS components are often highly specialized, contrary to multi-purpose devices in traditional networks. Sensors measure specific properties like power intake, pressure, temperature or audio. Similarly, actuators execute distinct actions, such as opening or closing a valve. In addition to specialized devices, the adoption of traditional IT components like office computers into ICS networks is increasing. This makes ICS networks a complex environment with a heterogeneous device composition. ICS follow a strictly compartmentalized architecture to secure the overall system [26]. This thesis focuses on the part of the network in which control devices communicate with sensors and actuators (Supervisory Control And Data Acquisition (SCADA) zone in Figure 2.2). In particular we analyze how the remote control influences physical device activity.

ICSs often use specialized and often proprietary communication protocols [1], [26]. Examples are the S7 protocol [29], MODBUS [30], Distributed Network Protocol 3 (DNP3) [31] or OPC-

UA [32], which fulfill specific requirements of different environments. In order to enable larger networks, a variety of those protocols can be embedded into the TCP/IP stack [1]. This stack uses Ethernet on the data link layer, the Internet protocol on the the network layer and either the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) on the transport layer. In this case, the used industrial protocol is in the payload of the transport layer protocol. Using the typical network stack allows managing distributed control systems that span a large geographical area, such as the smart grid. While the methods presented in Part II analyze how applications use the network, the methods presented in Part III analyze the instructions of the used industrial protocol.

Communication in ICS has distinct characteristics such as short packet sizes and periodic information polling or information publishing [22], [32], [33]. The protocols typically follow a primary-secondary architecture. For networked control, central control units send commands to sensors and actuators, which then execute the instructed activity or reply with the requested data. In addition to controlling devices, ICS networks typically exhibit periodic traffic due to constant polling of device states [34]. Primary devices send explicit instructions and requests to the secondary devices, which then execute them [35]. Due to a typically static setup, devices in ICS often have a static communication partner set [36], [37]. Part III of the thesis investigates the relationship between control traffic to remote controlled devices and their physical activity.

2.1.2 IoT Networks

Similar to ICSs, in the Internet of Things (IoT) sensors and actuators communicate and coordinate via a network and interact with the environment. The IoT has several application scenarios, e. g. activity trackers, smart homes. Smart homes allow users to combine multiple devices to automate processes in their home [38]. There are also industrial applications of the IoT in which devices coordinate in large networked systems, also called the Industrial Internet of Things (IIoT) [39], [40]. The IIoT focuses on the communication between industrial devices [39]. This includes the command and control of managed devices which this thesis focuses on.

Since the IoT is less regulated than ICSs, services in IoT typically need to provide fewer guarantees to their users than in ICS. The remote control of IoT devices uses protocols that run on top of the TCP/IP stack, such as the Message Queuing Telemetry Transport (MQTT) protocol [41]. Since IoT devices can be distributed over larger geographic areas, some IoT devices use LoRa [42], ZigBee [43] or 5G for the transmission of information. Those transmission channels influence properties such as the reliability or speed of the communication. Depending on the application environment, the communication in the IIoT needs to meet similar guarantees as in ICS [39]. IoT setups share similarities to ICS which we outline below. Devices receive and send measurements and commands via a network to instruct actuators to execute actions. Further, many IoT devices have the purpose of sensing or manipulating the environment. This implies that the relationship between physical changes and network traffic also exists in IoT environments [38]. Part III also analyzes how the absence of timing guarantees affects the accuracy of models that capture the relationship between control traffic and physical device behavior. This provides insights to which extend it is possible to apply the methods we investigate in this thesis to IoT environments.

2.1.3 Data Center Networks

In addition to ICSs and the IoT, we consider Data Centers (DCs) as an environment to apply our concepts. DC devices typically do not have the primary goal of interacting with the physical environment. However, some activity, such as booting a server also physically influences the environment. We show that our methods are also applicable beyond CPSs to improve the security of DC environments by monitoring multi-layer relationships.

DCs provide infrastructure for services and applications, e. g. computing power and storage [7]. Various services like Content Delivery Networks (CDNs) or video streaming platforms rely on DCs.

The infrastructure in a DC consists of two types of hardware. The first type is servers and network hardware at a large scale [7]. Those devices provide storage, computational power, and storage to DC users. The second type is infrastructure hardware, which physically manages the hosted hardware and its environment. Infrastructure hardware includes climate control for temperature and humidity, Air Condition (AC), airflow, and power control [24]. Due to the large scale of DCs, even simple operations such as power cycling a cluster of servers require an orchestration to avoid side effects such as power surges.

To run a DC, operators meticulously monitor the hosted hardware and network [16]. They do so by using additional software on DC devices, such as `checkmk` [44] or Icinga [45]. Tools like the Intelligent Platform Management Interface (IPMI) allow operators to remotely configure and access devices, even when they are turned off. Similarly, monitoring the physical properties of devices works by either using sensors on the hosted hardware itself or external sensors from which the information is collected in an additional management system. Using this combination of remote tools and distributed sensors enables the "lights out" [5] management paradigm with rare physical human interaction that many DCs follow. Depending on the instructed actions to DC devices, such as power-off or power-on, device activity also influences the environment, similar to actuators in ICSs and the IoT.

The monitoring of the physical properties of DCs is also essential for a constant and uninterrupted operation [24]. The methods presented in this thesis can also help to improve the monitoring of DCs by correlating information from management traffic with physical measurements. This implies the application of the presented methods extends beyond CPSs to other remote managed systems. We, therefore, apply our methods to a DC environment in Part III of this thesis.

2.2 Structure

The following sections introduce a logical structure and network topology of the considered environments. The logical structure helps to understand the purpose of devices and their communication. In addition, the logical structure helps to define multi-layer relationships. The network topology focuses on the implementation of the connectivity in the environments, highlighting vantage points to collect the relevant data.

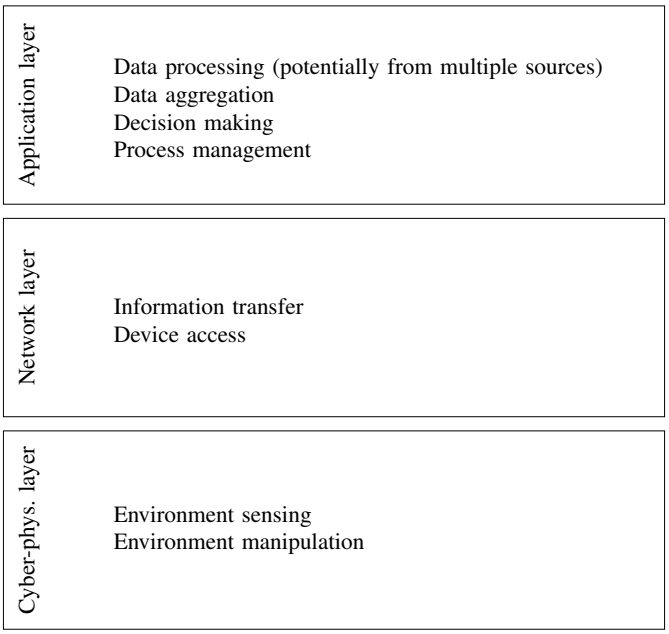

Figure 2.1: Three logical layers of highly automated systems, adapted from [46]

2.2.1 Logical Structure

The logical structure captures the purpose of the hardware and software running on devices. Like other works [46], [47], we partition the structure into three layers, (1) *cyber-physical layer*, (2) *network layer* and (3) *application layer*. The network layer in our logical structure combines the network architecture and communication protocols.

Figure 2.1 visualizes the layers of the logical structure. This thesis investigates the relationships between activity on different layers to make monitored systems more secure by identifying anomalies.

The *cyber-physical layer* acts as interface between the physical and cyber-world. The cyber-world encompasses all digital entities, i. e. digital sensor values, communication and data processing. The cyber-physical layer consists of sensors and actuators. Sensors and actuators ar I/O devices which convert physical quantities into electronic signals and vice versa [48]. This enables them to sense environmental properties, e. g., temperature. Actuators perform actions to influence are actively influence the environment when instructed, e. g., opening or closing valves in cooling systems or accelerating fans when turning on a server. In ICS and IoT, actuators are typically only built to execute one task. Since they do not handle complex computations but only execute a limited set of actions, they are often limited in computational resources [49]. In DCs, the cooling fans on a device have a similar purpose as actuators in CPSs Depending on the measured temperature, e. g., emitted by a running Central Processing Unit (CPU), fans accelerate to dissipate the heat in a server chassis. Changing the fan speed also has other side effects, such as changing the soundscape. The lowest layer in Figure 2.1 represents this layer.

The *network layer* implements the connectivity and enables information transfer between devices. In ICS and IoT networks, this includes connecting devices on the physical layer and control units and providing access to devices. Some environments rely on a middleware to simplify device interaction [32], [50]. In ICSs, the network layer implements two types of communication. First, there is communication between Programmable Logic Controllers (PLCs) and sensors and actuators on the physical layer. While this communication typically transfers information using serial communication, some protocols also run over Ethernet [1]. The second type of communication occurs between PLCs and other ICS components, such as central control devices. This communication is management traffic and typically relies on Ethernet-based protocols and the traditional TCP/IP stack [1], [32]. The communication enables the data transfer which includes sensor measurements and data traffic, as well as remote control via management traffic. This traffic is typically described as Supervisory Control And Data Acquisition (SCADA) [1]. In DCs, the network layer implements the connectivity between hosted devices and enables remote management capabilities. Typically there is a separation between management traffic and other traffic of DC users [51]. This separation can be logical (In-Band) or using a physical different network (Out-of-Band) [52]. In thesis the introduced methods focus on the management traffic in DC networks. We present methods to set events observed on the network layer into context of events on the other layers to identify abnormal system behavior.

The *application layer* implements the control and management of devices and the overall system. In ICSs and the IoT, the application layer is responsible for processing measurements from physical layer devices. Therefore, the application layer aggregates data from those devices, connected via a to process them and make decisions based on the information. For example, decisions include sending commands to actuators on the physical layer, which then change the environment and future measurements. In DC environments, we consider central monitoring and device management as the application layer. IoT and DC environments differ in this aspect from ICSs, since most devices can run their own control logic. In ICSs, applications are run on PLCs and other central control units. These receive sensor measurements and manage actuators on the physical layer. It is possible that ICSs implement a hierarchical structure in which one PLC manages multiple PLCs [1]. The application layer is also responsible for monitoring the devices, which runs on centralized nodes. These provide operators with a view of the state of the system, allowing them to make adjustments to the control logic or to provide instructions to individual devices.

The methods proposed in this thesis aim to monitor link events from the network layer to other layers. We refer to this link as multi-layer relationship. In Part II, we focus on the link between the application layer and the network layer, while Part III focuses on the relationship between the network layer and the cyber-physical layer.

2.2.2 Network Structure

ICSs, DCs and IoT networks typically have a hierarchical structure [7], [33], [53]. In IoT networks and ICS, physical layer devices communicate with aggregation devices, which forward the information to central control units. DCs arrange the network for hosted devices also in a

hierarchical structure to optimize the resilience and efficiency of the network for device communication.

On the lowest layer of the hierarchy are devices belonging to the cyber-physical layer of our logical structure. In ICSs and IoT networks those devices are typically sensors and actuators. In DCs those are compute and storage nodes that provide services and are remotely managed by operators. In ICS, the second layer of the hierarchy consists of PLCs, which manage the attached devices. In IoT networks, the second layer is typically a central hub connecting the devices in the network. The hierarchy in DC networks is mainly induced by the network topology. Since most devices are multi-purpose, there is no strong primary-secondary relation between DC devices in general. However, management devices have strict control over the deployed devices. Finally, it is common to enable some connection to distant devices via a gateway, allowing remote management. For example, this makes ICSs discoverable in Internet scans [54]. While this access enables remote management of the system, it also introduces new threat vectors. One method to reduce the risk of an attack is to use a strict segmentation of networks, e. g. separating office networks from production networks in ICS [1].

Figure 2.2 visualizes the generic hierarchy for an ICS. It shows the sensors and actuators on the lowest layer managing a physical process. Those devices have a physical connection to PLCs in which they exchange information via serial communication. Each PLC has a connection to the SCADA network, enabling it to communicate with other PLCs and central control units for monitoring. In addition to other PLCs, the network hosts devices that allow operators to monitor and mange the process via a Human Machine Interface (HMI). ICSs typically also use a data historian that records reported values and instructions over time [1]. A gateway separates the SCADA network from remote networks. These can either be a directly attached office network or the internet. ICSs often have a strict segmentation of the inner zone into subnetworks [26]. Figure 2.2 shows a simplified topology which does not reflect this segmentation of the inner zone.

This structure influences the data collection for the methods that analyze the relationship between network activity and device behavior. On the one hand, we analyze how applications running on devices influence network activity (Part II). In order to associate packets with local applications, this data collection needs to take place locally on individual hosts, e. g. the HMI. On the other hand, analyzing the relationship between control traffic and the changes on the cyber-physical layer (Part III) is possible from different vantage points. Since the control traffic specifies commands and destination devices, it is possible to collect information about network events from central points in the network, such as central switches in SCADA zone in Figure 2.2 [7], [33]

2.3 Possible Relationships

This thesis's central Research Question (RQ) is: *How can we improve the security in distributed systems by identifying and monitoring multi-layer relationships?* Thus, we introduce the relevant multi-layer relationships in the following by using our logical structure (Section 2.2.1).

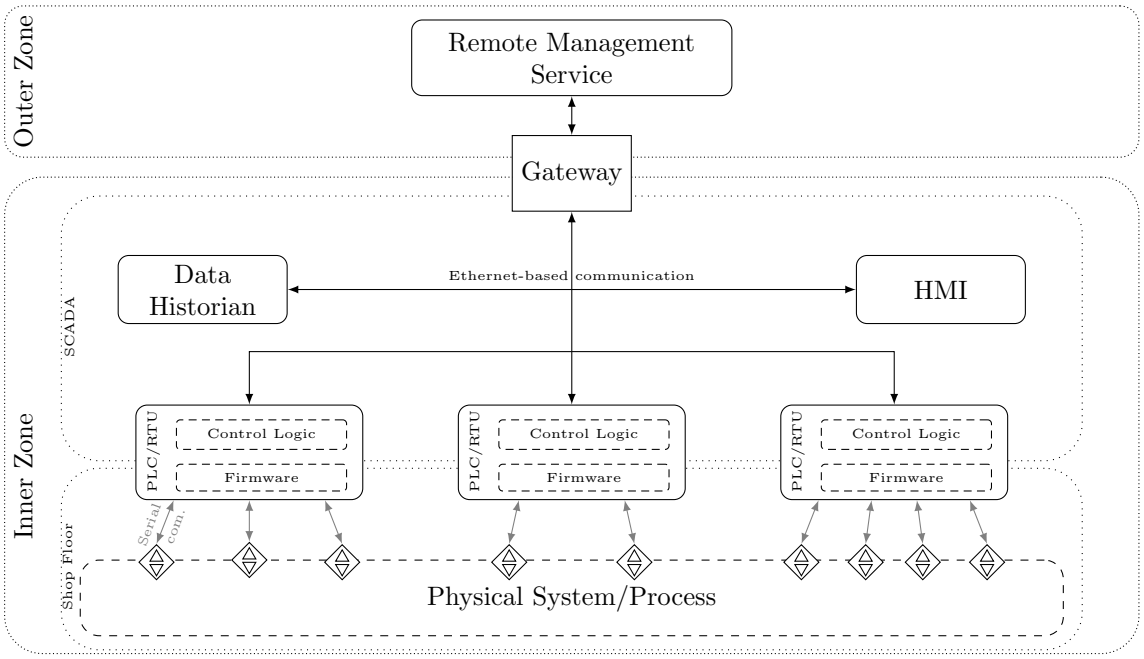

Figure 2.2: Generic topology of ICSs based on [26]

This section provides a general overview of relationships in highly automated networks. If a relationship is known beforehand, formal methods such as a rule set can help identify them during live operation or in network traces [11]. However, finding new relationships in a given data set is difficult as no formal identification method exists.

A variety of relationships occur within the network layer of our logical structure. The technology stack enables communication and also causes relations between flows. Protocols like HyperText Transport Protocol (HTTP) rely on a successful name resolution for a Fully Qualified Domain Name (FQDN) that a user provides. In this case, Domain Name System (DNS) queries and replies precede the flow containing the HTTP request, creating a relationship between HTTP and DNS flows [11]. In bi-directional communication, the first flow has a causal relationship to the flow containing the answer. Those relationships can also imply dependencies on other layers, e. g. of a browser on a local name resolution service. Since this thesis focuses on multi-layer relationships, we do not investigate those relationships.

The first multi-layer relationship is between applications and the associated network activity on a host. Thus, this relationship connects activity on our logical structure's network and application layers. Applications can cause network activity if they rely on remote services to execute tasks [3], [9]. Another reason for applications causing network activity is fetching remote information, e. g. sensor values in an ICS or server states in DCs. An application's network usage can exhibit unique features. Example are used communication protocols, periodic behavior, message patterns [4], [35], or flow sequences [11]. These properties can define typical application behavior that also depends on the application state. Knowledge about those characteristics

defined via the link between applications and network activity can help to correlate seemingly unrelated flows on the network layer. We investigate this relationship in Chapters 5 and 6.

The second multi-layer relationship type links control traffic on the network layer and physical device activity on the cyber-physical layer. In the setups we consider in this thesis–ICSs, the IoT and DCs–devices receive their instructions via the network. Those instructions can lead to device activity in which devices change their physical state, which causes measurable changes in the environment. For example, if a server in a DC receives the instruction to boot, it also changes its physical state. While the server starts, it emits sounds while its cooling fans accelerate, thus influencing the DC soundscape. Therefore, there is a connection between the control traffic instructing the server to boot and the change in the soundscape. We analyze this multi-layer relationship in Part III of the thesis.

We apply the methods in this thesis to increase the security of network-controlled systems by identifying anomalies. Therefore, we introduce AD in the following section.

2.4 Anomaly Detection

One major aspect of securing systems is the capability to detect unexpected states and unusual behavior [55]. Knowledge about relationships is an advantage to network and system operators and is key in order to identify such abnormal system states. Even after detecting abnormal behavior, knowing about relationships in devices is necessary to identify the root causes of errors. We apply our methods to identify and monitor multi-layer relationships that allow systems to be modeled in better detail for AD. The benefit of our approach is the ability to identify anomalies that Anomaly Detection System (ADS) that only focus on one layer cannot detect.

Deviations from the expected behavior of a system are called anomalies [56]. Common causes for anomalies are attacks, faults, or failures on hard- and software level. Depending on components malfunctioning due to measurement errors, faults, or attacks, a system can deviate in different ways. While faults typically show up in the monitoring, e. g. by a sensor reporting unusual values, attackers might try to obfuscate their activity [2], [10], [57]. However, the activity of attackers can also have side-effects that can be detected, e. g. due to slower response times [49]. Regardless of the cause, we aim to identify abnormal behavior by setting events from multiple layers into the context of each other.

Anomaly Detection Systems (ADSs) use algorithms that analyze observable features which describe the state or behavior of a system [56]. Based on the change or the value of these features, an ADS determines if a system operates normally or it performs in unintended ways, i. e. anomalous. Depending on the type and source of an anomaly, some features can indicate anomalies on some layers earlier than on others [8].

Most ADSs analyze information from one source. For example, Network Intrusion Detection Systems (NIDSs) only analyze network traffic. However, combining information from multiple sources allows for a faster detection of anomalies [8]. The combination of features from multiple layers also allows events to be set into context, providing enhanced reasoning capabilities.

However, the fusion of data from multiple sources for AD is challenging [8], e. g. due to clock synchronization issues.

There are two approaches to AD. AD algorithms either learn the properties of "normal" behavior, or behavior of known anomalies [56]. In the case an ADS knows "normal" system behavior, the algorithms determine an anomalous state by identifying deviations from the normal state the monitored system typically exhibits. In case the AD algorithm knows the characteristics of anomalous states, it identifies anomalies as soon as the monitored system shows similar behavior to an anomaly. While an AD method that has knowledge about "normal" behavior has the advantage of identifying all unknown behavior, operators often need to invest additional time to identify the type of the anomaly. In addition, those methods need to be retrained in case the definition of "normal" behavior changes. On the other hand, AD algorithms of the second type can classify anomaly types, they cannot identify anomalies that they were not trained for.

An ADS implements AD methods to identify anomalies. ADSs have a *model generation phase* and an *anomaly detection phase* [56], [58] which we describe in the following.

2.4.1 Model Generation Phase

The model generation phase creates a reference model of a system. The reference model describes a system's expected behavior, or characteristic behavior of known anomalies, depending on the AD approach. The reference model typically consists of a set of typical value ranges for features, rules, or a Machine Learning (ML) model that describe regular or anomalous behavior.

There are two approaches to generate reference models. The first method uses statistics and heuristics. Such methods require data collection from normal system behavior or behavior during anomalies. They extract information from collected data and apply different heuristics and statistical methods to extract key characteristics of behavior. *ML* and statistic based ADSs such as [59]–[62] use this approach. Such methods rely on data from a correctly functioning system or a labeled dataset of anomalies. To learn "normal" behavior, those methods require a benign dataset. If anomalies exist in the collected traces or an attacker manipulates the training data, the method learns these as expected behavior and cannot detect such behavior.
In many cases, creating a labeled dataset with anomalies is not feasible. On the one hand, creating real-world occurrences of system behavior during anomalies is not possible in production systems without interrupting the operation. On the other hand, it is challenging to automatically enhance an existing dataset with the corresponding labels. Finally, statistical approaches disregard rare occurrences as outliers. This can lead to a large number of False Positives (FPs), e. g. in which a system considers a rare but legitimate condition as an anomaly. Examples of such a behavior are device maintenance or updates. Furthermore, in case the AD has learned the complete system behavior, a change in the system requires a new complete training to incorporate the changes.

The second method is specification-based AD [25], [63]. Specification-based methods process existing documentation to create reference models. The documentation describes the monitored system, including deployed devices, expected interactions, and communication sequences, and typically seen value ranges for commands and measurements. From this information, an ADS

extracts features for its reference model.
Specification-based approaches to AD also have challenges. To extract the reference model, the methods need to process documentation from several sources. These sources potentially have different formats and need to be fused into a single reference model. In addition, the documentation reflects the specification of a system at a specific point in time. In case an undocumented change in the system occurs, it is challenging to incorporate such a system change into the reference model and keep it up to date.

2.4.2 Anomaly Detection Phase

During the AD phase, ADSs use a previously generated reference model to identify anomalous behavior during system operation. To identify anomalies, an ADS compares exhibited features or sequences in a monitored system to the values in the reference model. Based on the comparison outcome, the ADS raises an alarm. The comparison can happen in real time, as a sliding window, or in bulk at a specific point in time. [56] The mode of operation depends on the monitored data and goals of the AD.

2.4.3 Measuring the Performance of Anomaly Detection

There are several approaches to measure the performance of ADS [64]. Those measurements use statistics of each classification for their performance assessment. Each classification of an ADS is in one of four classes:

1. True Positive (TP) implies that the system correctly classifies a sample as an anomaly
2. False Positive (FP) is the result of falsely classifying a benign sample as anomaly
3. True Negative (TN) correctly classifies normal behavior, i. e. the absence of an anomaly
4. False Negative (FN) fails to detect an anomaly and classifies it as normal.

The rate of those metrics is typically used to assess the performance of an ADS. Well-performing ADS have high correct values and low false classification. There is typically a trade-off between a low FP rate and a low FN rate. An ADS, which is sensitive to small variations, potentially classifies benign observations as abnormal if they deviate from the expectation, leading to a lot of FPs. On the other hand, ADSs, which are insensitive to small variations, may classify an abnormal entry as normal behavior, leading to a high FN rate.

In order to set these rates into context, it is common to use the *accuracy*, *precision*, and *recall* metrics. The accuracy measures the overall correctness of all classifications:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

The *precision* measures the trustworthiness of anomaly classifications. It is calculated by

$$precision = \frac{TP}{TP + FP} \tag{2.2}$$

Finally, the *recall* measures how well a system detects anomalous data samples

$$recall = \frac{TP}{TP + FN} \tag{2.3}$$

2.5 Key Results

We introduced the environments we consider in this thesis and methods to structure them. We split the environment into a cyber-physical, network, and application layer. ICS, IoT, and DCs are typically remote-controlled environments that have a hierarchical network topology. In this thesis, we analyze multi-layer relationships, correlating events on the network with events on neighboring layers, i. e. on the cyber-physical and application layer. Those are caused by the network usage of applications, as well as due to control traffic of remote control that results in physical device activity. Due to their reliance on pre-defined algorithms and rules, it is easier to understand relationships in highly automated systems than environments, which are influenced mainly by human activity.

The network serves as a carrier of information to enable remote control. Due to its central role, events on the network can relate to events on other layers. This allows the monitoring of multi-layer relationships of a network-controlled system. Knowledge about these relationships makes it possible to create advanced models that describe a system's behavior. In this thesis, we propose methods to capture and model these relationships. We investigate the relationship between applications and their network usage in Part II. Due to the cyber-physical nature of the devices, the networked control also relates to physical activity. We analyze this relationship in Part III.

We apply our methods to secure network-controlled systems. A key aspect of secure computer systems is the capability to detect unintended behavior, e. g. via AD. Thus, we use the concepts proposed in this thesis to build more accurate ADSs, which are able to detect anomalies which cause inconsistencies in the reported and perceived activity on the different layers. Currently used ADS that only process information from one layer fail to identify such anomalies.

Chapter 3

Related Work

This chapter puts the contributions of existing work into the context of the goals of this thesis. The presented related work is a selection of related work of the individual contributions of this thesis. In contrast to the other chapters, which individually discuss related work in the specific context of their contribution, this chapter sets the considered papers into the context of the overall goal of this thesis. We classify related work into two types. The first type includes work directly or indirectly focusing on the multi-layer relationship between the network and application layers. The second type focuses on the multi-layer relationship between the network and cyber-physical layers of our logical structure. We discuss these approaches separately in the following sections.

3.1 Relationship Between Application and Network Activity

A variety of related work focuses on how hosts and applications use the network. This analysis covers the multi-layer relationship between our logical structure's application and network layers. We investigate this relationship in Part II of this thesis. We deem contributions relevant that link network activity with applications. The application domain considered in related work includes root cause analyses, dependency detection, and Anomaly Detection (AD) in various environments. Similar to most of the presented related work, we apply the methods in this thesis to the domain of AD. There are two types of works in this domain: The first type only relies on analyzing network traffic to reason about the activity of applications. The second type explicitly fuses information from network traffic observations and applications for enhanced reasoning about a system's behavior.

3.1.1 Network-only Approaches

Karagiannis et al. [9] present an approach to identify applications based on the payload in network flows. The evaluation shows that the presented framework could identify the application

type of most of the flows. Dai et al. [65] present an approach to fingerprint applications via characteristic network flows. The authors combine typical destination addresses with characteristic HyperText Transport Protocol (HTTP) header payloads to associate network traffic with applications. A drawback of these approaches is their reliance on the payload of packets. This information is inaccessible in encrypted network traffic. Therefore, we design our methods to work without access to payload information (Part II) or propose extensions that allow the decryption of traffic for parsing (Part III).
Asai et al. [11] present traffic causality graphs to identify causal relationships between flows on the network layer. For examples, traffic causality graphs allow to uncover that a HTTP request caused a preceding DNS query. The method uses temporal information and the IP-5-tuple for its analysis. The authors mention application profiling as a use case, allowing the identification of typically occurring flow combinations during the usage of an application. Piet et al. [4] show that it is possible to identify application layer protocols by analyzing packet sizes in flows on the network layer. Protocols cause unique patterns that allow their detection and inference of information about the application that causes them. This highlights that there can be unique features that applications exhibit when they create network activity. Pontes et al. [66] present a method to formalize typical network flows in an environment. The method identifies common combinations of flow features, such as typical protocol and port combinations. By finding common combinations, the strategy can identify unusual flows that mismatch typically occurring communication. As a result, the method enables to capture the characteristics of typical communication flows in a given dataset and identify abnormal flows.

Those approaches show a correlation between applications and activity on the network layer. However, while the presented methods help to characterize application behavior on based on network layer activity, they do not directly link network traffic to applications. A reliable link between network traffic and applications is essential for accurate reasoning about the relationship between application and network activity.

3.1.2 Application-Network Approaches

Popa et al. [3] analyze dependencies of applications on a host by processing traffic on the network layer. The authors create the link between applications and network traffic by periodically polling the Operating System (OS) to identify the relationship between ports and processes. Based on this link, the authors automatically identify commonly addressed hosts from the gathered data. This allows for the discovery of application dependencies to services on remote hosts. A shortcoming of the polling-based approach is that it can miss short lived-events that occur between polls. Haas et al. [67] combine distributed local information from the application layer and corresponding network activity. Based on this data collection, the authors present an extension to the zeek Intrusion Detection System (IDS) [68] for classifying flows. This approach increases the performance of the IDS to attribute flows to applications. Based on these insights, the authors created an experimental plugin [69] for zeek that implements the extension. The plugin uses similar data collection techniques to our approach presented in Chapter 5. Their work shows that correlating information across multiple layers helps to enhance the performance of Anomaly Detection System (ADS). Identifying the relationship between

individual applications and their associated network activity is not a goal of the presented related work. Running applications in isolated environments, such as virtual machines or containers, is also possible. The isolation of applications makes it possible to implicitly attribute network activity to them by attributing it to containers. Cilium Hubble [70] and Tetragon [71] use the extended Berkeley Packet Filter (eBPF) to monitor the activity of containers. This allows to identify abnormal application behavior if they run in such isolated environments. However, if multiple applications run within the same container, it is difficult to analyze which application is responsible for network activity. Our approach, allowing to directly link network packets and individual applications on a device, is presented in Chapter 5.

3.1.3 Summary

The related work presented in this section highlights that the multi-layer relationship between activity on the application and network layers of our logical structure can provide enhanced insights into the behavior of systems. Various use cases exist that require knowledge about the multi-layer relationship between applications and their network usage. Examples are dependency detection, network capacity planning, fault localization, and AD. Approaches that rely purely on network layer information lack a ground truth about which flows belong to individual applications. This thesis addresses this limitation by presenting a method that allows us to establish this link automatically and reliably in Chapter 5. Our method allows us to improve the reasoning capabilities of the above-presented approaches by automated fine-granular data collection. Existing methods that directly link network activity with applications highlight the benefits of this advanced information collection. However, to the best of our knowledge, research that analyzes the link between applications and network activity to characterize and formalize typical application behavior is missing. We address this shortcoming in Chapter 6 by introducing a method to formalize the typical network activity of applications.

3.2 Relationship Between Network Activity and Physical Behavior

The second class of related work focuses on the multi-layer relationship between physical device activity and remote control. We investigate this relationship in Part III of this thesis. Analyzing physical device activity by measuring physical attributes and changes is a standard approach in SC monitoring [72], [73]. Those monitor physical device behavior based on external measurements, such as power intake. The second body of related work correlates reported events with physical measurements to validate reported activity, which we refer to as event verification methods. Those methods allow to identify mismatching reported and and cyber-physical layer activity. Finally, we present attacks in which adversaries trick remote monitoring into perceiving a wrong physical state, as those are the types of anomalies which we aim to detect by using our methods.

3.2.1 Side-Channel Monitoring

Side Channel (SC) monitoring analyzes the relationship between digital and physical device activity to identify normal and anomalous behavior. Bayens et al. [74] present an approach to monitor the activity of an isolated 3D printer. The authors use an acoustic SC to monitor the motions of the printer during the printing process. The approach analyzes if the physical motions of the printer match the locally provided instructions to reason about the structural integrity of the printed object. This shows that audio, which is a byproduct of various physical operations, can be used to monitor device activity and motivates our use of acoustic signals as SC in Part III as a guiding example in our application to a Data Center (DC) environment. Han et al. [73] present an approach to measure the Electromagnetic (EM) of processors on Programmable Logic Controllers (PLCs) to validate the control flow of local code execution. A difference to our work is that those approaches aims at verifying the execution of locally executed code, while our work focuses on the execution of remotely instructed commands delivered via a network. Further, they analyze the activity of isolated devices to attribute changes to the monitored entities. We aim at addressing this limitation by using additional sources of information to allow to perform SC monitoring in shared environments.

Other approaches to use physical measurements to monitor a system's activity exist, particularly in DC environments. Levy et al. [24], Rogriguez et al. [75] and Ahuja et al. [76] propose to use a variety of different sensors to monitor DC environments. Those contributions highlight the importance of Data Center Infrastructure Management (DCIM) to ensure an optimal and uninterrupted operation of DC devices. Dayarathna et al. [77] and Borghesi et al. [78] identify anomalies by analyzing the power-consumption of DC devices. The goal of this thesis is different to those approaches. We analyze the relationship of changes in the environment and remote device management to identify mismatching states between remote instructed and executed device activity.

In this thesis, we show that using external measurements helps to identify scenarios in which the physical device activity differs from the reported activity. AD that purely relies on side-channel monitoring also has disadvantages. While those approaches can identify anomalous physical states, it is impossible to determine if activity was instructed by operators or is unintended behavior. Further, to attribute changes to a device, current methods require monitoring devices in isolation or device-specific measurements. The necessary isolation makes it difficult to use those approaches in a real-world setup or measurements that are influenced simultaneously by multiple devices. This can cause complex setups or expensive sensor placement for data collection. Chapter 7 presents a method to identify individual device activity in composite signals. We further address this limitation for AD by combining side-channel measurements with network layer activity. We show that this approach helps to enable side-channel monitoring for AD in shared environments.

3.2.2 Event Verification Methods

The typical application domain of event verification methods is cyber-physical environments, such as the Internet of Things (IoT). Birnbach et al. [2], [38] show that physical events can

simultaneously influence different physical measurements. Based on these insights, the authors propose Peeves, a system that checks if measured changes at multiple sensors are consistent in order to verify reported events. Similarly, Fu et al. [79] introduce HAWatcher, a system that correlates reported events to identify anomalous system behavior. The approach derives explainable system behavior in smart homes by including a semantic meaning of relationships between events. Ozmen et al. [57] identify constellations that allow adversaries to mislead systems like by Peeves and HAWatcher. The authors propose a strategy for optimized sensor placement. Taking locality into account allows to better distinguish between events that otherwise cause similar physical changes. This enhanced event identification reduces attacker capabilities to hide manipulations concerning reported or unreported events. These approaches show that correlating information from multiple sensors allows the identification of anomalous behavior. The presented approaches aim at verifying reported events by cyber-physical layer devices in IoT environments. Sahu et al. [8] propose to fuse information from sensors on multiple layers into a single data set for AD. The evaluation in an emulated environment shows that ADSs that simultaneously analyze fused information from multiple layers can identify anomalies faster than ADSs, which only process information from one layer. While the analyzed data set contains information from multiple layers, the study does analyze how changes on multiple layers from different sources relate to each other. This is different from our work in which we correlate activity on multiple layers to identify and monitor the relationship between them.

These examples show that combining information from the network layer and physical measurements has various advantages. First, it allows to identify system states in which physical and reported behavior differ. Second, it increases the detection speed of anomalies, as features on some layers indicate anomalies faster than others. These insights further motivate our approach to build methods for monitoring multi-layer relationships. Our approach to combining SC measurements with network traffic enables monitoring multiple devices with the same sensors while attributing changes to individual devices. The model presented in Chapter 8 of this thesis captures the relationship between networked control and physical device activity on the cyber-physical layer. We apply this model in Chapter 9 to validate device activity. Validating specific device activity differs from confirming events reported by individual sensors in event verification methods. It analyzes the specific changes to confirm previously instructed device activity on the cyber-physical layer. Further, we extend the application domain of such approaches to DC environments in which the purpose of devices is not primarily to change the environment. This way, we enhance the monitoring capabilities in these environments.

3.2.3 Attacks

The anomalies we consider in this thesis lead to inconsistencies between the network layer and the cyber-physical layer activity. In addition to faults, those inconsistencies can be the result of attacks. A well-known attack that causes such inconsistencies is the Stuxnet attack [10], which targeted centrifuges. In this attack, adversaries manipulated the physical behavior of centrifuges while reporting different behavior via the network to remote monitoring. This false reporting created a mismatch between behavior on the cyber-physical layer and the perceived device state in remote monitoring. Garcia et al. [80] implement a similar attack to misleading

the local control logic of PLC. The targeted PLC executed different physical device activity than reported. As a result, this attack misleads remote monitoring entities by targeting the firmware of PLCs in Industrial Control Systems (ICSs) (Figure 2.2). Both attacks hide physical device behavior from external entities and monitoring by simultaneously manipulating the input and output of devices. It is impossible for monitoring that relies on information from within the system to detect such attacks. The previously discussed papers [2], [38], [57], [79] also consider similar attacks which we refer to as "event masquerading" and "event spoofing". In event masquerading, adversaries use a similar strategy as in the Stuxnet attack to hide physical activity. In this case, they suppress notifications of local activity to remote monitoring. On the other hand, event spoofing reports non-existent physical device activity to remote monitoring. In both cases, there is a mismatch between the real and reported physical device activity.

Our work aims at detecting mismatches between activity on different layers, e. g., cyber-physical layer and reported activity, regardless of whether they occurred due to faults or intentional attacks. By correlating information from multiple sources and layers, particularly the management traffic for remote control, we create a cyber-physical model of environments in Chapter 8. This correlation allows the advancement of AD in those environments by enabling the detection of such states.

3.3 Key Results

Various related work implicitly uses multi-layer relationships to study and capture the behavior of network-controlled systems. Table 3.1 summarizes the discussed related work. There are two classes of related work. One class focuses on the relationship between applications and network traffic, while the second correlates physical activity with reported events. Analyzing network flows enables us to infer insights about the application causing it. However, many of those approaches lack a ground truth. This is because information collection that associates applications on a host with packets is challenging. This leads to two main contributions of this thesis in Part II. We address this limitation by proposing a novel method to collect information that associates packets with applications. Based on this data collection, we propose network application profiles that characterize applications that run natively on a host.

Related work on monitoring physical device behavior shows a relation between physical events and device activity. In particular, correlating changes perceived by multiple sensors helps to verify reported events by individual sensors in a system. A limitation of existing SC monitoring approaches is their reliance on isolated devices such that physical changes can be attributed to them. Our work in Part III of this chapter extends the state of the art in two ways. First, we show that it is possible to attribute physical changes in shared and noisy spaces to individual devices depending on the properties of the environment. Second, we enable the validation of instructed physical device activity instead of validating reported events by correlating physical changes with control traffic.

Table 3.1: Comparison of related work

Related Work	Cyber-phys. layer	Network layer	Application layer	Application	Environment
Karagiannis et al. [9]	✗	✓	✓	Traffic classification	Enterprise
Dai et al. [65]	✗	✓	✓	App. ident.	Home
Asai et al. [11]	✗	✓	✗	App. profiling	Home
Piet et al. [4]	✗	✓	✗	Protocol ident.	Enterprise
Pontes et al. [66]	✗	✓	✗	AD	Home
Popa et al. [3]	✗	✓	✓	Dependency det.	Enterprise
Haas et al. [67]	✗	✓	✓	AD	Enterprise
Hubble [70]	✗	✓	✓	AD	Container
Tetragon [71]	✗	✓	✓	AD	Container
Bayens et al. [74]	✓	✗	✗	AD	3D-printer
Han et al. [73]	✓	✗	✗	AD	ICS
Levy et al. [24]	✓	✗	✗	Monitoring	DC
Rogriguez et al. [75]	✓	✗	✗	Monitoring	DC
Ahuja et al. [76]	✓	✗	✗	Monitoring	DC
Dayarathna et al. [77]	✓	✗	✗	AD	DC
Borghesi et al. [78]	✓	✗	✗	AD	DC
Birnbach et al. [38]	✓	✓	✗	AD	IoT
Birnbach et al. [2]	✓	✓	✗	AD	IoT
Fu et al. [79]	✓	✓	✗	AD	IoT
Ozmen et al. [57]	✓	✓	✗	AD	IoT
Sahu et al. [8]	✓	✓	✗	AD	ICS
Stuxnet [10]	✓	✓	✗	Attack	ICS
Garcia et al. [80]	✓	✗	✗	Attack	ICS
This work	✓	✓	✓	AD	DC/ICS/IoT

Chapter 4

Effects of Attacks on Cyber-Physical Systems

Author's contribution: *This chapter is based on L. Wüstrich, M. Pahl, and S. Liebald, "Towards an Extensible IoT Security Taxonomy", in* IEEE Symposium on Computers and Communications, ISCC 2020, Rennes, France, July 7-10, 2020*, IEEE, 2020, pp. 1–6.* DOI: 10.1109/ISCC50000.2020.9219584*. The author conceived the concept and applied it to various attacks. The classification scheme was adapted for this thesis. The list of attacks was extended to cover the attacks considered in this thesis.*

The previously introduced multi-layer relationships can influence how an attack can affect a system. Attackers can abuse those relationships to reach their goals, enabling them to compromise the security of computer systems. However, as we show in this thesis, multi-layer relationships also allow us to extend the detection capabilities of Anomaly Detection Systems (ADSs), increasing the security of monitored systems.

This chapter provides an analysis of how attacks affect the security of a system by using the three logical layers introduced in Section 2.2.1. Our investigation provides an answer to *Research Question (RQ)1.2 Which relationships and types of dependencies exist in networks?* Understanding how attacks affect a system is important, because operators need to choose proper mitigation and prevention techniques. In particular, an attack can influence multiple layers at once due to multi-layer relationships. Setting events from multiple sources and layers into the context of each other makes it easier to identify attacks [8], [81]. It further allows to identify states where the behavior on one layer is inconsistent with the associated behavior on another. This motivates the main objective of the thesis to make systems more secure by monitoring multi-layer relationships.

Understanding the properties and effects of an attack requires a detailed analysis covering all logical layers. This section analyzes various attacks in cyber-physical environments and effects due

to multi-layer relationships. This provides insights into how multi-layer relationships influence the security of an environment.

4.1 Motivation

The collaboration between devices in the Internet of Things (IoT) allows the implementation of complex physical procedures by coordinating device activity, particularly on the cyber-physical layer [82]. The same holds true for other types of Cyber-Physical System (CPS), such as Industrial Control System (ICS). Adversaries can exploit relationships between system components to indirectly attack or interrupt a system. Understanding the threats and how attacks can affect a system is crucial to select suitable mechanisms to protect against them.

Security taxonomies allow a consistent labeling of attacks and threats. A consistent classification helps to understand and address similar attacks. Threat classifications are often non-intuitive. Existing taxonomies typically classify attacks by the methods used by an attacker and the goals it has instead of the effect an attack has on a system. While adversaries can use the same methods to attack different parts of a system, operators are also interested in understanding the effects of an attack on their system. Security taxonomies for traditional computer systems miss cyber-physical aspects. This creates a need for a taxonomy that classifies attacks based on the effects they have on a system.

This chapter introduces an extensible threat taxonomy based on the effects that an attack has on a system. The taxonomy uses two fundamental building blocks. The first building block is the attacked logical layer of the system (Section 2.2.1). Additionally, we use the fundamental security goals on each logical layer as a secondary building block. The resulting taxonomy enables the classification and comparison of attacks by their effects on a system's security.

Fundamental security goals are *availability*, *confidentiality*, *authenticity*, *accountability*, *integrity*, and *access control* [83]–[86]. Every security goal can be violated on each logical layer. Attacks are typically only carried out on a single layer [46] and aim at specific security goals. However, the effects of a successful attack can have an impact on other security goals on the other layers as well. For example, if an adversary can manipulate network packets, this can influence the decision-making of application-layer processes. The skills and knowledge of adversaries determine how likely attacks succeed [87], [88].

Operators need to classify threats and the effects of successful attacks on other system components in order to understand risks and prioritize attack prevention and mitigation strategies. Our proposed taxonomy differs from existing ones which primarily focus on techniques and goals of adversaries, such as Common Weakness Enumeration (CWE) [89], Common Vulnerability and Expsoures (CVE) [90] or Common Attack Pattern Enumeration and Classification (CAPEC)-IDs [91], [92]. Instead, it focuses on the impact of attacks on the security of a system.

4.2 Related Work

Diverse taxonomies to classify threats exist. Multi-layer taxonomies typically focus on CPSs and the IoT. A common approach is to split the domain into the three layers of our logical structure and group threats by the attacked layer [46], [47], [93], [94].

Andrea et al. [46] split the architecture into a perception/physical, network, and application layer. The authors group the attacks into the layers in which the attack occurs. The paper introduces the class of encryption attacks that aim to break the confidentiality of information. Encryption attacks are independent of the targeted layer. The introduction of a new attack class that is not in line with the layered architecture indicates that the functionality of the different layers is insufficient for a complete threat taxonomy. Rizvi et al. [47] combine IoT layers, threat vector, trust, and compliance into their approach. The purpose of each category varies strongly, making it hard to use. Further, since threats can be assigned to multiple classes, the classification is ambiguous. This makes it difficult to provide a distinct overview of IoT threats and to compare attacks. Our taxonomy solves this problem by providing a unified hierarchy for each layer.

The authors of [94], [95] provide a general overview of IoT security. In addition to the confidentiality, integrity, and availability security goals, they discuss general issues for the IoT like the heterogeneity of devices on the cyber-physical layer. Such issues also are relevant in the environments considered in this thesis in which heterogenous devices collaborate.

Mitrokotsa et al. [93] considers attacks on Radio Frequency Identification (RFID) systems, a subset of the IoT [40]. Teixeira et al. [88] provide an overview of attack models on CPSs. The authors define an attack space for networked systems and classify attacks depending on an attacker's system knowledge and the available resources. A holistic overview of IoT security is given in [96], [97]. While these publications also combine layers and security goals, they consider security goals that are not standardized.

Ronen et al. [98] categorize threats by the type of device utilization and functionality, i. e., reducing, misusing, or extending it to the adversary's benefit. The presented classification misses the effects of attacks on IoT systems.

A variety of surveys focus on different aspects of IoT security. Ammar et al. [99] focus on the security of IoT frameworks like Azure IoT or AWS IoT. The authors analyze the security goals of authenticity, access control, and confidentiality. Yang et al. [100] explore different authentication schemes for the middleware layer. The survey conducted in [101] solely focuses on trust in the IoT. It defines specific categories to assess and evaluate trust in different IoT frameworks. Stellios et al. [87] investigate attacker types and threats to critical infrastructures and services. The authors classify characteristics of adversaries with regard to their access and capabilities. They provide a threat assessment regarding risk, affected layers, and required attacker capabilities on networked systems. The lack of security goals prevents assessing how the security of an attacked system is affected.

A taxonomy created for threats on the web ecosystem [102] uses security goals for fine granular classification of attacks. Developed for the web, it is missing the cyber-physical aspect of the IoT. Rocchetto et al. [103] introduce a taxonomy for attacker models for CPSs. The presented work analyzes how types of attacks on CPSs affect an adversaries actions.

The presented works do not aim to compare attacks based on the effects they have on a system. This makes it hard to compare and classify attacks and solutions. Our proposed taxonomy covers all logical layers along with commonly used security goals. In particular, it allows to highlight multi-layer effects on a system. This way, it allows for the categorization of threats while remaining easy to use.

4.3 Common Security Goals and Attacker Types

In addition to the logical layers from Section 2.2.1, we consider fundamental security goals for our taxonomy. In this section, we introduce those goals and discuss how attacker capabilities influence the likelihood of a successful attack.

4.3.1 Security Goals

The security of a system can be assessed by analyzing by using common security goals. We define our security goals as the following: [83]–[86]

Availability: A system is called available when it can assure that information and communication functionalities are always available when expected.

Accountability: A system is called accountable if it removes the option of plausible deniability for actions.

Authenticity: An authentic system assures that all entities are credible and trustworthy. Authenticity can be verified by using an identity and other characteristics of entities.

Integrity: Integrity assures that information is not accidentally or maliciously altered without notice.

Confidentiality: Information is confidential when it is only disclosed to entities and processes with permission.

Access Control: Access control ensures that only authorized entities can access protected resources.

To ensure the security of a system, every security goal on all three layers needs to be fulfilled. Adversaries typically aim at specific security goals to achieve their objectives. Operators, therefore, have to prioritize and implement corresponding mechanisms to ensure all of their prioritized security goals are met, depending on the environment. The different tasks and limitations of each layer's components limit the choice of operators in the available mechanisms to achieve security goals [104]. For example, sensors on the cyber-physical layers have limited computing resources [49]. The limited computation power affects the cryptographic mechanisms that operators can choose to protect the integrity of measured values.

4.3.2 Attacker Properties

An attacker's properties influence the likelihood of a successful attack. Adversaries have *capabilities* and *knowledge* of an attacked system. Capabilities can be measured by (1) the *level of access*, and (2) *available resources* [87].

We distinguish between access levels *physical access* and *remote access*. Depending on the level of access, the adversary can execute different attacks. An example is the power consumption monitoring for executing a Side Channel (SC) attack [105]. An attacker can have a global or partial view of a system. For example, a compromised Programmable Logic Controller (PLC) in the structure presented in Figure 2.2 may give it access to the connected sensors and actuators but does not provide information about other PLCs in the network.

Additionally, the success of an attack depends on the available resources of an attacker [88]. The available resources determine how much time an adversary has and which tools it can use. Crafting sophisticated attacks requires more resources than simple attacks like replay attacks. An attacker with more resources is more likely to succeed.

Attacks can be categorized as *passive* or *active*. In passive attacks, attackers only observe the system and do not influence the targeted entity. For example, an attacker can apply cryptanalysis on observed network packets to extract their contents. An active attack influences the targeted system [106]. Active attackers can manipulate, inject, or drop information.

Depending on the attack, an adversary needs a minimum level of *knowledge* about a system. This includes knowledge about used protocols, devices, and device resources. Insider knowledge facilitates targeting specific parts of a system. *Without* knowledge of a system, an adversary can only perform basic attacks like cutting cables, damaging devices, or replaying messages. *Basic* system knowledge, e. g. insights on protocols, enables targeting specific components and vulnerabilities. *Advanced* knowledge, e. g. known communication patterns, enables developing and executing highly sophisticated attacks that are specifically adapted towards a targeted environment and allow it to hide its activity.

4.4 A Taxonomy for Attack Classification

We use the previously introduced building blocks to create our taxonomy for classifying attacks. The taxonomy categorizes an attack by

- targeted logical layer,
- affected security goals on other layers,
- required attacker knowledge, and
- required attacker capabilities.

Our taxonomy allows further sub-categorizing attacks according to each layer's violated security goals. This distinction makes it possible to accurately describe attacks affect the security of a system. Our approach allows comparing attacks regarding violated goals and difficulty of execution regarding attacker prerequisites and how they affect a system. Using the violated

security goals on each logical layer, comparing attacks, and prioritizing mitigation strategies is possible.

Our approach can also be used to assess which security goals are ensured by different protocols and frameworks. The layers and security goals imply which goals are protected. This can facilitate the creation of an overview of the state of a system's security and enables the identification of shortcomings of existing solutions. It also allows operators to identify gaps in currently implemented mechanisms.

We use a two-part scheme to highlight the affected security goals in the taxonomy. The first part of the scheme uses the targeted layer. The second part uses the previously introduced security goals. The following section introduces this naming scheme in detail.

4.4.1 Naming Scheme

Our naming scheme combines the targeted layer and affected security goals to capture the effect of attacks. For a better readability we use abbreviations of the affected parts. The target layer identifies which layer is affected by the attack:

- CP: Cyber-Physical
- NET: Network
- APP: Application

Attacks can affect security goals on multiple layers. However, they typically target only one layer. We use our naming scheme to capture how an attack affects a system. Since we want to identify how an attack affects the overall system, this part of our naming scheme captures which of the logical layers it affects.

The second used characteristic in the identifier is the security goal:

- AVAIL: availability
- AUTH: authenticity
- ACC: accountability
- INT: integrity
- CONF: confidentiality
- ACL: access control

Each identifier has two parts. The first part is a prefix that defines the affected layer. The second part is a suffix defining the security on this layer. Thus, each identifier has the structure of <layer>.<security goal>. In the following, we exemplify the construction of identifiers for the CP layer.

We use the CP.AVAIL class to denote attacks that target the security goal "availability" in the CP layer. This category includes attacks that render devices on the CP layer unavailable.

Examples are physical tampering and destruction of sensors or the interference of signals used for data transmission.

Attacks targeting the security goal "authenticity" are classified with the AUTH suffix. An example of an attack on the authenticity of the cyber-physical layer is RFID cloning [93]. Attacks that affect this security are denoted by CP.AUTH.

Threats that target a layer's "accountability" security goal are classified in the ACC category. This class includes attacks enabling adversaries to issue commands to devices on the cyber-physical layer without notice.

The INT category classifies attacks on the "integrity" security goal. An example for an attack in CP.INT is the manipulation of a sensor's measured values.

The class of CONF is used for attacks on the "confidentiality" security goal. This includes attacks on the cryptography used on the different layers. SC attacks [105] that e. g. analyze the power consumption of a device to extract secrets are also included in this class. ACL classifies attacks on the "access control" security goal. This class includes attacks that enable the unauthorized use of resources.

Following this scheme, the resulting classes for attacks at the cyber-physical layer are CP.AVAIL, CP.AUTH, CP.ACC, CP.INT, CP.CONF and CP.ACL for the CP layers. The NET and APP layers have the same sub-classes for attacks on the security goals on these layers with the respective prefixes.

Attacks violating multiple security goals can be classified using a combination of the categories. An attack that affects security goals on multiple layers (CP.ACC, CP.INT, CP.AVAIL, APP.INT) is the covert attack. Covert attacks use a combination of different attacks to succeed. Each used attack in a covert attack can also be classified independently.

Our approach makes it possible to intuitively assess the primary goal of an attack. The naming scheme highlights the affected layer, making it easy to assess the attacked part of a system.

4.4.2 Attack Classification

In the following, we classify attacks by applying our taxonomy.

Attacks on the cyber-physical layer

Attacks on the cyber-physical layer (CP) target both hardware and software. The distribution of sensors and actuators in the environment of CPSs facilitates adversary access.

> **Replay Attacks**: In replay attacks [107], an adversary retransmits previously captured secured signals and poses as the sender (NET.AUTH). Receiving a legitimate signal can cause a device to change its state. To execute the attack, the adversary does not need extensive knowledge about the system [88]. Access to devices and the capability to record and replay a signal are sufficient for this attack. This attack can also target the network layer by replaying intercepted messages via the network.

Sleep Deprivation Attack: This attack targets battery-powered devices [46], [108], [109]. In order to save energy, some devices have a sleep mode. The attacker interacts with the device to prevent it from entering the sleep mode, causing a targeted device to drain its battery. This renders the device unavailable (CP.AVAIL). For success, an adversary needs network access to interact with the node. It also needs knowledge about the commands it can send to the node to keep it from entering its sleep mode.

Physical tampering: Sensors and actuators can be physically damaged [46]. The adversary's goal is to render the device and its functionality unusable, which violates the goal of availability (CP.AVAIL). The attacker does not need any prior knowledge about the system to succeed.

Jamming Attacks: Various jamming attacks exist [110], [111] that create create physical interference. In wireless setups, adversaries jam the physical channel between the nodes in the network to make the communication between devices slow or impossible (CP.AVAIL). Jamming also affects the availability of the network can be used in the system or access to specific devices (NET.AVAIL). To succeed, adversaries need knowledge about the communication channels and physical proximity to the attacked system.

Covert Attack: Another class of attacks on the CP layer is covert attacks. In this attack, an adversary manipulates the input and output of devices simultaneously to hide the effects of its attack [112]. An example is the Stuxnet worm [10]. The adversary needs extensive knowledge of the attacked system to successfully carry out a covert attack. Since this attack consists of multiple components, each of the components can be classified on its own. For example, in Stuxnet, parts of the manipulate the input and output of sensor readings (CP.INT). Other parts issue commands on behalf of the controller (CP.ACC). The resulting damage made physical devices unavailable (CP.AVAIL). The manipulation of the device output also falsified the centralized control and monitoring (APP.INT) Depending on the goal of the covert attack, it also violates other security goals, which can be additional categories for this attack. As a result, covert attack can have various effects on the security of a system. In Stuxnet, those are, i. e. CP.AVAIL, CP.INT, CP.ACC and APP.INT. The Stuxnet attack caused inconsistent behavior on the logical layers. While the adversaries instructed devices locally to perform physical actions, the manipulated output caused a false reporting to the remote monitoring. Our work in Part III of this thesis aims to identify such inconsistencies.

Side Channel Attacks: Another class of attacks that predominantly targets the CP layer and devices on the cyber-physical layer due to their limited computing power and level of exposure are SC attacks. SC attacks extract the cryptographic key material used by the devices to encrypt and sign their data [105]. By analyzing physical characteristics like the device's power consumption during computations, the secret is inferred (APP.CONF) [113]. Depending on the SC, an adversary needs knowledge of the analyzed protocol and physical access to the device to execute the attack.

Calibration Parameter Tampering: Sensors and actuators are calibrated before they are deployed. An adversary that does calibration parameter tampering [114] changes

the devices' calibration to falsify the measurements. This attack violates the goal of integrity (CP.INT). Like covert attacks, successful calibration parameter tampering also affects the application layer's integrity (APP.INT).

Attacks on the network layer

Attacks in NET target the transport of information and mechanisms provided by the network layer. Therefore, this category contains attacks on the routing and addressing of information as well as attacks on network features.

Sybil Attack: In a Sybil attack, the adversary assumes multiple identities of other nodes (NET.AUTH) in the network [115]. An attacker uses these identities to influence the common decision-making of connected nodes in its favor. It needs network access and knowledge of the decision-making protocols to perform the attack.

Denial of Service: The denial of service attack targets the availability on the network layer (NET.AVAIL). There are several possibilities to execute this attack. One method is to flood the network with requests and traffic so the nodes become unresponsive [116]. To execute the attack, it is sufficient for the adversary to have network access. No further knowledge about the attacked system is required.

Sinkhole Attack: Another attack is the sinkhole attack [117]. In this attack, an adversary attracts as much traffic as possible by exploiting the used routing protocol. The adversary then decides how to proceed with the captured data, e. g. to drop it (NET.AVAIL). For this type of attack, the attacker needs network access. It also requires knowledge about the routing protocol used to exploit it. The sinkhole attack can additionally enable an adversary to manipulate the packets it captures (NET.INT).

Cryptanalysis: When attacking the confidentiality on the network layer (NET.CONF), an adversary also has various attack vectors. If the attacker knows the used cryptographic algorithms, it can take different efforts to extract the key through cryptanalysis. If it can intercept traffic and participate in the exchange of messages, it can also execute one of various man-in-the-middle (MitM) attacks [118]. There is also the possibility of decrypting captured traffic by brute-forcing all possible keys.

Command Injection: In command injection attacks, adversaries use the network to send malicious commands, e. g., to actuators. For this goal, attackers may impersonate the legitimate sender (NET.AUTH, NET.ACC). Physical devices then execute instructed commands as instructed and are unavailable for legitimate instructions in this time frame (CP.AVAIL). An attacker needs access to the network and knowledge about legitimate commands to succeed.

Event Spoofing: In event spoofing attacks, an attacker impersonates sensors or actuators (NET.AUTH) and reports wrong events (NET,ACC, NET.INT) to the application layer [38]. As a result, the attack affects the application layer decision-making as it processes falsely reported data (APP.INT). Similarly to covert attacks, this attack can

cause inconsistencies in reported behavior in the system, and real observed physical activity.

Event Masquerading: In event masquerading, an attacker removes information about an occurred event on the CP layer (CP.ACC) on the network layer (NET.INT). Similar to event spoofing, this also affects the integrity of the decision-making on the application layer (APP.INT). Further, dropping information from devices affects the availability of the targeted devices (CP.AVAIL). This attack also creates inconsistency between reported events and real physical activity, which we aim to identify in Part III of this thesis.

Like the CP layer, the attacker can use other SCs on the NET layer, such as timings, to draw conclusions on secrets used to encrypt information. However, these attacks are not targeted at the hardware but the software that runs on devices.

Attacks on the application layer

Attacks on the application layer aim at disrupting services that collect and process the data. Due to the variety of applications, there is no specific attack type for this layer, only a class of attacks that can be executed.

Phishing: In phishing attacks [119], an adversary attempts to trick unknowing victims into giving them confidential information. This allows the adversary to act on behalf of the victim (APP.AUTH) and execute actions in its name (APP.ACC).

DoS: Similar to NET, an attacker can perform a DoS attack on the application layer to attack single or multiple applications of the IoT and render them unavailable (APP.AVAIL).

Malicious Updates: If an attacker can modify the running application by using malware, a Trojan horse, or installing a malicious update, the integrity of the application layer is violated (APP.INT). In order to succeed, the attacker needs access to the network or the source from which the software is installed. An example of such an attack can be found in [120].

Cryptanalysis: Like NET, an adversary can also use cryptanalysis to attack the encryption schemes used by the software running on the application layer (APP.CONF).

Privilege Escalation: When an attacker can achieve privilege escalation, it can extend its capabilities to use the additional functionality of a system and access information (APP.ACL). Extensive knowledge of the software used and its internal functionality is needed to elevate its privilege.

Process Masquerading: In order to evade the detection on an end-system, an attacker can masquerade a process (APP.AUTH) on a machine as a benign process [121]. This way it is challenging for an ADS to identify the malicious process, as it hides its activity (APP.ACC) on the end host. Further, in case the process masquerades its network traffic as other benign traffic (NET.AUTH). To successfully execute the attack, an adversary needs remote control over the local process. Further, it needs knowledge about other running processes and used protocols to masquerade itself. This attack can cause inconsistencies between perceived application activity on the application layer and networking

Table 4.1: Classification of attacks using our scheme

Name	Target layer	Affected goals	Access level	Knowledge
Replay (signal)	CP	NET.AUTH	phys.	no
Sleep deprivation	CP	CP.AVAIL	net.	basic
Physical tampering	CP	CP.AVAIL	phys.	no
Jamming	CP	CP.AVAIL, NET.AVAIL	phys.	no
Covert attack	CP	CPP.ACC, CP.INT, CP.AVAIL, APP.INT	net.	advanced
SC attack	CP	APP.CONF	phys.	advanced
Cal. param. tamp.	CP	CP.INT, APP.INT	phys.	advanced
Replay (messages)	NET	NET.AUTH	net.	no
Sybil	NET	NET.AUTH	net.	advanced
DoS	NET	NET.AVAIL, CP.AVAIL	net.	no
Sinkhole	NET	NET.AVAIL, NET.INT	net.	advanced
Cryptanalysis	NET	NET.CONF, APP.CONF	net.	advanced
Command injection	NET	NET.AUTH, NET.ACC, CP.AVAIL	net.	advanced
Event spoofing	NET	NET.INT, NET.AUTH, NET.ACC, APP.INT	net.	advanced
Event masquerading	NET	CP.AVAIL, CP.ACC, NET.INT, APP.INT	net.	advanced
Phishing	APP	APP.AUTH, APP.ACC	net.	basic
DoS	APP	APP.AVAIL	net.	no
Mal. updates	APP	APP.INT	net.	advanced
Cryptanalysis	APP	APP.CONF	net.	advanced
Priv. escalation	APP	APP.ACL	net.	advanced
Proc. masquerading	APP	APP.AUTH, APP.ACC, NET.AUTH	net.	basic

activity on the network layer of our logical structure. Part II of this thesis presents an approach to detect such inconsistent states.

Table 4.1 classifies all discussed attacks. Our list of attacks is not exhaustive. However, it exemplifies how attacks can be classified and shows their impact on security goals on multiple layers if carried out successfully. By combining the different layers and the fundamentals of security, our taxonomy captures how attacks affect systems. If the taxonomy needs to be more fine-grained, each security goal class can be divided into the needed categories. Classifying new attacks can already be done using our taxonomy, as these attacks affect the fundamental security goals.

The introduced naming scheme uses fundamental properties of the cyber-physical logical architecture and fundamental security goals. By construction, this ensures the versatile applicability of the naming scheme. More advanced security goals can be considered as a combination of standard goals in the future.

To validate our naming scheme, we compared the classification of existing taxonomies like [46], [47]. While our taxonomy results in a similar classification, it has the benefit of highlighting how attacks affect the functionality of a system. Therefore, our naming scheme gives more structure to the attack space. This extends existing approaches or standards like CVE- or CAPEC-IDs [89], [92] where only numbers are used for classifying vulnerabilities and attack vectors.

In the remainder of the thesis, we aim to detect attacks by simultaneously monitoring multi-layer relationships. In particular, we analyze *masquerading*, *event spoofing*, *event masquerading* and *covert* attacks. Those attacks can affect multiple layers simultaneously. The methods that we propose show that setting events from multiple layers into context of each other makes it more difficult for adversaries to hide their activity and remain undetected.

4.5 Key Results

We analyzed the attack space in the considered cyber-physical environment. By considering the violated security across multiple layers, it is possible to understand how an attack can affect the security of a system. Providing a simple-to-use classification scheme for attacks can help make CPSs more secure. The presented scheme allows operators to assess how attacks influence the security of their system. Such a scheme was missing previously, as the existing approaches typically focus on the techniques and goals of attackers. Our naming scheme is based on the affected logical layers of the architecture introduced in Section 2.2.1 and the attacked security properties. Therefore, it is able to classify any attack. By construction, the presented taxonomy is also extensible to new security goals in the future.

Part II

Relationships Between Application Behavior and Network Activity

Chapter 5

Associating Processes and Packets

Author's contribution: *The contents of this chapter are based on L. Wüstrich, M. Schacherbauer, M. Budeus, D. F. von Künßberg, S. Gallenmüller, M. Pahl, and G. Carle, "Network Profiles for Detecting Application-Characteristic Behavior Using Linux eBPF", in* Proceedings of the 1st Workshop on eBPF and Kernel Extensions, eBPF 2023, New York, NY, USA, 10 September 2023*, ACM, 2023, pp. 8–14.* DOI: 10.1145/3609021.3609294*. The author contributed to the conception of the paper's research question, analysis, and methodology. The matcher is the result of two Bachelor's theses [122], [123] in which the matching framework was developed and evaluated. The data presented the evaluation were obtained as part of a Bachelor's thesis [123]. The presented figures were newly created by the author. The author was the advisor of both theses. He conceived the research questions, methodology, and evaluation strategy for both theses and contributed to the analysis and the evaluation procedure. This thesis contains an extended discussion of possible association methods, as well as more details on the results of the evaluation.*

This chapter focuses on the link between events from the network and application layer of our logical structure (Section 2.2.1). Collecting data about this link is a prerequisite to analyzing the multi-layer relationship between application behavior and associated network activity (Research Question (RQ)2). It is challenging to efficiently associate fully assembled packets to the causing applications since Operating Systems (OSs) provide the networking functionality. After an application provides an OS with data, the OS autonomously assembles and sends egress traffic or disassembles received ingress traffic. This creates a gap between the application and the networking layer. This chapter presents a framework to efficiently and reliably associate network traffic to applications, overcoming the gap between the network and application layer. The proposed method provides an answer to *RQ2.1: How can we reliably attribute network activity to applications?*

5.1 Motivation

Applications often communicate with remote services using the network [3], [65]. The dependency on remote services creates a relationship between applications and the communication of an end host. Knowing how applications use the network helps administrators in several use cases, such as Anomaly Detection (AD) [67], packet filtering [124], or service dependency detection [3].

To understand an application's communication behavior, collecting information about how they use the network is necessary. Applications often consist of multiple processes that occur during runtime. For example, the Chrome browser often creates multiple separate processes, e. g. for separate tabs. The methods discussed in the following aim to associate packets with processes of an application. However, performing this task efficiently and reliably is challenging as OSs typically provide the networking functionality to local processes. Since OSs autonomously perform the networking, it can be difficult to determine which process caused a packet observed at a Network Interface Card (NIC).

Existing approaches to associate processes and packets yield in a tradeoff between data completeness and additional device load. They either use *polling*, *logging*, *specialized tooling*, or *heuristic* based mechanisms to collect information.

Periodic polling relies on mappings from ports to applications in the OS [3]. By polling e. g. the `/proc`-file system, it is possible to identify which process is associated with a specific port. Packets from and to this port are then associated with the corresponding process. Polling leads to a tradeoff between additional Central Processing Unit (CPU) load due to a short polling periodicity and completeness of data to capture short-lived processes [3]. On Unix-based OSs, it is possible to use logging functionality such as the `auditd` service. However, using this service introduces a CPU overhead of up to 40 % when running applications such as Firefox [125]. Tools like osquery [126] require the installation of additional software that, similar to using `auditd`, introduces additional overhead on end-hosts. There is also the possibility to associate packets and processes based on flow information, e. g. the IP-5-tuple. However, it is impossible to reliably identify the associated application by analyzing the IP-5-tuple. For example, Firefox and Chrome browsers typically use the same destination ports for their users' requests (destination port 80 for HyperText Transport Protocol (HTTP), destination port 443 for HTTPS). While it is possible to identify separate processes by using the source port, it impossible identify specific browsers just by using the IP-5-tuple. A drawback of all the mentioned approaches is their reliance on Layer 4 (L4) information, such as ports. This makes associating traffic below L4, such as packets belonging to the Internet Control Message Protocol (ICMP), with processes impossible. Matching such traffic is necessary to identify applications that use ICMP tunneling to obfuscate their communication [127].

The extended Berkeley Packet Filter (eBPF) [128] allows to address these issues. eBPF enables the execution of custom code in addition to the execution of kernel functions. By hooking only two kernel functions, this approach approach observes all egress traffic of a host. Combined with ingress matching, this allows a more exhaustive association than polling based mecha-

nisms. Since OSs execute this additional code in kernel-space, this makes the usage of eBPF for data collection is more efficient than other methods, that run in user-space, e. g., logging. In addition to running more efficiently than existing approaches, the presented eBPF-based approach associates traffic to a process even if it does not have a complete IP-5-tuple, including packets without L4 information. Finally, since our approach is integrated into the kernel, it is application agnostic in contrast to specialized software. This makes a reliable and automated association of packets and processes possible by using eBPF. As a result, our approach removes the need for manual or heuristic approaches for packet association.

This chapter introduces a framework based on eBPF—called matcher in the following—to associate processes and packets on a host. The matcher associates all egress and ingress packets to their associated process. In our evaluation, we measure the performance impact of the matcher and outline its advantages over other approaches.

5.2 Reliably Associating Processes and Packets

Reasoning about the typical network usage of applications requires a reliable association between processes and network packets. OSs provide access to methods that allow processes to send and receive data. For example, on Linux systems, applications can use syscalls to send data, e. g., `write, sendmsg`. The application provides the data, which needs to be sent as a function argument. Afterward, the OS constructs the necessary packets and sends them to their destination. In addition to the packet construction, OSs mange additional mechanisms for relevant for data transmission. This includes the connection establishment in the Transmission Control Protocol (TCP) or potentially required name resolution via the Domain Name System (DNS). Since OSs autonomously handle all network requests from and to applications, it is difficult to attribute a packet to a process. This section presents existing methods to attribute network traffic to processes.

In general, the methods for the association of packets to processes should fulfill five properties:

1. The method should be able to observe all packets from and to a host.
2. The method should attribute all packets that have an associated process.
3. The association should be fast to not affect the user experience and allow quick responses from administrators. However, while being fast, the data collection should impose little computational overhead to avoid affecting monitored processes and other applications.
4. The attribution should be accurate to provide ground truth and allow correct reasoning about application behavior. In particular, it should be able to distinguish between individual processes and not be limited to detecting classes of applications, such as browsers.
5. The method should be easy to implement and generally compatible with all processes, removing the necessity to alter the code of analyzed applications.

In the following we introduce and discuss existing data collection methods in the context of these properties.

Methods to Associate Packets and Processes

Attributing packets to processes has been part of several studies in efforts to support administrators in various tasks. The approaches can be classified into four categories: polling the OS, logging, heuristics, or using additional tools.

The first category is frequently polling the OS [3], [67]. Methods using this approach periodically check the OS to identify which ports are associated with a process. If the method then observes a packet for or from that port, it uses this association to map the packet to the corresponding process. In Linux, the `/proc` file system holds information about processes [129], including their associated port for communication. Using information from `/proc` allows to attribute observed packets from and to this port to processes. While it is easy to implement, polling the OS has several drawbacks. First, polling causes a constant load on the system, regardless of the network activity. Since polling is independent of networking activity, it has to occur constantly. Each poll causes additional overhead at the OS to collect and deliver the information to the requesting program. Polling the OS does not ensure a complete information collection. Short-lived processes that start and stop between the polling interval go by unnoticed [3]. Therefore, adopters of this approach face a tradeoff between CPU overhead and potential data incompleteness. While a shorter polling interval reduces the likelihood of missing a process, it increases the CPU overhead due to the increased polling rate. Longer polling intervals reduce the overhead for obtaining information but make it more likely to miss short-lived processes. Since polling takes place regardless of ongoing network communication, polling-based approaches create constant overhead on monitored end-hosts.

The second approach is to integrate OS logging tools, such as the `auditd` tool or `ftrace` [125], [130]. These methods create a log entry whenever a selected syscall is executed. The log entry includes information about the process that used the syscall. From these logs, it is possible to attribute network activity to applications. However, associating a packet to a process during runtime requires sophisticated post-processing methods that process logs. This can cause unnecessary overhead for use cases that require real-time decisions, such as Intrusion Detection Systems (IDSs). Implementing a logging approach is similarly easy to implement as OS-polling. Contrary to polling, logging ensures the completeness of the collected data. This means logging observes the activity of all processes, including short-lived ones. Further, logging only happens when an application uses one of the monitored syscalls. Thus, in contrast to polling-based approaches, logging overhead only occurs on demand. However, logging-based approaches also have drawbacks. The additional logging activity causes a considerable amount of additional overhead for each monitored syscall. Related work reports that using `auditd` causes a 40 % CPU overhead for executing the Firefox browser [130]. To ensure a complete data collection, it is necessary to log the execution of many syscalls. Not all of these syscalls are exclusively used for networking purposes, such as `write`. Therefore, this approach leads to CPU-overhead even for applications that do not use networking functionality but use one of the monitored syscalls. By mixing networking and non-networking-related entries, post-processing applications need to identify the relevant log entries for their analysis.

Another approach to associate processes and packets is to use heuristics. The Internet Assigned Numbers Authority (IANA) [131] standardizes well-known ports, i. e., assigns protocols to them, such as HyperText Transport Protocol Secure (HTTPS) to port 443. Using this knowledge, it is possible to reason about the applications that use specific well-known ports in their communication. The method looks at the IP-5-tuple, particularly the well-known port used to identify an application. Observing all packets from and to a host allows for complete data collection. While the method can observe all network activity, it cannot associate packets without a well-known port or packets that do not use ports at all, e. g. ICMP packets, to a process. An example for a data stream without well-known ports is the data transfer in FTP passive mode [132]. Contrary to OS-polling and logging, this approach does not have to be locally implemented on the monitored end-host but can be performed by a remote entity. If implemented on the end host, a simple packet capture is sufficient and does not impose significant computational overhead. This approach allows for the complete collection of data on all network activities, similar to logging, and is easy to implement. While using heuristics to associate packets to processes causes low overhead, it also has disadvantages. The first disadvantage is a missing ground truth. While making assumptions about processes by looking at ports and protocols is possible, it does not yield a reliable association. For example, it is possible that applications wrongfully misuse well-known ports for other purposes and are, therefore, wrongfully classified. Further, it is impossible to distinguish applications that use the same well-known ports and protocols. For example, if multiple browsers are active at the same time, it is impossible to distinguish them by only using the IP-5-tuple.

It is also possible to use specialized tools to collect the desired information. Software like `osquery` [126] leverages one or multiple of the above-described methods or similar to collect information about the OS. Specialized tools have the advantage to allow a complete data collection tailored for specific purposes. Due to their specialization, they can efficiently collect data without creating additional overhead, such as the logging approach. A drawback of this approach is that it requires installing additional software on the host that it monitors. The software also needs to be tailored towards the requirements of the post-processing applications. After installation, the software must run on top of the usual software stack, causing constant overhead. The additional overhead for the association depends on the used association method for the data collection.

Finally, there is Linux eBPF [128]. eBPF is a technology allows the execution of additional custom code along with the execution of a kernel function. There are numerous kernel functions can be hooked. The selection appropriate kernel function is necessary in order to avoid affecting networking-unrelated operations while capturing all relevant events. The possibility of hooking kernel functions allows more fine-grained access to information than other approaches. We discuss the criteria of selecting appropriate kernel functions in Section 5.4.1. Depending on the selected kernel functions, it is possible to enable complete data collection and associate all traffic with a process. The complexity of the additional eBPF code determines the computational overhead. However, since eBPF executes the code in kernel-space, its execution is more efficient as it does not require a context switch to the user-space. Further, by only executing additional code along with specific kernel functions, this overhead only occurs on demand. A drawback

Table 5.1: Comparison of methods to associate processes and packets

Method	Implementation	Completeness	Overhead	Ground truth	Associate all packets	Used by
Polling	++	−−	−−	+	−−	Zeek-osquery [67], Macroscope [3], BLINC [9]
Logging	−	++	−	++	++	Protracer [130]
Heuristics	++	++	++	−−	−−	Traffic Causality Graphs [11]
Tools	−−	++	−	++	++	osquery [126]
eBPF	−	++	+	++	++	Hubble [70], Tetragon [71], Falco [133], Opensnitch [124]

of using eBPF is that it requires an extensive analysis of the kernel to identify relevant kernel functions. Further, depending on the goals, it involves the creation of complex code to efficiently collect the desired information.

Table 5.1 summarizes the methods and highlights the properties of each approach. The table shows that eBPF has advantages over other approaches. Thus, we use eBPF to develop our framework to associate packets and processes in Section 5.4.

5.3 Related Work

This section discusses how other applications use the previously introduced methods to associate packets and processes.

Polling: Haas et al. [67] extended the Zeek [68] Network Intrusion Detection System (NIDS) by combining process and network flow information. zeek-osquery relies on *osquery* [126]. Nowadays, osquery supports collection of information via eBPF; back then, osquery relied on polling `/proc` for information collection. The evaluation shows that the extension improves Zeek's classification accuracy. The paper does not report on the overhead induced by osquery. "nethogs" [134] and "netstat" [135] are tools to identify processes using network resources. They periodically poll `/proc` to identify Process-IDs (PIDs) and related sockets. nethogs matches packets to the PID via the IP-5-tuple. It, therefore, cannot match traffic that does not use L4. Popa et al. [3] use packet-process associations to identify service dependencies. With a period of 5 s, their software queries the OS's connection table to match PIDs to sockets, correlating processes and packets using the 5-tuple. Due to the chosen period, the approach cannot associate processes that occur in between polling intervals.

Logging: Ma et al. [125], [130] present methods to analyze logs based on the Linux Audit System. Their work identifies causal relations between syscalls and packets that belong to associated applications. The proposed methods reduce the CPU overhead due to logging from 40 % to 15 %.

Heuristics: Network traffic analysis has been used to identify characteristic process behavior.

BLINC [9] uses network flow information to identify applications. Asai et al. [11] analyze network flows to identify causal relations between them and profile applications. The approach uses the IP-5-tuple to aggregate packets into flows. Dai et al. [65] profile Android apps based on fingerprints of network flows, detecting characteristic network behavior that uniquely identifies and distinguishes apps.
eBPF: Falco [133] allows monitoring syscalls on end hosts and containers, collecting information from multiple sources. However, Falco does not provide access to the syscalls we identified for our collection of information. Cilium aims at analyzing and securing the network connectivity of Linux containers via eBPF. Cilium Hubble [70] and Tetragon [71] additionally allow associating activity to specific containers. While these tools utilize eBPF to monitor containers, they do not assign packets to applications running natively on the host. *OpenSnitch* [124] uses eBPF to associate packets to processes and enable process-based packet filtering decisions. OpenSnitch probes the `iptunnel_xmit`, `udp_sendmsg`, `udpv6_sendmsg`, and TCP their counterparts. Packets for which the kernel uses alternative functions, such as the ICMP packets used by the `ping` utility, go unnoticed and will not be considered for further analysis. This shows that it is crucial to identify the suitable kernel functions to observe all traffic.

Most related work identifies applications from network traffic or associate packets to processes to monitor dependencies or resource usage. This differs from our application in Chapter 6, which characterizes typical process behavior.

5.4 Packet Association via eBPF

This section presents how it is possible to associate processes and packets using eBPF. It first presents our assessment and our consideration for choosing relevant kernel functions to hook in Section 5.4.1. We then detail the general approach to collect information in Section 5.4.2. Sections 5.4.3, 5.4.4, and 5.4.5 present our approach to use the collected information for establishing the link between packets and processes.

5.4.1 Probed Kernel Functions

There are a variety of kernel functions that are involved in constructing and deconstructing packets [136]. For each layer of the network stack there exist kernel functions to add or remove packet headers, e. g. TCP and User Datagram Protocol (UDP) headers on L4, Internet Protocol (IP) on L3, and a frame header on L2. The kernel is also responsible for performing operational tasks, such as establishing a TCP connection via the 3-way handshake before an application can send data. To ensure data completeness, all egress and ingress traffic to applications must be associated with a process. This includes traffic on L2 and operational traffic by the kernel, such as Address Resolution Protocol (ARP).

Choosing the appropriate kernel functions is a tradeoff between the number of observed functions and parsing overhead. By hooking functions on higher layers, e. g. L3 and L4, fewer headers need to be parsed to extract information. However, this has the disadvantage that it requires more kernel functions to ensure data completeness than the relevant kernel functions on the lower layers. For example, capturing all egress packets belonging to a TCP stream requires

observing two kernel functions. The first function is needed to monitor the 3-way handshake, and the second function to monitor the egress of the application data. Solutions which only monitor higher layers via `kprobes` miss packets belonging to such operations. This leads to many kernel functions that need to be observed to guarantee a complete data collection. If the system monitors kernel functions on multiple layers of the TCP/IP stack, it is possible to create duplicate entries for the same packet. For example, hooking `tcp_v4_connect` for an outgoing TCP handshake, and also `__ip_queue_xmit` on the network layer to capture ICMP messages, the TCP message would be logged twice.

Another issue when hooking functions on higher layers is an uncertainty that a constructed packet is passed to a NIC for transmission. The packet construction may fail at one of the lower layers, in which case it is not emitted but logged as network activity of the responsible application. To ensure that only emitted packets are used for the association, it requires additional data structures that retain constructed packets and verify they were emitted by the NIC.

Hooking kernel functions closer to a NIC imposes more parsing overhead to extract information. Since the kernel has added all headers to the packet before handing it over to the NIC, it is necessary to parse all headers for information extraction. Since structured information can be parsed efficiently, our approach observes kernel functions on the data link layer. This approach also allows collecting information about low-layer protocols such as ICMP without additional overhead.

We identified two kernel functions on the data link layer. The first function is `__dev_queue_xmit`, the second function is `__dev_direct_xmit`. These are involved in constructing all egress traffic of a host. It also prevents duplicates due to redundancy in postprocessing applications.

The kernel usually calls the `__dev_queue_xmit` function to send packets. This call queues a constructed packet in the qdisc to send it via the NIC [136]. Linux 3.14 introduced the `PACKET_QDISC_BYPASS` flag [137]. The kernel uses the function `__dev_direct_xmit` to provide the packet to the NIC. This flag allows to bypass the `qdisc` and causes an immediate sending. Setting this flag introduces the risk of packet loss if the NIC's capacities to send packets are at their limit [137]. By probing these two functions, observing all fully assembled packets before they are passed to the NIC is possible. Matching ingress traffic is more challenging than matching egress traffic. After receiving a packet at the NIC, information about the associated process is not immediately available. We, therefore, do not use eBPF to match incoming traffic but use matching strategies based on the packet information. We describe those strategies in Section 5.4.3.

5.4.2 Data Collection

Our framework to associate packets and frames relies on two sources of information shown by the dotted lines in Figure 5.1.

The first source are the two previously discussed kernel functions probed by eBPF, providing information about egress traffic. Both functions have an `skb` as argument. The `skb` is a data structure in the kernel that references all information of a packet, including metadata [138]. We

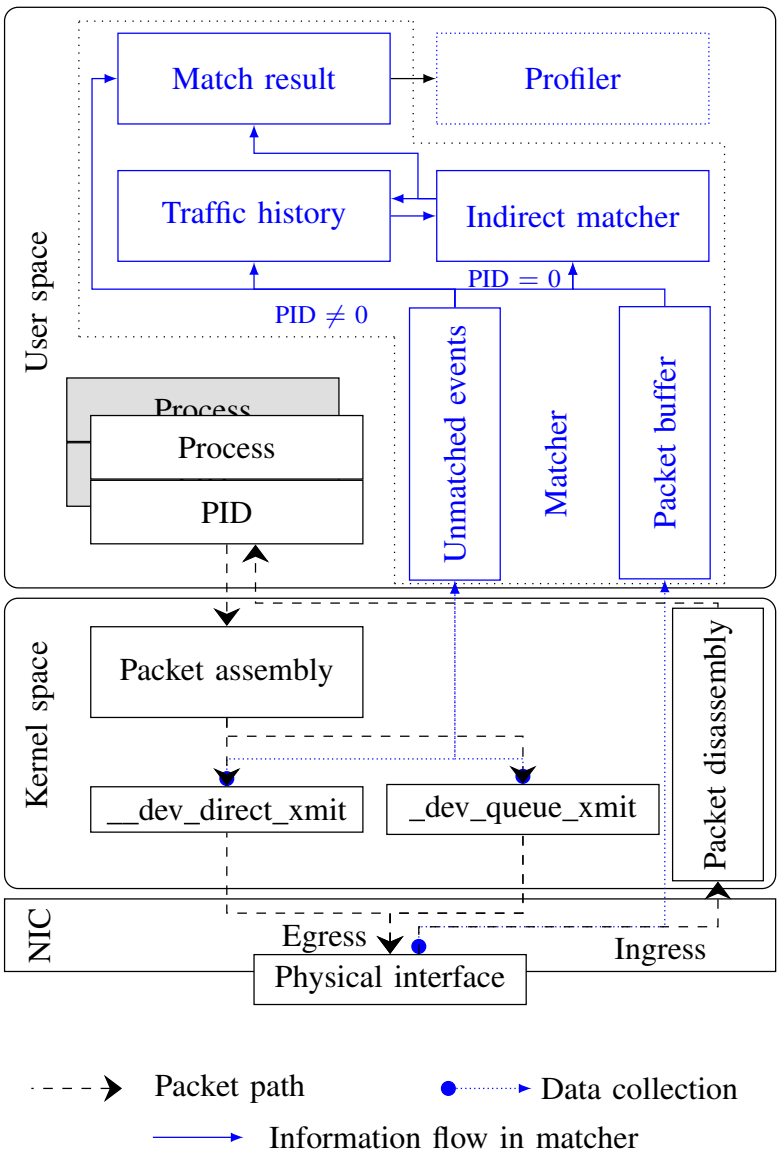

Figure 5.1: Architecture of the matcher (in the dotted box) and profiler. The general architecture of the matcher and its integration into the OS. The matcher collects information about egress traffic using eBPF at the two kernel functions. Data collection about ingress traffic happens via a raw socket that observes all traffic at a NIC. After the data collection, the matcher associates packets with processes using one of the matching strategies.

analyze this `skb` and instrument the Berkeley Packet Filter (BPF) to collect meta information in a `struct xmit_event`. This information includes a timestamp `ts`, the `PID`, the `TGID`, length, `copied_len`, and the `data` of the frame. We resolve the PID and Thread Group ID (TGID) to a process name via `/proc`.

The second source of information is a raw socket, serving as a network sniffer. This sniffer observes all ingress and egress traffic of the end-host interfaces. Efficiently extracting PID information via the kernel for ingress traffic is challenging since it is not inherently clear which process reads the memory. Thus, we analyze the properties of ingress traffic and match it to outgoing traffic via a set of strategies. Section 5.4.3 describes the strategies. Using those two sources of information allows for associating all ingress and egress traffic on a host.

5.4.3 Strategies for Traffic Matching

Egress: The information in the `xmit_event` extracted from the probed kernel functions includes the packet's content and the associated PID and TGID. This allows to directly associate all egress traffic to a process.

Ingress: We use multiple "indirect matching" techniques to associate ingress packets to PIDs. We assume that all ingress communication addressing a process on a host has at least one associated egress packet. Due to the typical command-response communication in the environments we consider in this thesis (Chapter 2), this assumption is reasonable. This assumption allows to use heuristics to match ingress traffic to previous matchings that were established by using eBPF. A limitation of this approach is that it cannot match ingress traffic to a process that does not have any egress traffic. After receiving a packet the PID of the program that processes its payload is unknown. One strategy is to use the information of the IP-5-tuple to match all packets of a flow to the same PID. In an enhanced version of this strategy for TCP, it is possible to use the sequence and acknowledgment numbers in the TCP header to associate packets. If a previously outgoing TCP segment of the same flow has the matching sequence number and length for a newly received acknowledgment, it is matched to the PID of the outgoing frame.

We also want to associate packets below L4 to a PID. Such packets lack layer 4 information, such as port numbers and a transport layer protocol. Thus, flow-based matching for such packets is impossible. Examples of such traffic are ICMP messages. Processing this traffic requires other matching strategies. We use two strategies to associate ICMP messages to PIDs.

The first strategy aims at matching ICMP error messages. Some ICMP error messages contain parts of the packet that caused the error message [139]. We compare the partial packet in the payload to recent egress packets. If the packet header is identical to a recent egress packet, we match the ICMP error to the corresponding process. Amongst others, this allows to match traffic of utilities such as `traceroute`.

The second matching strategy enables associating ICMP(v6) echo requests and replies to a process. Every ICMP(v6) echo request has a unique ID and sequence number [139]. Thus, we parse the header of ICMP(v6) echo replies and associate the packet to the PID of the belonging echo request or vice versa. This enables us to match traffic from utilities like `ping`.

Not all packets have have an associated process. Examples are broadcasted packets destined for other hosts in a network, such as discover messages in the Dynamic Host Configuration Protocol (DHCP)v4 [140].

The indirect matching introduces the need for additional memory and matching architecture to correlate ingress to egress traffic. This can increase the CPU and memory load of our approach. We investigate these effects in the evaluation.

5.4.4 Supporting Data Structures

We correlate the data from eBPF-hooked kernel functions and the packet sniffer to associate packets and processes. The matching framework uses two data structures in the user space to process the gathered data efficiently. These are a *traffic history* and a *packet buffer*.

The *traffic history* contains all recently matched packets and PIDs as events. There are two options to generate such an event. The first option is to use the information collected in `xmit_event` via eBPF for egress traffic. The second option is the result of an indirect matching strategy for ingress traffic. Egress traffic with a PID $\neq 0$ is added to the traffic history and logged immediately. Some packets have PID 0. Examples are ARP and TCP handshake packets. The process with PID 0 is a special process in Linux [141]. For simplicity, we assume that a matching to the process with PID 0 is an association with the kernel. We try to match egress traffic with PID 0 in indirect matching to reduce false matching to the kernel. For example, while the TCP handshake is handled by the kernel, its packets belong to the following flow that contains the actual data provided by the process. To avoid filling up the memory, the traffic history only contains recent matchings of the last 2 s. We limit the traffic history since correlated flows typically happen within a short time interval [11], we assume the same applies to packets. This value can be adjusted in our framework. Increasing the traffic history leads to an increased memory usage at a higher throughput as the matcher retains information longer in memory.

We use the *packet buffer* as data source for the indirect matching for ingress packets and PID 0 packets. Therefore, the packet buffer contains all recent and unmatched ingress packets. Similar to the traffic history, we keep packets in the packet buffer for a limited time; in our implementation 3 s.

The sources for the indirect matching are the traffic history, kernel events with PID 0, and the packet buffer. To match ingress and PID 0 events to egress traffic, we compare the packets in the packet buffer to the traffic history. If our matcher associates a packet from the packet buffer with a PID, it removes it from the packet buffer. If the indirect matching does not succeed after 3 s, the maximum lifetime of the packet buffer, previous PID 0 matchings are considered valid. We assume packets without matching within the 3 s do not have an associated process. By limiting the time to 3 s, we limit the memory consumption during runtime. Figure 5.1 shows the architecture of the matcher.

5.4.5 Matcher Architecture

This section describes how to implement our approach to efficiently and reliably associate packets and processes. After a process uses a syscall to send a packet, the kernel assembles it using vari-

ous kernel functions. At the end of the packet assembly, the kernel calls either `dev_queue_xmit` or `dev_direct_xmit` to send out fully assembled packets. We instrument eBPF to extract the information about the packet and PID by parsing the `skb` provided to the kernel function. These events are then stored in an "unmatched events" buffer (left side of Figure 5.1). The matcher then processes these events. If the extracted PID $\neq 0$, it is possible to directly identify the process that caused the construction and sending of the frame. These packets are directly matched, and the result is added to the traffic history. The traffic history contains recently matched events and is used for indirect matching. If the PID $= 0$, we assume that the kernel is responsible for this packet. Events initially associated with PID 0 are also enqueued for indirect matching to match later more fitting events. Indirect matching uses associations in the traffic history to associate ingress packets to a process. It uses an open raw socket to receive those packets. Therefore, the indirect matching uses one of the strategies described in Section 5.4.3. The matcher keeps the association to the kernel if it does not identify a more fitting process for a packet associated with PID 0. Ingress packets that cannot be matched do not receive an associated process. This can be, e. g., in the case of broadcasted packets in the network. The associations between packets and processes are also stored as output for postprocessing applications.

5.5 Evaluation

We conducted several experiments to evaluate if the proposed matcher fulfills the previously introduced requirements for packet-process associations. We use the pos framework [142] for automated and reproducible experiments. The framework was implemented in Python [122] and Rust [123]. The evaluation from both implementations show similar trends. However, due to the Global Interpreter Lock (GIL) and Python being an interpreted language, the Pyton implementation is slower than Rust. Thus, the results obtained in the same experiments with the Python implementation of the framework are several magnitudes slower. The following evaluation presents results from the implementation using Rust [143]. The Rust-based implementation uses redbpf [144] to instrument eBPF to hook the selected kernel functions. Both implementations only probe the `__dev_queue_xmit` function. For all experiments, the measurement setup consists of two directly connected hosts. Both hosts have an Intel Xeon D-1518 CPU, 32 GB RAM and runs Debian Bullseye (kernel v5.10.0-8-amd64). We first evaluate how the usage of the framework affects the operation of the monitored end host. We then evaluate the accuracy and completeness of the collected data.

Resource Overhead

We first evaluate the limits of the matcher at higher throughput rates. We use the `trafgen` tool to generate traffic at different speeds. During the traffic generation, we measure the CPU and memory usage in intervals of 5 s on the host. Each experiment has a 280 s duration.

The CPU overhead only depends on the throughput and is independent of the packet size. This is because the framework only processes packet headers and a fixed number of bytes from the payload. Thus, the packet size does not affect the association process on the CPU. Figure 5.2

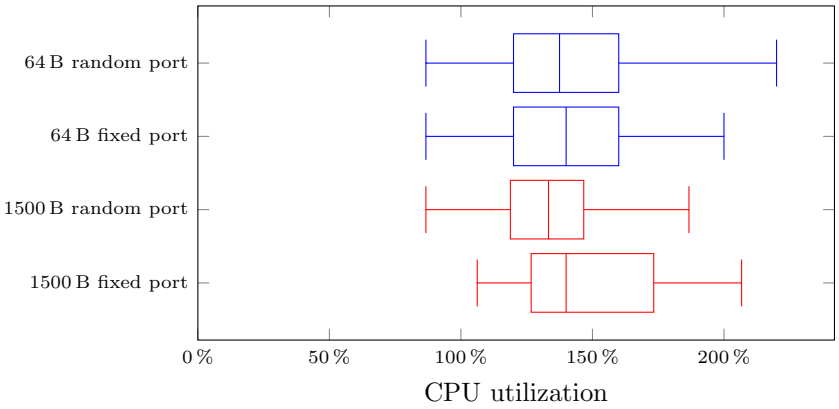

Figure 5.2: CPU utilization with a gap of 100 µs inter-frame gap

shows that the CPU usage is independent of the packet size. Matching packets with a size of 64 B (blue) uses roughly the same CPU as matching packets with a size of 1500 B (red) at the same inter-frame gap. On our machine, associating packets at a rate of 10 000 packets/s utilized approximately 140 % CPU, i. e. 1.4 of the host's cores. The graph also shows that the CPU usage ranges from 86.7 % to 220 %. This range is also independent of the packet size. A possible reason for those fluctuations are other running processes on the OS that run during the experiment execution. We also evaluated if there is a difference in the overhead if packets belong to the same connection (fixed ports), or different connections (random ports). The figure shows that the CPU overhead is the same.

To demonstrate that the CPU load depends on the number of packets, we repeated the experiment at lower emission rates and a packet size of 64 B. The results confirm our assumption that the CPU usage depends on the throughput (Figure 5.3). A 5 ms inter-frame gap between packets causes a CPU of about 13 %. If the inter-frame gap decreases, the packet rate and the CPU overhead increase. A gap of 2 ms leads to a CPU load of about 25 % which increases to 38 % at a gap of 1 ms. All of these values are below the 40 % overhead of standard logging-based approach mentioned in [125], [130].

The memory usage of the matcher depends on packet sizes, throughput, and port variation. The traffic history and packet buffer data structures cause this relation as they buffer information for a limited time to enable indirect matching. Constant packet rates and sizes to fixed ports lead to constant memory consumption during runtime (Figure 5.4). This is because these data structures retain entries for a predefined maximum time. In this scenario, packets cycle at an equilibrium that depends on the matching speed. If the matcher does not identify an associated process in time, it discards the packets after a maximum buffer time. Larger packets at the same packet rate lead to larger memory usage due to the packet buffer keeping packets for ingress matching. The traffic history contains more associations and processes if random ports are used. This increases the memory usage of the framework at the same packet size and speed (solid and opaque markers in Figure 5.4). Figure 5.4 shows the memory usage at a rate of 10 000 packets/s and a fixed port. When matching packets with a size of 1500 B (120 Mbit/s), the matcher

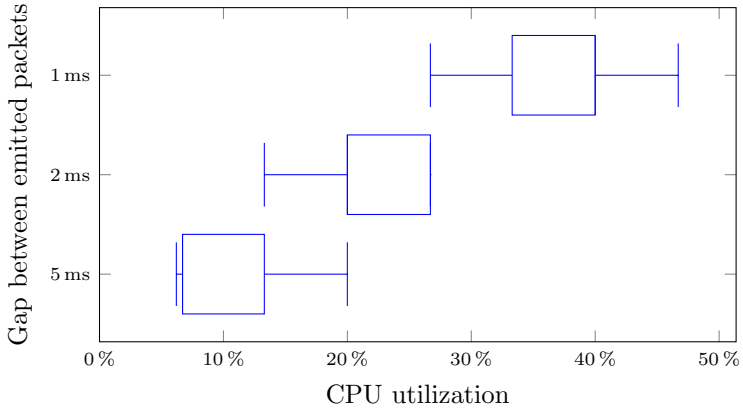

Figure 5.3: CPU load due to the matcher at different emission rates, all packets have a size of 64 B

.

requires approximately 102 MB RAM. A reduced packet size of 64 B at the same throughput leads to a memory consumption of 60 MB RAM.

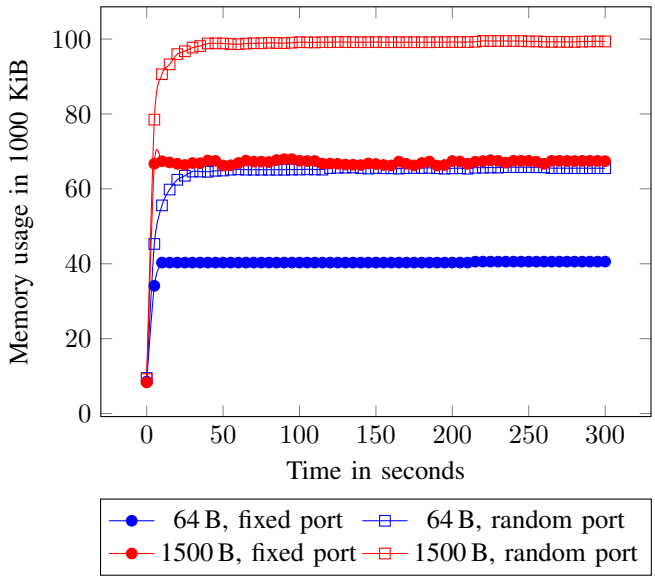

Figure 5.4: Average RAM usage with 100 µs inter-frame gap

Completeness and Accuracy

We conducted several experiments to assess the speed and accuracy of the association. We measure the time to match, i. e. the time it takes on average to associate a packet to a process, and the general ratio of associated packets to unassociated ones.

The packet size has the highest impact on the time to match. This is expected because packets are copied from the kernel space to the packet buffer in user space for indirect matching. Larger packets require a longer time to be copied to the user space. At a throughput of 10 000 packets/s

the Rust based implementation needs 7 µs to associate 64 B packets to processes. This time increases to 24 µs to associate 1500 B packets to processes. The increased time to match can induce issues for associating packets and processes at a higher throughput. If the matching takes too long, packets cannot be matched in time and are wrongfully matched to PID 0 or not associated at all.

The matcher has a matching rate of 99.7 % for up to 10 000 packets/s with a size of 64 B. For 1500 B packets, the matching accuracy stays at 99.9 % at 60 Mbit/s. The accuracy drops to 96.3 % at 120 Mbit/s. A matching rate of less than 100 % is realistic. Not all packets that a host receives have an associated process. Broadcast or multicast messages, such as DHCP, intended for different hosts, may be received at a NIC, but do not have an associated local process. A more problematic reason for non-associations in case the indirect matching happens too slow. However, these cases are less realistic to occur in real life since few applications cause a high throughput over an extended period of time.

By manual inspection, we assessed the accuracy of the matching. In our inspection, we did not identify any wrong associations. In our experiment, we observed wrong matchings in case of a CPU load $\geq$ 90 %. In this case, packets were associated with the matcher's process. This is a result of the used redbpf framework and not an issue of the method. We conclude that the framework fulfills all requirements we elicited in Section 5.2.

5.6 Context to Multi-Layer Relationships

The approach presented in this chapter shows how it is possible to efficiently collect data that allows measuring the relationship between the network and application layers of the architecture introduced in Section 2.2.1. Applications located on the application layer, which communicate with remote services, rely on the services provided by the network layer, causing events on this layer. However, OSs typically provide networking functionality to local applications. This separation of tasks leads to a gap between the two layers. The presented matcher addresses this gap between the application and network layers and allows to attribute packets to applications.

Existing approaches to overcome the gap between our architecture's network and application layers have limitations, such as incomplete data collection or a significant overhead. Our matcher overcomes these limitations to collect information about the network activity of individual processes. As a result, it is an essential tool that allows to measure the association between events on the application layer and events on the network layer. Based on this data collection, it is possible to analyze the multi-layer relationship between network and application layer events. This in turn allows to identify inconsistent behavior of devices on those layers. We present one method for such an analysis in Chapter 6.

5.7 Key Results

The association between packets and frames is challenging since OSs autonomously manage network activity for applications. Existing methods create significant computational overhead on the end host for non-networking activity or are incomplete. This chapter introduces a method to associate packets and processes by leveraging eBPF. In contrast to existing approaches, the usage of eBPF allows to associate packets to processes that do not use all fields of the IP-5-tuple, such as ICMP. This extends the capabilities for reasoning about causes of network traffic on a host to include lower level traffic.

Our evaluation indicates that the approach induces less CPU overhead than other approaches, such as logging-based methods. The induced memory and CPU overhead association depends on the throughput, packet size, and number of processes that actively use the network. The overhead only occurs when processes use the network.

Our approach bridges the user-space kernel gap and enables post-processing applications to reason about application behavior. The presented framework allows the collection of detailed information about the network activity of individual processes.

Chapter 6

Application Network Profiles

Author's contribution: *The contents of this chapter are mainly based on the sections on Network Profiles of the publication L. Wüstrich, M. Schacherbauer, M. Budeus, D. F. von Künßberg, S. Gallenmüller, M. Pahl, and G. Carle, "Network Profiles for Detecting Application-Characteristic Behavior Using Linux eBPF", in* Proceedings of the 1st Workshop on eBPF and Kernel Extensions, eBPF 2023, New York, NY, USA, 10 September 2023*, ACM, 2023, pp. 8–14.* DOI: *10.1145/3609021.3609294. The publication results from various theses in which the author acted as advisor, formulated the research question, methodology, and evaluation strategy. Parts of the publication are based on the master's thesis by Markus Schacherbauer [145] in which the framework for profiling applications was developed under the author's guidance. This thesis has an extended description of the proposed framework and a discussion of the advantages and use cases of network application profiles.*

This chapter focuses on the multi-layer relationship between our logical structure's network and application layers (Section 2.2.1). It presents a method to analyze the data collected by the matcher that Chapter 5 introduced, bridging the gap between those two layers. We introduce network application profiles, which allow us to reason about the link between events on the network layer and application activity. With this work we answer *Research Question (RQ)2.2: How can we determine typical application behavior and identify deviations?* Based on our insights, we use the proposed network application profiles to monitor the previously identified multi-layer relationship to identify anomalous application behavior. This allows us to detect process masquerading attacks that cause inconsistent activity on the network and application layers.

6.1 Motivation

The matcher introduced in Chapter 5 allows to collect detailed information about the network usage of individual processes. This enables to characterize an application's communication behavior. Several reasons can cause an application to use the network [3], [34]. Typical application profiles monitor hardware usage, such as memory consumption, function execution time, or Central Processing Unit (CPU) usage [146]. However, they do not capture information about communication behavior of software. Our work addresses this shortcoming by introducing application profiles that focus on their network activity.

A variety of use cases benefit from knowing typical application behavior. For example, it helps Network Intrusion Detection Systems (NIDSs) [67] to identify malicious flows or to identify which applications run on a device [65]. Knowledge about network dependencies and characteristic flows allows to identify applications [9]. Another use case is the identification of root causes of service outages when an application stops working due to failing remote services, e. g., during maintenance [3].

This motivates a fine-granular characterization of application behavior. *This chapter introduces application network profiles to capture and characterize application communication behavior.* Since the multifaceted characteristics describe general communication behavior, we combine a series of techniques to build application network profiles. These profiles can be used afterward to derive further insights about an application and perform Anomaly Detection (AD).

Our approach helps to highlight changes in typical application behavior, exceeding the capabilities of typical Machine Learning (ML)-based Anomaly Detection Systems (ADSs), often restricted to binary classification without automatic reasoning for the classification. Network application profiles allow the identification of attacks in which the application behavior is inconsistent with its network behavior. Network application profiles help to detect attacks such as masquerading attacks which influence application and network layer (Taxonomy, cf. Chapter 4). Our evaluation shows that even if an attacker masquerades a process and attempts to mimic typical network traffic emitted by the original application, the introduced traffic interrupts the typical behavior. Thus, by setting network flows into context with applications, network application profiles make it more difficult for adversaries to hide their activity since the behavior on the application and network layer need to match. Other work has shown that combining network data with processes also increases the accuracy of ADSs [67].

6.2 Facets of Application Communication

There are various aspects that describe the network behavior of a device. Therefore, a variety of methods exist to capture the characteristics of a host's communication behavior. We use similar techniques on a more fine-granular level to formalize the properties of application communication in application network profiles. We classify those properties into *objective* and *statistical* properties (Table 6.1).

Table 6.1: Relevant features of a network profile

Property	Type	Related Work
Network protocols	Objective	[3], [9], [11], [66], [124]
L4 destination ports	Objective	[3], [9], [11], [66], [124]
Associated processes	Objective	[147]
Sequences	Objective	[11], [65]
Communication partners	Statistic	[34], [65]
Avg. number of flow bytes	Statistic	[151]
Avg. packet size	Statistic	[150]
Avg. flow duration	Statistic	[66], [150], [151]
Periodicity	Statistic	[7], [34]

Objective properties capture factual aspects of communication behavior. Those include used protocols, ports, and remote services [3], [9], [66]. For example, browsers rely on Domain Name System (DNS) resolvers to identify the destination Internet Protocol (IP) address of a request in the HyperText Transport Protocol (HTTP) [11]. Other applications rely on pre-defined endpoints to fetch and process data. This results in a set of common communication partners [3], [65]. In industrial setups, devices may have static sets of communication partners [36]. Further, there are flow sequences due to dependencies such as the relationship between DNS and HTTP [11]. An application potentially has a set of related processes on the end host running it. In the previous example, browsers rely on Operating System (OS) services for name resolution, such as `systemd-resolved` on Linux. Therefore, applications can have a set of related processes that occur along with the main application. This process set also characterizes application behavior that can be used for malware detection [147]. Similar to sequences of network flows, there can exist specific interaction patterns between applications. While such patterns are particularly predictable in fully automated processes, variations of the sequences are common in user-driven applications.

Statistical properties capture meta-information about communication. Those properties depend on the environment and usage of an application [148]. Statistical reasoning is necessary since most traffic is nowadays encrypted making deep packet inspection impossible. Periodicity is a characteristic property in various scenarios, such as industrial environments [7], [34]. Periodic behavior also occurs in traditional networks, e. g. due to Network Time Protocol (NTP) [149] or periodic data fetching. Other statistical properties cover flow characteristics. These occur either on a packet level, e. g., average packet size [150] or packet size distribution, or on the flow level [151]. Examples are the average flow length or number of communication partners. Statistical features also include ratios of traffic properties. Examples of ratios are the ratio of egress to ingress traffic, or usage of L4 protocols. Such ratios are often scenario-specific; for instance, the ingress and egress traffic volume ratio is typically asymmetric in client-server scenarios but more balanced in peer-to-peer scenarios. Since such characteristics highly depend on the scenario, we exclude them from our network profiles.

6.3 Creating Network Profiles

The previous section introduced relevant properties that define an application's communication behavior. Various mechanisms exists to capture individual properties, such as periodicity or commonly used ports. However, currently there is no technique that captures all properties at once. Therefore, in order to create network profiles which capture all relevant characteristics, we combine multiple techniques.

There are several methods to identify each of the relevant properties individually. However, in order to create an accurate application behavior profile, it is necessary to set individual characteristics into context of each other. For example, while an application may use both destination ports 53 and 443, and the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP), it is important to register typical combinations of characteristics. While the combination of UDP and port 53 may be normal behavior, a UDP flow to destination port 443 is unusual. Thus, setting properties into context makes the characterization of application behavior more accurate than listing valid values of individual properties.

An intuitive approach to identify typical combinations of feature values in large data sets is the use of ML. ML methods, such as auto encoders and Neural Networks (NNs), can learn statistical properties in data, including typically co-occurring values to classify observations. Similar to AD methods (Section 2.4), ML methods have an initial training phase. In this phase, the models learn patterns and characteristics in a given data set. After the training phase, the models can classify newly observed data. However, the reason for a classification of ML methods is often unclear to humans [81]. Thus, the use of such approaches makes it challenging for humans to understand the causes of a classification during runtime. One of the goals of network application profiles is to formalize typical behavior, making it possible for network operators to reason about an application's behavior. Therefore, our proposed profile creation and behavior analysis combines methods that infer statistical information while retaining human interpretability. Since the reasoning of ML methods like NNs are not human interpretable, we do not use such methods to create network profiles. In the following, we explain the techniques we use to create network profiles.

6.3.1 Characteristic Flow Features

A fundamental aspect of application communication are typical features of its network flows. These can be characterized by the typically used L3 addresses, ports, and transport layer protocols. In addition, the average payload size and flow duration are characteristic features of an application's behavior or used higher layer protocols [4]. It is possible to capture information about commonly used ports and protocols by creating lists and combine them in rules. However, while rules capture valid feature combinations, they do not contain information about the likelihood of their occurrence. As a result, such lists do not help operators to reason how common an observed characteristic for a flow associated with an application is. One method that allows to capture and combine multiple features is the *Energy-based Flow Classifier (EFC)* [66]. The EFC generates a fully connected and weighted graph for each flow. Each the vertex of the graph represents an individual characteristic of the flow. The weight of each vertex represents the like-

lihood of the characteristic. Edges represent the link indicating the combination of individual characteristics while its weight indicates the likelihood of the combination. By design, a low weight indicates a high likelihood, while a high weight indicates a rare occurrence, i. e. a low likelihood. Similar to NNs, EFCs analyze data to derive common occurrences and combinations of characteristics in an automatic way. However, in contrast to a NN, the weights of the graphs created by an EFC allows a human interpretable explanation of the classification.

We integrate multiple characteristics we consider important (Table 6.1) into the EFC of each application's network profile. Those are the L4 protocol, ports, number of ingress and egress packets and bytes, and flow duration. Unusual characteristics have a low likelihood and result in a "high" energy, indicating unusual behavior. Due to the modular design of the graph and weights in an EFC, it is possible to identify which characteristic is unusual and causes a high energy. We provide flows belonging to individual applications as input for the EFC, collected by the framework introduced in Chapter 5. By using this approach, we can learn typical combinations of characteristics exhibited by network flows belonging to an application. The result is an application-specific EFC, which makes up one building block of each application's network profile.

6.3.2 Characteristic Flow Sequences

To identify typical sequences of the flows exhibited by an application, we use *Episode Rule Mining (ERM)* [152]. ERM allows to extract reoccurring ordered sequences of events, called episodes, from a list of events. In addition to the order, ERM introduces a temporal threshold in which events can occur in an episode. The threshold enables the implementation of restrictions on the time that can pass between flows in an episode. If more time than the threshold passes, ERM does not consider them to be part of an episode. Therefore, choosing the threshold causes a tradeoff that can affect the quality of the result. Large thresholds can lead to episodes that are too long, while low thresholds can end episodes too early, missing parts of the sequence. We choose the threshold dynamically depending on the profiled application. The resulting rules from ERM describe those episodes. In our approach, episodes describe characteristic flow sequences that occur while running an application.

ERM has advantages over other pattern mining approaches, such as Frequent Itemset Mining (FIM) [153], Association Rule Mining (ARM) [154], or Sequential Rule Mining (SRM) [155]. FIM, as used by Popa et al. [3], identifies the existence of correlated flows but not their sequence order. This limits the length of identifiable sequences to 1. ARM allows to identify if multiple flows typically occur together. This allows to validate if a set of flows related to an application occurs across multiple captures. However, ARM does not identify the order in which flows appear. SRM addresses this shortcoming and allows to also identify flow orders. Both ARM and SRM disregard the temporal distance between flows. However, we assume that relevant flow sequences occur within a short time frame. This is in line with related work, which assumes that flows with a causal relationship occur within 1 s [11]. While slicing helps to address this issue, it introduces the risk of splitting a sequence. ERM allows to specify a sliding window, in which dependencies in an event list can appear. This allows to limit the time frame in which sequences have to occur.

We identify flow sequences in two steps. ERM requires an event list as input. We generate a list of flow-events from the matcher output. Each event has the timestamp of the first packet of the represented flow, and information about the flow itself. If multiple observation periods exist, we aggregate them into one list and rescale timestamps to indicate relative time differences between flows. To avoid the identification of unrelated sequences across different observation periods, we insert an artificial buffer between those periods. The buffer exploits the maximum time between flows in a sequence which can be specified. In the second step, we apply ERM to the created event list to identify the sequences. To extract episode rules, we use the EMMA algorithm [156]. If there are two flows to destinations A and B, and it is expected that B occurs within a time frame after A with a pre-defined *minimum confidence*, the algorithm creates a rule $A \Rightarrow B$. The output of the module is a set of rules that capture both common flow sequences and typical communication partners of an application. This allows to put flows classified by the EFC into the context of each other.

6.3.3 Common Communication Partners

Particular devices in Industrial Control Systems (ICSs) and the Internet of Things (IoT) can have a fixed set of communication partners [36], [37]. However, applications in other environments can also have common communication partners due to dependencies [3]. We aim to identify common communication partners that we include in our application network profiles. Since traffic is typically encrypted, it is usually impossible to reason about dependencies by inspecting the payload of packets in flows. To avoid this limitation, we use an approach that only analyzes relevant headers of a packet in a flow, i. e., destination IP, destination port and L4 protocol.

We aim to create a list of common communication partners, e. g., due to static dependencies [3], in our application network profiles. Static dependencies cause an application to access a fixed remote service denoted by a fixed IP and destination port. There are several approaches to identify common communication partners. One method is to analyze the source code of applications or configuration files [157]. However, this approach is infeasible as some applications have closed source code and do not rely on configuration files. Thus, we choose a statistical approach to identify common communication partners.

One approach to identify common communication partners to analyze the frequency of flows. If an application contacts a server frequently, this approach is likely to classify it as a common communication partner. A drawback of this approach is that it assigns a low likelihood to flows that occur less frequently. However, a dependency can also result in less frequent flows, e. g., during an initial authentication during its startup. A frequency-based analysis would not identify such flows as common communication partners.

We use an approach that estimates the likelihood of applications accessing a remote service. Therefore, instead of using the frequency of communication flows, we measure whether an application contacts a remote service during its execution. We measure multiple runs of an application. For each run, we analyze if it contacts specific remote services. After the information collection, we analyze the contacted services and the number of runs in which the application

accesses them. For each communication partner, we calculate $f = \frac{\text{partner contacted}}{\text{number of runs}}$. If f exceeds a threshold, we classify it as a common communication partner.

Choosing this threshold is a tradeoff between falsely marking communication partners as characteristic partners and missing dependencies. A higher threshold requires more deterministic communication with a service for inclusion in the common communication partner set. However, a low threshold can falsely include communication partners as it can include partners that the application accesses in fewer runs. In our experiments, we chose a threshold of 0.8.

From this analysis, we generate a list of potential communication partners for the network profile.

6.3.4 Associated Processes

An application's communication can also influence other OS services by causing them to communicate or change their communication behavior. Therefore, it is relevant to identify such processes affected by an application on an OS as well. Identifying the associated processes of an application is challenging. Associated processes of application depend on its type and purpose, the host OS, and its configuration. There are various approaches to identify such dependencies. These include configuration file and code inspection [157], log mining [158], and other monitoring approaches, such as network monitoring [159]. Manual code inspection and configuration file analysis depend on the accessibility of information. Such information is not available for all applications, limiting the applicability of those approaches. Log mining approaches and network monitoring correlate events to identify relationships between them. As discussed in Section 5.2, those approaches have advantages and limitations.

Application network profiles focus on network activity. The matcher presented in Chapter 5 allows to efficiently collect information about network activity. It generates a list of processes that use the network. This allows us to limit the analysis to processes that interact with the network. Thus, the matcher allows us to use a combination of log mining and network monitoring by processing that list. We use a heuristic approach to identify processes that the analyzed application influences. Therefore, we first generate a baseline of typical OS behavior and communication. After generating the baseline, we start the profiled application and analyze changes in the behavior of the OS.

In order to generate the baseline scenario, we first monitor a clean OS without running applications with the matcher for a few hours. The baseline identifies default OS processes and their communication behavior, e. g. NTP. In the second step, we run the profiled application and identify relevant changes. Relevant changes include new processes and their communication and how existing processes change their communication behavior. The changes in network activity can be caused by the profiled application itself or related processes such as `systemd-resolved`. For example, browsers can increase the use of `systemd-resolved` due to additional DNS requests to serve HTTP requests. We repeat the second step multiple times to increase the likelihood of identifying the correct processes. From the differences between the baseline and the second measurement, we create a list of processes associated with the profiled application. We include the list of associated processes in the network profile.

6.3.5 Periodic Communication

Finally, the network profiles also consider periodic communication behavior of applications. Periodicity is often prevalent in fully automated processes, such as NTP or in ICSs [34]. We use a *periodogram* [160] to identify periodic communication. Instead of manually identifying significant periods, we automatically identify them by using the Fisher g-statistic [161]. We generate the periodogram for a recorded trace by encoding the traffic of an application as a boolean time series. If the application caused network flows during a short period, we encode it as 1 and 0 otherwise. In our implementation, we use bins with a length of 2 s. By introducing this abstraction, it is possible to avoid bursty activity-biased period detection [162].

6.3.6 Resulting Profiles

The combined methods allow to create a holistic description of typical application-related network activity (Table 6.1). The used EFC allows to identify and validate typical characteristics of individual flows occurring in the context of applications. This includes commonly used transport layer protocols and ports, as well as statistical information about packet sizes and duration of flows. The chosen method allows us to contextualize the characteristics of those flows. In order to set flows into the context of each other, we use ERM. ERM identifies typical flow sequences that an application may exhibit, potentially setting individual flows into the context of each other. We identify associated processes via a heuristic approach. Further, we use two heuristic approaches to formalize characteristic application behavior. First, we identify common communication partners. Second, we also measure changes in the host's communication activity. By measuring changes in communication activity, we identify the effects an application has on the communication activity of other services. Finally, we use an automatic analysis of periodograms of network activity to identify periodic behavior.

Network profiles aggregate information from multiple techniques. An extension for additional techniques to include new characteristics is possible by design.

6.4 Profiling Applications

We implement our profiling framework in Python, realizing the techniques described in Section 6.3. The framework is modular and extensible, allowing to add new methods for analyzing communication behavior in the future. Like an ADS, the framework has a reference generation phase and an AD phase. In the reference generation phase, the framework creates a profile for an application using previously collected data. In the AD phase, the framework periodically processes newly collected data in a sliding window approach on an end-host to monitor application behavior.

We use the matcher presented in Chapter 5 for the data collection. The output of the matcher is a list of packets and associated processes. Some of the methods used require different data granularity. Therefore, the profiler framework pre-processes the association list to prepare the data for the implemented analyzing techniques. The pre-processing creates an additional flow-level view of the information, aggregating flows using the IP-5-tuple. After pre-processing the data,

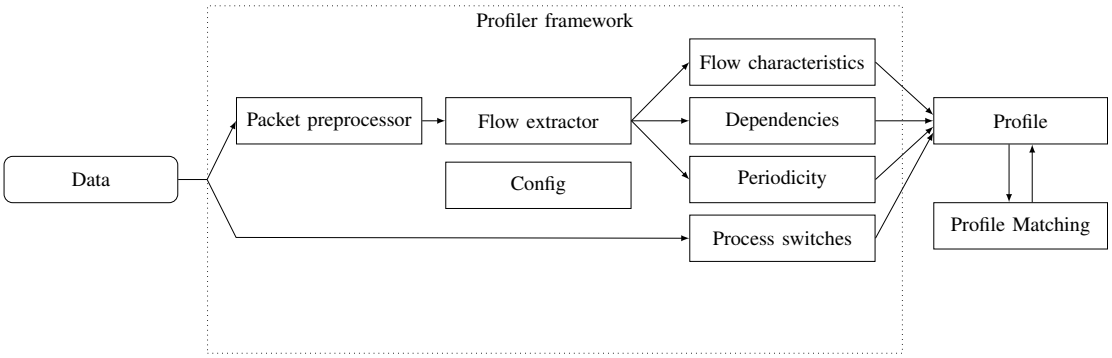

Figure 6.1: Procedure of creating an application profile

the profiler processes the captured information. Some of the mechanisms require configuration parameters, e. g. the threshold for identifying common communication partners. Choosing the right values depends on the environment and profiled application. Thus, the values can be adapted based on the considerations described for the individual techniques (Section 6.3). The framework stores those in a config file that the mechanisms use during their runtime. The profiler then applies the chosen techniques to extract characteristic application behavior. During the profile generation, the profiler stores the results in an application profile. The profile contains the result for each analysis, which the profiling framework uses to monitor applications in the future. Figure 6.1 presents the procedure.

The resulting size of an application's network profile depends on the complexity of an application's characteristics. For example, a profile of an NTP demon, which only uses one protocol and periodic requests to a single destination but exhibits no communication sequences, has a profile with the size of 116 kB. In contrast, the profile of a Chrome browser contains more complexity, e. g., due to various sequences uncovered by ERM has a size of ≈16 MB in our experiments.

Monitoring application behavior after the creation of its profile follows a similar procedure. After pre-processing the data, the profiler applies the same analytics to derive or identify the exhibited characteristics. It then compares the observed characteristics to the expected ones in the application profile and creates a report. The "profile matching" block in Figure 6.1 represents this step. The application behaves as expected if the characteristics are consistent with the profile. Otherwise, the report contains information about how the monitored behavior differs from the profile, e. g., if the periodicity differs. In addition to highlighting abnormal behavior, this formalization helps operators understand how an application behaves differently than expected in case of abnormal behavior. On the other hand, it formalizes typical application behavior in a human-interpretable way.

6.5 Identifying Abnormal Application Behavior

In this section, we use network profiles for AD. We identify a malicious process that takes part in simulated botnet communication and masquerades itself as NTP daemon, mimicking NTP traffic. Masquerading to avoid detection is a common strategy [121].

We automate all experiments using the pos framework [142] similar to the evaluation in Chapter 5. The experiment setup consists of two directly connected hosts. Both hosts have an Intel Xeon D-1518 CPU and 32 GB RAM and run Debian Bullseye (kernel v5.10.0-8-amd64). We run the matcher and profiler on the first host. In addition, the first host runs applications that cause communication and are monitored by our framework, an NTP daemon in our experiment. The second host runs software to serve and reply to queries from the applications running on the first host.

We first create a network profile of the `ntpd` process, an NTP daemon. The data basis for the profile is ten 5 min traces of packet-process associations for `ntpd`. The EFC method identifies that `ntpd` has short-lived flows sending UDP packets to a single destination. Each NTP flow has source and destination port 123, in line with the associated RFC 1305 [163]. The method for identifying periodic behavior extracts a periodicity of 62 s to 75 s for the flows. The real periodicity of the flow is between 67 s and 69 s. Longer and more traces of application behavior allow a more accurate estimation of periodic behavior. Since NTP uses individual flows for each request, the ERM approach identifies no sequences that `ntpd` exhibits. Therefore, this method does not create any sequence rule for the profile of `ntpd`.

After creating the profile, we implement a Command and Control (C2) server running on the second host of our setup. We also implement a botnet agent on the monitored host. The botnet agent uses UDP and destination port 123 on the C2 server to contact its control server to mimic NTP traffic. On the local host, the botnet agent masquerades itself as `ntpd`. Like other botnets, the agent contacts the C2 server in periodic intervals [164]. The C2 server responds to the agent's queries. Besides communication, we do not simulate any further behavior of the botnet agent and C2 server.

We then collect new traces in which both the botnet agent and `ntpd` process run concurrently. Due to the masquerading of the botnet agent, the matching framework associates the agent's flows for the communication with the C2 server to `ntpd`. Using our taxonomy (Chapter 4), the masquerading of the process affects the authenticity of the perceived application on the host system (APP.AUTH). In addition, the attack also makes it impossible to identify which process caused the network activity (NET.AUTH). Further, since it is no longer clear which application caused the network activity on the host, the attack also affects the APP.ACC security goal. The result of the attack is a mismatch between application activity on the host and the network activity, which is the result of the concurrent activity of the botnet and `ntpd`.

In the following section, we describe how this affects the perceived behavior of `ntpd`, resulting in the detection of abnormal behavior.

Characteristic Flow Features: The EFC part of the profile assigns all flows from the botnet agent a "high" energy, indicating unusual flows. This has two reasons. First, `ntpd` uses 123 for both the source and destination port. However, the botnet agent uses an ephemeral port as source port. This differs from `ntpd`, resulting in a higher energy than benign flows. The second reason is the packet size of the botnet traffic. NTP traffic typically has a constant packet length due to the fixed information it contains. In our experiment, the botnet traffic has differing packet lengths of up to 1200 B. Therefore, the EFC also highlights those abnormal packet sizes of the

botnet traffic. As a result, only 56.69 % of the observed flows match with typical characteristics exhibited by NTP that the previously built EFC observed.

Flow Sequences: The botnet traffic occurs independently of the typical NTP traffic. As a result, the ERM does not identify additional flow sequences. Therefore, the additional botnet traffic does not cause unknown flow sequences.

Common Communication Partners: The botnet agent communicates with the C2 server. As a result, it appears if the `ntpd` process communicates with the NTP server and the C2 server. The profiler detects the C2 server as an additional common communication partner of the NTP process. However, the profile of `ntpd` does not contain the C2 server as a dependency. This indicates unusual activity of `ntpd`, since it communicates with a new, previously unknown, entity. The new destination IP address of the agent's communication with the C2 server does not fit the typical communication behavior of `ntpd`.

Associated Processes: The botnet activity does not cause any other processes to change their behavior. Therefore, the associated processes are identical to the ones stored in the network profile.

Periodic Communication: The traffic from the botnet agent to the C2 server changes the periodic behavior of the `ntpd`. The additional flows of the botnet agent disrupt the periodic flows of the `ntpd` occurring approximately every 67 s. There are three possibilities in which this can affect the periodicity of `ntpd`. First, if the agent times its communication with the C2 server to match the typical periodicity, the ERM algorithm detects a new flow sequence. Second, similar to the first case, the botnet can attempt to match the periodicity of `ntpd` and send out data at the same period in the middle of each regular interval. In this case, the periodicity detection would identify periodic behavior at exactly half of the typical period, highlighting unusual behavior. In the third scenario, the botnet agent disrupts the periodic behavior of `ntpd`. While both the agent and `ntpd` process cause periodic flows, the combination results in a non-periodic sequence of C2 traffic and NTP flows. The third scenario occurred in our experiment. Therefore, the analysis could not identify any periodic behavior of `ntpd`, deviating from the expected periodicity.

In our experiment, using application network profiles allows to identify several characteristics in which `ntpd` changes its behavior due to the botnet traffic. By formalizing the typical behavior of applications, network profiles provide operators the means to identify how monitored behavior differs from the expectation. In addition, due to the association with processes, it allows an operator to identify malicious applications from network traffic. In contrast to ML-based approaches, the proposed network profiles are human-interpretable. While other ADS can identify malicious network flows by identifying unusual features or periodicity, no mapping allows those systems to identify the processes causing those flows. Thus, our method can provide additional information, allowing operators to identify malicious processes.

The presented approach also has limitations. Host behavior changes when multiple applications run concurrently. In particular, challenges arise if the applications have the same local processes associated with them. E. g. if both Firefox and Chrome run simultaneously, it is impossible to

identify which application triggered a DNS resolution. This issue can be addressed by creating host network profiles that characterize host behavior with applications running concurrently or enhancing the tracing of local process dependencies. Another issue is that applications that rely on user input may exhibit different behavior depending on the user. If the user influences an application's behavior, the application profile implicitly describes a user's patterns, which raises data protection concerns. Therefore, using network application profiles works better for fully automated environments, such as the environments introduced in Chapter 2, than in environments with high user activity.

6.6 Context to Multi-Layer Relationships

The introduced network application profiles analyze associations between packets and processes collected by the matcher introduced in Chapter 5. They use several techniques to extract and monitor the relationship between events on the network and application layer of our logical structure (Section 2.2.1). The techniques for creating and using network application profiles contextualize activity from the application and network layers. In particular, automated processes exhibit predictable behavior on the network layer. The relationship between network activity and applications allows for determining if observed network activity on the network layer fits active processes on the application layer.

We evaluated the applicability of network application profiles to detect masquerading attacks in which an adversary mimics another application. We can classify the attack from our evaluation by using our taxonomy (Chapter 4). The attack targets the application layer as it hides a malicious local process. It affects the APP.AUTH and APP.ACC security goals, as processes on the local machine cannot be identified anymore. Due to the induced network activity, the attack also affects the NET.AUTH security goal, as it is no longer possible to identify the application responsible for the communication. As a result, this attack causes a mismatch of activity on the application and network layers.

By masquerading the malicious process, an attacker can trick local monitoring of the application layer as it fails to identify unusual processes. Similarly, it can hide its communication by mimicking other common communication. An NIDS fails to identify malicious applications by only monitoring the network layer. Since network application profiles monitor the relationship between the application and network layers, they can identify the introduced mismatch between the application and network layers. The adversary's additional activity breaks the local process' expected communication activity, allowing network application profiles to detect anomalous behavior.

This shows that our approach to contextualize related activity from the application and network layers helps to identify anomalies that cannot be identified by monitoring only one of the layers.

6.7 Key Results

This chapter introduced application network profiles to capture and formalize typical application network behavior answering RQ2.2. We can determine typical application behavior by identifying various characteristics that need to be considered and put into context with each other. These characteristics include information about an application's typical network flows, flow sequences, associated processes, and potential periodic behavior. Influences on those characteristics are the technology stack, user input, automation, or dependencies on local and remote services. In particular, automation and user interaction influence the characteristics of applications, causing them to be unique to their environment. We can then identify deviations from this behavior by comparing exhibited characteristics during the application's runtime to previously determined typical characteristics.

No method captures all identified characteristics of an application's communication behavior at once. Thus, we introduce an approach to create application network profiles. The approach combines multiple techniques to capture and formalize application behavior accurately and sets identified properties into context with each other. The implemented methods automatically extract typical characteristics of application behavior from process-packet associations and aggregate them into a profile. In addition to listing typical feature value ranges, the profile sets properties into the context of each other while remaining human-interpretable. Application network profiles have a modular structure, allowing a future extension to include new characteristics or the adaption of individual methods to improve a profile's accuracy.

Our application to an AD use case shows that application network profiles can accurately extract typically exhibited application behavior and identify deviations during runtime. Our approach can connect malicious flows to their causing processes by bridging the gap from processes to network packets. This helps operators identify if applications change their behavior in unintended ways or detect processes mimicking other locally installed applications. As a result, our approach can identify anomalies that cause a mismatch between application and network layer activity, e. g., due to an attack. This allows the detection of attacks that ADS, which only monitor activity on one layer, cannot identify.

Part III

Relationships Between Network and Physical Activity

Chapter 7

Identifying Physical Device Activity in Composite Signals

Author's contribution: *This chapter is mainly based on joint work published in L. Wüstrich, S. Gallenmüller, M. Pahl, and G. Carle, "AC/DCIM: Acoustic Channels for Data Center Infrastructure Monitoring", in* 2022 IEEE/IFIP Network Operations and Management Symposium, NOMS 2022, Budapest, Hungary, April 25-29, 2022*, IEEE, 2022, pp. 1–5.* DOI: 10.1109/NOMS54207.2022.9789839*. The author conceptualized the research question and designed and executed the data collection and analysis of the experiments. This thesis has an extended characterization of the properties of the Data Center (DC) soundscape, a formal definition of the procedure, and a discussion on the choice of the used parameters.*

Identifying physical device activity is the first fundamental building block of our approach to monitor the multi-layer relationship between physical and network activity. Therefore, this chapter focuses on physical device activity on the cyber-physical layer of our logical structure (Section 2.2.1). We characterize the properties of physical device activity on the cyber-physical layer and introduce a method to identify it. This allows us to set events, defined by physical device activity, on the cyber-physical layer into the context of events on the network layer in Chapter 8 and 9.

Devices in industrial environments rarely operate in isolated setups but in shared spaces with multiple devices operating concurrently. Regardless of the setup, device activity can cause changes to the environment, in particular actuators in Cyber-Physical Systems (CPSs). However, activity of devices that do not have the primary purpose to influence the environment can cause measurable physical changes, such as a booting server in a DC. If multiple devices operate in the same space, those changes overlap, resulting in a composite signal. Measurements of composite signals make it challenging to attribute changes to individual device activity. This

chapter proposes a method to identify physical changes due to physical device activity in composite signals, providing an answer to Research Question (RQ)3.1 *How can we characterize and identify physical device activity in composite signals?* We apply this method to advance the monitoring capabilities in DCs, showing one application area of our approach.

7.1 Motivation

The activity of devices influences the physical environment. In traditional computing systems, Central Processing Units (CPUs) emit Electromagnetic (EM) waves, draw power, produce audio, or emit heat. Measuring physical changes induced by devices allows to reason about the activity of the monitored devices. The source of such information is called a Side Channel (SC). In SC-attacks, adversaries infer secrets [105], [165], [166] or reconstruct procedures [74] by monitoring physical device properties. SC attacks allow them to obtain information they should not have access to. It is also possible to use SCs for monitoring device activity and detect unusual behavior. SC-monitoring allows to validate the control flow integrity of software on CPUs [167]. This emphasizes the direct relationship between physical changes and computations on devices.

The intent to alter the environment by actuators extends this relationship in CPSs to physical device activity. Thus, it is possible to use similar physical measurements to reason about the activity of actuators in CPSs. Actions executed by actuators create unique physical changes that identify them [168]. This makes SCs suitable for monitoring and validating physical device activity in cyber-physical scenarios. Similar to validating the execution of program code, it is possible to validate physical device activity by measuring environmental changes. However, in shared environments, it is necessary to attribute changes to individual devices to validate their behavior.

Monitoring is essential for the management of computer systems. It helps to identify a device's state, validate device behavior, and identify anomalies. If a device behaves differently than expected, the corresponding changes are different than expected in the monitoring, allowing operators to investigate the cause. One environment that heavily relies on device monitoring are DCs. In the following sections, we use the DC environment as a guiding example to use side-channel monitoring of composite signals. However, the proposed method can also be used in other environments.

DCs house servers at a large scale to provide computing power and data storage capabilities to customers [169]. DC monitoring encompasses the hosted devices and the environment in which the devices are located. Monitoring the DC environment is part of Data Center Infrastructure Management (DCIM) [24]. The availability of DCs heavily influences our daily lives since they provide data for various applications. This motivates meticulous monitoring, enabling operators to efficiently manage DCs and identify anomalies early. DCIM encompasses physical measurements such as energy consumption, temperature, or humidity [24]. Using these information channels allows to identify problems and minimize service outages. Some features can indicate problems earlier than others. For example, if the air condition fails and the temperature rises, devices will keep operating until they shut down due to overheating.

Some DC devices offer extended monitoring capabilities. On the one hand, those capabilities are measurements via onboard sensors or extended functionality via additional software like the Intelligent Platform Management Interface (IPMI) [6]. However, not all hardware has onboard sensors or supports these standards. In addition, IPMI-like services can also introduce vulnerabilities and thus increase the attack vector of DCs [5]. Therefore, services like IPMI may be disabled, limiting the monitoring capabilities of DC operators.

This limitation forces operators to rely even more on DCIM to infer the state of DCs. We investigate the suitability of audio-SCs to monitor the state and activity of DC devices. Audio-signal monitoring offers attractive benefits: the collection is fast, well-understood, accessible, affordable, and non-intrusive. It can provide a way to detect device states without introducing a new attack vector to hosted devices.

Extracting usable data about individual device activity in composite signals is challenging. The soundscape of a DC contains the audio of all devices in proximity, including switches, servers, or air conditioning, blended into a single information channel. In this work, we propose methods to extract valuable information from the typical soundscape of a DC. We then demonstrate how our insights can be applied to monitor specific events in the real world. We show that we can extract information about server behavior from the audio signal collected in a DC and detect device error codes in a composite audio stream.

7.2 Related Work

The private sector drives most progress in DCIM. DC monitoring typically focuses on power intake and climate control to ensure an optimal environment for hosted devices. Table 7.1 summarizes commonly used DCIM features and related work based on them.

Depending on the used features, DCIM may require expensive hardware, e. g., power meters or heat image cameras. Measurement hardware must be typically placed at critical locations, e. g., the power lines, to collect accurate information. However, adding such measurement hardware retrospectively to an existing setup can be challenging. Installing sensors like power meters requires a complete shutdown of devices, thus interrupting the operation. In contrast, the still unused audio channels can be measured non-intrusively, flexibly, and inexpensively through microphones. While previous works utilize various side channels for DCIM, none use audio to the best of our knowledge.

Levy et al. [24] present a holistic approach for DCIM via Internet of Things (IoT) sensors. They propose using IoT sensors to include physical properties in the monitoring. Proposed properties are vibration in cabinets, differential air pressure, and water pressure of cooling systems. Ahuja [76] analyzes the importance of airflow management in DCs to optimize cooling capabilities. Rodriguez [75] uses a wireless sensor network to support its DCIM. In addition to temperature, deployed sensors monitor humidity to optimize cooling capabilities. The authors do not aim to monitor individual device activity but collect information about the shared infrastructure.

Table 7.1: Physical features of DCIM used by related work

Feature	RW
Power consumption	[24], [77], [78]
Heat	[24], [170]–[172]
Airflow	[24], [76]
Humidity	[24], [75]
Vibration, water/air pressure	[24]
Audio	None

Many approaches monitor power consumption at various levels for Anomaly Detection (AD). Dayarathna et al. [77] investigate methods to model the energy consumption of DCs. Their survey presents techniques to model power consumption on different levels of a DC. To detect anomalies, Borghesi [78] analyzes the power consumption of single nodes in HPC environments. The presented prototype achieves an anomaly detection accuracy of 95 %. Our work differs from these approaches, defining typical behavior by monitoring explicit device activity.

Marwah et al. [170] present an approach to predict equipment failure based on heat emission of DC devices. Similarly, Lee et al. [171], [172] use thermal measurements to detect anomalies in DCs. While these works highlight that SCs help identify anomalies, they do not aim at characterizing environmental changes due to device activity.

To our knowledge, no related work aims to identify individual device activity in composite signals for DCIM. In addition, to the best of our knowledge, acoustic channels are still an untapped source of information for DCIM (Table 7.1).

7.3 Data Center Infrastructure Management

This section gives an overview of devices used throughout DCs and methods to monitor them. As Chapter 2 states, DCIM combines sophisticated monitoring and management tools. Those allow to monitor large-scale Information Technology (IT) equipment hosting. These tools aim to capture the state of devices and enable operators to make changes when necessary. Each device has a physical state and a closely coupled cyber state. For example, a server performing complex computations will increase the clock of its CPUs. High computational load affects a device's power intake and thermal emission, influencing cooling. Since many devices operate in the same environment, signals from this environment are composite, as they are affected by all devices. For example, temperature is an aggregated value due to the heat that the surrounding devices emit. In the following sections, we describe the features of both state types and the means to monitor them.

Data Center Devices: In addition to IT equipment like servers, routers, and switches, DCs host additional hardware for their operation. Power supply units, power distribution units, switchgear, electrical panels, and generators ensure constant power provisioning for hosted devices. To manage the climate, DCs use complex cooling systems consisting of chillers, cooling

towers, and air conditioning units. Facility management DCs often use automation systems for managing their building infrastructure. To address emergencies, DCs have fire systems [24] or redundant power supply systems in place.

This work focuses on the *cyber-physical layer* of our logical structure (Section 2.2.1). We further partition the cyber-physical layer into two layers. The first layer is the *device layer* consisting of IT equipment for computational and data storage capabilities. The second layer is the *environmental layer* for managing the environment for the device layer. Operators constantly monitor and manage both layers to ensure an efficient and optimal operation of DCs.

Device Monitoring and Management: To manage IT equipment, operators use tools to monitor and change the state of DC devices. While most tools rely on data collected within a host Operating System (OS), there exist additional tools that work without a running OS. There are various tools to monitor the state of machines on the cyber level. Tools running on top of an OS can monitor information on a device, e. g., CPU load, memory consumption, etc. Example for information collection requiring a running OS, tools like `checkmk` [44] or Nagios [173]. Operators also collect information about data flow between hosted nodes. Tools like Tivoli Netcool/OMNIbus [174] or Grafana [175] accumulate the collected data. Collecting the distributed information at a central node allows to present a holistic overview of a DC's state [176].

Operators must be able to manage devices without physical interaction, even when shut down or have no running OS. Tools like IPMI [6] or iDRAC [177] provide API-like access over the network. We investigate the relationship between remote control of devices via such tools and device activity in Chapters 8 and 9.

Environment Monitoring and Management: Environment monitoring and management address the physical properties of DCIM. These include power supply and climate control. Distributed sensors measure the environment throughout DCs. For power management, power meters measure the power intake. Hardware vendors also provide tools for remote management of power supply. A DC's climate control ensures optimal room conditions for hosted hardware. To prevent devices from overheating, IT equipment has onboard cooling like fans. There are additional devices for climate control to handle the heat generated by the many devices in DCs. These are additional air conditioning units, cooling towers, and airflow management units. Via distributed sensors located on devices and throughout infrastructure, DCIM collects information about temperature and humidity. It then uses this information to adjust and optimize the climate. Further, it is possible to extend DCIM by other sensor measurements to collect information about the DC environment, e. g., the vibration of a cabinet [24].

DCIM combines tools to collect information at both the device and environmental level to present a holistic view of a DC's state.

7.4 The DC Soundscape

We aim to enable reasoning about individual physical device activity in composite DCs signals. While we introduce audio as new SC for monitoring the DC, the presented approach is also applicable to other SCs.

This section introduces the properties of the DC soundscape. Since DCs house many devices in the same environment, the soundscape is a composite audio signal. Figure 7.1 shows a spectrogram from an audio sample in a DC. DC devices emit noise due to physical onboard devices. Servers have spinning fans that emit noise at characteristic frequencies. Since devices often operate continuously, fans mostly spin at the same speed, causing constant sound. The bright horizontal lines in the spectrogram of Figure 7.1 show this constant behavior. Dynamic changes in the soundscape stand out due to the constant operation. Such changes occur when devices perform an activity, e. g., when servers change the speed of their onboard cooling fans, e. g., when a server boots. The change in the speed of fans creates a trace in the soundscape. This trace creates unique patterns in spectrograms [178] and differs from the characteristic frequencies of the constantly cooling fans, making this activity stand out. The spectrogram in Figure 7.1 shows the pattern of a booting server at the 6 s mark, highlighted in the blue boxes. The trace ends in the characteristic lines since the fans speed up to regular operation. Before accelerating the fans, a buzzer on the server emits a short beep lasting around 0.5 s highlighted in the blue circle. Other sounds emitted by devices might be beep codes to provide information about a device's state, such as errors. Such error codes are currently not perceivable by existing methods that do not use audio as a source of information.

While cooling fans from servers, routers, and switches make up most of the soundscape, other devices such as Air Condition (AC) also emit sound. Similar to servers, these mostly emit constant noise. AC devices emit less noise than the many housed computing devices. Therefore, AC does not impact the soundscape meaningfully. Similarly, since DC are mostly remotely managed [5], human activity rarely affects the DC soundscape.

To conclude, the DC soundscape is a composite audio signal containing sound from multiple nearby devices. Due to constantly spinning fans, most of the DC devices emit noise at characteristic frequencies. Chapter 2 points out that physical human interaction in DCs is rare. With low human interaction and supporting infrastructure, such as AC, not heavily impacting the soundscape, DC devices make up the majority of the soundscape. Therefore, device activity that deviates from the characteristic frequencies creates unique patterns in the spectrogram and stands out in the audio signal. This makes the DC soundscape suitable for device monitoring individual device activity in a composite signal.

7.5 Acoustic DCIM

For the remainder of this thesis, we define physical device activity by the changes it induces in the environment. In case of analyzing a DC soundscape, device activity is defined by the changes induced to this soundscape.

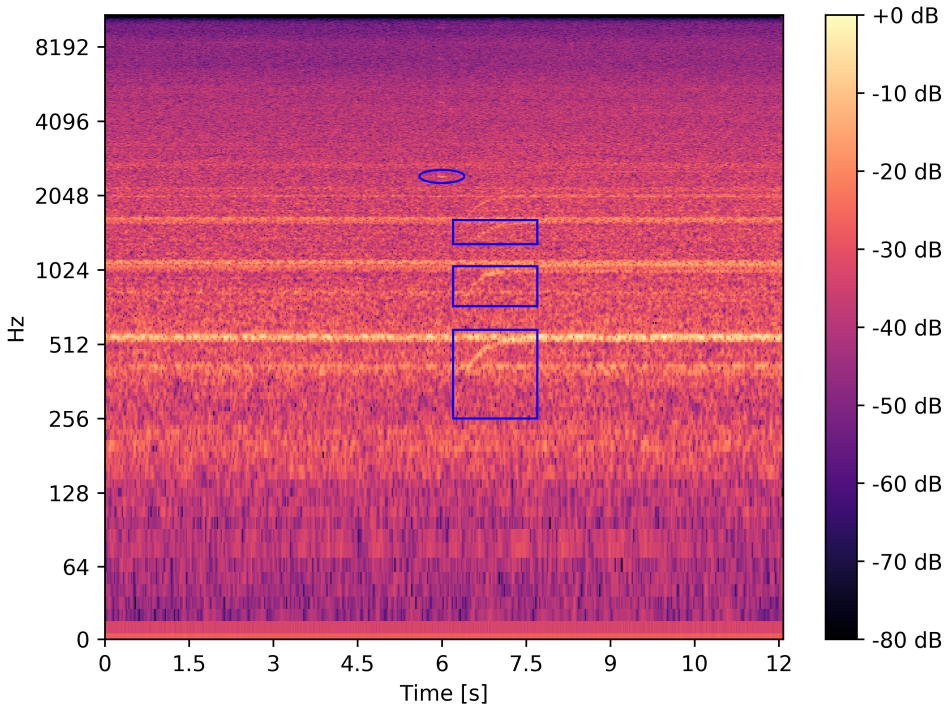

Figure 7.1: Log power spectrogram of a recording in our DC. The ellipsis marks a beep that is emitted by a booting server. The rectangles mark the fans of a server spinning up during its startup.

To effectively use acoustic channels for DCIM, our method needs to extract and identify individual device behavior from a mixed signal. Due to the large amount of background noise in the DC soundscape, single devices do not stick out in an analysis of a signal's amplitude and loudness. We, therefore, rely on spectral analysis of the soundscape to identify device activity.

Our method uses two steps to identify individual device activity. We manually perform spectral analysis in the first step (Section 7.5.1). This allows identifying relevant frequency bands and limits the remaining analysis to relevant frequency ranges in which activity influences the soundscape. In the second step (Section 7.5.2), our method automatically extracts the time frame during which an activity takes place.

7.5.1 Spectral Analysis

While most physical operations emit noise, sounds do not affect all frequencies of a spectrum. For example, CPUs emit noise at high frequencies when doing their computations [179]. Buzzers or fans emit sound on other frequencies (Figure 7.1).

To identify relevant frequency bands, we perform spectral analysis. A power spectrogram shows the amount of energy a frequency carries in a time frame. Figure 7.1 shows the full spectrogram of a recording from our DC. A visual inspection of the figure shows distinct frequencies constantly carrying high energy (brighter areas) while others carry low energy (darker areas). Spinning fans emit noise, creating high-energy frequencies. Analyzing a spectrogram over a time frame enables the identification of energy changes on specific frequencies. The recording in Figure 7.1 contains a server's startup. At around 6 s, a buzzer emits a short high-pitched beep at 2500 Hz. The traces starting 6.5 s show the acceleration of the server's fans to their normal operating speed until 7.2 s. Since multiple identical servers run in the background, the traces end in significant frequencies. The ellipsis in Figure 7.1 marks the beep in the spectrogram; the squares mark the accelerating fans.

Device activity can cause two types of patterns in a spectrogram: (1) single-frequency events and (2) multi-frequency events. In the first case, activity changes the soundscape for a certain period on a single frequency, e. g., due to beep codes. The second case induces changes to the spectrogram on multiple frequencies. In most cases, activity does not affect the whole frequency band, but only a subset [180], [181] Since those patterns indicate the changes in the spectrogram, we use them to define and identify device activity in an audio SC.

Since the patterns only occur on a specific frequency band, we can limit the analysis to this frequency band in the next step. Thus, we manually apply a band-pass filter after identifying the relevant frequency bands on the spectrum where activity can occur. Examples of relevant frequency bands are the frequencies of onboard buzzers for error codes or in which fan speeds operate. This allows to filter out non-relevant frequencies for the next step and to reduce the effects of background noise. By reducing noise, actual device activity stands out more.

7.5.2 Time Frame Extraction

After identifying the relevant frequency band, the second step identifies the activity time frame in a composite signal. Using our previously introduced definition, this is the time frame in which a device induces changes to the soundscape.

Spectral Flatness (SF) [182] indicates how tone-alike a signal is by value in $[0, 1]$. The closer SF is to 1, the more tone-alike the signal is. Due to the high background noise, the DC soundscape has a low SF. The SF throughout the recording in Figure 7.1 does not exceed 0.0005. Using SF is only possible in otherwise silent environments where the monitored activity starts and ends in silence, which is not the case for DCs and other shared environments. As a result, it is impossible to use SF to determine the beginning or end of an activity.

Due to the constant background noise, we can detect changes in audio signals by calculating the Root Mean Square (RMS) Energy. The RMS Energy can be calculated by processing the result of a Short Term Fourier Transform (STFT) from a signal, which allows the calculation of the energy of individual frequencies. The RMS Energy aggregates the energy of each STFT slice [183].

$$RMSE_x = \sqrt{\frac{1}{N} * \sum_n |x(n)|^2} \tag{7.1}$$

The result is the total energy a sample of an audio signal carries across all frequencies, indicating how loud a signal is. In an otherwise constant signal, device behavior causes changes in the RMS Energy as it adds or removes audio from the overall signal. In particular, operations such as buzzers and additional audio sources affect the RMS Energy. The middle plot in Figure 7.2 shows the RMS Energy of a sample recording in which a server turns on. While the signal fluctuates due to noise, the average RMS Energy is constant in periods with no device activity. After the server is turned on, the overall level of RMS Energy remains higher than before when fewer devices are active.

We smooth the curve to identify trends in the noisy RMS Energy over time. We perform the smoothing via a moving average. We specify the moving average with a window-size w. Thus, we calculate the smoothed RMS Energy as

$$RMSE_{i_{smoothed}} = \frac{\sum_{n=i-w/2}^{i+w/2} RMSE_n}{w} \tag{7.2}$$

The choice of w introduces a tradeoff between noise reduction and sensitivity to short-lived events. A large w can remove short-lived spikes or gaps caused by short-lived events, making it impossible to detect them. A small w may not mitigate the effects of fluctuations in the RMS Energy due to noise, resulting in false activity detection. Therefore, selecting the correct window size is a tradeoff between sensitivity and noise reduction. Another effect to consider when choosing w is the size of the frequency band. A smaller frequency band covers less frequencies on which noise can occur. Thus, smaller frequency bands require less smoothing to mitigate

the effects of noise than large frequency bands. Therefore, the size of window w also depends on the frequency band size determined in the first step. Thus, the choice for the size of the moving average window w depends on the type of the searched event and the frequency band size covered by the RMS Energy calculation.

The final step identifies the start and end of activities from the smoothed RMS Energy curve. Our approach automatically determines time stamps of the start and end of an activity through a threshold-based method. Therefore, we analyze how the RMS Energy changes over time by its difference d in a time window $w_{\{s,e\}}$.

$$d = |RMS_i - RMS_{i+w_{\{s,e\}}}| \tag{7.3}$$

Windows for identifying the start of activity are denoted by w_s, while we denote those for identifying the end of activity with w_e. If the RMS Energy changes more than a defined threshold t_s of energy in a given time window w_s, we mark the beginning of an event. If $d > t_s$, we assume an activity started; otherwise, we assume noise-induced changes in the RMS Energy. Similarly, we assume an activity has ended if the change in RMS Energy d remains below a threshold t_e in a window w_e. We use $d \leq t_e$ to determine the end of an activity.

Similar to the previous choice of w, choosing an appropriate d depends on the detection and use case requirements. A low d can increase the method's sensitivity but can cause the false attribution of noise to device activity. On the other hand, a high d can lead to failing to detect activity in the first place while reducing the number of False Positive (FP) for detecting activity. Therefore, the expected fluctuation of the smoothed RMS Energy due to noise needs consideration when choosing an appropriate d. It further limits the activity that can be detected. Changes due to device activity need to exceed d.

While threshold-based methods are easy to implement, they have several drawbacks. Thresholds and window sizes must be chosen on a use case basis, requiring manual tuning. Similar to the choice of the moving average window, the time frame in which changes have to occur also affects the accuracy of activity detection. Despite the drawbacks, our method can reliably extract the beginning and end of actions of single devices in our DC.

Identifying a device's activity's time frame can then help identify relevant subsequences from measurements for further processing. For example, image recognition on spectrograms can help to identify the type of an event [178]. In Chapter 9, we integrate the proposed method into an Anomaly Detection System (ADS) to validate device activity.

7.6 Experiments

We conducted several measurements in a DC environment to evaluate the feasibility of our approach. Initially, we demonstrate the soundscape during regular operation. Afterward, we apply our methods to failure detection. The air-conditioned DC consists of multiple servers, managed switches, and routers. We conducted our measurements during regular operation with

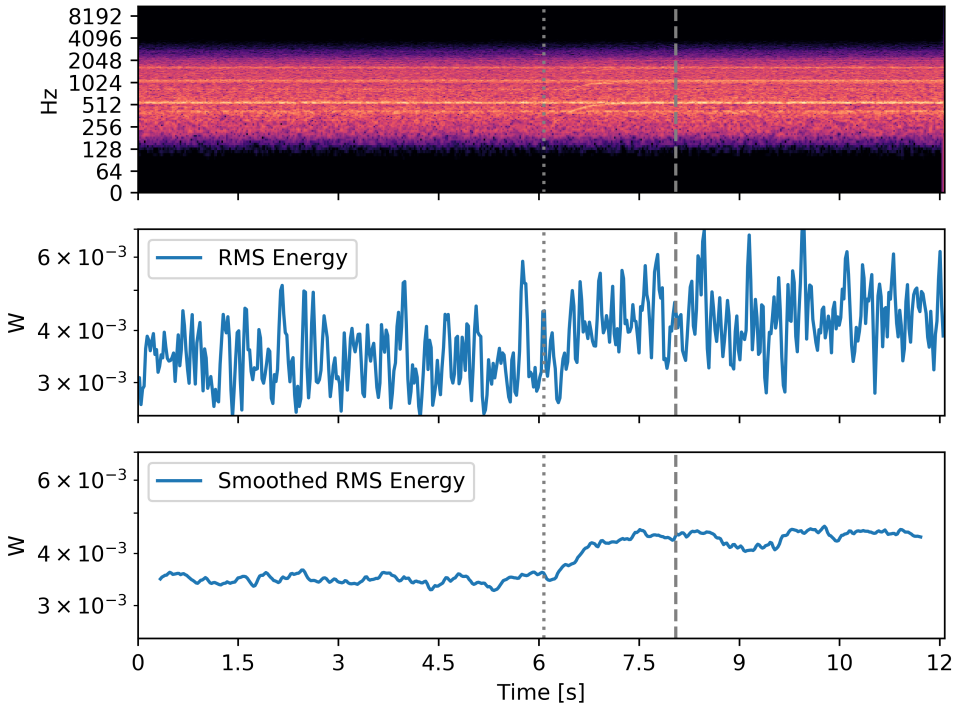

Figure 7.2: A server spinning up its fans during boot. The top part shows the log power spectrogram of the filtered signal. The middle plot shows the RMS energy for the spectrogram. The bottom plot shows the smoothed RMS energy. The dotted and dashed lines mark the start and end of the identified event.

normal background noise. For our measurements, we set up a Blue Yeti X [184] microphone and recorded the audio signal at approximately a distance of 1.5 m.

Normal operation: In our first experiment, we instruct individual devices to execute commands that physically alter their behavior. These are power-off, power-on, or reboot activities since they cause the devices to speed up or slow down their cooling fans. For each operation, we created an audio recording. To analyze recordings, we implemented the proposed approach with the librosa python library [185].

Figure 7.1 shows the analysis of our first experiment in which a server boots. The recording has a length of 12 s. During the startup, the server emits a short beep and then accelerates its fans until they reach the intended spinning speed. As a result, the recording contains a short, single-frequency event and a longer multi-frequency event. A server performs a startup at the 6 s mark. After booting, the server remains turned on and is available to users.

We first aim to identify the time frame during which the server accelerates its fans and preceding beep. The accelerating fans cause energy traces in multiple frequency bands. Those are between 256 Hz and 512 Hz, 512 Hz and 1024 Hz, and 1300 Hz and 1600 Hz. The beep occurs at 2450 Hz. Since multiple devices of the same type are running in the background, the noise traces merge with significant frequencies in the background noise.

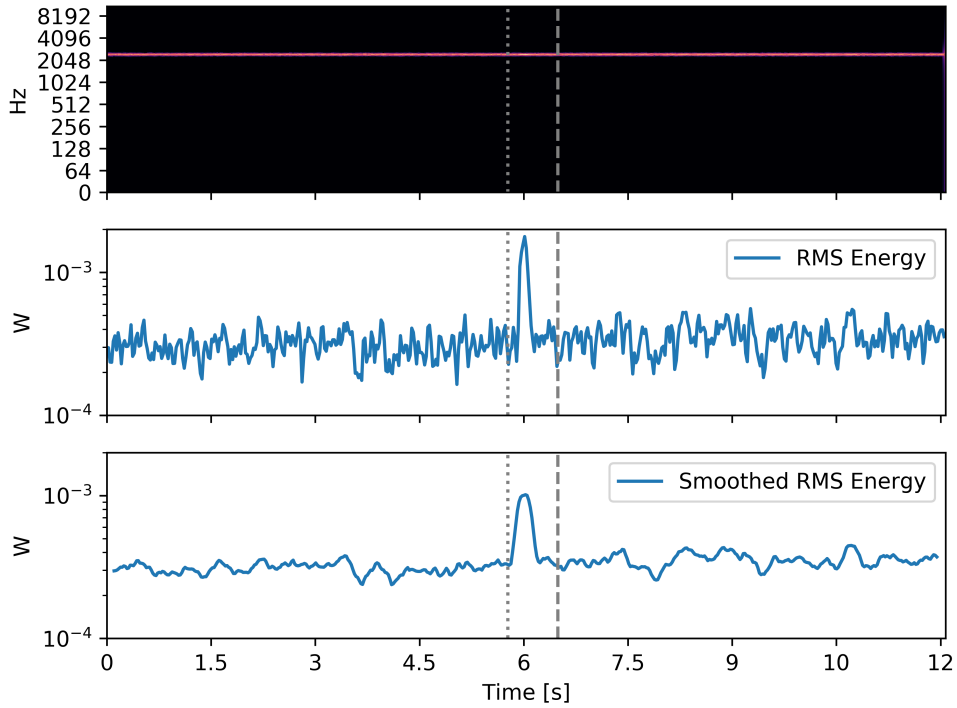

Figure 7.3: Detection of a short single-frequency event in our first experiment

To detect the accelerating fans, we limit the frequency band to 256 Hz–1700 Hz using the first step of our approach. The upper part of Figure 7.2 shows the limited frequency band used for calculating the RMS Energy. The middle part of the figure shows the calculated RMS Energy of the filtered signal and the smoothed RMS Energy in the lower part. Since we calculate the RMS Energy for a wide frequency band, the moving average for the smoothing considers 30 neighboring samples. The RMS Energy remains constant until it increases during the server's startup. The RMS Energy stays constant after the fans reach operational speed. Using the second step of our method, we automatically identify the period of the server's physical startup, around 1.7 s. The dotted and dashed lines in Figure 7.2 show the detected duration of the startup.

Our experiment implicates that the number of running devices affects the total RMS Energy. However, we have not yet investigated if it is possible to infer the number of running devices from the RMS Energy. To identify short single-frequency events, we limit the frequency band to the frequency of beep 2400 Hz–2500 Hz. Since we consider fewer frequencies for single-frequency event detection, the calculated RMS Energy is less noisy. Thus, we reduce the smoothing in the second step. Figure 7.3 visualizes this part of the experiment. We can detect this type of activity by making minor adjustments to the thresholds t_s and t_e and window sizes w_s and w_e. The dashed lines show that our method correctly identifies the beep.

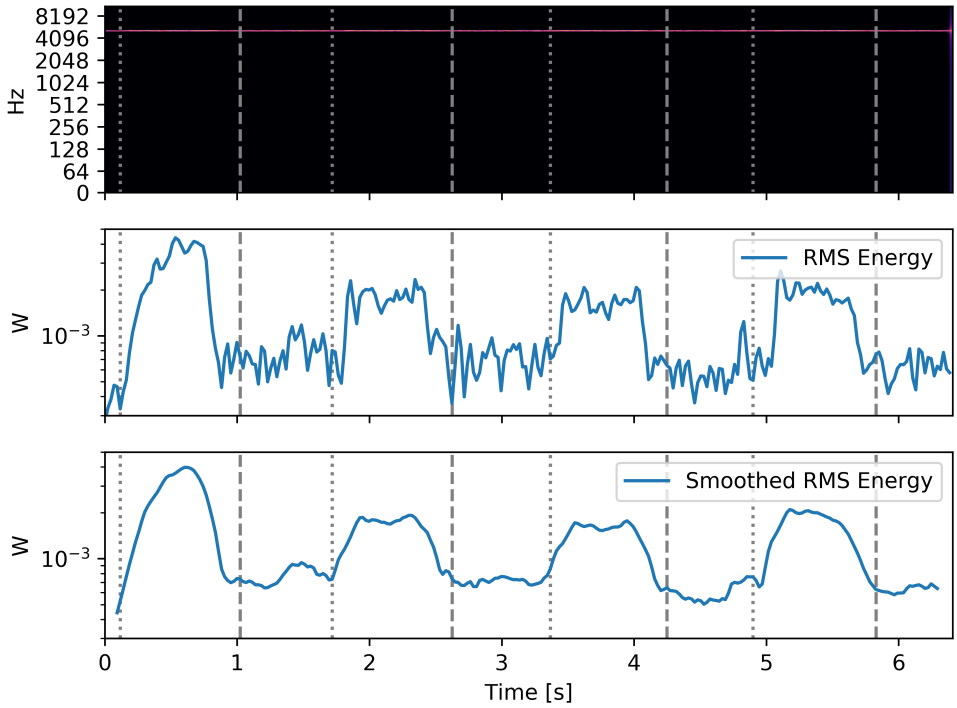

Figure 7.4: Error code emitted by a server's buzzer due to a faulty fan

Failure detection: After demonstrating acoustic channel analysis during regular operation, we apply our method to identify erroneous server states. Some boards indicate erroneous states by a beep code. We unplug a server's fan such that the server detects it as faulty. As a result, the manipulated server emits a series of beeps to indicate the issue. Each beep has a length of roughly 1 s at 5150 Hz. Since the server uses different buzzers for those beeps they occur on a different frequency than the short beep that happens during the server's startup in the previous experiment.

We reduce the smoothing similar to the first experiment to detect such short-time events. Since the event occurs at 5150 Hz, we limit the frequency band to 5100 Hz–5200 Hz. Figure 7.4 shows the results of the second experiment. The lower plots show that the RMS Energy spikes during the server's buzzer activity. Our method automatically detects each beep (dotted and dashed lines in Figure 7.4). This shows that we could reliably extract the error code from the composite audio signal.

Our evaluation shows that acoustic side channels are applicable to detect the activity of individual devices in the DC soundscape for DCIM despite a noisy background. Our approach allows to identify the time frames in which activity takes place.

7.7 Discussion

We conducted two experiments addressing two important use cases to show the potential and relevancy of our approach for DCIM. A server's fan activity can give insights into its state and indicate potential issues. The early detection of a failing fan can help to prevent further hardware damage, e. g., by overheating. The detection of fan activity requires monitoring a wide frequency band, which is potentially noise-prone. However, our experiments show that we can successfully identify such activity from a noisy signal without OS-level information or external tools like IPMI. Our second experiment revolves around the detection of beep codes emitted by buzzers. Devices can emit beeps when booting or use beeps to indicate the presence and type of errors, e. g., faulty RAM or failing fans. Since buzzers emit noise at distinct frequencies, we can limit the monitored spectrum to these frequencies to detect activity. Reducing the analyzed frequency band also reduces the background noise, making it easier to detect relevant events. Our experiments show that we can identify error codes purely by analyzing audio without additional tools on devices.

7.8 Conclusion

Composite signals are affected by multiple devices at once, making it difficult to identify individual device activity. However, depending on the sources that affect the signal, the presented approach shows that extracting information about individual device activity is possible. If surrounding devices emit constant noise, it is possible to attribute all changes in the mixed signal to individual active devices.

We present a method that can extract device behavior from mixed audio signals. Despite a challenging DC environment, in which a multitude of devices create a blended information channel and noisy signal—we can successfully extract the behavior of individual devices.

We apply our method to a scenario: DCIM. Using audio for DCIM has several advantages: it is inexpensive and easy to set up thanks to widespread usage in consumer hardware. Audio can be recorded with high granularity and accuracy, and its measurement is non-intrusive. Our measurements have shown that we can identify the sounds emitted by a server during regular operation, such as fan noise or beeps. More importantly, we applied our proposed methodology to detect failing equipment in a DC. Our automatic detection mechanism identified a server's beep error codes without additional tools. These experiments demonstrate the feasibility of acoustic channel DCIM.

7.9 Context to Multi-Layer Relationships

The work presented in this chapter focuses on events on the cyber-physical layer of the logical architecture (Section 2.2.1). As pointed out in Chapter 2, physical device activity on the cyber-physical layer can cause changes to the environment. We interpret changes caused by device activity as events on the cyber-physical layer. In contrast to events on the network layer, such as network flows or packets, events on the cyber-physical layer are not as well defined. In

particular, it can be challenging to identify events in shared spaces with multiple simultaneously operating devices, which can cause concurrent events. However, depending on the activity, the events on the cyber-physical layer are unique. This chapter introduces a method to characterize and identify cyber-physical layer events.

Our work demonstrates that, depending on the properties of the environment, it is feasible to identify physical device activity as events in composite signals. Identifying events on the cyber-physical layer is essential to be able to correlate them with events on other layers. With the ability to identify events in composite signals, this work extends our approach's application area to non-isolated devices. In Chapter 8 we propose a model that correlates cyber-physical and network layer events.

7.10 Key Results

This chapter presents a method to identify individual device activity in composite signals. Physical measurements of the environment can provide information about device activity. However, physical measurements collect composite signals in scenarios in which multiple devices operate in proximity, such as DCs. We investigate under which circumstances it is possible to reason about individual device behavior in such signals. We identified that the DC soundscape exhibits the necessary characteristics to make this approach possible. While multiple devices operate in proximity, the DC soundscape has static noise. This makes it possible to identify dynamic changes in the DC soundscape that implicate physical device activity. Therefore, to show its potential, we apply our method to audio as SC in a DC environment. The method can identify the time frame of activity and server error codes without on-device tools.

Our work provides an answer to RQ3.1, *How can we characterize and identify physical device activity in composite signals?* We can define the physical device activity by the changes it induces to a signal, given that it is the main cause of the change. Analyzing general trends in a composite signal makes it possible to identify time frames in which activity takes place. If the activity is unique, it is possible to reason about device states, e. g. by analyzing error codes emitted by onboard buzzers.

In case the primary purpose of a device is to actively influence the environment, such as actuators in Industrial Control System (ICS) or the IoT, devices cause measurable changes. However, also devices that do not primarily aim at influencing the environment can cause such changes due to physical activity. We have shown that it is possible to identify time frames in the composite audio signals in which physical activity of individual devices takes place. We integrate this approach into an ADS in Chapter 9.

Chapter 8

Relating Network-Traffic and Physical Device Behavior

Author's contribution: *Section 8.1 is based on L. Wüstrich, L. Schröder, and M. Pahl, "Cyber-Physical Anomaly Detection for ICS", in* 17th IFIP/IEEE International Symposium on Integrated Network Management, IM 2021, Bordeaux, France, May 17-21, 2021*, T. Ahmed, O. Festor, Y. Ghamri-Doudane, J. Kang, A. E. S. Filho, A. Lahmadi, and E. R. M. Madeira, Eds., IEEE, 2021, pp. 950–955. The author conceived the research question and led the analysis and theoretical model of the publication. Section 8.3 contains content from two publications, L. Wüstrich, S. Gallenmüller, S. M. Günther, G. Carle, and M. Pahl, "Shells Bells: Cyber-Physical Anomaly Detection in Data Centers", in* NOMS 2024 IEEE Network Operations and Management Symposium, Seoul, Republic of Korea, May 6-10, 2024*, IEEE, 2024, pp. 1–10.* DOI: 10.1109/NOMS59830.2024.10575124 *and L. Wüstrich, L. Schröder, and M. Pahl, "Cyber-Physical Anomaly Detection for ICS", in* 17th IFIP/IEEE International Symposium on Integrated Network Management, IM 2021, Bordeaux, France, May 17-21, 2021*, T. Ahmed, O. Festor, Y. Ghamri-Doudane, J. Kang, A. E. S. Filho, A. Lahmadi, and E. R. M. Madeira, Eds., IEEE, 2021, pp. 950–955. The author conceived the main construct and formalization of the model in those papers. In addition to the publications, this chapter introduces an analysis of the Intelligent Platform Management Interface (IPMI) protocol and an evaluation of the accuracy of the model's prediction.*

This chapter investigates the multi-layer relationship between our logical structure's network and cyber-physical layer. This relationship links control traffic and physical device activity that causes environmental changes. As discussed in Chapter 2, our considered environments–Industrial Control Systems (ICSs), Data Centers (DCs), and the Internet of Things (IoT)–typically rely on remote control. Devices receive instructions over a network and execute corresponding physical device activity, thus influencing the environment. Chapter 7 shows that

we can detect changes in the environment caused by physical device activity without additional information from the network. However, if multiple devices operate in the same space, it is challenging to attribute events on the cyber-physical to specific devices.

This chapter addresses this challenge by introducing a model that captures the relationship between the network and cyber-physical layers. Therefore, we investigate Research Question (RQ)3.2, *What is the relation between network control traffic and physical device activity?* For this purpose, we analyze the structure of remote control in the considered environments. We then present an approach to correlate events from a network, i. e. control traffic, with physical changes due to device activity.

8.1 Motivation

Sensors and actuators in Cyber-Physical Systems (CPSs) constantly communicate via a network (Chapter 2). Examples are power grids, chemical, water treatment, or nuclear plants [26]. Similarly, DC devices have a similar remote management, allowing operators to instruct them to perform actions from a distance. In both scenarios, operators extensively monitor system components and send instructions via the network. Since instructions can to physical activity, e. g. by actuators, control traffic dictates the future behavior of devices in the system.

Modeling the relationship between network activity and physical changes has various advantages. In Anomaly Detection (AD), it allows new types of anomalies in which device behavior does not conform with the communication on the network. If a device performs different actions than reported, it creates a mismatch between its physical state and perceived cyber-state.

Cyber-physical models can support system operators to understand system behavior. In this case, the model captures the relationship between control traffic and expected physical changes. The model allows operators to understand when and how device activity changes a system's environment.

Fusing information from multiple independent sources into a single model is challenging [8]. It requires to set variables into context and to identify causalities and correlations. Physical measurements are often noisy and can contain information from multiple devices (Chapter 7). Thus, they need to be pre-processed to reduce noise before they can be processed. Combining features helps to create a context of the state of an observed system and gives operators enhanced reasoning capabilities. For example, while two features may be individually in valid feature ranges, their combination may be abnormal [81].

Most models, particularly for Anomaly Detection System (ADS), rely only on cyber or real-world features [56]. However, using a combination of cyber and physical features for AD has advantages [8], [186]. For AD, depending on the cause of an anomaly, physical features can indicate abnormal behavior earlier than cyber features or vice versa. For example, a malfunctioning device may acknowledge an operation even though it failed to perform the instructed action. Such behavior can not be detected by models that rely only on cyber features. Other physical issues of devices, such as abrasion, also cannot be detected by cyber-only ADS. An

attack that exploited this weakness is Stuxnet [10]. Stuxnet altered the device input and output of centrifuges in attacked systems. Centrifuges would spin at a suboptimal speed while reporting a different spinning speed to the remote monitoring. Since the monitoring relied purely on cyber-only features, it did not detect the tampering and displayed a false state. The monitoring would have benefited from additional physical monitoring to identify the abnormal behavior not reported by the centrifuges. Similarly, DC devices may report a wrong state to the remote monitoring.

Vice versa, an ADS, which relies exclusively on physical measurements, cannot detect a loss of control immediately. For example, the method presented in the previous chapter can identify physical activity. However, it fails to identify if a device becomes unresponsive. This shows that combining information from multiple independent sources has essential benefits for improving AD [8].

Physical actions typically take time and have a continuous transition from one state into another. In the previous chapter, on a server startup, fans accelerate until they reach normal speed. In the cyber-state, the device switched from an "off" state to an "on" state. Models need to reflect this transition and physical change over a period of time. In the application domain of AD, procedures over a time window make it challenging to enable real-time AD since collecting data about the activity is necessary.

In this chapter, we investigate the relationship between control traffic and changes in the physical environment. We also analyze how consistent the relationship is. Based on our insights, we propose a model that combines features from physical measurements and network traffic. The next chapter uses this model to implement AD for composite signals.

8.2 Related Work

Related work fields for creating cyber-physical models are control systems engineering and AD. Choi, Lee, Aafer, *et al.* [187] build a cyber-physical model of a robot vehicle. They compare predictions of a robotic vehicle's control algorithm and real physical measurements to detect jamming attacks. In our work, we do not have access to the control algorithm on devices. Instead, we only observe the control messages between the controller and devices. Further, we aim to model the behavior of actuators in the system instead of external influences on the system. Birnbach, Eberz, and Martinovic [38] combine measurements from multiple sensors and events reported via the network in IoT environments. The work shows that physical events simultaneously influence different Side Channels (SCs). Evaluating if independent sensors measuring different physical signals change as expected in line with a reported event allows the detection of spoofed events. Our work follows a similar approach. However, contrary to validating reported events retrospectively, we model future changes based on instructed activity. Sahu, Mao, Wlazlo, *et al.* [8] fuse measurements from multiple sensors and the cyber-layer to model the behavior of power systems. The findings emphasize that cyber and physical activity are closely intertwined. Our approach aims to model the effects of individual and concurrent device activity instead of the behavior of the complete system. Stock and Schel [186] propose

a hybrid fingerprinting scheme for cyber-physical production systems. The authors propose to combine physical and cyber-information to build a fingerprint of CPS. Our work exceeds this approach by investigating how network traffic affects the physical environment due to induced device activity.

8.3 Modeled Scenario

We want to model the changes in the environment that occur due to remote-controlled device activity. Our scenario allows multiple devices to operate simultaneously. Concurrent activity leads to a total change, which our model predicts.

We define the activity of a device as an action α. Actions are physical operations that can influence the environment. We define those actions by the physical effects they induce. We consider physical changes, in which the effects of actions add up. Examples are the sound or power intake of devices. This makes predicting the total change due to concurrent device activity easier to model.

Actions can occur in a variety of systems. In our model, we consider actuators in IoT systems, ICSs, as well as the operation of servers in DC environments. The previous chapter introduced examples of possible actions of DC devices, power-off and power-on operations or error beep codes that influence the soundscape.

Devices receive their instructions over the network in our model. The specialization of devices often limits the possible action set. Valves can only open or close, while servers can turn off, on, or adapt cooling fan speeds. This limitation reduces the possible variations of the expected environmental changes.

The protocols for managing devices often have a request-response design, particularly in industrial networks [1]. Management devices send commands to the controlled devices, processing the command and returning a reply. The reply typically indicates the execution status, the device state, or an acknowledgment that the command was received. Two examples are MODBUS [30], and IPMI [6].

Our model uses the networked control to correlate events from the network with actions. Thus, it uses the control traffic to predict future environmental changes. We assume that devices execute instructed activity deterministically. In the following section, we analyze the characteristics of control traffic and how they determine the actions that are executed.

8.4 Characteristics of Networked Control

A key building block of our model is control traffic. Industrial environments typically rely on specialized protocols to monitor and manage devices [26]. ICSs, like power generation networks or water treatment plants, use IEC-61850, DNP3, MODBUS, or the Siemens S7 protocol [35], [188]. IoT networks use protocols like MQTT. DC operators use IPMI [6] or iDARC [177] to

remote control devices, even when they are turned off. While those protocols follow different implementations, they have similar design principles.

Protocols for networked control typically have a primary-secondary structure [1]. The primary is a central control unit that monitors the slave via polling [34]. The messages typically request information about specific registers [35]. The secondary then responds with the requested information.

Central control units can also instruct secondaries to execute actions. In this case, the central control unit sends control messages to managed nodes, uniquely specifying the actions. For example, in MODBUS, the primary writes new values to Programmable Logic Controller (PLC) registers, which in turn causes actuators to change their state. In IPMI, the management device has predefined packet structures and command values [6] that specify the intended instructions. Slaves then reply with a response code that indicates the outcome of the instructed activity [6], [49].

In the following, we analyze IPMI in more detail, as we use it in all applications in the remaining chapters.

The IPMI Protocol

The Intelligent Platform Management Interface (IPMI) [6] is used to manage server hardware remotely [189]. It is a User Datagram Protocol (UDP)-based request-response protocol for requesting information from devices or making them execute commands. Its second version, IPMIv2.0, extends the Remote Management Control Protocol (RMCP). In each IPMI session, two entities communicate. The communicating parties are a management device and a server. The management device acts as primary, the server as secondary device. Both devices must share a *Pre-Shared Key (PSK)* for a successful handshake. Each IPMI session consists of two phases. During the first phase, the communication partners perform a handshake to establish session secrets. The communication partners use these secrets to encrypt, and integrity protect following traffic. In the second phase, the parties exchange data, i. e. IPMI commands and responses.

IPMI Handshake

The IPMI handshake (Figure 8.1) has the goal of negotiating cipher suites and the establishment of shared secrets. The communicating parties establish these by using values that are exchanged in handshake messages and a PSK.

The IPMI handshake consists of six messages. The first two messages contain an *RMCP+ Open Session Request* to the client and a corresponding *RMCP+ Open Session Response*. The first RMCP+ message contains a *remote console session ID*, and lists supported authentication, integrity protection, and encryption algorithms. The second RMCP+ message contains a choice of cipher suites and a *managed system session ID*.

The remaining four messages follow the Remote Authenticated Key-Exchange Protocol (RAKP). This exchange aims to establish key material for the chosen cipher suites. *RAKP Message 1*

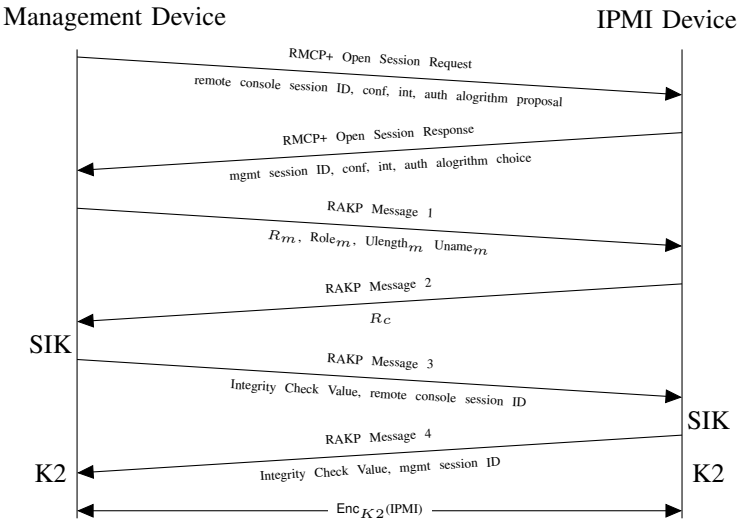

Figure 8.1: IPMI handshake and information in each message, cf. [6], [189]

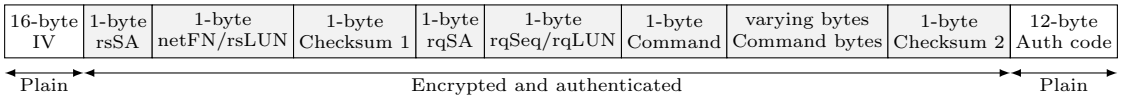

Figure 8.2: Structure of an IPMI request packet

contains a random R_m, a requested privilege level $Role_m$, length of username $Ulength_m$ and username $Uname_m$ itself. The answer is *RAKP Message 2*, containing a client random R_c and parameters to authenticate the handshake. Both parties generate a *Session Integrity Key (SIK)* from this information. The management uses the SIK to calculate an integrity check value. The integrity check covers the exchanged values and session id from the RMCP+ session request. It sends an integrity check value and the remote console session ID in *RAKP Message 3*. The IPMI device also performs an integrity check. It replies with a corresponding integrity check value (using the same SIK) and the managed system session ID in *RAKP Message 4*. Both parties use the same algorithm chosen in the RMCP+ session response to calculate the SIK. Using PSK KG, $SIK = HMAC_{KG}(R_m|R_c|Role_m|Ulength_m|Uname_m)$. The partners also use SIK and a specified constant $const2$ to generate key material $K2$ for confidentiality protection. $K2$ yields from $HMAC_{SIK}(const2)$. Both parties use $K2$ to encrypt and decrypt subsequent traffic. [6], [189]

All handshake messages are transmitted in plaintext. Therefore, an external party that observes the handshake can extract the relevant values to derive a shared secret. If the external party knows the PSK between the management device and server, such as operators, it is possible to derive the cryptographic session key material. This allows the external party, such as a monitoring entity, to perform deep packet inspection and extract information from the following IPMI control traffic.

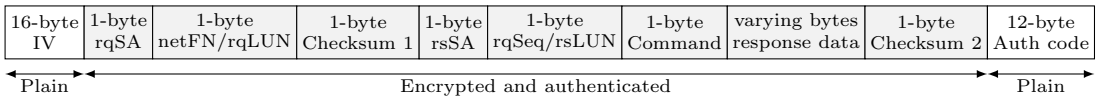

Figure 8.3: Structure of an IPMI response packet

IPMI Control Traffic: After the handshake, IPMI encapsulates commands in IPMI requests and replies in IPMI responses. All IPMI requests and responses have a predefined structure (Figures 8.2 and 8.3).

IPMI commands consist of a network function (netFN), command bytes, and additional data of variable length. The specification [6] defines valid values for each field and which instruction they specify. There are network functions for several parts of server hardware, including server chassis, bridge, sensors, and application-level requests. The command bytes (and additional data) specify the explicit commands within the group, such as *power-down* or *power-up*. Figures 8.2 presents the structure of IPMI requests.

IPMI responses have a similar structure. Responses repeat the netFN and command bytes of the associated command. The IPMI specification defines values for valid values for the response data. These indicate the success or failure when executing the instructed command on the server. In case of a failure, the response can also include additional information about the reason for the failure. Figure 8.3 shows the structure of IPMI request and response packets.

IPMI is an example of how control traffic can explicitly define which action a device should execute. It also shows how operators can decrypt and parse control traffic from independent vantage points.

8.5 Correlating Network Traffic and Environmental Changes

In the previous section, we introduced the characteristics of protocols to control remote devices and IPMI as a guiding example. Similar to IPMI, other control protocols uniquely specify the commands operators send to devices. Thus, the payload in control messages dictates behavior in the near future, such as the boot of a server. As a result, the control traffic also implicates environmental changes, such as sounds in a DC's soundscape.

In the following, we introduce our model to capture this relationship. The model combines two different layers. The first layer is the cyber-layer c, incorporating all logical layers of a system (Chapter 2). Thus, the cyber-layer includes information from network traffic, i. e. the control traffic. In our guiding example, the control traffic are IPMI commands and responses. The second layer is the environment of the cyber-physical layer of our logical structure, capturing the changes in the physical environment in which devices reside. Our model captures the environment as a physical SC signal p.

We define commands observed in c as γ. Similarly, the effects of actions on the environment are measured by p as α. A γ in the model captures all defining characteristics of the command

Table 8.1: Variable overview

Variable	Description
c	cyber layer capture
p	physical layer capture
γ	a cyber layer command
α	execution of a γ on the physical layer
c_γ	cyber layer capture that only contains γ
p_γ	unfiltered side-channel capture containing an α that belongs to γ
p'_γ	filtered side-channel capture
p_α	filtered side-channel signal that only contains α belonging to a γ
ts_{a_i}	a timestamp of an α or γ, $a \in \{\alpha, \gamma\}, i \in \mathbb{N}$
C	list of γ in a c_n
$p_{\Delta t_p}$	side-channel capture during Δt
$p'_{\Delta t_p}$	filtered side-channel capture during Δt
np	noise profile of static background noise
o_γ	offset between a γ and its accompanying α
f_γ	function code of a γ sent via IPMI
v_γ	variable command bytes of an IPMI command
τ_α	execution time of an α
d_γ	L3 destination address of target device of γ

that identify the instructed action. These depend on the underlying protocol. In our IPMI example, γ consists of the IPMI netFN code f_γ and variable command bytes v_γ. From the network packet carrying the command, a γ has a destination Layer 3 (L3) device address d_γ. The destination device is relevant since the same command executed by different devices can cause different environmental changes. For example, different fans can create different sounds in the soundscape. Finally, every command has a timestamp ts_γ at which an observer logs it. We refer to a cyber capture as c_γ in case it only contains γ and no other traffic.

Similarly, we want to model environmental changes due to an action α. We capture three aspects of every action. The first aspect is a timestamp at which ts_α at which α starts to affect the environment. Second, a duration τ_α indicates how long an α induces environmental changes. Third, a reference how α changes the environment.

Chapter 7 introduces a method to identify those features in a composite SC signal p. Using this method, we can extract all these features about environmental changes due to an action. Similar to c_γ, the activity after γ only contains changes due to command execution in a capture p_γ. Since SC signals are typically noisy, it is often necessary to filter this signal. We refer to an SC capture as p'_γ. We denote the part p'_γ that only contains changes due to α by p_α. It is possible to model some sorts of noise, e. g. white noise [190]. We capture a model of this noise in a noise profile np.

To model the relationship between the two events, we rely on a temporal offset o_γ. This offset describes the time frame between the observation of a command γ and the beginning of environmental changes of the devices due to the executed α. We calculate o_γ by using the two

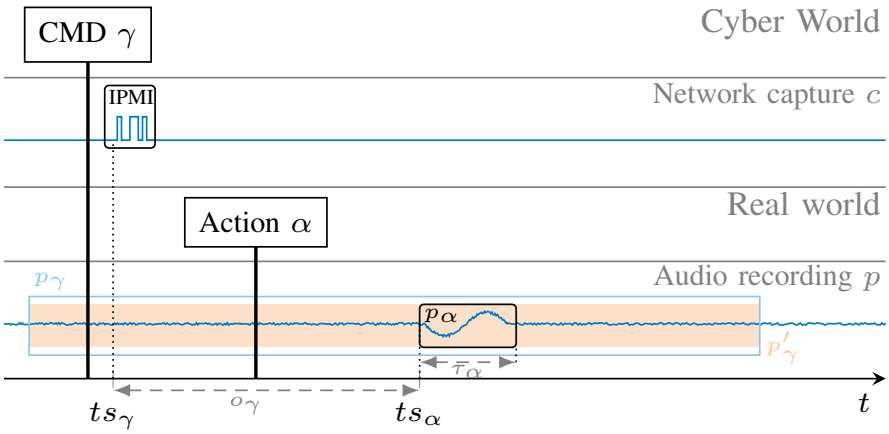

Figure 8.4: Illustration of events and variables and their relationship in our model applied to IPMI and audio

timestamps ts_α and ts_γ

$$o_\gamma = ts_\alpha - ts_\gamma \tag{8.1}$$

This enables the prediction of when an activity changes the environment based on observed control traffic. By using the reference p_α, the model can predict how the environment changes after o_γ. As stated in Section 8.3, we consider signals in which the effects of activity add up. Therefore, by adding the effects on the environment to the signal, the model can predict the interplay between concurrent activities.

Figure 8.4 illustrates our cyber-physical model and our variable and event naming at the example of IPMI. After an operator issues a command γ on its local machine. IPMI delivers the command via the network to its destination. The command is visible as c_γ via the network. When the device receives the command, it performs the instructed activity α. The system concurrently monitors the environment. After the offset o_γ, the model predicts changes p_α in the environment due to the activity.

Table 8.1 provides an overview of the variable names.

8.6 Measuring the Accuracy of the Relation

In this section, we want to measure how consistent the offset o_γ is. In particular, we want to analyze how consistent o_γ is and how reliable it is as an indicator to predict changes in the environment.

Setup: We measure the typical offset in a DC environment. The setup consists of two devices: a management device and a controlled server. The management device sends instructions to servers via IPMI. The management device has an attached Blue Yeti X [184] microphone.

The management server simultaneously has a packet capture running on the interface on which the IPMI commands are transmitted and the microphone. Our implementation runs on the

management server. It implements two modules. The first module derives the shared secret between servers and the management device by parsing the IPMI handshake described in Section 8.4. It extracts IPMI commands and ts_{IPMI} from the control traffic. The second module implements the method introduced in Chapter 7 to identify changes in the soundscape and when they occur, starting at ts_{α}.

We use the management device to instruct servers in the DC via IPMI to perform power-on and power-off activities. We then measure the time offset o_{IPMI} by using Equation 7.1. We repeat this 6 times for both the power-on and power-off actions. While this is not a representative sample size, it provides an initial estimate of our assumption. Collecting more data can be problematic since it occupies resources from other users and introduces wear and tear on devices. IPMI does not provide timing guarantees, in contrast to many industrial environments. Thus, models of environments with stricter timing guarantees, such as an ICS, are more accurate.

Results: The offset o_{IPMI} for both commands was in a constant command-specific time window. For power-on operations, the average offset between the IPMI instruction and change in the environment is 3.36 s and a median of 3.29 s. The standard deviation was around 1.24 s. The minimum o_{IPMI} for power-on is 1.42 s, while the maximum is 5.53 s. In 4 of the 6 samples, the change in the soundscape happened within 0.6 s of the predicted timestamp.

For power-off operations, the average o_{IPMI} is 1.57 s and the median is 1.35 s. The standard deviation was around 0.53 s The minimum o_{IPMI} for power-off is 1.13 s, while the maximum is 2.66 s. Thus, the model's prediction of the power-off is more accurate than that of the power-on operation.

Our measurements support the assumption that control traffic helps to predict future environmental changes. The accuracy depends on the modeled system's action and timing guarantees. However, even in environments with no timing guarantees, such as DCs, our model could predict changes in the soundscape correctly within a time window of 0.6 s. Models for environments with stricter timing guarantees, such as ICS, are likely more accurate.

8.7 Context to Multi-Layer Relationships

This chapter focuses on the relationship between activity on our logical architecture's cyber-physical and network layers (Section 2.2.1). Due to the remote control via the network, events on the network layer containing instructions for cyber-physical layer devices precede the physical changes in the environment. Network layer events are control traffic that contains explicit instructions for devices. We use the method presented in Chapter 7 to identify cyber-physical layer activity. This allows us to develop a model that captures this temporal relationship, linking the activity on both layers. The introduced model predicts changes on the cyber-physical layer based on observed network layer activity.

The temporal offset between events—control traffic on the network layer and device activity on the cyber-physical layer—is constant within a time window. The size and variations within this time window depend on the environment. However, the predictable offset allows predicting

an event on the cyber-physical layer based on observing specific instructions on the network layer. This highlights that there exists a relationship between the events of both layers, which we capture in our cyber-physical model.

Our cyber-physical model extracts device instructions from control traffic on the network layer. In addition, it identifies extracts cyber-physical layer events by using the approach presented in Chapter 7. The model captures the relationship between cyber-physical and network layer events by automatically determining the typical offset between them.

8.8 Key Results

This chapter presents a model that combines control traffic and physical measurements in network-controlled environments. The control traffic allows the prediction of device activity, which leads to environmental changes. Based on knowledge about the activity, modeling the changes using observed control traffic is possible. Thus, the answer to RQ3.2 *What is the relation between network control traffic and physical device activity?* is that control traffic in network controlled systems typically specifies future physical device behavior. As a result, it is possible to predict when a device performs physical activity and how this activity influences the environment.

For additive SCs, such as audio, our model, built upon this approach, can also predict changes in composite channels in which multiple devices operate simultaneously. In this scenario, it is possible to model the effects of concurrent device activity by summing up the effects of individual device activity. To validate our model and measure a real-world example, we use the proposed method in Chapter 7 to measure the typical offset between command and change and the environment. Our measurements show that even in environments with no timing guarantees– such as DCs–devices perform instructed actions within a certain time window.

Chapter 9

Monitoring Physical Behavior in Composite Side Channels

Author's contribution: *This chapter is based on joint work culminating in the two papers L. Wüstrich, S. Gallenmüller, S. M. Günther, G. Carle, and M. Pahl, "Shells Bells: Cyber-Physical Anomaly Detection in Data Centers", in* NOMS 2024 IEEE Network Operations and Management Symposium, Seoul, Republic of Korea, May 6-10, 2024*, IEEE, 2024, pp. 1–10. DOI: 10.1109/NOMS59830.2024.10575124 and L. Wüstrich, L. Schröder, and M. Pahl, "Cyber-Physical Anomaly Detection for ICS", in* 17th IFIP/IEEE International Symposium on Integrated Network Management, IM 2021, Bordeaux, France, May 17-21, 2021*, T. Ahmed, O. Festor, Y. Ghamri-Doudane, J. Kang, A. E. S. Filho, A. Lahmadi, and E. R. M. Madeira, Eds., IEEE, 2021, pp. 950–955. In both papers, the author led the research by conceptualizing the approach and leading the analysis. The author further performed the measurements and evaluated the results. This thesis contains an additional discussion of the plausibility of the results and an assessment of possibilities to adjust the training of the Convolutional Neural Network (CNN) to improve its real world performance.*

The previous chapters analyzed the multi-layer relationship between the network and cyber-physical layers. This chapter applies the model introduced in Chapter 8 to monitor this relationship to detect anomalous system states. We use the model to predict system behavior based on control traffic and compare it with actual behavior. The introduced model to set events from the network and cyber-physical layers into the context of each other makes it possible to identify states with a mismatch between behaviors on different layers. This allows us to advance the Anomaly Detection (AD) capabilities in cyber-physical environments in which devices are remote-controlled. Thus, this chapter addresses Research Question (RQ)3.3 *How can we use network control traffic to validate physical device activity in composite signals?* by building upon the insights obtained in the previous chapters.

9.1 Motivation

Physically correct device behavior is essential for the integrity of a system. Incorrect device behavior, e. g. a valve not closing, can cause damage [10], [80], emergency shutdowns, or even have fatal consequences. Examples are Internet of Things (IoT) devices in smart homes [38], devices in Industrial Control Systems (ICSs) [80], or servers in Data Centers (DCs) [5]. Existing physical validation approaches use Side Channel (SC) monitoring to ensure the integrity of the software flow [73], [165], [191]. However, they are not aimed at validating physical activity. Further, those approaches address controlled environments with isolated devices and low background noise. Such environments are unrealistic to exist in practice. In reality, devices often operate in dense and interconnected setups, such as server rooms [5] or ICSs [1], [80]. In those setups, multiple devices simultaneously influence SC measurements, which results in a composite signal. Composite signals render existing approaches for isolated devices infeasible, making it difficult to validate the activity of individual devices. Chapter 7 investigated how it is possible to identify device activity. *This chapter presents an approach to use the model presented in Chapter 8 for validating physical device activity and detecting anomalies in composite signals.*

Via remote management, users can instruct devices to execute physical actions, which change the environment. Actions range from opening a valve or turning on a light to the controlled power-on of servers. While remote capabilities simplify device management, they introduce new attack vectors [5], [120]. Examples are covert attacks, such as Stuxnet [10] and Harvey [80]. In covert attacks, adversaries simultaneously manipulate device input and output to hide their activity. This enables spoofing non-existent or masquerading existing events [38], falsifying the information shown in remote monitoring. The taxonomy described in Chapter 4 highlights that those attacks affect multiple layers at once. However, remote management also provides an opportunity to enable AD on composite signals. In the previous chapter, we have shown that control traffic can be used as an additional source of information, indicating future device behavior and changes in composite SC signals.

This chapter has three main contributions.

1. A novel method to validate device activity in composite signals using the previously introduced cyber-physical model. We use the model to predict changes in composite SC signals and validate physical behavior. Using information from the network enables to validate if environmental changes are consistent with network-instructed activity. This permits using composite physical measurements such as audio for AD. Our approach addresses the limitations of current SC monitoring methods, which require isolated devices for their analysis [73], [74].
2. We apply our method to Intelligent Platform Management Interface (IPMI) traffic and acoustic side channels. Sound is a less popular SC for attacks or monitoring than others, such as Electromagnetic (EM) emissions. Audio is challenging since it is affected by environmental reflections and is easily influenced by multiple sources, leading to a composite signal. DC environments are frequently considered in AD applications [59], [78].

3. We evaluate our application in a real DC environment with multiple nearby devices. The evaluation has a semi-synthetic part, showing the method's robustness and capability to detect attacks in a controlled environment. We then use our system in a real-world setting, showing its practicality. We show that acoustic side channels provide a cheap and easy-to-implement method with advantages over existing approaches [78], [192]. Our evaluation shows that it is possible to train a machine-learning model with semi-synthetic data and apply it in the real world.

We present the threat model and considered anomalies in Section 9.2. Section 9.3 presents a general approach to use the model from Chapter 8 for AD. Section 9.4 introduces the environment and used SC for the implementation of the approach. Section 9.5 applies the general approach to IPMI and acoustic-channel AD, i. e. a DC environment. Section 9.6 shows our evaluation in a real-world setup. Section 9.7 discusses generalizability and limitations. Section 9.8 compares related work.

The proposed method observes physical activity via an SC. To detect anomalies, we construct an SC signal containing expected changes and compare it with the real SC measurements, by using the model introduced in Chapter 8.

9.2 Threat Model

We assume an attacker that can compromise an unlimited number of remote-controlled devices on the cyber-physical layer (Section 2.2.1) in the system. Attackers have full control over compromised devices and can locally issue commands but cannot alter the firmware, specifying how a device executes instructed actions. To hide activity, the attacker can manipulate device input and output. The attacker can, therefore, spoof and masquerade activities [38]. While manipulating the instructions on the device, the attacker reports the expected responses via the network to the system's monitoring to hide its activity. Harvey [80] and Stuxnet [10] exemplify real-world occurrences of such covert attacks. Such attacks cause a mismatch between network and cyber-physical layer activity.

We can classify the effects of the attacks by using our taxonomy (Chapter 4). The attacks target the cyber-physical layer. Manipulating the input and output of the devices violates the CP.INT and CP.ACC security goals. By also interfering with control traffic, e. g., dropping, injecting, or manipulating packets between the remote management and local device, the attack affects the NET.AUTH, NET.ACC and NET.INT security goals.

Our Anomaly Detection System (ADS) should be able to validate correct physical device activity, i. e. "normal" operations, and identify anomalies in which devices' physical and reported activity differ. Those are "action spoofing," "action masquerading," and "early" and "delayed" execution of instructed activity. We define "action masquerading" as executing actions without a corresponding command. "Action spoofing" is the not-execution of an action or executing an action with an abnormal offset to an issued command, i. e. an early or delayed execution. While an attacker can manipulate device input and output, it cannot control the physical effects of an action execution. When a device executes actions, it changes the environment regardless

of its legitimacy. Our ADS observes management commands sent to devices via a network. Adversaries cannot compromise the external monitoring entity and falsify the reference model. We assume that all devices behave as expected during the reference creation.

9.3 Side-Channel Based Anomaly Detection

We have established that devices performing activities affect the environment. An example is the sound of an opening valve or spinning fans. Measurements can capture activities from multiple nearby devices depending on the SC. This enables monitoring multiple devices at once. Attributing changes in a composite signal to the activity of individual devices requires additional information. Our approach addresses this problem by using information from the cyber layer to analyze SC measurements.

We use the same naming convention as our model in the previous chapter (Table 8.1). When a device receives a command γ, it executes a corresponding activity. We call the execution of γ on the physical layer an action α. In networked systems, devices can receive a limited number of commands and execute corresponding actions. This is due to highly specialized devices or rudimentary capabilities without an already running Operating System (OS). A device causes measurable physical changes during the execution of an α. The characteristics of the changes in an SC signal depend on the executed α. Our model considers composite SC measurements, containing all changes due to αs that devices in proximity execute. Our method uses the model of Chapter 8 to predict SC signal changes from observed commands. It then compares measurements to the expected signal to identify anomalies. As a typical ADS, our approach has a model generation phase and an AD phase.

To use composite SCs for AD, we need to address challenges in three areas:

① processing of SC information,

② processing of cyber layer information, and

③ model creation and validation.

Area ①: Challenges in this area are characterizing changes due to device activity α and distinguishing α from background noise. We define p_α as the change in the SC signal due to the execution of an action. Physical signals are often noisy. This noise can falsify the reasoning about collected measurements. It is necessary to remove this noise before extracting information about an α or performing AD. We categorize three types of noise: (1) static background noise, (2) dynamic noise emitted by surrounding devices, and (3) dynamic noise from unknown sources.

Our approach considers types of noise (1) and (2) for AD. Each noise type needs an independent step for removal. The noise removal techniques for each noise type depend on the used SC. Techniques to remove static noise include limiting the considered frequency spectrum (as in Chapter 7) or using statistical noise models. Addressing noise of type (2) requires a dynamic method. Such methods need additional information about device activities that influence the

measured SC signal. Our approach uses information from the cyber layer to provide the required information. The dynamic method occurs only during the AD phase.

During model generation, methods in ① remove noise and extract SC changes that characterize an individual α for use during AD. During the AD phase, ① removes background noise from the SC signal. Both phases forward the resulting information to ③.

Area ②: We process network control traffic on the cyber layer to extract information about a command γ. This requires a characterization of information that uniquely identifies γ. First, it is necessary to distinguish traffic that contains information about γ from irrelevant network packets. This is possible, e. g., by filtering traffic for a specific protocol and port combination. The specific characterization of γ on the application layer depends on the protocol. This requires decrypting and parsing the packet payload. Finally, it is necessary to identify the device that executes the corresponding action. In order to characterize a γ, we use the L3 destination address and protocol-specific information. Fusing information from multiple sources requires a common identifier [8]. Our approach relies on temporal information. Therefore, we extract a timestamp ts_γ specifying the observation time of γ. We generate a command list C that contains information about all γ.

Area ③: Areas ① and ② provide independent information about α and γ. To combine this information, Area ③ needs to fuse information from both layers to perform AD. Several approaches exist to fuse data and correlate information from multiple data sources [8]. We use temporal information to correlate a γ and its corresponding α. Timings in industrial networks are often constant due to stable network topologies and resource overprovisioning [33], [49]. Our results from Chapter 8 implicate that using the time between observing a γ and the change on the physical layer due to the execution of α is suitable to map the two events. After building a mapping for all γ-α pairs, it is possible to use this information for AD. Creating a mapping for all pairs leads to a modular global reference model consisting of all sub-models. Our method dynamically constructs an expected SC signal during AD based on observed γ. The dynamically constructed signal reflects the expected mixed signal containing all devices' observations. In the final step, we compare the expected to the actual SC signal to identify deviations from the expected behavior.

We detail the model generation and AD in the following.

9.3.1 Reference Model Generation

The model generation phase creates a reference model for the AD phase. To do so, the reference model needs to capture the global behavior of the monitored system. In many cases, ADSs derive the reference model from the global observation of all devices at once. While this helps to incorporate relationships between devices into the reference, it requires a new reference creation when a system changes. In addition, changes in composite SC signals are often unique, depending on the α a device executes. This can lead to a large reference model. To address these issues, we use a modular reference model. The global reference model consists of a set of sub-models. Each sub-model describes the behavior of an individual device that executes an α belonging to a γ. Thus, the model generation maps all γ to a corresponding α in the monitored system.

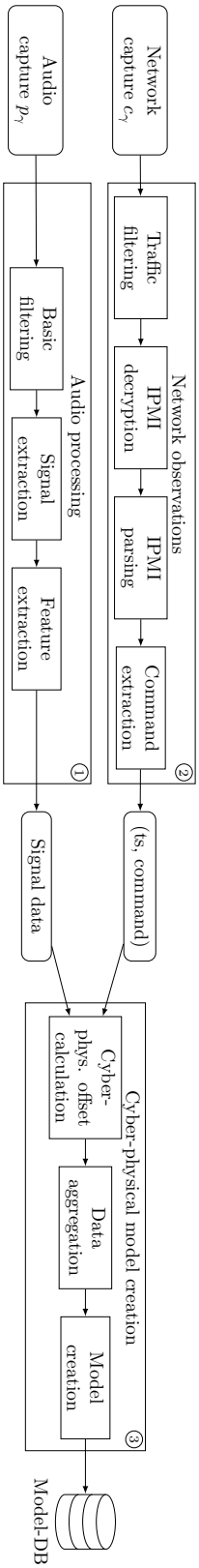

Figure 9.1: For creating a sub-model the first steps extract layer-specific features from c_γ and p_γ. The third step fuses the information into a cyber-physical model.

During AD, our approach rearranges the sub-models dynamically on demand based on events on the cyber layer.

The sub-model generation is a three-step process (Figure 9.1). The first two steps happen in parallel. They process an SC signal p_γ (①) to extract the characteristics of an α, as well as a cyber signal c_γ (②) to extract γ. The third step correlates information about α and γ to generate a γ-α-mapping as cyber-physical model (③). An example for p_γ is an audio recording of α, while c_γ is a network capture that contains γ that triggered α.

Accurate AD requires a precise characterization of expected activity. Unrelated noise in references can decrease AD performance. Therefore, the physical layer model generation must generate an accurate characterization of an α from an SC signal p_γ. The tasks during model generation are removing background noise from p_γ and extracting α. To reduce the amount of noise that can falsify the characterization, an isolated environment to record p_γ that contains α is desirable. Creating an isolated environment during model building is often unrealistic. However, we have shown in Chapter 7 that even if an environment has static background noise, it is possible to detect the effects of α. We use this strategy to extract the changes from p_γ. It is still possible to build robust fingerprinting methods from noisy signals [193], [194].

The first step removes static background noise from p_γ. If p_γ was captured in a non-isolated environment, the responsible module needs to apply sophisticated filtering to remove background noise. The filters for removing background noise are SC specific. We refer to the filtered SC signal as p'_γ. After generating p'_γ, the next step is to extract characteristics describing α. Similar to the filters, the characterization of an α is specific to the SC. For the sub-model in the global reference, our method extracts the SC signal changes due to α from p'_γ as p_α. An α has additional metadata, e. g. a timestamp ts_α that marks when α begins to influence the SC signal. Since our method relies on temporal information to map a γ to α, it is essential to identify ts_α. The metadata further includes a duration τ_α specifying how long it influences the SC.

In the second step, we identify the γ that causes the execution of α on a device on the cyber layer. From a cyber layer capture c_γ, we extract information that specifies γ. The characterization includes the L3 address of the device d_γ that executes α and application layer information that specifies γ. This information depends on the protocol used to control d_γ, e. g. an IPMI command. For the temporal correlation of γ to α in the next step, our method extracts the timestamp ts_γ when γ was observed.

We correlate γ and α to generate the sub-model in the third step. Our method maps γ to α via the timestamps of the events, i. e., ts_γ and ts_α. The accurate mapping between the events requires timestamps ts_γ and ts_α to be created by a synchronized clock. Unsynchronized clocks can introduce inaccuracies [8] and decrease AD performance. The mapping is the offset calculated using Equation 8.5, i. e. the time between γ and when α starts to affect the SC. This way, we connect γ and α for use during AD. We store each sub-model in a model database (model-DB). Each sub-model consists of p_α, extracted features about α, γ, and a mapping that relates p_α to a γ via the temporal information o_γ. Our approach implements the model presented in Chapter 8 to build a reference for all possible actions in the monitored system.

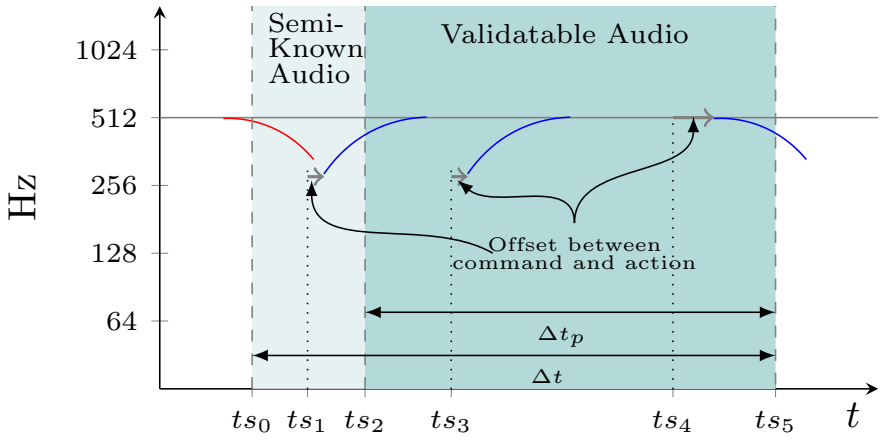

Figure 9.2: A visualization of the time frames Δt, Δt_p, and events in an example with an acoustic side-channel. The ADS analyzes the time frame Δt from ts_0 to ts_5 and observes commands at ts_1, ts_3 and ts_4. The gray arrows indicate the offset between the offset and the change in the signal. The ADS can predict the blue traces as it observes the relevant commands between ts_0 and ts_5. The ADS cannot predict the red trace since the command causing the red trace occurs before ts_0. Thus, the time frame from ts_0 to ts_2 is semi known audio that can contain predictable and unpredictable changes. We can only validate activity in Δt_p from ts_2 to ts_5 since we can predict all changes in this time frame.

The complete model-DB makes up the global reference model of the ADS. In case there are multiple devices of the same type, it is possible to reuse a sub-model. Since the devices then execute the same actions, it is sufficient to adapt the destination address of the sub-model. This increases the transferability of our method. Figure 9.1 illustrates the process of creating a sub-model.

9.3.2 Anomaly Detection

The AD phase analyzes if observations on the physical and cyber layers are consistent. Our ADS can identify anomalies caused by the attacks in our threat model (Section 9.2) or faults. For this task, our approach dynamically builds a reference SC signal based on observed commands γ. It then compares the reference to the measurements for AD.

The identification of anomalies can happen on live observations or in pre-defined intervals. Since activities on the physical layer take place over a period of time, our approach performs the AD periodically as a sliding window. We call the analyzed time window Δt. Our method cannot validate the entire time frame of Δt on the physical layer. We refer to the validatable part as Δt_p. The validatable part Δt_p starts after an offset to the start of Δt and ends at the same time as Δt. This is the case since the execution of an action α, e. g. booting a machine, takes time (Figure 7.1). If an α started before the beginning of $p_{\Delta t_p}$ but has not yet finished, $p_{\Delta t_p}$ includes a part of α at the beginning. The offset has to be long enough such that a generated reference includes all known sources that could influence $p_{\Delta t_p}$. The time during the offset is semi-known since it can contain known and unknown activity.

Figure 9.2 visualizes the time frames for acoustic SCs. The figure shows important changes in the sound frequency spectrum over time. Δt starts at ts_0, Δt_p at ts_1. Both time frames end

at ts_3. The time frame between ts_0 and ts_1 is semi-known. An example of an unknown source is the red trace. This action cannot be validated since the corresponding command happened before the start of Δt. We can only validate the time frame of Δt_p, i. e., starting after ts_1 until ts_3. During Δt, there are three commands on the network at timestamps ts_1, ts_3, and ts_4. The corresponding events happen after an offset marked by the blue traces. The gray line at 512 Hz marks a characteristic frequency on which surrounding devices constantly emit a sound. To ensure a continuous AD, the maximum shift of the sliding window is Δt_p.

The AD phase consists of four steps outlined below. The first two steps happen simultaneously.

1. Side-Channel Processing: Removing static background noise from $p_{\Delta t_p}$. It uses the same techniques to filter noise as the model generation. We target environments with static background noise and dynamic noise from nearby devices. We do not address unknown dynamic noise since including them in a constructed SC signal is impossible. We consider such activity as anomalous. We refer to the filtered SC signal as $p'_{\Delta t_p}$.

2. Control Traffic Processing: Identifying the γ that devices received during Δt from $c_{\Delta t}$. It is necessary to distinguish control traffic from other traffic. This can happen, e. g., by only parsing packets of the relevant protocols and matching defined characteristics from the model generation. This process results in a list C of all γ with their timestamp ts_γ.

3. Reference Signal Construction: Dynamically constructing a composite reference SC signal r. Creating a dynamic reference distinguishes our approach from existing ones, which often compare measurements against fixed pre-recorded traces. We construct an accurate signal containing all relevant signal changes using Algorithm 1. In the previous step, we identified all γ that induce device activity. From the model generation, we know how the physical activity of a device receiving γ affects the SC signal. Therefore, we can construct r based on the observed γ and their ts_γ in C and the sub-models in the model-DB. We initialize r with no initial changes and the start of r ts_r to the start of Δt. We then look up each command γ in C in the model-DB. If γ does not exist in the model-DB, we do not know the effects of it on the SC and ignore it. If the model-DB contains all γ that affect the SC, a γ that is not in the model-DB does not influence the SC. If γ exists in the model-DB, we know its effects α_γ on the SC. We know ts_γ when γ occurs from C, the offset o_γ from γ to SC changes and the effects. We use this information to add α_γ to r at $ts_\gamma + o_\gamma$. By repeating this process for all $\gamma \in C$, we construct r to contain all expected activity during Δt. For the following AD step, we only keep the part of r during Δt_p.

4. Activity Validation: Validating device activity by comparing r to $p'_{\Delta t_p}$. A reliable validation is subject to multiple challenges. One challenge is defining the similarity between a reference and an observed signal. The best metric depends on the chosen SC. It is a result of the distance calculation between the two signals. This distance calculation needs to be robust against noise and temporal shifts. While filtering in the previous steps reduces noise, there is no guarantee for completely removing noise. There are several potential causes for temporal shifts. Offsets o_γ estimate when to expect changes in the SC signal. However, they cannot deterministically predict exact timings. Even in ICSs, there are minor deviations, despite timing guarantees [1], [49]. Other setups, such as DC environments, provide less strict time guarantees

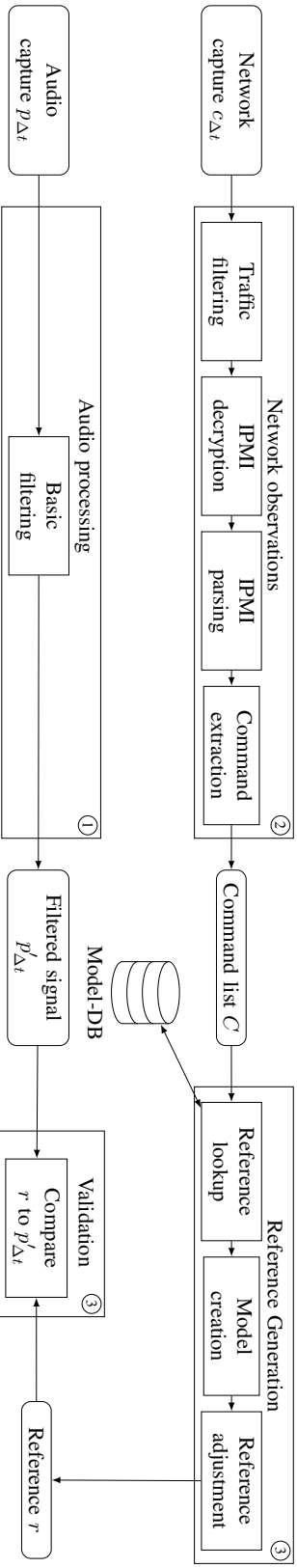

Figure 9.3: The AD phase creates a dynamic reference from control traffic using the model-DB to construct a reference for comparison with real observations.

Algorithm 1 Algorithm to generate a dynamic reference

```
1: initialize start of ts_r as start of Δt
2: initialize r as empty
3: procedure constructReference(C, model-DB)
4:     for γ ∈ C do
5:         if c ∈ model-DB then
6:             retrieve o_γ from model-DB
7:             retrieve α_γ from model-DB
8:             add α_γ to r at timestamp ts_γ + o_γ
       return r during Δt_p
```

than ICSs (Chapter 8). As a result, larger temporal shifts are possible in such systems. This means that the amount of acceptable deviation from an o_γ, which distinguishes normal from abnormal behavior, depends on the monitored system. These aspects affect the choice of a comparison method for validation.

Comparing r and $p'_{\Delta t_p}$ can lead to a positive or negative validation. A positive validation indicates correct behavior, i. e., a high similarity between the constructed signal and recording. A negative validation is the result of low similarity and indicates an anomaly. Figure 9.3 illustrates the steps to perform AD of a time frame.

9.3.3 Advantages and Challenges

The proposed approach to set control traffic into the context of physical changes has several advantages over existing ADSs. (1) The method enables using composite SC signals for AD. The use of information from the cyber layer allows predicting changes in the SC signal, even when multiple devices operate simultaneously. (2) Our approach can use different SCs and information on the cyber layer and even combine multiple SC. The choice of the SC only affects the filtering. The method is similarly adaptable to protocols on the cyber layer. (3) Our ADS works passively after model generation. The ADS has no impact on the monitored system after the global model generation. Our method only observes regular activity and does not require interaction with monitored devices. This is especially important in systems with devices that have limited resources [49]. (4) Other works show that ADSs using multiple layers detect anomalies faster than ADSs only operating on a single layer, e. g., false data injection [8] or sequence attacks [195]. (5) The approach can identify anomalies on both layers by combining network traffic and physical observations. (6) To remain undetected during attacks, an adversary must simultaneously manipulate SC and the cyber layer. (7) Our ADS is modular. The global model consists of sub-models that describe individual actions. Modularity makes replacing parts of the global model easy. If a device is patched or replaced, only the mappings for this device need replacement in the model-DB.

However, our approach also has limitations. (1) Changes in the SC signal due to actions must be unique. If all actions cause similar changes on the chosen SC, the ADS can only validate that an action happened. For classes where actions have the same signature, our approach can only distinguish between but not within classes. (2) Some actions may share similar sub-

sequences. Distinguishing such actions requires more sophisticated methods in the validation step to avoid false positives. (3) If the system does not return to its original state after an action, the ADS must keep track of its state. In this case, the same command on the cyber layer may trigger different actions. The global reference model then needs to include information about a system's state in the mapping, increasing the size of the model-DB. (4) Depending on the SC, additional, possibly unknown, noise sources must be considered and removed. The chosen SC affects the required amount and types of filters, which influences the signal loss and the validation. (5) Since noise is non-deterministic, the validation requires a robust method to reliably compare noisy signals to a generated reference. (6) Depending on the guarantees given by the system, the generation of the reference model is more accurate. For example, the timing guarantees in ICS can lead to a more accurate r. Systems with less strict timing guarantees, such as DCs, require validation methods that can mitigate such inaccuracies. (7) Finally, our approach cannot differentiate if the cause of an anomaly is an attack or a fault. Therefore, the framework needs an extension to support further root cause analysis for a detected anomaly.

9.4 Application Environment Characteristics

This section introduces the environment of our application to DCs and acoustic SCs.

Scenario and Application Environment: Chapter 7 highlights that DC environment has unique properties. DC environments are most of the time remote managed [5], leading to rare human interaction. Therefore, all noise and physical changes are due to known device activity. The soundscape of DC environments is constant, making individual device activity stand out. We, therefore, apply our ADS approach to this environment.

Acoustic Side-Channels—Challenges and Advantages: Power-intake [105], [196], EM [197], and time [49], [196] are commonly used SCs. Sound is a byproduct of many physical activities [74], but less popular than other SCs. In contrast to power intake and EM SCs, sound propagates over a distance of several meters. Reflections, obstacles, and the openness of a space influence the spreading of sound waves. Due to the range of sound distribution, an audio signal can contain sound from multiple nearby devices. It is challenging to isolate the sound of individual devices in such a composite signal. Acoustic channels, therefore, require a thorough pre-processing and often additional information to isolate individual activity.

A major challenge is noise filtering. Noise can be static or dynamic. Several techniques exist to filter an audio signal [198]. Active noise-canceling techniques aim to remove monotonous noises or are performant in a narrow band of the complete frequency spectrum. The removal of dynamic sound, e. g. the vocals of a lead singer from a song require complex methods [199]. These require additional information about the source and nature of a sound. An example is information about devices in proximity that perform actions. Unknown noise sources include all sounds for which no such information is available. Knowledge about the scenario helps to address these challenges. Bandpass filters limit the frequency band to the expected spectrum of sound. This can improve the performance of advanced filtering methods and focus on a specific activity.

Despite the challenges, the use of mixed audio signals has advantages. Given separation capabilities, a single microphone can monitor multiple devices. Microphones can be added retrospectively to a system without affecting existing infrastructure. In addition, microphones are more affordable than other measuring equipment, such as heat cameras, EM-measurement devices, or power meters. Even cheap microphones provide high resolution at a high sampling rate.

9.5 Application to IPMI and Audio

This section applies the concept of Section 9.3 to a specific scenario. We use it to build an ADS for DC environments using IPMI traffic and an acoustic SC.

Audio is suitable to identify processes and actions [178], [179], [200]. Microphones can record audio passively and can be retrospectively added to environments. Our architecture addresses challenges identified by previous work on multi-layer ADSs [8], [195]. In this application, the changes in the signal p_α describe the sound a device emits when it executes α. Actions create specific patterns in the soundscape [201]. The action set in our scenario consists of power-off and power-on of servers [6].

On the cyber layer, our ADS analyzes IPMI traffic. IPMI is a standardized protocol for managing servers, e. g. in DCs (Section 8.4). Our ADS use a Pre-Shared Key (PSK) to derive a shared secret $K2$[‡] from the handshake. After the handshake, the devices use $K2$ to encrypt and integrity protect their communication. Since operators know the PSK, an observing device can extract the relevant information from an observed handshake to calculate $K2$. After calculating $K2$, we can decrypt and parse the IPMI communication.

The management device can then send requests to DC devices to execute commands or query information. A targeted device replies with the requested data or a status code indicating the success or failure of an executed command. Each IPMI command has a distinct *netfunction (netFN)*, command, and command bytes (Section 8.4). Depending on the success of the execution, devices send out specific response codes indicating the result [6]. The combination of netFN, command, command bytes, and L3 information characterize a γ.

9.5.1 Reference Model Generation

Our architecture defines the global reference model as the set of all sub-models. It requires a sub-model for all commands that trigger an action. Each sub-model maps an IPMI command γ to the sound of the corresponding action α.

The inputs for the model generation are audio recordings p_γ and network captures c_γ. Following our method, the ADS processes p_γ and c_γ separately. The following paragraphs describe the process for creating one sub-model.

[‡] The IPMI standard defines the shared secret as $K2$ [6]

Audio Processing: In an optimal case, the recording of p_γ containing the sound of α happens without background noise. While this is, in many cases, not practical, it is possible to remove static noise from audio. The soundscape of DCs fulfills this static property (Chapter 7). Therefore, recording p_γ when no other servers execute measurable actions is sufficient.

We first remove background noise to extract accurate information about α. A common source is electric noise from microphones. In DCs, nearby device fans and climate control devices cause the majority of static background noise (Chapter 7). Device activity does not affect the whole frequency band at once [74]. We limit the considered frequency band to frequencies on which DC devices emit sound to 400-900 Hz. Therefore, we apply low- and high-pass filters to p_γ.

Based on our method introduced in Chapter 7, we use the *Root Mean Square (RMS) Energy* as an indicator to identify the device activity. We use this method to extract the timestamp ts_α of α's start and duration τ_α from p_γ.

We further exploit the static soundscape of DCs to remove noise. We calculate a noise profile np describing the static noise in p_γ before ts_α. The next step uses np to remove static background noise from p_γ. Applying these filtering steps results in a noise-free p'_γ. The final step uses ts_α and τ_α to extract p_α from p'_γ.

Command Extraction: The model generation simultaneously processes c_γ. Network traffic in c_γ only relates to γ, i. e., containing only the relevant IPMI command. The first step filters all non-IPMI traffic from c_γ. After analyzing the handshake between the management and target devices, it derives the shared secret $K2$. The module then parses the IPMI request to extract γ. In addition to the γ itself, the module extracts the L3 destination address of the target device d_γ and timestamp ts_γ. The command extraction forwards the information to the event correlation module.

Event Correlation: To calculate the time from command observation to the execution of a physical action, the module calculates the offset (Equation 7.1). Capturing all data on the same device can prevent synchronization issues [8].

DCs often physically and logically separate management traffic. The DC infrastructure overprovisions network resources for management traffic to ensure a fast and reliable method for operators for device access. Further, the static setup of DCs leads to IPMI packets being typically routed via the same hops. This keeps the temporal distance between observations on the cyber and physical layer constant. Therefore, the network-induced delay in this scenario is negligible, similar to ICS scenarios [49]. While there are similarities to ICS, the given time guarantees in DCs are not as strict as in industrial environments. However, it is possible to estimate when physical changes occur.

We store all information from audio-processing, command extraction, and o_γ as sub-model in the model-DB. Since ts_γ and ts_α are specific to the recording, the model-DB only stores o_γ. Table 9.1 lists all features stored per action in the model-DB.

Table 9.1: Example of a model-DB: every row is a sub-model

entry	exec. offs	device	netFN	cmd bytes	phys. ref.	duration
1	o_{γ_1}	d_1	f_1	v_1	p_{α_1}	τ_1
⋮	⋮	⋮	⋮	⋮	⋮	⋮
m	o_{γ_m}	d_m	f_m	v_m	p_{α_m}	τ_m

9.5.2 Anomaly Detection

In the AD phase, the system validates that recorded audio is consistent with the observed IPMI traffic. Therefore, the ADS dynamically creates a reference audio signal containing all expected changes using the model introduced in Chapter 8. The following section describes the validation of a single time frame Δt. The AD moves this time frame periodically as a sliding window to ensure continuous monitoring

According to our methodology, the first two steps separately process the audio recording and network traffic during Δt. The SC input $p_{\Delta t_p}$ is an audio recording and $c_{\Delta t}$ a network capture.

On the physical layer, we remove static background noise. We use a noise profile to remove static noise to generate $p'_{\Delta t_p}$. Further, we apply the same bandpass filter to $p_{\Delta t_p}$ as during the model generation phase. The filtered $p'_{\Delta t_p}$ is a composite signal containing sound from multiple devices executing actions.

The second step removes non-IPMI traffic from $c_{\Delta t}$. It decrypts the IPMI traffic after observing the handshake described in Section 8.4 using the PSK. After decrypting the traffic, it extracts all γ and their timestamps ts_γ from the remaining packets to create the list C. The module then forwards C to construct the expected SC signal.

In the third step, we construct the dynamic reference audio signal r from all γ in C using Algorithm 1. Sometimes, $p'_{\Delta t_p}$ includes sound from actions that started before the beginning of $p_{\Delta t_p}$. The reference r must include these partial α at the beginning of $p_{\Delta t_p}$. It is, therefore, only possible to validate the sub-interval Δt_p of Δt. Section 9.3.2 outlines the differences between these two time frames, and Figure 9.2 visualizes them. The offset has the length of the maximum of $\tau_{\alpha_i} + o_{\gamma_i}$ of all sub-model entries $i \in 1, \ldots, m$ in the model-DB. The module builds an r for Δt considering all $\gamma \in C$. This ensures that r includes all expected sounds in the recording. The start of r and Δt are identical. For all IPMI commands $\gamma \in C$, the module looks up the reference in the model-DB. If there is no reference for γ, it discards the command as irrelevant for generating r. If a γ in C exists in the model-DB, it retrieves the reference p_α and offset o_γ. The module then adds p_α at timestamp $ts_\gamma + o_\gamma$ to r. After repeating this process for all $\gamma \in C$, r is complete. Algorithm 1 shows this procedure.

The validation compares the last part of r spanning Δt_p. If there is no γ in C, the reference r is empty.

Figure 9.4 shows an example of an r compared to an real recording. The figure shows the spectrogram of an audio audio signal of a booting server. The upper part of the figure shows the real observed signal after its filtering, while the lower part shows the constructed reference

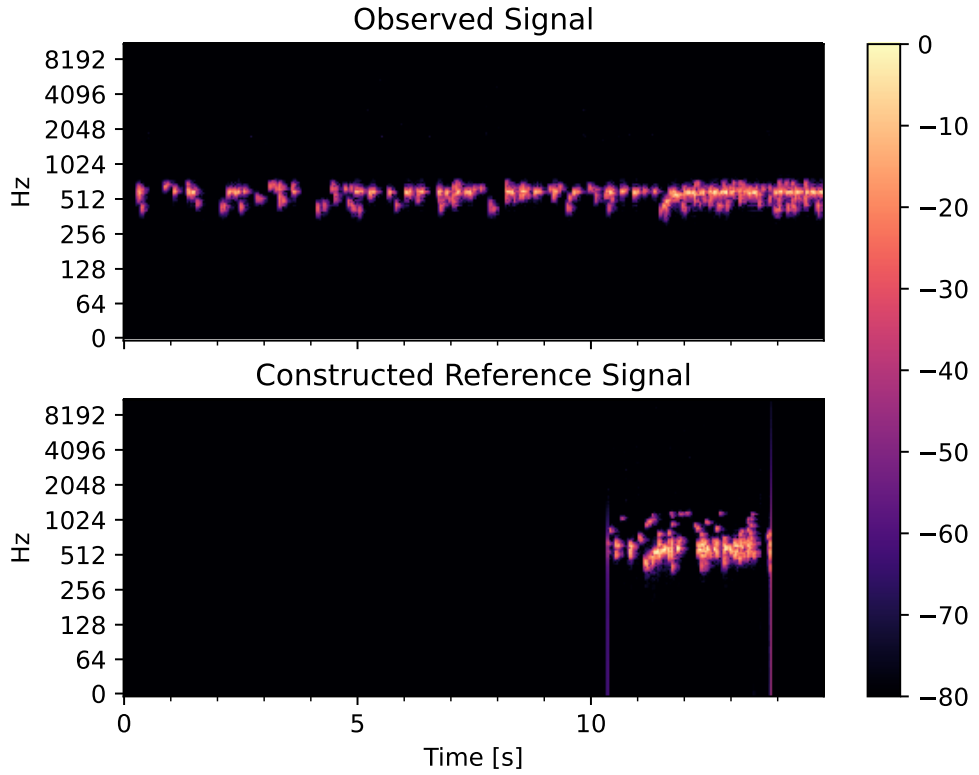

Figure 9.4: A real signal and its reference with a server turning on at 11 s

based on the observed IPMI traffic. The figure also shows the effects described in Section 9.3.2. The real observed signal still contains noise even after filtering. Since IPMI offers no temporal guarantees, temporal shifts can occur (Chapter 8.6). In the shown example, the accelerating fans influence the soundscape 0.5 s later than predicted by the model. We consider those effects in the validation described below.

The fourth step compares r to $p'_{\Delta t_p}$. There are a variety of methods for comparing audio methods. Audio fingerprinting [202], Dynamic Time Warping (DTW) [203], image recognition [178], and Machine Learning (ML) [204] are common techniques. This paragraph compares them and motivates our chosen validation method. Section 9.3.2 emphasizes that a reliable validation requires temporal and noise robustness.

Audio fingerprinting offers a high noise robustness [202]. However, due to the calculation of the fingerprint, temporal shifts lead to low similarity. This effect is measurable when an action in a recording shifts by an offset ≥ 10 ms to its reference counterpart. This makes audio fingerprinting only suitable for deterministic environments in which it is possible to accurately predict device activity or in which fixed snippets need comparison.

DTW calculates a distance between signals [203]. While DTW is robust to temporal shifts and speed variations, it requires that signals have the same structure. This can cause issues in case of action masquerading and spoofing attacks, which can change the structure of the compared signals due to missing or added device activity. In addition, noise can result in a high

Table 9.2: Comparison of similarity calculation methods for audio signals

Method	Noise	Temporal	Comparison	Reasoning
Fingerprinting	✓	✗	✓	% matching hashes
DTW	✗	✓	✓	distance
RMS Energy	✗	✓	✓	distance
Image recognition	✓	✓	✗	events
ML	✓	✓	✓	events/classes

distance calculation of DTW, even if most of the signal is similar [205]. Thus, noise affects the performance of DTW.

It is also possible to compare aggregated signal information, e. g., the **RMS Energy** [16]. Noise affects RMS-Energy-based comparisons similar to DTW. Since the RMS-Energy is an aggregated value, it is useful to identify time frames when changes occur. Aggregating information into a single value makes it impossible to identify which frequencies changed. This makes it challenging to distinguish activities that cause changes that affect the aggregated value in a similar manner. Depending on the distance metric to compare the RMS-Energy of two signals, this method is robust to temporal shifts.

Image recognition like [178] identifies events in spectrograms. Events cause geometric patterns [201] which the methods identify. These patterns must stand out from the background to identify them via image recognition. Noisy environments, such as DCs, alleviate the contrast of patterns to the background and reduce the efficiency of such approaches. While image recognition can identify events, its purpose is not to compare complete spectrograms with multiple components.

ML methods can be robust to noise and temporal shifts and can even identify melody variations. Examples are the identification of audio sequences, and variations of musical themes [204]. Depending on the chosen ML method, it is possible to classify the result of the comparison. A drawback of this approach is that the amount of acceptable delay needs to be specified at training time.

Table 9.2 presents the capabilities of the considered comparison methods. Due to the superior robustness, our validation relies on an ML-based approach. The high resolution and audio sampling rate lead to high dimensional input data for ML. The processing of such data requires data preprocessing and preparation.

Our approach applies a procedure to reduce the dimensionality of the audio input while retaining important information about changes due to activity. Figure 9.5 shows the procedure, Figure 9.6 visualizes how the steps affect an examplary audio signal of a booting server. In general, the procedure reduces the resolution of the frequency band and identifies deviations from the average energy on a frequency.

The process initially performs a *Short Term Fourier Transform (STFT)* on the audio signal. Similar to other works [201], [206] we choose a window size of 2048 for the STFT. $|\Delta t|$ denotes the number of STFT time frames of the audio during Δt and $|F|$ the frequency band size. The

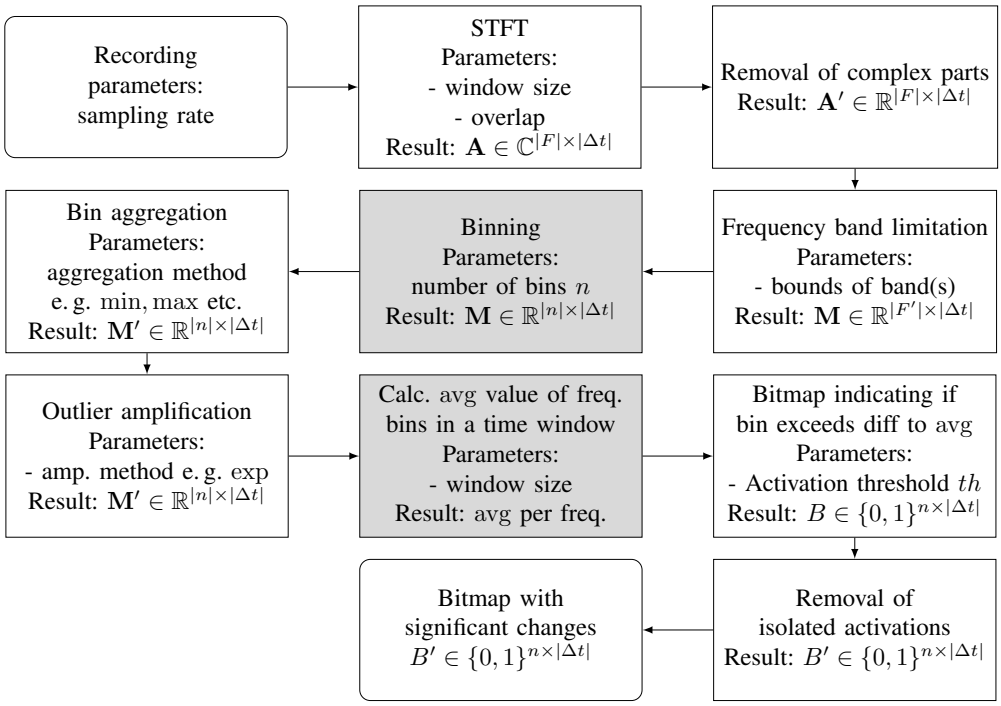

Figure 9.5: Preprocessing chain of audio input for the ML validation. Operations in gray boxes do not directly affect the signal but are relevant operations for the following steps.

result of the STFT is a matrix $\mathbf{A} \in \mathbb{C}^{|F|\times|\Delta t|}$ holding Fourier coefficients as elements. A 15 s recording at a sampling rate of 44.1 kHz leads to $|\Delta t| = 646$, each frame covering 23.21 ms. An $a_{ij} \in \mathbf{A}$ describes the amplitude of frequency $i \in F$ of a frame $j \in |\Delta t|$. Since the elements of $\mathbf{A}$ are complex, ordering them is impossible. To compare the $a_{ij} \in \mathbf{A}$, the next step takes their absolute values $|a_{ij}|$ to create a matrix $\mathbf{A}' \in \mathbb{R}^{|F|\times|\Delta t|}$. Since device activity does not affect all frequencies $f \in F$ (see Section 7.5.1), the process reduces the frequency band to F'. The reduced frequency band F' is a subset of F, e. g., the lower quarter. This results in $|F'| < |F|$ and a matrix $\mathbf{M} \in \mathbb{R}^{|F'|\times|\Delta t|}$.

To further reduce the dimensionality of the CNN input, we aggregate the remaining frequency band F' of each time step $j \in \Delta t$ into n bins. The number of bins $n \in \{1, ..., |F'|\}$ influences information loss due to value aggregation. When $n = 1$, all frequencies in F' are aggregated into a single bin. If $n = |F'|$, all frequencies in F' are considered individually. This results in a tradeoff between the resolution and dimensionality of the input. After a series of experiments, we choose $n = 128$.

The method calculates a single value for each bin using aggregation methods such as max, min, avg, or sum. Our validation aims to identify significant changes an activity causes to the frequency band. We calculate the avg of each bin as representative. This step initially makes the method susceptible to noise causing random spikes in a bin. We address those outliers in a later step. After reducing the dimensionality in the previous steps, the remaining steps aim at identifying signal changes. Each representative element of a bin is amplified to separate

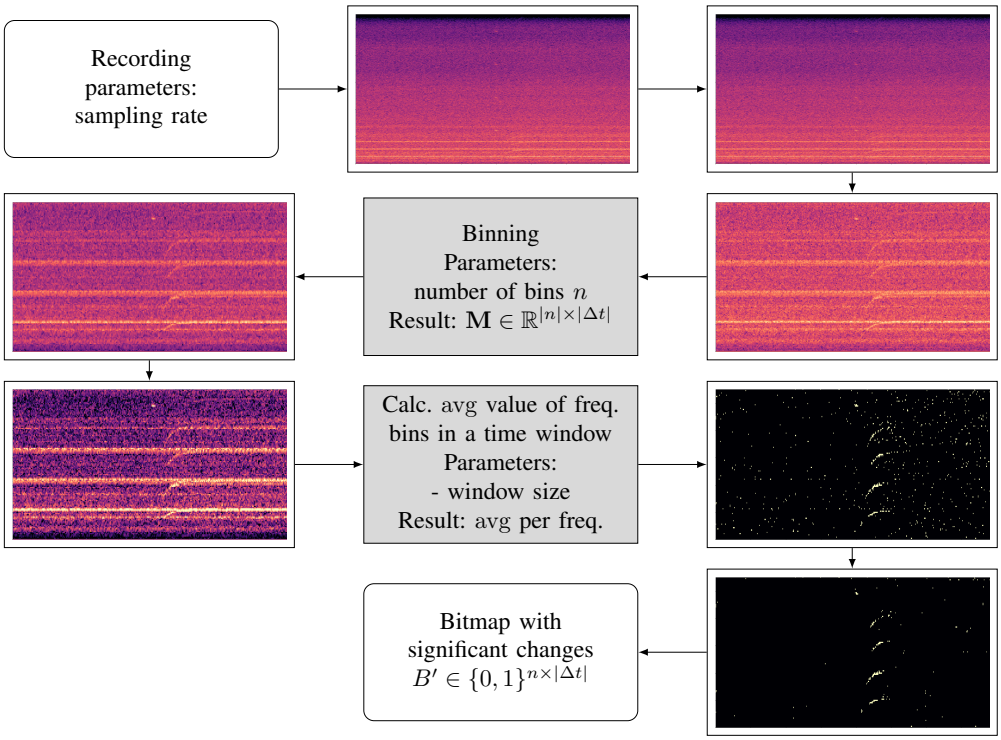

Figure 9.6: Preprocessing chain of audio input for the ML validation at an example audio signal

outliers. The amount of amplification depends on the scenario. While separation is important, our method limits the amplification effect via a scenario-specific ceiling for the later steps.

The following steps consider individual frequency bins b_{fj} of a frequency f during all $j \in |\Delta t|$. To identify outliers on a frequency bin b, the procedure calculates the average value of each frequency bin over Δt as $a_{b_f} = \sum_{j=0}^{|\Delta t|} \frac{b_{fj}}{|\Delta t|}$. It then calculates a distance $|b_{fj} - a_{b_f}|$ and compares it to a threshold th. If $|b_{fj} - a_{b_f}| \leq th$, it is marked as 0, otherwise as 1. The sox utility filters noise in a similar way [207]. The choice of th depends on a variety of factors. One factor is how many events stand out from the background noise. A lower difference to the background demands a smaller th as otherwise expected bits do not get activated. The second factor is the number of events in Δt. More events increase the average, reducing the difference between peaks and the average. Therefore, th needs to be set accordingly. The method stores all marks in bitmap $B \in \{0,1\}^{n \times |\Delta t|}$ highlighting the changes.

Random noise spikes can impact the calculation of bin representatives and cause wrongly activated bits in the bitmap. These occur randomly and in isolation. Similar to Pham et al. [201], we found that activity causes patterns, i. e., larger, connected, and activated fields. To remove random noise, we deactivate isolated bits. We define an isolated bit by a maximum number of activated bits in proximity. The choice of this number requires thorough considerations. A small maximum activation number can lead to noise remaining in the bitmap. However, an activation number that is too high leads to the false removal of action patterns. In particular small patterns, e. g., caused by a short beep, can be wrongfully filtered out if the activation number

is too high. The definition of proximity requires a similar discussion. A too small proximity distance wrongfully disregards valid signals, too high values do not filter out noise. The isolated filtering step results in the final bitmap $B' \in \{0,1\}^{n \times |\Delta t|}$.

The procedure highlights changes relative to background noise. This makes it adaptive and robust to changes in the base-level of background noise. The preprocessing and signal construction for the ML-based validation create two bitmaps. Each bitmap highlights the changes in one audio artifact—the reference and the recorded signal. Since bitmaps represent images, we use a CNN. Various image processing applications show that CNNs are particularly suited for tasks with image-like input. The structure of our CNN is similar to the LeNet [208] and ConvNet [209]. We use four convolutional layers with max-pooling layers after the second and the fourth layer. There are two more fully connected layers after the second pooling layer for the classification. The input for the CNN is a concatenation of both bitmaps into a validation bitmap. In our setup, the size of the validation bitmap is in $\{0,1\}^{256 \times 646}$. We categorize the input into the five classes of the attacker model (Section 9.2). If the CNN classifies the bitmap as normal, the ADS validates the time frame as "normal". Otherwise, it raises an alarm and shows the classification.

9.5.3 Application-Specific Challenges

Using acoustic SCs introduces challenges. First, sound wave dispersion depends on the environment. Therefore, references for the model-DB need to be created within the monitored system. Changes in the environment, e. g., the setup of new structures that affect the propagation of sound can require a new training of the model-DB. Second, audio filtering is complex and requires additional knowledge about an environment. The choice of filters depends on the environment and monitored devices. Filtering introduces information loss. Careful tuning is necessary to avoid automatically filtering out important information. In particular, filtering dynamic sound sources is computationally expensive [178]. Dynamic filtering methods require additional domain knowledge.

9.6 Evaluation

Our evaluation measures the capabilities of our approach. We first evaluate the AD capabilities in a controlled experiment using synthetic data from a real-world DC environment. In a second evaluation, we show the proposed ADS functionality in a real setup.

Implementation: We implemented or system (cf. Section 9.5) in Python, using *librosa* [185] and *(py)sox* [210], [211] for audio analysis and processing. The prototype uses *pypacker* [212] to process network traffic. The CNN validation is implemented in *pytorch* [213].

Test Environment: The DC environment providing data for our evaluation consists of 33 servers and 4 actively cooled switches in a single air-conditioned room. We can control all devices via IPMI. There are several types of servers. Of each server type, there are multiple identical devices. In the soundscape, the server types mainly differ in the cooling fans, emitting sound on specific frequencies (Chapter 7). The frequency ranges of fans identify each server

type. Typically, the servers continuously run network experiments. The typical noise level in the DC is around 72.6 dB(A). The ADS runs on a Linux machine with Debian 11 installed, an Intel Xeon 1265L CPU, and 16 GB RAM.

Experiment Setup: Our experiments involve three device types. The first device type encompasses management devices issuing commands to servers. Management devices can monitor other devices from within the network by analyzing IPMI replies. We send instructions from management devices to other devices with the command line tool *ipmitool* [214]. The second class of devices are servers. These receive IPMI commands from management devices to execute actions. After building the model-DB, the servers can be fully compromised according to our threat model. The third class is a monitor running our ADS. The monitor captures real-time traffic between management devices and servers on its network interface. In our scenario, the management device and the monitor run on the same machine. We use a Blue Yeti X [184] microphone in cardioid recording mode for audio recording at the monitor. The microphone records the frequency band of 20 Hz to 20 kHz at a sampling rate of 44.1 kHz. The placement of the microphone is approximately 0.3 m in front of the server racks. Figure 9.7 shows the schematic and real setup.

Reference Model Generation: To construct the model-DB, we send *power off* and *power on* commands via IPMI. At this stage, we assume the devices execute the instructed actions as intended. These actions cause server fans to stop or start. As already shown in previous chapters, the change in fan speed changes the soundscape. The monitor extracts the relevant features from the recording and maps it to an IPMI command to a destination device. The reference creation takes place *during normal operation*, i. e., without interrupting the general operation. To simplify the reference creation, we limit the frequency band to 400 Hz–900 Hz, the spectrum of the fans. The model additionally applies noise filtering via sox with a strength of 0.05. The model-DB contains 8 references for 4 devices, covering only a subset of all devices. We use the same model-DB for the synthetic and real-world evaluation. It is possible to reuse existing models for devices making identical noises.

Metrics: We evaluate the performance by analyzing commonly used metrics [74], [215] (Chapter 2.4.3).

9.6.1 Synthetic Model Training and Evaluation

Model Training with Generated Data: The generation of sufficient samples to train ML can be problematic. If there are multiple actions, the combination of these actions creates an exponentially growing combination space. Recording audio in a real environment requires time, e. g., 1000 recordings with a length of 15 s are equivalent to 4.16 h. Recorded devices are blocked from being used by other users during the data generation. In addition, the repeated command execution causes wear and tear. This makes the generation of a real world data set challenging. We, therefore, create a semi-synthetic dataset by rearranging references from the previously generated model-DB. This is possible since we consider activities to be deterministic. This enables the generation of labeled traces from a few recordings for training. We train the CNN validation module on such a dataset.

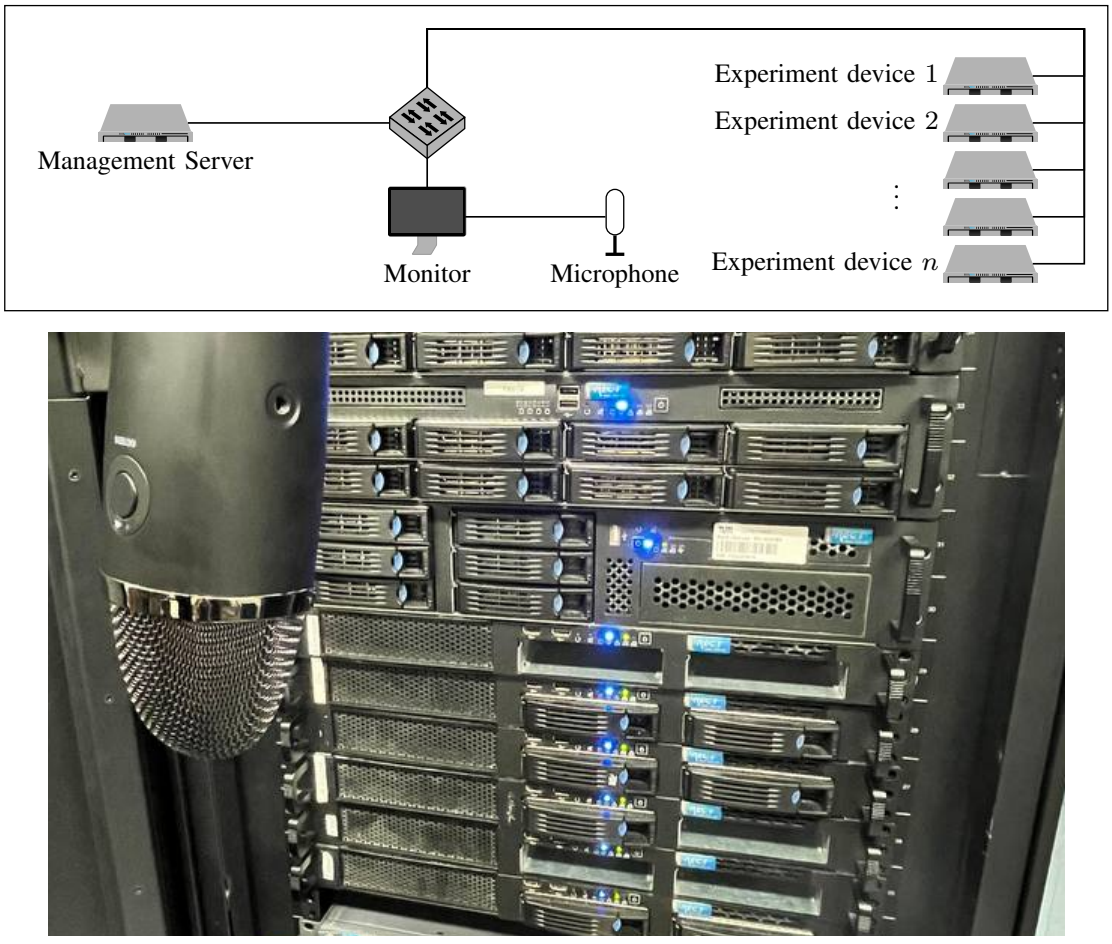

Figure 9.7: Schematic and real experiment setup

The dataset contains 72600 recordings and references spanning 302.5 h. References contain 0 to 2 simultaneous actions from the model-DB. Traces of 1 action show the possibility to identify a single device in a composite signal, and 2 actions show the performance if multiple devices operate concurrently. There are 160 different references for 1 and 80 for 2 actions. This ensures a balance of spoofing and masquerading traces with 1 and 2 actions.

To generate the labels and data, we temporally shift or remove actions compared to their reference to artificially construct anomalous recordings. Depending on the change in the recording, we automatically assign a label. If the reference and recording contain the same actions at the same time stamps, the time frame is labeled "normal". The labeling requires a definition of acceptable delay and early execution. In our scenario, we consider the execution within a time frame of 0.5 s to the expected execution acceptable and as "normal". To avoid False Negative (FN) classifications for early and delayed execution anomalies, we choose a slightly stricter threshold than the typical variation observed in Section 8.6. If an event shifts 0.5 s to 1.5 s, it is labeled as early or delayed execution. Temporal shifts of more than 1.5 s are not in the dataset and are considered either action spoofing or masquerading. Multiple repetitions of the model-building process show that this is a reasonable assumption. Entries in which references contain more actions than the recording are labeled as "spoofing". Similarly, entries with fewer actions in the reference than the recording are labeled "masquerading."

The distribution of classes in the data set is 15000 normal, 9200 samples for each delay and early execution, and 19200 for each of the masquerading and spoofing classes. The train-test split of the dataset is 80/20, yielding 58080 entries for training and 14520 for testing. We train the model for 20 epochs with a learning rate of 0.01. A non-improving loss indicated overfitting after 11 epochs. The remainder of the evaluation uses the model trained for 11 epochs.

Results: Table 9.3 lists the classification results. The chosen model has an overall accuracy of 98.62 % on the synthetic test dataset. The model achieves the highest precision for detecting action spoofing and masquerading with a precision of 99.66 % and 99.63 %. The model correctly classifies early and delayed execution with a precision of 98.74 % and 99.21 %. It shows the worst performance at classifying normal conditions with a precision of 95.46 %. The recall (Section 2.4.3) for all classes implicates how well the CNN can detect each anomaly class. The trained CNN has the highest recall for "spoofing" (≈ 0.9924) and "masquerading" (≈ 0.9898) anomalies. In the other anomalies, the recall is also high with ≈ 0.9823 for "normal", ≈ 0.9813 for "delayed execution" and ≈ 0.9771 for "early execution" samples. This shows that the proposed approach can identify all of the relevant anomalies, as well as distinguish them from normal device activity.

Our evaluation shows a limitation of our approach due to the preprocessing step. Multiple identical operations within the same time frame merge into a single pattern in the bitmaps. Therefore, the current model cannot distinguish if a pattern results from an individual or two simultaneous activities, creating a shadowing effect.

9.6.2 Real-World Evaluation

This section evaluates the approach's applicability in the earlier described environment.

Table 9.3: Validation results of the CNN model

Class/Classified	Normal	Early	Delay	Spoof.	Masqu.	Recall
Normal (3000)	2947	23	14	8	8	0.9823
Early (1920)	40	1876	0	0	4	0.9771
Delay (1920)	35	0	1884	1	0	0.9813
Spoofing (3840)	27	0	1	3811	1	0.9924
Masquerading (3840)	38	1	0	0	3801	0.9898
Precision	0.9546	0.9874	0.9921	0.9976	0.9963	

Setup: We reuse the setup shown in Figure 9.7. We then simulate anomalies in each of the four anomaly classes. To simulate early and delayed execution of events, we add or subtract 1 s to the offset of the sub-models in the model-DB. For the event masquerading attack simulation, we remove the sub-model for a monitored device from the model-DB such that the constructed reference using the presented algorithm does not contain the expected change. If an IPMI managed device receives a command to turn into a state in which it already is, e. g. power on, it responds with a success message. We use this behavior to simulate event spoofing for power-off and power-on operations.

We create four types of time frames: normal with 0 actions, normal with 1 action, action spoofing, and action masquerading. Due to the missing guarantees for maximum execution times in IPMI , we include "early" and "delayed" action execution in "normal" cases. We confirm the classification decision of the CNN by visual inspection. We collect 20 "normal" time frames and 20 recordings for each attack. In total, we validate 80 independent 15 s time frames, making up 20 min.

Results: The results show that the model can accurately classify real-world measurements. All 20 of the time frames with 0 actions were classified correctly as "normal". The model does not classify random noise that remains ofter filtering as spoofed activities. The remaining analyzed time frames have at least one action in either the reference, recording, or both.

A visual inspection of the remaining 20 "normal" time frames confirms 15 of the classifications by the CNN. The offset between IPMI commands and actions varied as expected. The CNN correctly identified minor deviations from the expectation as either early or delayed execution in three out of three cases. In six other cases, the deviation from the expectation was larger than the specified acceptable difference of 0.5 s. The CNN correctly classified those six time frames as "spoofing". In six other cases, the CNN falsely classified a time frame as "spoofing".

The model successfully detected 19 of 20 spoofed events. In the misclassified time frame, the model classified the activity once as "normal" due to patterns in the recording of remaining noise. This highlights the importance of filtering and change extraction as input for the validation.

The model did not identify any of the action masquerading attacks. A visual inspection of the bitmaps of the reference and recording shows that the construction method correctly highlights device activity. In 4 cases, the remaining patterns in the recording are indistinguishable from noise patterns, which is why we deem these four classifications correct.

Table 9.4: DC evaluation results

Class/Classified	Normal	Early	Delay	Spoof.	Masqu.
Normal (40)	25	1	2	12	0
Spoofing (20)	1	0	0	19	0
Masqu. (20)	20 (4)†	0	0	0	0 (16)†

† In combination with RMS Energy.

This indicates an issue with the ML validation step. We argue that an improved training set can address this issue, e. g., if it includes more examples of masquerading. It is possible to combine multiple validation methods to identify such attacks. For example, the RMS-Energy-based approach used in the model building can identify the time frames in which activity occurs. If there is no expected activity but the RMS-Energy-based method identifies an action masquerading takes place. With this extension, our approach could correctly identify masquerading in 16 time-frames. We point out, that identifying masqueraded activity with the RMS-Energy-based only works if there is no control traffic.

Table 9.4 summarizes the results. In our quantitative evaluation, 57 (73 in combination with RMS-Energy) of the 80 time frames were classified correctly. This shows that the presented approach could successfully validate device activity and identify anomalies in a composite signal. However, the capabilities to detect anomalies varies on the anomaly type. While the model can identify "spoofing", it fails to identify "masquerading" anomalies.

9.6.3 Plausibility of the Real-World Measurements

This section discusses the plausibility of the real-world evaluation and puts the results from the real-world measurements into context with the synthetic evaluation. We determine if the detection accuracy in the real-world measurements is realistic given the accuracy of the CNN model in the synthetic evaluation. The CNN model classifies each time frame separately into one of the anomaly classes or as "normal". The classification of each time frame is either correct or wrong. In addition, the CNN classifies time frames independently. This means the classification of one time frame does not influence the result of classifying other time frames.

Since every classification is true or false, we use a binomial distribution to assess the plausibility of the real-world experiments. The overall accuracy of the CNN in the synthetic evaluation using 14520 traces (Section 9.6.1) is 0.9862. Thus, we assume that 0.9862 is the probability that the CNN correctly classifies a time frame.

Overall accuracy: To achieve the accuracy of 0.9862 from the synthetic evaluation, the CNN model needs 78.896 (approx. 79) correctly classified time frames. In order to pass the typically assumed threshold of 0.05 [216], at least 78 observations need to be classified correctly. However, the CNN model correctly assessed 57 of the 80 time frames in the real-world evaluation. Combined with RMS Energy, this number increases to 73 of 80 correctly classified time frames. The numbers of correctly classified time frames are below the expectation of 79. This implies that the ADS is less accurate in the real-world setup.

The likelihood of observing the observed number of correct classifications for the standard model by using the assumed accuracy is:

$$P(X = 57) = \binom{80}{57}(0.9862)^{57} \cdot (0.0138)^{23} \approx 5.1 \cdot 10^{-24}$$

Analogously, the likelihood for observing 73 correctly classified time frames (by using the combination with RMS Energy) is:

$$P(X = 73) = \binom{80}{73}(0.9862)^{73} \cdot (0.0138)^{7} = 0.000109794$$

Both values are below the threshold of 0.05. Given the accuracy of 0.9862 from the synthetic evaluation, the observed number of successes is unlikely to occur. We, therefore, conclude that the proposed ADS is less accurate in real-world measurements. From the observed classifications, an accuracy of $\frac{57}{80} = 0.7125$ (or $\frac{73}{80} = 0.9125$) is more plausible. The decreased accuracy fosters the need for adjustments before production-ready deployment of the ADS.

Per class accuracy: The real-world evaluation shows that the CNN model performs differently depending on the anomaly type. Thus, we also investigate the plausibility per class. In this case, we use the recall for each class by using the classifications from the synthetic experiments (Table 9.3). In real-world experiments, it is difficult to distinguish the "normal", "delayed", and "early execution" classes due to a lack of guarantees given by the monitored system. Therefore, we only assess the "masquerading" and "spoofing" anomaly classes as those are the most crucial ones.

The ADS failed to detect any of the "masquerading" anomalies in the real-world measurements. Thus, it does not work in its current state for detecting "masquerading" anomalies in the real world. In particular, this contrasts the observed recall of 0.9898 in the synthetic evaluation. We propose methods to address this issue in the paragraph "Assessment of the plausibility".

In the real world evaluation, the ADS correctly identified 19 of 20 "spoofing" anomalies which has a likelihood of

$$P(X = 19) = \binom{20}{19}(0.9924)^{19} \cdot (0.0076)^{1} \approx 0.13149$$

which is above the threshold of 0.05. This means that the ADS can detect "spoofing" anomalies similarly in the synthetic and real-world experiments.

Assessment of the plausibility: A lower accuracy in the real world is realistic for several reasons. Real-world measurements contain more background noise not present in simulations, such as in our synthetic evaluation. While our proposed approach ADS uses filtering to remove noise, the signal retains some remaining noise. In particular, our approach has no mitigation to filter unexpected dynamic noise (Section 9.3). While the server room of the real-world evaluation has an active Air Condition (AC), we noticed that the room's overall loudness, and therefore the

amount of background noise, depends on the seasons. This is because the builtin AC cools the server room relatively to the outside temperature, e. g., $-3\,^{\circ}\mathrm{C}$ instead of keeping the temperature fixed. Therefore, the overall temperature in the server room for the experiments is higher due to more prolonged exposure to the sun and higher outside temperatures during those months. As a result, during warm summer, the servers are louder than during winter due to a higher speed of the onboard cooling fans. Therefore, the amount of filtering used when processing a time frame may need adaption depending on the room temperature, which may increase during higher temperatures and decrease during lower temperatures.
The real-world measurements are not performed in a fully controlled environment. The activity of other devices in the environment that are not part of the model-DB can affect the audio recordings. This activity appears as dynamic noise our system currently does not address, leading to potential False Positives (FPs) of "masquerading" cases. Another factor is that the recordings in the real-world setup can have slight variations, which is different to the fixed traces in the synthetic evaluations. Reasons include the mentioned noise or a slightly different microphone placement during the reference recording and real-world evaluation. Another reason for a worse performance in the real world can be effects due to the model training, e. g. the earlier described shadowing. Those effects can lead to a slight change in the input data for the CNN, which can degrade its performance [217], [218].

Our evaluation highlights the challenges when using ML models trained with synthetic data in real world scenarios. However, there are multiple ways to mitigate unexpected effects in real-world environments. This can potentially improve the performance of synthetically trained ML models in real-world settings. In particular, adjustments during the training phase can improve the cyber-physical model and increase the performance of the ADS.

First, we can use enhanced noise models that add noise to the training samples during the training data generation. By incorporating expected noise into the training data, it is possible to mitigate the effects of unfiltered noise of real-world recordings. This can improve the performance of the ADS in all classes.

A second option is recalibrating the acceptable temporal deviations from the predicted activity. As pointed out in Chapter 8, the expected deviation depends on the guarantees given by the system. Increasing the acceptable time window in which devices can execute an instructed action improves the classification of "normal" operation. With an increasing acceptable deviation, the ADS is less strict about temporal deviations of the real-world measurements to its predicted model. Thus, this approach reduces the FP rate for "early" and "delayed" execution anomalies. However, an increasing window decreases the capability to distinguish those anomalies from "normal" execution. Therefore, the choice of acceptable delay depends on the prioritization of the system operator.

Third, to address minor deviations in the measurements, it is possible to increase the number of recordings for each action in the training data generation. This way, the training set contains more variations of the expected activity, allowing for more deviations in the execution. In particular, using recordings from different microphone positions allows to mitigate the effects of different sensor placements.

Finally, creating more training samples of classes in which the ADS under-performs is possible. By supplying more samples of those classes, the CNN can learn better to detect those scenarios. This potentially increases the performance of detecting anomalies, such as "masquerading".

9.6.4 Sensitivity

To evade detection, attackers must ensure that control traffic and SC measurements match. Action spoofing requires corresponding physical changes. This requires physical access to the environment. Action spoofing is possible in combination with action masquerading. By forcing another device to perform an action that causes identical changes, both signals match. There is an analogous approach to evade the detection of action masquerading. One way to address this is to add localization capabilities, e. g. multiple microphones. Another way to avoid detection of action masquerading is to manipulate the execution of actions by limiting physical changes, e. g. lowering the speed of fan deceleration. Since our system filters small changes as noise, an attacker can evade detection. In our setup, we can only detect activities with a signal-to-noise ratio (SNR) higher than $-4.8\,\mathrm{dB}$. Such low-level manipulations require firmware-level access to the compromised device to alter how a device executes an action. Finally, it is possible to masquerade actions by shadowing other legitimate actions without localization. However, the fixed offsets require attackers to accurately predict actions, which is challenging [38].

9.7 Discussion

This section discusses the generalizability and limitations of the presented approach.

Generalizability: This chapter applied the approach to a DC scenario. However, it can be used in other environments with other SCs and protocols. The requirements for a successful adaption are the observability of commands and deterministic device behavior. It is possible to parse other control traffic on the cyber layer, e. g. MODBUS [30]. Similarly, using other additive side channels, e. g., shared power meters or temperature sensors, is also possible. While validation methods, such as audio fingerprinting, are SC-specific, ML approaches are flexible and adaptable. A system that guarantees execution times allows stricter validation methods.

As shown in the evaluation, the modular approach allows to create of a synthetic dataset that can be used to train a model for ML. Since only changes in the SC signal are relevant for the validation, we can retroactively equip environments with our method and retrain the model on demand.

Limitations: If the monitor cannot decrypt encrypted network traffic, observing commands without an additional channel is impossible. The temporal offset occurring between commands and related activities should be predictable in order to enable an estimation when changes in a physical signal occur. Our approach can only reliably predict expected changes in an SC signal with a deterministic mapping between a command and an action. It is only possible to monitor actions where the changes due to activity stand out compared to noise. Finally, our evaluation shows that the current implementation could be more robust to shadowing. The changes due to one action are indistinguishable from changes of the same action being executed by multiple

devices at once. It is also impossible to distinguish actions that cause the same changes in the SC measurements. This can be addressed by using multiple sensors that, for example, enable localization as presented by [57].

9.8 Related Work

Our approach performs cyber-physical AD. It combines information from multiple layers, particularly related is SC based AD.

AD in CPSs: Luo et al. [219] provide an overview of ML based AD methods for Cyber-Physical Systems (CPSs). The input to the presented modeling approaches are features from various observations that can be made from within a CPS. Some approaches, such as [220], [221], directly use observations related to the physical process to create the reference model. Wang et al. [222] use accumulated process logs to derive models for an observed IoT system. Some approaches use the observed communication for finding patterns [223], [224]. Specification-based anomaly detection processes existing documentation to create a model. Caselli et al. [25], as well as Sekar et al. [225] present how documentation can be leveraged to build reference models. While those methods model the behavior system, they disregard the relationship between physical behavior and the cyber layer.

Side-Channel Based AD and Verification: A variety of publications propose SC monitoring for AD. Most approaches verify the control flow integrity of code execution on devices. These works show that SCs are suitable for monitoring device activity. In contrast to monitoring code execution, our approach validates correct physical activity. Commonly used side channels are EM and power consumption. Han et al. [73] and Aubel et al. [167] measure EM emission to validate the correct control flow of programs. Boggs et al. [226] and Nazari et al. [227] use an EM SC to detect malicious code execution. Han et al. [191] and Lui et al. [165] present power intake as an SC to verify the correct execution of software. Dupuis et al. [228] review the effectiveness of power SCs for detecting hardware trojan horses. Bolboacă et al. [229] and Formby et al. [49] measure the time an ICS device requires to execute commands to detect anomalous behavior of devices. In this work, the authors exploit the deterministic behavior of devices and used protocols to implement a timing side channel for AD. Our work does not monitor the control flow of software on devices but the behavior when executing a physical action. Closest to our work is [38], in which Birnbach et al. combine measurements from multiple sensors to verify the execution of a reported physical event. The work demonstrates that physical events cause measurable changes on multiple side channels. The authors combine multiple SCs to detect spoofed events. Our work exceeds those capabilities by validating specific physical activity. We consider simultaneous activities of multiple devices and also aim to detect masquerading of actions.

Multi-Layer AD: Several works show the advantages of using information from multiple layers for AD. However, no approach aims to validate whether a monitored device correctly executes instructed activity on the physical layer. Sahu et al. [8] fuse cyber and physical information to identify false command and measurement injections in power systems. Yang et al. [195]

analyze if the power consumption and perceived state of ICS devices match to identify attacks. Carcano et al. [215] model MODBUS traffic and command values to track if a command chain puts a system into a critical state. Close to our work is [187] by Choi et al., which creates a physical and a cyber model of robotic vehicles. Mismatches between those models indicate abnormal behavior. Our approach differs by using network control traffic to dynamically predict expected changes in the system and compare them with real measurements for AD.

Sound Classification Analyzing audio for AD is closely related to sound classification. The biggest challenges in this field are the filtering and enrichment of recordings. An overview of the properties of sound fingerprinting is given in [230]. A large portion of current research focuses on music fingerprinting [231], in which a song should be identified from a short sample [232]. The field of environmental sound classification has not received much attention until 2014. Chachada and Kuo [233] give an overview about challenges for environmental sound classification. Piczak [234] argues the feasibility of using ML for environmental sound recognition. Salamon and Bello [235] successfully use ML to classify events. While we also classify audio into normal and anomalous classes, the sound database for the comparison is fixed in the mentioned works. This differs from our work, in which we dynamically generate the reference.

Audio-Based AD: Contrary to our approach, most works using acoustic SCs target isolated devices and do not consider simultaneous device activity. Audio is used in various SC attacks [179], [236]. Applications that use audio for monitoring are often related to manufacturing. Al Faruque et al. [237] and Hojjati et al. [238] reconstruct a product's manufacturing processes and shape from audio recordings in a factory. Müller et al. [200] present a method for AD on audio signals via image recognition. Using image recognition on an audio spectrogram, the authors identify anomalous device behavior in industrial systems. Bayens et al. [74] and Chhetri et al. [239] use an acoustic SC and control signals to detect attacks in additive manufacturing. These compare printing instruction code for 3D printers and audio of the printing process to detect attacks or identify malicious fill patterns. By combining measurements with network traffic, our approach is less intrusive than analyzing local code execution. Further, it enables the consideration of the activity of multiple devices at once in one blended signal.

AD in DCs: Most AD approaches for DCs rely on network traffic monitoring or collect information on end hosts. Garg et al. [59] present an ML-based approach to detect anomalies from DC network traffic. The method addresses complex network topologies, such as DC networks, but does not consider physical device behavior. Borghesi et al. [78] monitor the power consumption of DC devices. The authors train an autoencoder to learn the relationship between physical and software properties during operation in an IPMI-managed DC to detect anomalous power consumption states. In Chapter 7, we analyzed the spectrogram of a DC soundscape to identify device activity and error codes without access to technologies such as IPMI. Only a few ADS in DCs use information from multiple layers. Hernandez et al.[192] propose to combine information from the Simple Network Management Protocol (SNMP), motherboard sensors, and external power distribution units to detect malware on single devices. The authors highlight the requirement of high temporal and spatial resolution for the investigated sensors to detect malware reliably and the potential requirement for custom-built monitoring software. Our approach

exceeds these proposals by combining information from multiple layers and actively predicting changes due to control traffic instead of relying on polled data.

9.9 Conclusion

This chapter presents a novel approach to modular cyber-physical AD. We identify anomalies by setting control traffic into the context of physical changes. Our method leverages control traffic on the network to validate device activity in a composite SC signal. By utilizing control traffic, we predict changes in composite signals. After introducing our concept in Section 9.3, we applied it to IPMI and audio as SC in Section 9.5.

The proposed method is non-intrusive since it does not interact directly with components after generating a reference model. During AD, it allows a dynamic rearrangement of sub-modules of references on demand. Using information of multiple layers for AD makes it harder for attackers to evade detection. To remain undetected, they need to manipulate two information sources. Our evaluation (Section 9.6) shows that our system is robust to noise and minor temporal inaccuracies in the prediction. In synthetic experiments, our method achieves an accuracy of 98.62 %. Our experiments in a real setting confirm the synthetic results.

Our contribution opens up new possibilities for AD. It allows using SC-based AD in systems while multiple devices operate in proximity. It also enables using novel SCs for AD that simultaneously contain information from multiple devices.

9.10 Context to Multi-Layer Relationships

This chapter introduces an ADS that monitors the relationship between events on the network and cyber-physical layers of our logical architecture (Section 2.2.1). Monitoring this relationship makes it possible to identify anomalous states in which the events of those layers mismatch, e. g. due to attacks or faults. An example of such a state is a device confirming the execution of an instructed action via the network without actually performing it.

Our ADS utilizes the model that captures the relationship between network and cyber-physical layer events described in Chapter 8. It uses the model to predict events on the cyber-physical layer and compares the prediction with the actual observations. By setting the events from the cyber-physical and network layer into context, the ADS has advanced anomaly capabilities. An ADS that only processes network layer information cannot validate if a remote-controlled device behaves as reported. Similarly, an ADS that only monitors the cyber-physical layer can process sensor measurements but cannot validate if changes align with reported activity to remote management. The proposed ADS addresses these limitations by contextualizing events from multiple layers via their relationship.

We show the functionality of our ADS to detect attacks that disrupt this relationship, “event spoofing” or “event masquerading”. Using our taxonomy (Chapter 4), those attacks target the cyber-physical layer. In “event spoofing” attacks, adversaries report events to the monitoring from devices, affecting the NET.AUTH, NET.ACC and NET.INT security goals. On the other

hand, "event masquerading" attacks hide device activity from the monitoring, affecting the CP.INT, CP.ACC, NET.INT security goals. By monitoring the typical relationship between cyber-physical and network layer events, our ADS can identify the mismatch on both layers and detect the anomalies resulting from these attacks.

This highlights how monitoring the relationship between events of multiple layers enhances AD capabilities.

9.11 Key Results

In this chapter, we demonstrated that it is possible to perform cyber-physical AD using the model established in Chapter 8. The answer RQ3.3 *How can we use network control traffic to validate physical device activity in composite signals?* is outlined in the following. We can validate individual device activity by constructing a reference signal that contains expected environmental changes based on previously observed control traffic. By comparing changes in the constructed signal to real measurements, it is possible to validate whether devices altered the environment as expected.

By considering all control traffic, we can construct a signal that contains the activity of multiple simultaneously operating devices. The dynamic rearrangement sets the presented method apart from existing SC-monitoring approaches, which use fixed pre-recorded traces as a reference for the comparison.

The dynamically constructed signal can be used as a reference to identify anomalies in which the physical behavior of devices is different from the reported one in composite measurements. This exceeds the capabilities of typical SC-based work, which mostly considers isolated devices in their analysis. Combining independent sources allows the detection of covert attacks as a new class of attacks, such as Stuxnet. In those attacks, attackers intentionally manipulate the input and output of devices to mislead the monitoring, which only uses information from within a system.

The leading causes for differences between the real and constructed signals are noise and temporal inaccuracies in the change prediction. Based on these effects, we identify that ML is most suitable for this comparison due to its resilience to such differences.

We implemented our approach for IPMI control traffic and audio SCs. Our evaluation in synthetic experiments shows that our approach can successfully detect discrepancies between reported activity and physical changes. The evaluation in a real-world DC environment confirm the synthetic results, however, with less accuracy depending on the anomaly type.

Chapter 10

Conclusion

This chapter summarizes our insights for making distributed systems more secure by monitoring multi-layer relationships. It answers the Research Questions (RQs) stated Chapter 1. Throughout the thesis, we focused on two multi-layer relationships. The first relationship is the link between applications and network communication, the second relationship is between control traffic and physical device activity. Based on our insights, we proposed two novel methods for Anomaly Detection (AD), which set network events into the context of events on other layers to spot unusual activity. The first method profiles the typical network usage of applications, allowing the identification of abnormal communication behavior on a process level. The second method sets control traffic and environmental changes into the context of each other, allowing to reason about the physical activity in composite measurements. We evaluated the methods in real-world scenarios, showing their capabilities to detect attacks in which the network activity does not match the behavior on other layers. The presented methods introduce novel ways to reliably detect and classify anomalies in networked systems and set the foundation for new research in the area of anomaly detection.

10.1 Key Results

In this section, we present the key results of the thesis. We present those results by answering the initially stated Research Questions (RQs) from Chapter 1. Since all sub-RQs aim at answering the overarching RQ1, we provide an answer to this question last.

RQ1.1: What distinguishes an industrial network from a traditional network? We analyzed typical components, characteristics, and protocols of industrial, Internet of Things (IoT) and Data Center (DC) networks in Chapter 2. It is common that such networks have a centralized, hierarchical topology with a central device monitoring and management. The networks are typically static, with a most traffic being machine-to-machine (M2M) traffic. Devices are remote managed via networked control, in which central control units send instructions to devices, which then change their state. This leads to environments with low human interaction, often following

predefined procedures. Devices in industrial and IoT networks often have the primary purpose to alter the environment and manage a physical process. While DC devices do not have the primary purpose of physical device activity, they can also execute actions that cause physical changes in the environment. This is in contrast to traditional networks. The cause of device activity in such environments is human activity. This can lead to dynamic changes and exhibit diurnal patterns in observed network traffic. In addition, devices constantly join and leave the network depending on user movement. This makes such traditional networks less predictable than the networks we studied in this thesis.

The networked control in industrial environments is an essential building block throughout the thesis, as it is one of the main causes of the investigated relationships. Part II analyzes how applications use the network depending on their state and how this causes network flows. Part III uses the networked control to predict environmental changes.

RQ1.2: Which relationships and types of dependencies exist in networks? We investigated and analyzed multi-layer relationships in Chapter 2 and how attacks on systems affect their security on multiple layers in Chapter 4. We then identified three types of relationships, two of which span multiple layers. The first relation is the application-network relationship. This relationship is caused by dependencies on remote services by other applications, causing network flows to other hosts for fetching and sending data. In industrial environments, application-network relationships are often fixed due to static setups and continuous management efforts. This leads to predictable behavior of applications and their network usage. The second relationship type occurs within the network due to the used protocol stack. Since those relationships do not span across multiple layers, we do not analyze them in detail in this thesis. The third type of relationship is between control traffic and physical changes in the environment. We consider network-controlled scenarios in which devices receive instructions over the network and execute physical activity. The control traffic is related to the timing and type of the physical change that independent sensors can measure. We investigate methods to identify and monitor those relationships for AD to set activity on different layers into the context of each other. Part II of the thesis focused on the application-network relationship. We analyzed the relationship between control traffic and physical changes in Part III of the thesis.

After analyzing the application environment and underlying scenario, Part II of the thesis investigated the relationship between applications and network activity.

RQ2.1: How can we reliably attribute network activity to applications? A prerequisite to reasoning how applications use the network is associating packets and processes reliably. However, since Operating Systems (OSs) typically provide networking functionality as a service to running applications, it is difficult to associate network traffic from a Network Interface Card (NIC) to a process. We analyzed different approaches to achieve this task and identified that this association with existing solutions requires a trade-off between data completeness and Central Processing Unit (CPU) overhead. Based on our analysis, we presented a new framework based on the extended Berkeley Packet Filter (eBPF). By executing additional code when the kernel executes relevant functions, it is possible to reliably associate packets and frames emitted by a host without significant additional load. The proposed approach has advantages over exist-

ing methods, which rely on polling the OS or creating log entries based on specific syscalls, causing significant CPU overhead. Our framework can associate traffic to packets at speeds up to 100 Mbit/s. This allows the collection of detailed information on how applications use the network.

The proposed framework can also be used for the data collection for other methods such as traffic causality graphs [11], thus increasing their insights. We use our framework in Chapter 6 to build network profiles of applications to capture application behavior.

RQ2.2: How can we determine typical application behavior and identify deviations? We identified that application behavior on the network is defined by a multitude of aspects. On the network level, relevant characteristics range from used protocols and typical flow characteristics to common flow sequences and periodic behavior. Locally, applications rely on services provided by the OS, which are associated with applications. For automated applications, it is possible to extract this behavior by combining multiple techniques that allow extracting and formalizing the individual characteristics and setting them into context. We created a framework to build such network application profiles. Network application profiles are human interpretable constructs that capture and model typical application behavior. We apply network application profiles for AD using the previously mentioned strategy to collect data that associates packets and processes. Our evaluation shows that network profiles can accurately capture the typical behavior of processes and identify deviations. The proposed approach has advantages over existing Anomaly Detection Systems (ADSs) since it allows us to attribute malicious flows to applications. In addition, in contrast to other Machine Learning (ML) based approaches, which often only classify into anomalous and expected behavior, the network application profiles provide insights into how the communication behavior of an application differs from the expectation.

Part III of the thesis focused on the relationship between control traffic and physical device activity.

RQ3.1: How can we characterize and identify physical device activity in composite signals? We can define device activity in composite signals by the changes the activity induces on it. This activity typically creates unique traces that can be identified in physical measurements, even if other devices operate in the same space and influence the signal. In particular, if devices in proximity create a constant noise level, it is possible to identify dynamic changes due to individual device activity. We applied our method to acoustic channels in a DC environment. Our experiments show that we can identify typical device activity such as power-off and power-on operations, and error codes emitted by buzzers without the help of advanced standards such as Intelligent Platform Management Interface (IPMI). Despite the influence of multiple devices on the recorded audio signal, we could identify individual device activity. Based on our research, we developed a method to identify time frames of device activity and to characterize it in composite signals. This is a first step to increase the application domain of Side Channel (SC) monitoring methods to make them suitable for the surveillance of non-isolated devices. We use the resulting knowledge about possible device activity to validate device behavior in DC environments where multiple devices concurrently influence the environment.

RQ3.2: What is the relation between network control traffic and physical device activity? The control traffic in Cyber-Physical Systems (CPSs), Industrial Control Systems (ICSs), and DCs can be used as a strong indicator for future device behavior. Based on this relationship, predicting changes in isolated and shared environments is possible. Control traffic specifies which device and action should be executed. This allows us to predict the changes and unique traces the previously introduced method (RQ3.1) uncovered. Based on this relationship, we introduced a cyber-physical model that predicts changes in the environment based on networked control traffic to monitored devices. This model works for shared environments in which the effects of device activity add up or counterbalance. The model's accuracy depends on the system's guarantees, its determinism, and how noise can be filtered. In highly deterministic systems, such as ICS, our model can accurately predict when a change in the environment occurs. In less deterministic environments, such as DCs, the control traffic only provides an estimate within a time frame when a change happens. This has effects on post-processing applications that provide services based on this model. However, the proposed model, incorporating control traffic and physical changes, enhances the reasoning capabilities of operators about the behavior of a system. This includes CPSs, but also DCs in which devices do not have the primary purpose of physically changing the environment. We used this model in Chapter 9 to build a cyber-physical ADS.

RQ3.3: How can we use network control traffic to validate physical device activity in composite signals? Using the model proposed Chapter 8 allows to predict physical changes based on control traffic. In a way, this model poses as an incomplete digital twin, reflecting the changes in the real environment. In case the model can predict changes, e. g. because all components in the monitored system are remote-controlled and there are no other influences, the model contains all expected changes, even in composite measurements. Comparing the changes in the model with the measured changes in the real world enables the validation of individual device activity. Based on this insight, we propose a new cyber-physical ADS. The proposed ADS detects anomalies in which device activity differs from network-instructed or reported activity. By combining independent sources of information, the ADS can detect masqueraded activity and spoofed, i. e. uninstructed activity that otherwise would not show up in monitoring that only observes one layer. We evaluated our ADS in a real-world DC environment, showing that even in environments without low guarantees, it is possible to validate device activity. Depending on the system's given guarantees, it is possible to choose different validation mechanisms, such as audio fingerprinting. Our analysis shows that ML is most robust to noise in the measured signal and temporal inaccuracies of the model for AD. This shows that the relationship captured by the introduced model (Chapter 8) allows extended reasoning about a system's behavior and to identify anomalies. In particular it allows to identify anomalies which lead to mismatches between reported and real physical behavior, e. g. due to attacks like Stuxnet.

We conclude and answer the overarching *RQ1: How can we improve the security in distributed systems by identifying and monitoring multi-layer relationships?* We can identify relationships by first analyzing activity on individual layers and set them into the context of events that occur on other layers. Events can be induced by application activity, causing communication with remote services or control traffic that implies future physical changes. Based on this relationship, it is possible to build models that contextualize events across multiple layers by correlation.

Models that capture the multi-layer relationships in systems allow enhanced reasoning about the activity and state of individual devices, even in composite physical measurements. This thesis presents two models: (1) The first model is an application network profile that associates network activity and application behavior. (2) Second, we proposed a cyber-physical model that maps control traffic to physical changes. We apply those models to perform AD and identify unexpected activity. By combining information from multiple layers, the proposed approaches exceed the AD capabilities of existing ADSs which only operate on a single layer. In particular, they allow the detection of attacks in which attackers mimic correct behavior on a single layer to mislead monitoring systems. The new detection capabilities help make network controlled systems more secure, increasing the difficulty for adversaries to hide their activity. Once operators identify anomalous behavior, they can take action to contain and eradicate the identified threats.

10.2 Future Work

The results of this thesis open up new venues for future research. We presented the value of identifying and using multi-layer relationships to improve the security of distributed systems. New research could explore the underlying causes of the identified relationships. Examples of causes, e. g. in CPSs or the IoT could be relationships induced by sequences in physical process management. Knowledge about the causes can allow automated root cause analysis in case of system outages. The ability to automatically identify root causes allows the development of mechanisms to support the recovery of a system after an outage.

The proposed matcher in Chapter 5 uses eBPF to associate egress traffic to processes. While the method already shows advantages over existing systems, future research could extend the matcher to associate ingress traffic to processes via eBPF. Using eBPF for associate ingress traffic could further reduce the overhead of the matcher imposed on the monitored system by removing the need for the packet buffer. As a result, this could reduce the overhead of methods that use the matcher in the data collection while improving their accuracy due to improved data quality. Future research could run experiments to further quantify the advantages of eBPF based data collection over other approaches. Highlighting the advantages motivates further adoption of the approach into other post-processing applications, increasing their efficiency. The matcher opens up further avenues for new ADS-schemes, such as a decentralized hybrid ADS as proposed by [67]. A decentralized information collection can help to enhance the capabilities of ADS by providing an extended view of a system's state. The benefit can be a more accurate classification of a system's state than the one of an ADS that uses fewer vantage points for information collection, leading to better AD capabilities.

Network application profiles already pose as a starting point to formalize application behavior. Future research is needed to extend the profiles to capture more information and set it into context with each other. This enables the creation of a more accurate description of application behavior, enhancing the reasoning capabilities of network operators. The resulting insights can help optimize capacity planning or predict the effects of service outages.

The proposed cyber-physical model in Part III of the thesis shows that control traffic allows to predict changes in physical environments. In our application, we modeled the changes in a DC environment. While this environment has desirable properties for modeling physical changes, it does not provide any guarantees to its users. Therefore, future research could analyze how accurate the proposed models are in environments with stricter guarantees. This would enable the exploration of different comparison methods in the validation phase, potentially leading to better accuracy and precision of the ADS. Further, the concept could be expanded to other SCs or combinations of multiple SC, enabling similar benefits as reported by other approaches [38], [57], [79]. This would further extend the detection capabilities of SC monitoring approaches.

Finally, this thesis considers the relationship between applications and communication, as well as the relationship between control traffic and physical behavior. Thus, future endeavors can explore creating a holistic system model covering all layers at once. By enabling the creation of models that describe the behavior and changes in environments and applications running within them, operators can simulate the effect of attacks and faults on a system. The models can help to generate realistic labelled data sets for creating ML based ADSs that reflects anomalous system behavior, without interrupting real world systems. This allows to develop advanced AD strategies that have a contextual awareness of changes in a monitored environment.

Chapter A

Appendix

A.1 List of Acronyms

AC Air Condition
AD Anomaly Detection
ADS Anomaly Detection System
ARM Association Rule Mining
ARP Address Resolution Protocol
BPF Berkeley Packet Filter
C2 Command and Control
CAPEC Common Attack Pattern Enumeration and Classification
CDN Content Delivery Network
CNN Convolutional Neural Network
CPS Cyber-Physical System
CPU Central Processing Unit
CVE Common Vulnerability and Expsoures
CWE Common Weakness Enumeration
DC Data Center
DCIM Data Center Infrastructure Management
DHCP Dynamic Host Configuration Protocol
DNP3 Distributed Network Protocol 3
DNS Domain Name System
DTW Dynamic Time Warping
eBPF extended Berkeley Packet Filter
EFC Energy-based Flow Classifier
EM Electromagnetic
ERM Episode Rule Mining
FIM Frequent Itemset Mining

FN False Negative
FP False Positive
FQDN Fully Qualified Domain Name
GIL Global Interpreter Lock
HMI Human Machine Interface
HTTP HyperText Transport Protocol
HTTPS HyperText Transport Protocol Secure
IANA Internet Assigned Numbers Authority
ICMP Internet Control Message Protocol
ICS Industrial Control System
IDS Intrusion Detection System
IIoT Industrial Internet of Things
IoT Internet of Things
IT Information Technology
IP Internet Protocol
IPMI Intelligent Platform Management Interface
M2M machine-to-machine
ML Machine Learning
MQTT Message Queuing Telemetry Transport
NIC Network Interface Card
NIDS Network Intrusion Detection System
NN Neural Network
NTP Network Time Protocol
OS Operating System
PID Process-ID
PLC Programmable Logic Controller
PSK Pre-Shared Key
RAKP Remote Authenticated Key-Exchange Protocol
RFID Radio Frequency Identification
RMCP Remote Management Control Protocol
RMS Root Mean Square
RQ Research Question
SC Side Channel
SCADA Supervisory Control And Data Acquisition
SF Spectral Flatness
SRM Sequential Rule Mining
STFT Short Term Fourier Transform
TCP Transmission Control Protocol
TGID Thread Group ID
TN True Negative
TP True Positive
UDP User Datagram Protocol

A.2 List of Figures

A.3 List of Tables

Bibliography

Work with author's contribution

[14] L. Wüstrich, M. Pahl, and S. Liebald, "Towards an Extensible IoT Security Taxonomy", in *IEEE Symposium on Computers and Communications, ISCC 2020, Rennes, France, July 7-10, 2020*, IEEE, 2020, pp. 1–6. DOI: 10.1109/ISCC50000.2020.9219584.

[15] L. Wüstrich, M. Schacherbauer, M. Budeus, D. F. von Künßberg, S. Gallenmüller, M. Pahl, and G. Carle, "Network Profiles for Detecting Application-Characteristic Behavior Using Linux eBPF", in *Proceedings of the 1st Workshop on eBPF and Kernel Extensions, eBPF 2023, New York, NY, USA, 10 September 2023*, ACM, 2023, pp. 8–14. DOI: 10.1145/3609021.3609294.

[16] L. Wüstrich, S. Gallenmüller, M. Pahl, and G. Carle, "AC/DCIM: Acoustic Channels for Data Center Infrastructure Monitoring", in *2022 IEEE/IFIP Network Operations and Management Symposium, NOMS 2022, Budapest, Hungary, April 25-29, 2022*, IEEE, 2022, pp. 1–5. DOI: 10.1109/NOMS54207.2022.9789839.

[17] L. Wüstrich, L. Schröder, and M. Pahl, "Cyber-Physical Anomaly Detection for ICS", in *17th IFIP/IEEE International Symposium on Integrated Network Management, IM 2021, Bordeaux, France, May 17-21, 2021*, T. Ahmed, O. Festor, Y. Ghamri-Doudane, J. Kang, A. E. S. Filho, A. Lahmadi, and E. R. M. Madeira, Eds., IEEE, 2021, pp. 950–955.

[18] L. Wüstrich, S. Gallenmüller, S. M. Günther, G. Carle, and M. Pahl, "Shells Bells: Cyber-Physical Anomaly Detection in Data Centers", in *NOMS 2024 IEEE Network Operations and Management Symposium, Seoul, Republic of Korea, May 6-10, 2024*, IEEE, 2024, pp. 1–10. DOI: 10.1109/NOMS59830.2024.10575124.

[19] M. Barbeau, J. García-Alfaro, C. Lübben, M. Pahl, and L. Wüstrich, "Resilience via Blackbox Self-Piloting Plants", in *Proceedings of the 29th Computer & Electronics Security Application Rendezvous co-located with the 7th European Cyber Week (ECW 2022), Rennes, France, November 15-16, 2022*, G. le Guernic, Ed., ser. CEUR Workshop Proceedings, vol. 3329, CEUR-WS.org, 2022, pp. 35–46.

[20] A. Piccoli, M. Pahl, and L. Wüstrich, "Group Key Management in Constrained IoT Settings", in *IEEE Symposium on Computers and Communications, ISCC 2020, Rennes, France, July 7-10, 2020*, IEEE, 2020, pp. 1–6. DOI: 10.1109/ISCC50000.2020.9219619.

[21] S. Paiho, J. Kiljander, R. Sarala, H. Siikavirta, O. Kilkki, A. Bajpai, M. Duchon, M.-O. Pahl, L. Wüstrich, C. Lübben, *et al.*, "Towards Cross-Commodity Energy-Sharing Communities – A Review of the Market, Regulatory, and Technical Situation", *Renewable and Sustainable Energy Reviews*, vol. 151, p. 111 568, 2021.

References

[1] E. D. Knapp, *Industrial Network Security: Securing Critical Infrastructure Networks for Smart Grid, SCADA, and other Industrial Control Systems*, Third Edition. Syngress, 2024.

[2] S. Birnbach, S. Eberz, and I. Martinovic, "Haunted House: Physical Smart Home Event Verification in the Presence of Compromised Sensors", *ACM Transactions on Internet of Things*, vol. 3, no. 3, 18:1–18:28, 2022. DOI: 10.1145/3506859.

[3] L. Popa, B.-G. Chun, I. Stoica, J. Chandrashekar, and N. Taft, "Macroscope: End-Point Approach to Networked Application Dependency Discovery", in *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, J. Liebeherr, G. Ventre, E. W. Biersack, and S. Keshav, Eds., ser. CoNEXT '09, Rome, Italy: Association for Computing Machinery, 2009, 229–240, ISBN: 9781605586366. DOI: 10.1145/1658939.1658966.

[4] J. Piet, D. Nwoji, and V. Paxson, "GGFAST: Automating Generation of Flexible Network Traffic Classifiers", in *Proceedings of the ACM SIGCOMM 2023 Conference, ACM SIGCOMM 2023, New York, NY, USA, 10-14 September 2023*, H. Schulzrinne, V. Misra, E. Kohler, and D. A. Maltz, Eds., ACM, 2023, pp. 850–866. DOI: 10.1145/3603269.3604840.

[5] A. Bonkoski, R. Bielawski, and J. A. Halderman, "Illuminating the Security Issues Surrounding Lights-Out Server Management", in *7th USENIX Workshop on Offensive Technologies, WOOT '13, Washington, D.C., USA, August 13, 2013*, J. Oberheide and W. K. Robertson, Eds., USENIX Association, 2013.

[6] - *IPMI - Intelligent Platform Management Interface Specification Second Generation*, Rev. 1.1, Intel, Hewlett-Packard, NEC, Dell, Oct. 2013.

[7] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild", in *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010, Melbourne, Australia - November 1-3, 2010*, M. Allman, Ed., ACM, 2010, pp. 267–280. DOI: 10.1145/1879141.1879175.

[8] A. Sahu, Z. Mao, P. Wlazlo, H. Huang, K. R. Davis, A. E. Goulart, and S. A. Zonouz, "Multi-Source Multi-Domain Data Fusion for Cyberattack Detection in Power Systems", *IEEE Access*, vol. 9, pp. 119 118–119 138, 2021. DOI: 10.1109/ACCESS.2021.3106873.

[9] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel Traffic Classification in the Dark", in *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, R. Guérin, R. Govindan, and G. Minshall, Eds., ACM, 2005, pp. 229–240. DOI: 10.1145/1080091.1080119.

[10] R. Langner, "Stuxnet: Dissecting a Cyberwarfare Weapon", *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011. DOI: 10.1109/MSP.2011.67.

[11] H. Asai, K. Fukuda, and H. Esaki, "Traffic Causality Graphs: Profiling Network Applications Through Temporal and Spatial Causality of Flows", in *2011 23rd International Teletraffic Congress (ITC)*, Å. Arvidsson, G. de Veciana, S. H. Low, C. R. Kalmanek, and D. Medhi, Eds., IEEE, IEEE, 2011, pp. 95–102.

[12] Linux manual page, *auditd(8)*, https://linux.die.net/man/8/auditd, Accessed: 2024-07-08.

[13] L. Zeng, Y. Xiao, and H. Chen, "Auditing overhead, auditing adaptation, and benchmark evaluation in linux", *Security and Communication Networks*, vol. 8, no. 18, pp. 3523–3534, 2015. DOI: 10.1002/SEC.1277.

[22] R. R. R. Barbosa, R. Sadre, and A. Pras, "Difficulties in Modeling SCADA Traffic: A Comparative Analysis", in *Passive and Active Measurement - 13th International Conference, PAM 2012, Vienna, Austria, March 12-14th, 2012. Proceedings*, N. Taft and F. Ricciato, Eds., ser. Lecture Notes in Computer Science, vol. 7192, Springer, 2012, pp. 126–135. DOI: 10.1007/978-3-642-28537-0_13.

[23] S.-H. Leitner and W. Mahnke, "OPC UA–service-oriented architecture for industrial applications", *ABB Corporate Research Center*, vol. 48, pp. 61–66, 2006.

[24] M. Levy and J. O. Hallstrom, "A New Approach to Data Center Infrastructure Monitoring and Management (DCIMM)", in *IEEE 7th Annual Computing and Communication Workshop and Conference, CCWC 2017, Las Vegas, NV, USA, January 9-11, 2017*, IEEE, 2017, pp. 1–6. DOI: 10.1109/CCWC.2017.7868412.

[25] M. Caselli, E. Zambon, J. Amann, R. Sommer, and F. Kargl, "Specification Mining for Intrusion Detection in Networked Control Systems", in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, T. Holz and S. Savage, Eds., USENIX Association, 2016, pp. 791–806.

[26] K. Stouffer, J. Falco, and K. Scarfone, "Guide to Industrial Control Systems (ICS) Security", *NIST Special Publication*, vol. 800, no. 82, pp. 16–16, 2011.

[27] K.-D. Kim and P. R. Kumar, "The Importance, Design and Implementation of a Middleware for Networked Control Systems", in *Networked Control Systems*, A. Bemporad, M. Heemels, and M. Johansson, Eds. London: Springer London, 2010, pp. 1–29, ISBN: 978-0-85729-033-5. DOI: 10.1007/978-0-85729-033-5_1.

[28] I. Burgetova, P. Matousek, and O. Rysavý, "Anomaly Detection of ICS Communication Using Statistical Models", in *17th International Conference on Network and Service Management, CNSM 2021, Izmir, Turkey, October 25-29, 2021*, P. Chemouil, M. Ulema, S. Clayman, M. Sayit, C. Çetinkaya, and S. Secci, Eds., IEEE, 2021, pp. 166–172. DOI: 10.23919/CNSM52442.2021.9615510.

[29] *ISO Transport Service on top of the TCP Version: 3*, RFC 1006, 1987. [Online]. Available: https://www.rfc-editor.org/info/rfc1006.

[30] A. Swales, "Open MODBUS/TCP Specification", *Schneider Electric*, vol. 29, pp. 3–19, 1999.

[31] "IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3)", *IEEE Std 1815-2012 (Revision of IEEE Std 1815-2010)*, pp. 1–821, 2012. DOI: 10.1109/IEEESTD.2012.6327578.

[32] O. P. Communication, *Unified Architecture*, https://opcfoundation.org/about/opc-technologies/opc-ua/, Accessed: 2024-07-09.

[33] N. O. Silva, M. Rosso, E. Zambon, J. den Hartog, and A. A. Cárdenas, "From Power to Water: Dissecting SCADA Networks Across Different Critical Infrastructures", in *Passive and Active Measurement - 25th International Conference, PAM 2024, Virtual Event,*

March 11-13, 2024, Proceedings, Part I, P. Richter, V. Bajpai, and E. Carisimo, Eds., ser. Lecture Notes in Computer Science, vol. 14537, Springer, 2024, pp. 3–31. DOI: 10.1007/978-3-031-56249-5_1.

[34] R. R. R. Barbosa, R. Sadre, and A. Pras, "A First Look into SCADA Network Traffic", in *2012 IEEE Network Operations and Management Symposium*, F. D. Turck, L. P. Gaspary, and D. Medhi, Eds., IEEE, IEEE, 2012, pp. 518–521. DOI: 10.1109/NOMS.2012.6211945.

[35] A. Kleinmann and A. Wool, "Accurate Modeling of The Siemens S7 SCADA Protocol For Intrusion Detection And Digital Forensic", *J. Digit. Forensics Secur. Law*, vol. 9, no. 2, pp. 37–50, 2014. DOI: 10.15394/JDFSL.2014.1169.

[36] R. R. R. Barbosa, R. Sadre, and A. Pras, "Flow Whitelisting in SCADA Networks", *International Journal of Critical Infrastructure Protection*, vol. 6, no. 3-4, pp. 150–158, 2013. DOI: 10.1016/J.IJCIP.2013.08.003.

[37] A. Lemay, J. Rochon, and J. M. Fernandez, "A Practical Flow White List Approach for SCADA Systems", in *4th International Symposium for ICS & SCADA Cyber Security Research 2016, ICS-CSR 2016, 23 - 25 August 2016, Queen's Belfast University, UK*, T. Brandstetter and H. Janicke, Eds., ser. Workshops in Computing, BCS, 2016. [Online]. Available: http://ewic.bcs.org/content/ConWebDoc/56475.

[38] S. Birnbach, S. Eberz, and I. Martinovic, "Peeves: Physical Event Verification in Smart Homes", in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds., ACM, 2019, pp. 1455–1467. DOI: 10.1145/3319535.3354254.

[39] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, Opportunities, and Directions", *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018. DOI: 10.1109/TII.2018.2852491.

[40] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey", *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010. DOI: 10.1016/J.COMNET.2010.05.010.

[42] M. C. Bor, J. E. Vidler, and U. Roedig, "Lora for the internet of things", in *Proceedings of the International Conference on Embedded Wireless Systems and Networks, EWSN 2016, Graz, Austria, 15-17 February 2016*, K. Römer, K. Langendoen, and T. Voigt, Eds., Junction Publishing, Canada / ACM, 2016, pp. 361–366. [Online]. Available: http://dl.acm.org/citation.cfm?id=2893802.

[43] *ZigBee Specification*, ZigBee Document 05-3474-21, ZigBee Alliance, Aug. 2015.

[44] checkmk, *checkmk - Infrastructure and Application Monitoring*, https://checkmk.com/, Accessed: 2024-07-02.

[45] Icinga, *Icinga*, https://icinga.com/, Accessed: 2024-07-02.

[46] I. Andrea, C. Chrysostomou, and G. C. Hadjichristofi, "Internet of Things: Security Vulnerabilities and Challenges", in *2015 IEEE Symposium on Computers and Communication, ISCC 2015, Larnaca, Cyprus, July 6-9, 2015*, IEEE Computer Society, 2015, pp. 180–187. DOI: 10.1109/ISCC.2015.7405513.

[47] S. Rizvi, A. Kurtz, J. R. Pfeffer, and M. Rizvi, "Securing the Internet of Things (IoT): A Security Taxonomy for IoT", in *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 12th IEEE International Conference*

On Big Data Science And Engineering, TrustCom/BigDataSE 2018, New York, NY, USA, August 1-3, 2018, IEEE, 2018, pp. 163–168. DOI: 10.1109/TRUSTCOM/BIGDATASE.2018.00034.

[48] Y. Zhang and K. Rasmussen, "Detection of Electromagnetic Interference Attacks on Sensor Systems", in *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, IEEE, 2020, pp. 203–216. DOI: 10.1109/SP40000.2020.00001.

[49] D. Formby, P. Srinivasan, A. M. Leonard, J. D. Rogers, and R. A. Beyah, "Who's in Control of Your Control System? Device Fingerprinting for Cyber-Physical Systems", in *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, The Internet Society, 2016.

[50] M. Pahl, "Distributed Smart Space Orchestration", Ph.D. dissertation, Technical University Munich, 2014, ISBN: 978-3-937201-46-7.

[51] F. P. Tso, S. Jouet, and D. P. Pezaros, "Network and Server Resource Management Strategies for Data Centre Infrastructures: A Survey", *Computer Networks*, vol. 106, pp. 209–225, 2016. DOI: 10.1016/J.COMNET.2016.07.002.

[52] E. Saitović and I. Ivanović, *Network Monitoring and Management Recommendations*, 2011.

[53] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture", in *Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Seattle, WA, USA, August 17-22, 2008*, V. Bahl, D. Wetherall, S. Savage, and I. Stoica, Eds., ACM, 2008, pp. 63–74. DOI: 10.1145/1402958.1402967.

[54] J. M. Ceron, J. J. Chromik, J. Santanna, and A. Pras, "Online Discoverability and Vulnerabilities of ICS/SCADA Devices in the Netherlands", *CoRR*, vol. abs/2011.02019, 2020. DOI: 10.48550/arXiv.2011.02019.

[55] C. Pascoe, S. Quinn, and K. Scarfone, *The NIST Cybersecurity Framework (CSF) 2.0*, EN, 2024. DOI: https://doi.org/10.6028/NIST.CSWP.29.

[56] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey", *ACM Computing Surveys*, vol. 41, no. 3, 15:1–15:58, 2009. DOI: 10.1145/1541880.1541882.

[57] M. O. Ozmen, R. Song, H. Farrukh, and Z. B. Celik, "Evasion Attacks and Defenses on Smart Home Physical Event Verification", in *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*, The Internet Society, 2023.

[58] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection", in *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berkeley/Oakland, California, USA*, IEEE Computer Society, 2010, pp. 305–316. DOI: 10.1109/SP.2010.25.

[59] S. Garg, K. Kaur, N. Kumar, G. Kaddoum, A. Y. Zomaya, and R. Ranjan, "A Hybrid Deep Learning-Based Model for Anomaly Detection in Cloud Datacenter Networks", *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 924–935, 2019. DOI: 10.1109/TNSM.2019.2927886.

[60] M. Ozay, I. Esnaola, F. T. Y. Vural, S. R. Kulkarni, and H. V. Poor, "Machine Learning Methods for Attack Detection in the Smart Grid", *IEEE Transactions Neural Networks and Learning Systems*, vol. 27, no. 8, pp. 1773–1786, 2016. DOI: 10.1109/TNNLS.2015.2404803.

[61] J. Inoue, Y. Yamagata, Y. Chen, C. M. Poskitt, and J. Sun, "Anomaly Detection for a Water Treatment System Using Unsupervised Machine Learning", in *2017 IEEE International Conference on Data Mining Workshops, ICDM Workshops 2017, New Orleans, LA, USA, November 18-21, 2017*, R. Gottumukkala, X. Ning, G. Dong, V. Raghavan, S. Aluru, G. Karypis, L. Miele, and X. Wu, Eds., IEEE Computer Society, 2017, pp. 1058–1065. DOI: 10.1109/ICDMW.2017.149.

[62] M. Kravchik and A. Shabtai, "Detecting Cyber Attacks in Industrial Control Systems Using Convolutional Neural Networks", in *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and Privacy, CPS-SPC@CCS 2018, Toronto, ON, Canada, October 19, 2018*, D. Lie and M. Mannan, Eds., ACM, 2018, pp. 72–83. DOI: 10.1145/3264888.3264896.

[63] P. Garcia-Teodoro, J. E. D. Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges", *computers & security*, vol. 28, no. 1-2, pp. 18–28, 2009. DOI: 10.1016/J.COSE.2008.08.003.

[64] D. M. W. Powers, "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation", *CoRR*, vol. abs/2010.16061, 2020. DOI: 10.48550/arXiv.2010.16061.

[65] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "NetworkProfiler: Towards Automatic Fingerprinting of Android Apps", in *Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14-19, 2013*, IEEE, 2013, pp. 809–817. DOI: 10.1109/INFCOM.2013.6566868.

[66] C. F. T. Pontes, M. M. C. de Souza, J. J. C. Gondim, M. Bishop, and M. A. Marotta, "A New Method for Flow-Based Network Intrusion Detection Using the Inverse Potts Model", *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1125–1136, 2021. DOI: 10.1109/TNSM.2021.3075503.

[67] S. Haas, R. Sommer, and M. Fischer, "Zeek-osquery: Host-Network Correlation for Advanced Monitoring and Intrusion Detection", in *ICT Systems Security and Privacy Protection: 35th IFIP TC 11 International Conference, SEC 2020, Maribor, Slovenia, September 21–23, 2020, Proceedings 35*, M. Hölbl, K. Rannenberg, and T. Welzer, Eds., ser. IFIP Advances in Information and Communication Technology, Springer, vol. 580, Springer, 2020, pp. 248–262. DOI: 10.1007/978-3-030-58201-2_17.

[68] zeek, *The Zeek Network Security Monitor*, https://zeek.org/, Accessed: 2024-08-22.

[69] zeek, *Zeek Agent*, https://github.com/zeek/zeek-agent-v2, Accessed: 2024-07-03.

[70] Cilium Hubble, *Hubble - Network, Service & Security Observability for Kubernetes using eBPF*, https://github.com/cilium/hubble, Accessed: 2024-07-03.

[71] Cilium Tetragon, *tetragon - eBPF-based Security Observability and Runtime Enforcement*, https://github.com/cilium/tetragon, Accessed: 2024-07-03.

[72] Y. Han, I. Christoudis, K. Diamantaras, S. Zonouz, and A. Petropulu, "A Survey of Side Channel Based Code Execution Monitoring Systems",

[73] Y. Han, S. Etigowni, H. Liu, S. A. Zonouz, and A. P. Petropulu, "Watch Me, but Don't Touch Me! Contactless Control Flow Monitoring via Electromagnetic Emanations", in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., ACM, 2017, pp. 1095–1108. DOI: 10.1145/3133956.3134081.

[74] C. Bayens, T. Le, L. Garcia, R. A. Beyah, M. Javanmard, and S. A. Zonouz, "See No Evil, Hear No Evil, Feel No Evil, Print No Evil? Malicious Fill Patterns Detection in Additive Manufacturing", in *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, E. Kirda and T. Ristenpart, Eds., USENIX Association, 2017, pp. 1181–1198.

[75] M. G. Rodriguez, L. E. O. Uriarte, Y. Jia, K. Yoshii, R. Ross, and P. H. Beckman, "Wireless Sensor Network for Data-Center Environmental Monitoring", in *2011 Fifth International Conference on Sensing Technology*, IEEE, 2011, pp. 533–537. DOI: 10.1109/ICSensT.2011.6137036.

[76] N. Ahuja, C. W. Rego, S. Ahuja, S. Zhou, and S. Shrivastava, "Real Time Monitoring and Availability of Server Airflow for Efficient Data Center Cooling", in *29th IEEE Semiconductor Thermal Measurement and Management Symposium*, IEEE, 2013, pp. 243–247. DOI: 10.1109/SEMI-THERM.2013.6526836.

[77] M. Dayarathna, Y. Wen, and R. Fan, "Data Center Energy Consumption Modeling: A Survey", *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 732–794, 2016. DOI: 10.1109/COMST.2015.2481183.

[78] A. Borghesi, A. Libri, L. Benini, and A. Bartolini, "Online Anomaly Detection in HPC Systems", in *IEEE International Conference on Artificial Intelligence Circuits and Systems, AICAS 2019, Hsinchu, Taiwan, March 18-20, 2019*, IEEE, 2019, pp. 229–233. DOI: 10.1109/AICAS.2019.8771527.

[79] C. Fu, Q. Zeng, and X. Du, "HAWatcher: Semantics-Aware Anomaly Detection for Appified Smart Homes", in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. D. Bailey and R. Greenstadt, Eds., USENIX Association, 2021, pp. 4223–4240.

[80] L. Garcia, F. Brasser, M. H. Cintuglu, A. Sadeghi, O. A. Mohammed, and S. A. Zonouz, "Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit", in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*, The Internet Society, 2017.

[81] C. Fung, E. Zeng, and L. Bauer, "Attributions for ML-based ICS Anomaly Detection: From Theory to Practice", in *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*, The Internet Society, 2024.

[82] M. Pahl and S. Liebald, "Information-Centric IoT Middleware Overlay: VSL", in *2019 International Conference on Networked Systems, NetSys 2019, Munich, Germany, March 18-21, 2019*, IEEE, 2019, pp. 1–8. DOI: 10.1109/NETSYS.2019.8854515.

[83] C. Eckert, *IT-Sicherheit: Konzepte-Verfahren-Protokolle*. De Gruyter Oldenbourg, 2023, ISBN: 978-3110996890.

[84] R. von Solms and J. V. Niekerk, "From Information Security to Cyber Security", *computers & security*, vol. 38, pp. 97–102, 2013. DOI: 10.1016/J.COSE.2013.04.004.

[85] T. R. Peltier, *Information Security Risk Analysis*. Taylor & Francis, 2010, ISBN: 978-0849308802.

[86] R. S. Sandhu and P. Samarati, "Access Control: Principles and Practice", *IEEE Communications Magazine*, vol. 32, no. 9, pp. 40–48, 1994. DOI: 10.1109/35.312842.

[87] I. Stellios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz, and J. López, "A Survey of IoT-Enabled Cyberattacks: Assessing Attack Paths to Critical Infrastructures and Services", *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3453–3495, 2018. DOI: 10.1109/COMST.2018.2855563.

[88] A. Teixeira, D. Pérez, H. Sandberg, and K. H. Johansson, "Attack Models and Scenarios for Networked Control Systems", in *1st International Conference on High Confidence Networked Systems (HiCoNS - at CPS Week 2012), HiCoNS '12, Beijing, China, April 17-18, 2012*, S. S. Sastry, T. Basar, S. Amin, and G. Karsai, Eds., ACM, 2012, pp. 55–64. DOI: 10.1145/2185505.2185515.

[89] R. A. Martin, "Common Weakness Enumeration", *Mitre Corporation*, 2007.

[90] The MITRE Corporation, *CVE*, https://www.cve.org/, Accessed: 2024-09-13.

[91] The MITRE Corporation, *CAPEC*, https://capec.mitre.org/, Accessed: 2024-09-13.

[92] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, "Mitre ATT&CK: Design and Philosophy", *Technical report*, 2018.

[93] A. Mitrokotsa, M. R. Rieback, and A. S. Tanenbaum, "Classification of RFID Attacks", Q. Z. Sheng, Z. Maamar, S. Zeadally, and M. Cameron, Eds., pp. 73–86, 2008.

[94] R. Mahmoud, T. Yousuf, F. A. Aloul, and I. A. Zualkernan, "Internet of Things (IoT) Security: Current Status, Challenges and Prospective Measures", in *10th International Conference for Internet Technology and Secured Transactions, ICITST 2015, London, United Kingdom, December 14-16, 2015*, IEEE, 2015, pp. 336–341. DOI: 10.1109/ICITST.2015.7412116.

[95] K. Sha, W. Wei, T. A. Yang, Z. Wang, and W. Shi, "On Security Challenges and Open Issues in Internet of Things", *Future Generation Computer Systems*, vol. 83, pp. 326–337, 2018. DOI: 10.1016/J.FUTURE.2018.01.059.

[97] M. Leo, F. Battisti, M. Carli, and A. Neri, "A Federated Architecture Approach for Internet of Things Security", in *Euro Med Telco Conference, EMTC 2014, Naples, Italy, November 12-15, 2014*, IEEE, 2014, pp. 1–5. DOI: 10.1109/EMTC.2014.6996632.

[98] E. Ronen and A. Shamir, "Extended Functionality Attacks on IoT Devices: The Case of Smart Lights", in *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, IEEE, 2016, pp. 3–12. DOI: 10.1109/EUROSP.2016.13.

[99] M. Ammar, G. Russello, and B. Crispo, "Internet of Things: A Survey on the Security of IoT Frameworks", *Journal of Information Security and Applications*, vol. 38, pp. 8–27, 2018. DOI: 10.1016/J.JISA.2017.11.002.

[100] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A Survey on Security and Privacy Issues in Internet-of-Things", *IEEE Internet Things Journal*, vol. 4, no. 5, pp. 1250–1258, 2017. DOI: 10.1109/JIOT.2017.2694844.

[101] Z. Yan, P. Zhang, and A. V. Vasilakos, "A Survey on Trust Management for Internet of Things", *Journal of Network and Computer Applications*, vol. 42, pp. 120–134, 2014. DOI: 10.1016/J.JNCA.2014.01.014.

[102] C. M. R. da Silva, R. B. Rodrigues, R. J. G. B. de Queiroz, V. C. Garcia, J. Silva, D. Gatti, R. E. Assad, L. M. do Nascimento, K. Brito, and P. B. C. de Miranda, "Towards a Taxonomy for Security Threats on the Web Ecosystem", in *2016 IEEE/IFIP Network Operations and Management Symposium, NOMS 2016, Istanbul, Turkey, April 25-29, 2016*, S. Oktug, M. Ulema, C. Cavdar, L. Z. Granville, and C. R. P. dos Santos, Eds., IEEE, 2016, pp. 584–590. DOI: 10.1109/NOMS.2016.7502862.

[103] M. Rocchetto and N. O. Tippenhauer, "On Attacker Models and Profiles for Cyber-Physical Systems", in *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II*, I. G. Askoxylakis, S. Ioannidis, S. K. Katsikas, and C. Meadows, Eds., ser. Lecture Notes in Computer Science, vol. 9879, Springer, 2016, pp. 427–449. DOI: 10.1007/978-3-319-45741-3_22.

[104] W. Trappe, R. E. Howard, and R. S. Moore, "Low-Energy Security: Limits and Opportunities in the Internet of Things", *IEEE Security & Privacy*, vol. 13, no. 1, pp. 14–21, 2015. DOI: 10.1109/MSP.2015.7.

[105] F. Standaert, "Introduction to Side-Channel Attacks", in *Secure Integrated Circuits and Systems*, ser. Integrated Circuits and Systems, I. M. R. Verbauwhede, Ed., Springer, 2010, pp. 27–42. DOI: 10.1007/978-0-387-71829-3_2.

[106] R. L. Barnes, B. Schneier, C. Jennings, T. Hardie, B. Trammell, C. Huitema, and D. Borkmann, "Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement", *RFC*, vol. 7624, pp. 1–24, 2015. DOI: 10.17487/RFC7624.

[107] P. F. Syverson, "A Taxonomy of Replay Attacks", Tech. Rep., 1994, pp. 187–191. DOI: 10.1109/CSFW.1994.315935.

[108] O. A. Osanaiye, A. S. Alfa, and G. P. Hancke, "Denial of Service Defence for Resource Availability in Wireless Sensor Networks", *IEEE Access*, vol. 6, pp. 6975–7004, 2018. DOI: 10.1109/ACCESS.2018.2793841.

[109] M. Pirretti, S. Zhu, N. Vijaykrishnan, P. D. McDaniel, M. T. Kandemir, and R. R. Brooks, "The Sleep Deprivation Attack in Sensor Networks: Analysis and Methods of Defense", *International Journal of Distributed Sensor Networks*, vol. 2, no. 3, pp. 267–287, 2006. DOI: 10.1080/15501320600642718.

[110] K. Pelechrinis, M. Iliofotou, and S. V. Krishnamurthy, "Denial of Service Attacks in Wireless Networks: The Case of Jammers", *IEEE Communications Surveys & Tutorials*, vol. 13, no. 2, pp. 245–257, 2011. DOI: 10.1109/SURV.2011.041110.00022.

[111] A. Mpitziopoulos, D. Gavalas, C. Konstantopoulos, and G. E. Pantziou, "A Survey on Jamming Attacks and Countermeasures in WSNs", *IEEE Communications Surveys & Tutorials*, vol. 11, no. 4, pp. 42–56, 2009. DOI: 10.1109/SURV.2009.090404.

[112] C. Schellenberger and P. Zhang, "Detection of Covert Attacks on Cyber-Physical Systems by Extending the System Dynamics with an Auxiliary System", in *56th IEEE Annual Conference on Decision and Control, CDC 2017, Melbourne, Australia, December 12-15, 2017*, IEEE, 2017, pp. 1374–1379. DOI: 10.1109/CDC.2017.8263846.

[113] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007, ISBN: 978-0-387-30857-9.

[114] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. M. Zanchettin, and S. Zanero, "An Experimental Security Analysis of an Industrial Robot Controller", in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, IEEE Computer Society, 2017, pp. 268–286. DOI: 10.1109/SP.2017.20.

[115] J. R. Douceur, "The Sybil Attack", in *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, Eds., ser. Lecture Notes in Computer Science, vol. 2429, Springer, 2002, pp. 251–260. DOI: 10.1007/3-540-45748-8_24.

[116] A. D. Wood and J. A. Stankovic, "Denial of Service in Sensor Networks", *Computer*, vol. 35, no. 10, pp. 54–62, 2002. DOI: 10.1109/MC.2002.1039518.

[117] C. Karlof and D. A. Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures", 2-3, vol. 1, 2003, pp. 293–315. DOI: 10.1016/S1570-8705(03)00008-8.

[118] G. N. Nayak and S. G. Samaddar, "Different Flavours of Man-in-the-Middle Attack, Consequences and Feasible Solutions", in *2010 3rd International Conference on Computer Science and Information Technology*, IEEE, vol. 5, 2010, pp. 491–495. DOI: 10.1109/ICCSIT.2010.5563900.

[119] T. Moore and R. Clayton, "An Empirical Analysis of the Current State of Phishing Attack and Defence", in *6th Annual Workshop on the Economics of Information Security, WEIS 2007, The Heinz School and CyLab at Carnegie Mellon University, Pittsburgh, PA, USA, June 7-8, 2007*, 2007.

[120] E. Ronen, A. Shamir, A. Weingarten, and C. O'Flynn, "IoT Goes Nuclear: Creating a ZigBee Chain Reaction", in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, IEEE Computer Society, 2017, pp. 195–212. DOI: 10.1109/SP.2017.14.

[121] Mitre ATT&CK, *Masquerading*, https://attack.mitre.org/techniques/T1036/, Accessed: 2024-07-02, 2017.

[122] M. Budeus, *Mapping Network Flows to Applications for Profile Creation*, Bachelor's thesis advised by Lars Wüstrich and Sebastian Gallenmüller, Garching, Germany, 2021.

[123] D. F. von Künßberg, *Mapping Network Flows to Local Processes Efficiently*, Bachelor's thesis advised by Lars Wüstrich and Sebastian Gallenmüller, Garching, Germany, 2022.

[124] S. Margaritelli, *OpenSnitch*, https://github.com/evilsocket/opensnitch, Accessed: 2024-07-03.

[125] S. Ma, J. Zhai, Y. Kwon, K. H. Lee, X. Zhang, G. F. Ciocarlie, A. Gehani, V. Yegneswaran, D. Xu, and S. Jha, "Kernel-Supported Cost-Effective Audit Logging for Causality Tracking", in *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*, H. S. Gunawi and B. C. Reed, Eds., USENIX Association, 2018, pp. 241–254.

[126] osquery, *Performant Endpoint Visibility*, https://osquery.io/, Accessed: 2024-07-03.

[127] A. Singh, O. Nordström, C. Lu, and A. L. M. dos Santos, "Malicious ICMP Tunneling: Defense against the Vulnerability", in *Information Security and Privacy, 8th Australasian*

Conference, ACISP 2003, Wollongong, Australia, July 9-11, 2003, Proceedings, R. Safavi-Naini and J. Seberry, Eds., ser. Lecture Notes in Computer Science, vol. 2727, Springer, 2003, pp. 226–235. DOI: 10.1007/3-540-45067-X_20.

[128] *eBPF*, https://ebpf.io/, Accessed: 2024-07-03.

[129] A. M. Bishop, "The /proc file system and procmeter", *Linux Journal*, vol. 1997, no. 36es, 5–es, 1997.

[130] S. Ma, X. Zhang, and D. Xu, "ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting", in *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, The Internet Society, 2016.

[131] *Internet Assigned Numbers Authority*, https://www.iana.org/, Accessed: 2024-07-03.

[132] S. M. Bellovin, *Firewall-Friendly FTP*, RFC 1579, 1994. DOI: 10.17487/RFC1579.

[133] Falco, *Detect security threats in real time*, https://falco.org/, Accessed: 2024-07-03.

[134] A. Engelen, *Nethogs*, https://github.com/raboof/nethogs, Accessed: 2024-07-03.

[135] Linux manual page, *netstat(8)*, https://linux.die.net/man/8/netstat, Accessed: 2024-07-03.

[136] A. K. Chimata, "Path of a Packet in the Linux Kernel Stack", *University of Kansas*, 2005.

[137] Linux manual page, *packet(7)*, https://man7.org/linux/man-pages/man7/packet.7.html, Accessed: 2024-07-01.

[138] *struct sk_buf*, https://docs.kernel.org/networking/skbuff.html, Accessed: 2024-07-12.

[139] *Internet Control Message Protocol*, RFC 792, 1981. DOI: 10.17487/RFC0792.

[140] R. Droms, *Dynamic Host Configuration Protocol*, RFC 2131, 1997. DOI: 10.17487/RFC2131.

[141] David Anderson, *What is PID 0?*, https://blog.dave.tf/post/linux-pid0/, Accessed: 2024-09-13.

[142] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle, "The pos Framework: A Methodology and Toolchain for Reproducible Network Experiments", in *CoNEXT '21: The 17th International Conference on emerging Networking EXperiments and Technologies, Virtual Event, Munich, Germany, December 7 - 10, 2021*, G. Carle and J. Ott, Eds., ACM, 2021, pp. 259–266. DOI: 10.1145/3485983.3494841.

[143] D. von Künßberg, *network-matcher*, https://github.com/undvikar/network-matcher, Accessed: 2024-07-03.

[144] redbpf, *Rust library for building and running BPF/eBPF modules*, https://github.com/foniod/redbpf, Accessed: 2024-07-03.

[145] M. Schacherbauer, *Profiling Applications via their Network Flows*, Master's thesis advised by Lars Wüstrich and Sebastian Gallenmüller, Garching, Germany, 2023.

[146] Intel, *Intel VTune Profiler*, https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html, Accessed: 2024-07-03.

[147] R. Canzanese, S. Mancoridis, and M. Kam, "System Call-Based Detection of Malicious Processes", in *2015 IEEE International Conference on Software Quality, Reliability and*

Security, QRS 2015, Vancouver, BC, Canada, August 3-5, 2015, IEEE, 2015, pp. 119–124. DOI: 10.1109/QRS.2015.26.

[148] S. Floyd and V. Paxson, "Difficulties in Simulating the Internet", *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, pp. 392–403, 2001. DOI: 10.1109/90.944338.

[149] D. L. Mills, "Network Time Protocol (NTP)", *RFC*, vol. 958, pp. 1–14, 1985. DOI: 10.17487/RFC0958.

[150] D. Murray, T. Koziniec, S. Zander, M. W. Dixon, and P. Koutsakis, "An Analysis of Changing Enterprise Network Traffic Characteristics", in *23rd Asia-Pacific Conference on Communications, APCC 2017, Perth, Australia, December 11-13, 2017*, IEEE, 2017, pp. 1–6. DOI: 10.23919/APCC.2017.8303960.

[151] S. Bauer, B. Jaeger, F. Helfert, P. Barias, and G. Carle, "On the Evolution of Internet Flow Characteristics", in *ANRW '21: Applied Networking Research Workshop, Virtual Event, USA, July 24-30, 2021*, ACM, 2021, pp. 29–35. DOI: 10.1145/3472305.3472321.

[152] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of Frequent Episodes in Event Sequences", *Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 259–289, 1997. DOI: 10.1023/A:1009748302351.

[153] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach", *Data Mining and Knowledge Discovery*, vol. 8, no. 1, pp. 53–87, 2004. DOI: 10.1023/B:DAMI.0000005258.31418.83.

[154] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining Association Rules between Sets of Items in Large Databases", in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 26-28, 1993*, P. Buneman and S. Jajodia, Eds., ACM Press, 1993, pp. 207–216. DOI: 10.1145/170035.170072.

[155] R. Agrawal and R. Srikant, "Mining Sequential Patterns", in *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, P. S. Yu and A. L. P. Chen, Eds., IEEE Computer Society, 1995, pp. 3–14. DOI: 10.1109/ICDE.1995.380415.

[156] K. Huang and C. Chang, "Efficient Mining of Frequent Episodes from Complex Sequences", *Information Systems*, vol. 33, no. 1, pp. 96–114, 2008. DOI: 10.1016/j.is.2007.07.003.

[157] N. Joukov, V. Tarasov, J. Ossher, B. Pfitzmann, S. Chicherin, M. Pistoia, and T. Tateishi, "Static Discovery and Remediation of Code-Embedded Resource Dependencies", in *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management, IM 2011, Dublin, Ireland, 23-27 May 2011*, N. Agoulmine, C. Bartolini, T. Pfeifer, and D. O'Sullivan, Eds., IEEE, 2011, pp. 233–240. DOI: 10.1109/INM.2011.5990696.

[158] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using Magpie for Request Extraction and Workload Modelling", in *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, E. A. Brewer and P. Chen, Eds., USENIX Association, 2004, pp. 259–272.

[159] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl, "Automating Network Application Dependency Discovery: Experiences, Limitations, and New Solutions", in *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10,*

2008, San Diego, California, USA, Proceedings, R. Draves and R. van Renesse, Eds., USENIX Association, 2008, pp. 117–130.

[160] Y. Wang, J. Li, and P. Stoica, *Spectral Analysis of Signals: The Missing Data Case* (Synthesis Lectures on Signal Processing). Morgan & Claypool Publishers, 2006. DOI: 10.2200/S00001ED1V01Y200508SPR001.

[161] R. A. Fisher, "Tests of Significance in Harmonic Analysis", *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, vol. 125, no. 796, pp. 54–59, 1929.

[162] N. Heard, P. Rubin-Delanchy, and D. J. Lawson, "Filtering Automated Polling Traffic in Computer Network Flow Data", in *IEEE Joint Intelligence and Security Informatics Conference, JISIC 2014, The Hague, The Netherlands, 24-26 September, 2014*, IEEE, 2014, pp. 268–271. DOI: 10.1109/JISIC.2014.52.

[163] *Network Time Protocol (Version 3) Specification, Implementation and Analysis*, RFC 1305, 1992. DOI: 10.17487/RFC1305.

[164] B. AsSadhan, J. M. F. Moura, and D. E. Lapsley, "Periodic Behavior in Botnet Command and Control Channels Traffic", in *Proceedings of the Global Communications Conference, 2009. GLOBECOM 2009, Honolulu, Hawaii, USA, 30 November - 4 December 2009*, IEEE, 2009, pp. 1–6. DOI: 10.1109/GLOCOM.2009.5426172.

[165] Y. Liu, L. Wei, Z. Zhou, K. Zhang, W. Xu, and Q. Xu, "On Code Execution Tracking via Power Side-Channel", in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds., ACM, 2016, pp. 1019–1031. DOI: 10.1145/2976749.2978299.

[166] Y. Wang, R. Paccagnella, E. T. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, "Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86", in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds., USENIX Association, 2022, pp. 679–697.

[167] P. V. Aubel, K. Papagiannopoulos, L. Chmielewski, and C. Doerr, "Side-Channel Based Intrusion Detection for Industrial Control Systems", in *Critical Information Infrastructures Security - 12th International Conference, CRITIS 2017, Lucca, Italy, October 8-13, 2017, Revised Selected Papers*, G. D'Agostino and A. Scala, Eds., ser. Lecture Notes in Computer Science, vol. 10707, Springer, 2017, pp. 207–224. DOI: 10.1007/978-3-319-99843-5_19.

[168] R. Shah, M. Ahmed, and S. Nagaraja, "Fingerprinting Robot Movements via Acoustic Side Channel", *CoRR*, vol. abs/2209.10240, 2022. DOI: 10.48550/ARXIV.2209.10240.

[169] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding Data Center Traffic Characteristics", *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, 2010. DOI: 10.1145/1672308.1672325.

[170] M. Marwah, R. Sharma, and C. Bash, "Thermal Anomaly Prediction in Data Centers", in *2010 12th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, IEEE, 2010, pp. 1–7. DOI: 10.1109/ITHERM.2010.5501330.

[171] E. K. Lee, H. Viswanathan, and D. Pompili, "Model-Based Thermal Anomaly Detection in Cloud Datacenters", in *IEEE International Conference on Distributed Computing in Sensor Systems, DCOSS 2013, Cambridge, MA, USA, May 20-23, 2013*, IEEE Computer Society, 2013, pp. 191–198. DOI: 10.1109/DCOSS.2013.8.

[172] E. Lee, H. Viswanathan, and D. Pompili, "Model-Based Thermal Anomaly Detection in Cloud Datacenters Using Thermal Imaging", *IEEE Trans. Cloud Comput.*, vol. 6, no. 2, pp. 330–343, 2018. DOI: 10.1109/TCC.2015.2481423.

[173] W. Barth, *Nagios: System and Network Monitoring*. No Starch Press, 2008, ISBN: 978-1593271794.

[174] IBM, *Introduction to Tivoli Netcool/OMNIbus*, https://www.ibm.com/docs/en/netcoolomnibus, Accessed: 2024-07-03.

[175] *Grafana: The open observability platform*, https://grafana.com/, Accessed: 2024-07-03.

[176] A Aimar, A. A. Corman, P Andrade, S Belov, J. D. Fernandez, B. G. Bear, M Georgiou, E Karavakis, L Magnoni, R. R. Ballesteros, *et al.*, "Unified Monitoring Architecture for IT and Grid Services", in *Journal of Physics: Conference Series*, IOP Publishing, vol. 898, 2017, p. 092 033. DOI: 10.1088/1742-6596/898/9/092033.

[177] - *iDRAC - integrated Dell Remote Access Controller*, Dell.

[178] V. Boddapati, A. Petef, J. Rasmusson, and L. Lundberg, "Classifying Environmental Sounds Using Image Recognition Networks", in *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 21st International Conference KES-2017, Marseille, France, 6-8 September 2017*, C. Zanni-Merk, C. S. Frydman, C. Toro, Y. Hicks, R. J. Howlett, and L. C. Jain, Eds., ser. Procedia Computer Science, vol. 112, Elsevier, 2017, pp. 2048–2056. DOI: 10.1016/j.procs.2017.08.250.

[179] D. Genkin, A. Shamir, and E. Tromer, "Acoustic Cryptanalysis", *Journal of Cryptology*, vol. 30, no. 2, pp. 392–443, 2017. DOI: 10.1007/s00145-015-9224-2.

[180] J. Haitsma and T. Kalker, "A Highly Robust Audio Fingerprinting System", in *ISMIR 2002, 3rd International Conference on Music Information Retrieval, Paris, France, October 13-17, 2002, Proceedings*, 2002.

[181] S. Belikovetsky, Y. A. Solewicz, M. Yampolskiy, J. Toh, and Y. Elovici, "Digital Audio Signature for 3D Printing Integrity", *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 5, pp. 1127–1141, 2019. DOI: 10.1109/TIFS.2018.2851584.

[182] N. Madhu, "Note on Measures for Spectral Flatness", *Electronics letters*, vol. 45, no. 23, pp. 1195–1196, 2009. DOI: 10.1049/el.2009.1977.

[183] J. O. Smith, *Mathematics of the Discrete Fourier Transform (DFT)*. W3K Publishing, 2007, ISBN: 978-0-9745607-4-8.

[184] Blue Microphones, *Blue - Yeti X*, https://web.archive.org/web/20230331185228/https://www.bluemic.com/en-us/products/yeti-x/, Accessed: 2024-07-03.

[185] B. McFee, C. Raffel, D. Liang, D. P. W. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and Music Signal Analysis in Python", in *Proceedings of the 14th Python in Science Conference 2015 (SciPy 2015), Austin, Texas, July 6 - 12, 2015*, K. Huff and J. Bergstra, Eds., scipy.org, 2015, pp. 18–24. DOI: 10.25080/MAJORA-7B98E3ED-003.

[186] D. Stock and D. Schel, "Cyber-Physical Production System Fingerprinting", *Procedia CIRP*, vol. 81, pp. 393–398, 2019, 52nd CIRP Conference on Manufacturing Systems

(CMS), Ljubljana, Slovenia, June 12-14, 2019, ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2019.03.068.

[187] H. Choi, W. Lee, Y. Aafer, F. Fei, Z. Tu, X. Zhang, D. Xu, and X. Deng, "Detecting Attacks Against Robotic Vehicles: A Control Invariant Approach", in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds., ACM, 2018, pp. 801–816. DOI: 10.1145/3243734.3243752.

[188] D. Formby, A. Walid, and R. A. Beyah, "A Case Study in Power Substation Network Dynamics", *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, 19:1–19:24, 2017. DOI: 10.1145/3084456.

[189] O. Gasser, F. Emmert, and G. Carle, "Digging for Dark IPMI Devices: Advancing BMC Detection and Evaluating Operational Security", in *Traffic Monitoring and Analysis - 8th International Workshop, TMA 2016, Louvain la Neuve, Belgium, April 7-8, 2016*, A. Botta, R. Sadre, and F. E. Bustamante, Eds., IFIP, 2016.

[190] J. O. Smith, *Spectral Audio Signal Processing*. W3K Publishing, 2011, ISBN: 978-0-9745607-3-1.

[191] Y. Han, M. Chan, Z. Aref, N. O. Tippenhauer, and S. A. Zonouz, "Hiding in Plain Sight? On the Efficacy of Power Side Channel-Based Control Flow Monitoring", in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds., USENIX Association, 2022, pp. 661–678.

[192] J. M. Hernández, L. Pouchard, J. T. McDonald, and S. J. Prowell, "Developing a Power Measurement Framework for Cyber Defense", in *Cyber Security and Information Intelligence, CSIIRW '13, Oak Ridge, TN, USA, January 8-10, 2013*, F. T. Sheldon, A. Giani, A. W. Krings, and R. K. Abercrombie, Eds., ACM, 2013, p. 28. DOI: 10.1145/2459976.2460008.

[193] C. M. Ahmed, M. Ochoa, J. Zhou, A. P. Mathur, R. Qadeer, C. Murguia, and J. Ruths, "NoisePrint: Attack Detection Using Sensor and Process Noise Fingerprint in Cyber Physical Systems", in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, J. Kim, G. Ahn, S. Kim, Y. Kim, J. López, and T. Kim, Eds., ACM, 2018, pp. 483–497. DOI: 10.1145/3196494.3196532.

[194] C. M. Ahmed, J. Zhou, and A. P. Mathur, "Noise Matters: Using Sensor and Process Noise Fingerprint to Detect Stealthy Cyber Attacks and Authenticate sensors in CPS", in *Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018*, ACM, 2018, pp. 566–581. DOI: 10.1145/3274694.3274748.

[195] A. Yang, X. Wang, Y. Sun, Y. Hu, Z. Shi, and L. Sun, "Multi-Dimensional Data Fusion Intrusion Detection for Stealthy Attacks on Industrial Control Systems", in *IEEE Global Communications Conference, GLOBECOM 2018, Abu Dhabi, United Arab Emirates, December 9-13, 2018*, IEEE, 2018, pp. 1–7. DOI: 10.1109/GLOCOM.2018.8648131.

[196] U. Rührmair, X. Xu, J. Sölter, A. Mahmoud, M. Majzoobi, F. Koushanfar, and W. P. Burleson, "Efficient Power and Timing Side Channels for Physical Unclonable Functions", in *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th Inter-*

national Workshop, Busan, South Korea, September 23-26, 2014. Proceedings, L. Batina and M. Robshaw, Eds., ser. Lecture Notes in Computer Science, vol. 8731, Springer, 2014, pp. 476–492. DOI: 10.1007/978-3-662-44709-3_26.

[197] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM Side-Channel(s)", in *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, B. S. K. Jr., Ç. K. Koç, and C. Paar, Eds., ser. Lecture Notes in Computer Science, vol. 2523, Springer, 2002, pp. 29–45. DOI: 10.1007/3-540-36400-5_4.

[198] A. A. Milani, I. M. S. Panahi, and P. C. Loizou, "A New Delayless Subband Adaptive Filtering Algorithm for Active Noise Control Systems", *IEEE Trans. Speech Audio Process.*, vol. 17, no. 5, pp. 1038–1045, 2009. DOI: 10.1109/TASL.2009.2015691.

[199] Z. Rafii, A. Liutkus, F. Stöter, S. I. Mimilakis, D. FitzGerald, and B. Pardo, "An Overview of Lead and Accompaniment Separation in Music", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, no. 8, pp. 1307–1335, 2018. DOI: 10.1109/TASLP.2018.2825440.

[200] R. Müller, F. Ritz, S. Illium, and C. Linnhoff-Popien, "Acoustic Anomaly Detection for Machine Sounds based on Image Transfer Learning", in *Proceedings of the 13th International Conference on Agents and Artificial Intelligence, ICAART 2021, Volume 2, Online Streaming, February 4-6, 2021*, A. P. Rocha, L. Steels, and H. J. van den Herik, Eds., SCITEPRESS, 2021, pp. 49–56. DOI: 10.5220/0010185800490056.

[201] P. Pham, J. Li, J. Szurley, and S. Das, "Eventness: Object Detection on Spectrograms for Temporal Localization of Audio Events", in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*, IEEE, 2018, pp. 2491–2495. DOI: 10.1109/ICASSP.2018.8462062.

[202] A. Wang, "An Industrial Strength Audio Search Algorithm", in *ISMIR 2003, 4th International Conference on Music Information Retrieval, Baltimore, Maryland, USA, October 27-30, 2003, Proceedings*, 2003.

[203] M. Müller, "Dynamic Time Warping", *Information Retrieval for Music and Motion*, pp. 69–84, 2007.

[204] M. Krause, M. Müller, and C. Weiß, "Towards Leitmotif Activity Detection in Opera Recordings", *Trans. Int. Soc. Music. Inf. Retr.*, vol. 4, no. 1, pp. 127–140, 2021. DOI: 10.5334/TISMIR.116.

[205] X. Song, Q. Wen, Y. Li, and L. Sun, "Robust Time Series Dissimilarity Measure for Outlier Detection and Periodicity Detection", in *Proceedings of the 31st ACM International Conference on Information& Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, M. A. Hasan and L. Xiong, Eds., ACM, 2022, pp. 4510–4514. DOI: 10.1145/3511808.3557686.

[206] D. Genkin, N. Nissan, R. Schuster, and E. Tromer, "Lend Me Your Ear: Passive Remote Physical Side Channels on PCs", in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds., USENIX Association, 2022, pp. 4437–4454.

[207] U. Klauer, *Message on sox mailing list (2013-06-27): noise reduction algorithm*, https://sourceforge.net/p/sox/mailman/sox-users/?viewmonth=201306&viewday=27, Accessed: 2024-07-03.

[208] Y. Lecun and Y. Bengio, "Convolutional Networks for Images, Speech, and Time-Series", English (US), in *The handbook of brain theory and neural networks*, M. Arbib, Ed. MIT Press, 1995.

[209] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep Learning", *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. DOI: 10.1038/NATURE14539.

[210] R. Bittner, E. Humphrey, and J. Bello, "Pysox: Leveraging the Audio Signal Processing Power of sox in Python", in *Proceedings of the International Society for Music Information Retrieval Conference Late Breaking and Demo Papers*, 2016.

[211] *SoX - Sound eXchange*, http://sox.sourceforge.net/, Accessed: 2024-07-03.

[212] M. Stahn, *pypacker - The fastest and simplest packet manipulation lib for Python*, https://gitlab.com/mike01/pypacker, Accessed: 2024-07-03.

[213] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library", in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 8024–8035.

[214] IBM, *Ipmitool*, https://github.com/ipmitool/ipmitool, Accessed: 2024-07-02.

[215] A. Carcano, A. Coletta, M. Guglielmi, M. Masera, I. N. Fovino, and A. Trombetta, "A Multidimensional Critical State Analysis for Detecting Intrusions in SCADA Systems", *IEEE Transactions on Industrial Informatics*, vol. 7, no. 2, pp. 179–186, 2011. DOI: 10.1109/TII.2010.2099234.

[216] R. A. Fisher, *The Design of Experiments*. Oliver and Boyd Edinburgh, 1935.

[217] K. R. Weiss, T. M. Khoshgoftaar, and D. Wang, "A Survey of Transfer Learning", *Journal of Big Data*, vol. 3, p. 9, 2016. DOI: 10.1186/S40537-016-0043-6.

[218] H. Shimodaira, "Improving Predictive Inference Under Covariate Shift by Weighting the Log-Likelihood Function", *Journal of Statistical Planning and Inference*, vol. 90, no. 2, pp. 227–244, 2000, ISSN: 0378-3758. DOI: https://doi.org/10.1016/S0378-3758(00)00115-4.

[219] Y. Luo, Y. Xiao, L. Cheng, G. Peng, and D. D. Yao, "Deep Learning-based Anomaly Detection in Cyber-physical Systems: Progress and Opportunities", *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, 106:1–106:36, 2022. DOI: 10.1145/3453155.

[220] J. Giraldo, D. I. Urbina, A. A. Cárdenas, J. Valente, M. A. Faisal, J. Ruths, N. O. Tippenhauer, H. Sandberg, and R. Candell, "A Survey of Physics-Based Attack Detection in Cyber-Physical Systems", *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, 76:1–76:36, 2018. DOI: 10.1145/3203245.

[221] W. Aoudi, M. Iturbe, and M. Almgren, "Truth Will Out: Departure-Based Process-Level Detection of Stealthy Attacks on Control Systems", in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto,*

ON, Canada, October 15-19, 2018, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds., ACM, 2018, pp. 817–831. DOI: 10.1145/3243734.3243781.

[222] Q. Wang, W. U. Hassan, A. Bates, and C. A. Gunter, "Fear and Logging in the Internet of Things", in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*, The Internet Society, 2018.

[223] M.-K. Yoon and G. F. Ciocarlie, "Communication Pattern Monitoring: Improving the Utility of Anomaly Detection for Industrial Control Systems", in *NDSS Workshop on Security of Emerging Networking Technologies*, 2014.

[224] S. J. Saidi, A. M. Mandalari, R. Kolcun, H. Haddadi, D. J. Dubois, D. R. Choffnes, G. Smaragdakis, and A. Feldmann, "A Haystack Full of Needles: Scalable Detection of IoT Devices in the Wild", in *IMC '20: ACM Internet Measurement Conference, Virtual Event, USA, October 27-29, 2020*, ACM, 2020, pp. 87–100. DOI: 10.1145/3419394.3423650.

[225] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou, "Specification-Based Anomaly Detection: A New Approach for Detecting Network Intrusions", in *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, V. Atluri, Ed., ACM, 2002, pp. 265–274. DOI: 10.1145/586110.586146.

[226] N. Boggs, J. C. Chau, and A. Cui, "Utilizing Electromagnetic Emanations for Out-of-Band Detection of Unknown Attack Code in a Programmable Logic Controller", in *Cyber Sensing 2018*, International Society for Optics and Photonics, vol. 10630, 2018. DOI: 10.1117/12.2304465.

[227] A. Nazari, N. Sehatbakhsh, M. Alam, A. G. Zajic, and M. Prvulovic, "EDDIE: EM-Based Detection of Deviations in Program Execution", in *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*, ACM, 2017, pp. 333–346. DOI: 10.1145/3079856.3080223.

[228] S. Dupuis, G. D. Natale, M. Flottes, and B. Rouzeyre, "On the Effectiveness of Hardware Trojan Horse Detection via Side-Channel Analysis", *Information Security Journal: A Global Perspective*, vol. 22, no. 5-6, pp. 226–236, 2013. DOI: 10.1080/19393555.2014.891277.

[229] R. Bolboaca, B. Genge, and P. Haller, "Using Side-Channels to Detect Abnormal Behavior in Industrial Control Systems", in *15th IEEE International Conference on Intelligent Computer Communication and Processing, ICCP 2019, Cluj-Napoca, Romania, September 5-7, 2019*, S. Nedevschi, R. Potolea, and R. R. Slavescu, Eds., IEEE, 2019, pp. 435–441. DOI: 10.1109/ICCP48234.2019.8959745.

[230] P. Cano, E. Batlle, T. Kalker, and J. Haitsma, "A Review of Audio Fingerprinting", *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, vol. 41, no. 3, pp. 271–284, 2005. DOI: 10.1007/S11265-005-4151-3.

[231] R. Typke, F. Wiering, and R. C. Veltkamp, "A Survey of Music Information Retrieval Systems", in *ISMIR 2005, 6th International Conference on Music Information Retrieval, London, UK, 11-15 September 2005, Proceedings*, 2005, pp. 153–160.

[232] A. Wang, "The Shazam Music Recognition Service", *Communications of the ACM*, vol. 49, no. 8, pp. 44–48, 2006. DOI: 10.1145/1145287.1145312.

[233] S. Chachada and C. J. Kuo, "Environmental Sound Recognition: A Survey", pp. 1–9, 2013. DOI: 10.1109/APSIPA.2013.6694338.

[234] K. J. Piczak, "Environmental Sound Classification with Convolutional Neural Networks", in *25th IEEE International Workshop on Machine Learning for Signal Processing, MLSP 2015, Boston, MA, USA, September 17-20, 2015*, D. Erdogmus, M. Akçakaya, S. S. Kozat, and J. Larsen, Eds., IEEE, 2015, pp. 1–6. DOI: 10.1109/MLSP.2015.7324337.

[235] J. Salamon and J. P. Bello, "Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification", *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279–283, 2017. DOI: 10.1109/LSP.2017.2657381.

[236] M. Backes, M. Dürmuth, S. Gerling, M. Pinkal, and C. Sporleder, "Acoustic Side-Channel Attacks on Printers", in *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, USENIX Association, 2010, pp. 307–322.

[237] M. A. A. Faruque, S. R. Chhetri, A. Canedo, and J. Wan, "Acoustic Side-Channel Attacks on Additive Manufacturing Systems", in *7th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2016, Vienna, Austria, April 11-14, 2016*, IEEE Computer Society, 2016, 19:1–19:10. DOI: 10.1109/ICCPS.2016.7479068.

[238] A. Hojjati, A. Adhikari, K. Struckmann, E. Chou, T. N. T. Nguyen, K. Madan, M. S. Winslett, C. A. Gunter, and W. P. King, "Leave Your Phone at the Door: Side Channels that Reveal Factory Floor Secrets", in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds., ACM, 2016, pp. 883–894. DOI: 10.1145/2976749.2978323.

[239] S. R. Chhetri, A. Canedo, and M. A. A. Faruque, "KCAD: Kinetic Cyber-Attack Detection Method for Cyber-Physical Additive Manufacturing Systems", in *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD 2016, Austin, TX, USA, November 7-10, 2016*, F. Liu, Ed., ACM, 2016, p. 74. DOI: 10.1145/2966986.2967050.