# FingerCI: Writing Industrial Process Specifications from Network Traffic

Filipe Apolinário[a], Nelson Escravana[a], Éric Hervé[b], Miguel L. Pardal[c], Miguel Correia[c]

[a]*INOV-INESC INOVAÇÃO, R. Alves Redol 9, Lisbon, 1000-029, Portugal*
[b]*Alstef Group, 104 Bd de la Salle, Boigny-sur-Bionne, 45760, France*
[c]*INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, R. Alves Redol 9, Lisbon, 1000-029, Portugal*

**Abstract**

Critical infrastructures (CIs) are often targets of cyber-attacks, requiring accurate process specifications to identify and defend against incidents. However, discrepancies between these specifications and real-world CI conditions arise due to the costly process of manual specification by experts.

This paper introduces FingerCI, a method for automatically generating CI process specifications through network traffic analysis and physical behavior modeling. By defining a Specification Language that integrates with existing systems, FingerCI extracts industrial process specifications without infrastructure changes or downtime. The specifications include a behavior model that validates physical correctness.

We evaluated FingerCI on a digital twin of an airport baggage handling system, achieving 99.98% fitness to observed behavior. Our method improves cybersecurity and fault detection with high accuracy.

*Keywords:* cyber-physical systems, network profiling, process mining, critical infrastructures, specification-based intrusion detection systems

*Email addresses:* filipe.apolinario@tecnico.ulisboa.pt (Filipe Apolinário), nelson.escravana@inov.pt (Nelson Escravana), eric.herve@alstefgroup.com (Éric Hervé), miguel.pardal@tecnico.ulisboa.pt (Miguel L. Pardal), miguel.p.correia@tecnico.ulisboa.pt (Miguel Correia)

## 1. Introduction

Critical Infrastructures (CIs) [50], including those in energy, transportation, and water distribution, are typically managed through an Industrial Control System (ICS) architecture [47]. Such an architecture has three layers: supervisory, control, and physical. The *supervisory layer* manages the system, whereas the other two implement the industrial processes. The *control layer* comprises devices such as Programmable Logic Controllers (PLCs), which operate industrial equipment, i.e., the *physical layer*. Given that most control layer devices must work continuously and without interruption with a projected lifespan of more than 15 years [14], many devices used in CIs contain *legacy code*. This implies that their maintenance tends to be expensive, as it involves engineers with unusual expertise and tools that have stopped being supported.

These maintenance challenges lead to obsolete or missing and difficult to create *specifications* of the control system behavior [48]. The lack of specifications prevents the use of *Specification-Based Systems* (SBSs). For example, with increasing exposure of CIs to *cyber-physical attacks* [12, 33, 52], the lack of specification makes it harder for CI operators to identify faults caused by these attacks and their effect on industrial processes. In contrast, the existence of a specification of the behavior of the control layer allows the use of a *Specification-Based Intrusion Detection System* (SBIDS) [32, 31], a type of SBS that can detect when the behavior deviates from a specification. Specifications also allow for quality of service monitoring [10] and the implementation of digital twins [14, 3].

There are a few methods for *inferring specifications*, but they usually require collecting logs from all devices involved [54, 58, 29, 53, 40, 41, 39]. This is not practical in CIs due to three challenges: (1) accessing control layer devices that usually are on a restricted network; (2) preprocessing different formats of data logs to be analyzed by process discovery methods; and (3) the nonexistence of logs in many control devices (due to old software), leading to incomplete process models.

This article proposes FingerCI, which addresses these challenges relying only on network traffic. FingerCI is a semi-supervised approach that constructs a specification, or *fingerprint*, of a CI system based on network traffic inspection. This specification represents the *normal behavior* of the CI system. The fingerprints produced by FingerCI have the important characteristic of being *understandable* to experts, who can verify and modify them

if needed. FingerCI performs *network reconnaissance* based on protocol dissection to interpret the CI network protocols and extract a *network model* of the CI. The information gathered is further analyzed using process discovery techniques [53] to find the expected sequencing of CI network activities. Moreover, data is analyzed using physical behavior modeling to infer the conditions for activities to occur. Automated construction of system specifications from network traffic inspection also raises the technical challenge of defining a specification language that identifies the CI process information in a format that can link information present on multiple network protocols. The language defined in this work provides the missing link that allows current process discovery and behavior analysis to extract the features to model the CI process. This work addresses this challenge and shows the feasibility of its approach with FingerCI, which leverages the proposed language in a semi-supervised method.

An experimental evaluation of the proposed solution was conducted using a digital twin of a *Baggage Handling System* (BHS), on the simulation platform that the actual BHS providers[1]use to test their systems on contractual operating conditions. FingerCI provided *fast* - only took 12 seconds to fingerprint a capture of eight hours of network traffic - and *correct* - with 99.98% fitness - specification models that capture the CI in a format that is understandable by experts. FingerCI specifications were integrated with an SBIDS, the *Business Process Intrusion Detection System* (BP-IDS) [32, 31], and a commonly used specification-based process management system called *Multi-Perspective Process Explorer* (M-PE) [36]. The integration with SBSs did not require software changes to accommodate FingerCI specifications. The experiments show accurate detection of anomalies in the BHS operation.

The main contributions of the paper are:

- Definition of a Specification Language that identifies the CI process information based on network data. This specification language builds on top of existing popular graph specification languages and can be integrated with existing specification-based systems.

- The design of FingerCI to extract industrial process specifications

[1]BHSs provider companies are few and specialized, with little research released to the public due to their security sensitivity. Research focusing on airport security is commonly considered classified information. For example, this article required validation before submission for publication was cleared [15].

from network traffic. FingerCI can write specifications by inspecting network traffic, in a passive way, without requiring changes to the infrastructure and downtime of the CI systems. The specifications generated include behavior models that provide the necessary state variables to validate CI physical correctness. These models when integrated with existing specification-based systems can be used to monitor the CI and determine invalid physical states.

- Integration of FingerCI with two SBSs (BP-IDS and M-PE)[2] for improved cyber security and fault detection. This integration shows the added value of FingerCI in writing specifications for the CI, reducing input needed from experts with accurate detection results and a low false-positive rate.

## 2. Literature Review

Critical Infrastructures have been subjected to a wide range of cyber-physical incidents [22, 23]. These attacks circumvent cybersecurity protections, compromise control layer devices and perform unauthorized operations over the physical layer. Such attacks typically exploit unpatched IT vulnerabilities on supervisory equipment. Once compromised, the malware performs privileged operations over ICS physical assets. Notorious examples of such malware attacks are Stuxnet[16] and CrashOverride [28]. In most cases, security incidents targeting CIs are constructed by abusing legitimate actions to compromise physical devices to an invalid state (e.g., continuously turning off physical equipment to cause denial of service, or altering equipment configuration, like equipment operation speed, to compromise quality of service and safety). Determining when these legitimate actions are abused typically requires a tight security specification.

Several works have proposed SBSs to secure CI control layer devices. Some SBSs model the conditions for the secure operation of physical devices, e.g., unmanned aircraft systems [37] and medical systems [38]). Others specify how network communications should be exchanged between ICS components (e.g., Modbus [20], PCOM [44]). Some model power grid components,

[2] A previous short paper [4] briefly addresses the architecture and integration with BP-IDS. Contributions 1 and 3 are new, and 2 and 4 are improved with the methodology and integration with another detection system (M-PE). This work also adds important process model validation, performance evaluation, and lessons learned.

such as advanced metering infrastructures [6] and automatic generation controllers [49]). Another SBS models railway train movement updates and positional tracking using business processes [32, 31]. Although SBSs are important for protecting CIs from control-layer cyberattacks, they demand the laborious work of enumerating all system state conditions, a task that demands expert knowledge to avoid flaws in the specification. FingerCI aids in inferring the CI security specification based on network traffic to prevent CI invalid states.

FingerCI is related to specification mining [2]. Specification mining infers system models based on observations, e.g., on system logs [18], traces [8], or network communication [7, 57]. FingerCI combines two specification mining areas: procedure analysis and fingerprinting. This combination enables a reliable specification that explains how control layer operations influence the physical behavior of the CI.

Procedure analysis techniques can be divided into invariant rules and process discovery approaches. Invariant rules [18, 46, 59] reflect the behavior of the system that is always verifiable when the system behaves correctly. These rules are generated in two steps. First, extraction of features of the environment based on the values reported in the CI sensor and actuator logs. Second, estimation of invariant rules based on the acceptable ranges of sensor and actuator values. These rules can be obtained based on finding maximum and minimum values [59, 46], or based on machine learning estimators [18]. Some works discover invariants that correlate sensor and actuator values to represent a holistic CI state according to a threshold of support and confidence [18, 46, 34]. Process discovery identifies sequences of activities based on logs and creates a process specification based on these sequences [40, 41, 39, 57]. FingerCI favors process discovery since it offers a more understandable specification model. However, FingerCI goes beyond current approaches that apply process discovery to critical infrastructures. Most of these approaches model the process based on logs stored in control units (which are difficult to access in a real CI) [40, 41, 39]. Some process discovery approaches use network traffic to construct the process [57] but offer a minimal view of the industrial process since they cannot parse CI protocols. FingerCI dissection rules offer a novel in-depth feature extraction mechanism capable of parsing a wide range of CI protocols and offer process discovery over network data. In the evaluation, we show FingerCI process discovery applied in a CI to create understandable process models.

The fingerprinting performed by FingerCI to construct network and

behavior models is related to mechanisms applied in IDSs. Some sensor fingerprints are based on estimating errors in sensor measurements (e.g., power grid sensors [18]). Other sensor fingerprints are based on physics formulas (such as the power system formula [33]). Other sensor fingerprints measure clock skewness in sensor readings (e.g., electronic control units, ECUs, message transmissions [9]). Others extract statistics of sensor values (like average and standard deviations of ECUs [26] or water control systems [1]. Some control-layer messages can be fingerprinted by measuring the time taken by the mechanical actuators to perform actions [19]. FingerCI behavior models are based on physical fingerprinting since they represent the physical state conditions for the execution of the CI process. The mixture of process discovery with physical fingerprinting is part of the novelty of FingerCI.

Caselli et al. present an alternative specification mining system [7]. That system builds specifications based on documentation specific to the BACnet protocol, while FingerCI is based on the assumption that there is no up-to-date documentation, something that we have observed many times in real CIs, so it takes information from the network. FingerCI also addresses a more difficult challenge: obtaining the specification of a critical process. This requires modeling the sequence of operations and their correct behavior, whereas [7] provides neither since its detection rules do not profile operations.

Table 1 compares FingerCI with the related work. Our method supports the most common industrial protocols, including MODBUS, Profinet, or IEC 61850. It can be easily expanded to accommodate proprietary protocols such as aviation-specific protocols, like the IATA RP1745 [24] (see Section 6). The FingerCI behavior model is also novel because previous works that employed process discovery techniques for anomaly detection only evaluated the causal relationship between activities based on their execution [54, 58, 29, 53, 40, 41, 39]. The behavior model complements the specification by adding valid conditions for activities to occur, e.g., the state of variables or devices. The behavior model is constructed based on PLC network traffic and is vital to validate the operation of each process. For each system variable, the behavior model specifies: the expected value before the industrial process operation is performed (i.e., operation precondition), and the value after the operation is carried out (i.e., operation postcondition). This novel behavior model extends the previous work [7] that attempted to construct behavior specifications automatically, since the previous work only identified attacks that performed unauthorized operations or changed parameters on devices, whereas the FingerCI behavior model can identify attacks that can

even perform authorized operations on CI devices to leave the system in an invalid physical state. Although validation was performed on the Aviation domain, the application of FingerCI in other ICS-based domains should be straightforward (see Section 7.2 for more details on generalizability).

| Method | Procedure Analysis | Fingerprinting | Data Source | Protocol | Application Sector |
|---|---|---|---|---|---|
| [18] | Invariant | Physical | Device | None | Energy |
| [40, 41, 39] | Process | - | Device | None | Energy |
| [57] | Process | - | Network | TCP/IP | ICT |
| [46] | Invariant | Physical | Device | None | Transportation |
| [59] | Invariant | Physical | Device | None | Energy |
| [7] | Invariant | Network | Network | BACnet | ICT |
| FingerCI | Process | Network & Physical | Network | Multiple | Transportation |

**Table 1:** Comparison of FingerCI with related work.

## 3. FingerCI

This work aims to create a fingerprint representation of the processes and devices present in a CI. FingerCI inspects traffic and captures the interactions between CI devices.

As shown in Figure 1, FingerCI operates in four stages. The first builds a *registry of CI operations* ($R_o$) by annotating packet captures using an expert-defined ontology. The second stage builds a *network model* ($M_n$) containing identification rules based on regular expressions (REGEXP) that identify CI operations and devices in $R_o$. The third stage creates a *process model* ($M_p$) using process discovery techniques; the model represents the sequence of activities in $R_o$. The fourth stage creates the *behavior model* ($M_b$), composed of rules that reflect the correct conditions for the activities to occur on $R_o$. With these three models – network, process, and behavior models –, SBSs can, for instance, identify operations and detect process faults.

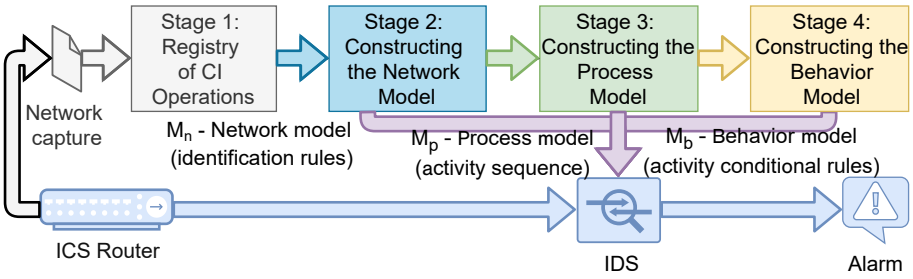


**Figure 1:** Example of FingerCI fingerprinting a CI network and feeding an IDS with the specification.

### *3.1. Assumptions*

FingerCI considers two assumptions: (a) it is possible to record all critical operations using network packet captures; (b) packets are not encrypted in a way that prevents their inspection.

Regarding assumption (a), the location(s) where the network packet captures are obtained have to be those where the SBS will be introduced. For example, if the SBS is an SBIDS, the locations to capture the network traffic should be those where the SBIDS sensors will be placed.

About assumption (b), we are interested in the case where the tool analyzes CI network traffic. In particular, we consider that it has access to the traffic of critical systems that work directly with the physical equipment of the CI (i.e., control layer devices) and perform operations that change the physical behavior of the CI (e.g., change the state of a lever or a valve). To profile the physical behavior of the CI, FingerCI assumes it is capable of accessing deciphered network traffic so that the physical state can be tracked before and after critical operations are performed. Assumption (b) is satisfied by a large range of CI systems, as most data on ICS control layers is not encrypted (they are mostly composed of legacy systems protected by firewall and access control). If ciphered communications should also be fingerprinted, we assume that the CI can decipher network communication while recording the network capture. This assumption is reasonable as the organizations that will deploy FingerCI will typically have full control of their infrastructure, so also access to the cryptographic keys necessary to decrypt the traffic when

recording the packet capture[3].

### *3.2. Background*

The design of FingerCI was inspired by a case study of an airport Baggage Handling System (BHS). BHSs guarantee that bags placed at check-in counters are routed to the correct destinations, typically airplanes, but also implement safety measures, so they are critical infrastructures for air travel. BHSs offer three main services: baggage tracking, screening, and sortation. *Baggage tracking* services keep track of bags on the conveyor by collecting real-time bag location data. *Baggage screening* services ensure that bags pass through security evaluation procedures and that suspicious bags are filtered into a designated unclear chute to ensure flight safety. *Baggage sortation* services decide the correct route for the bags to reach their destination based on bag check-in, flight information, and screening results. We show throughout the article that FingerCI is adequate to generate a specification that includes the three BHS services.

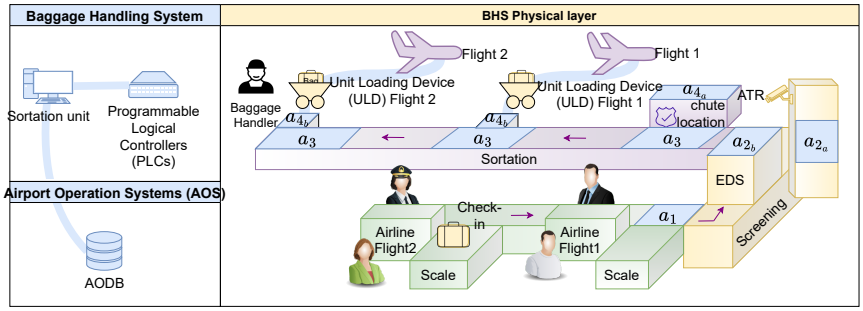


**Figure 2:** Architecture of a BHS system. On the left in blue the airport digital systems, on the right in yellow group the physical components involved in baggage handling. Blue boxes $a_1$ to $a_4$ correspond to Figure 3 industrial process.

As shown in Figure 2, BHSs are connected to Airport Operation Databases (AODBs)[4]. From there, BHSs obtain flight departure schedules and baggage check-in information. BHSs are also connected to *airport physical equipment*

[3]For example, Wireshark allows users to provide decryption keys to record packet captures over ciphered data `https://wiki.wireshark.org/HowToDecrypt802.11`

[4]`https://dcs.aero/airport-operational-data-base-aodb/`

to identify bags and their position, validate their security screening status, and change the flow of the bags to ensure the correct route. BHSs detect new bags introduced or removed from conveyors and keep track of the bags. The control units identify bags based on their tag with *Automatic Tag Readers* (ATRs). *Explosive Detection Systems* (EDSs) screen bags for explosives. Finally, control units alter the bag trajectory to ensure the correct route, using *bag sorters* (e.g., diverters and pushers).

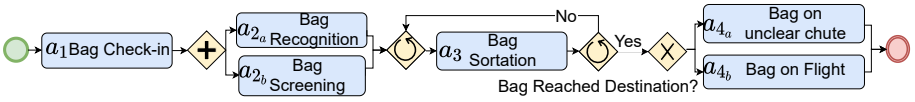


**Figure 3:** Industrial process representation of BHS activities.

The BHS activities for routing bags to their destination can be modeled as the industrial process described in Figure 3 using Business Process Modeling Notation (BPMN) [13]. In BPMN, operations are represented by *activities* (rectangles), which can be connected by an arrow ($\rightarrow$) indicating a sequence. BPMN can also represent decision paths in the sequence of activities, using *gateways* (diamonds). *Parallel gateways* (diamonds with a + sign) represent paths that occur in any order. *Exclusive gateways* (diamonds with $\times$) represent alternative paths based on predictable conditions. *Loop gateways* (diamonds with $\circlearrowleft$) represent paths that can occur multiple times.

The first BHS activity ($a_1$) begins when the bags are checked in and inserted into the conveyor; this initiates the bag tracking. The next activities happen in parallel (+ gateway) and correspond to the ATR recognizing the bag ($a_{2a}$) by scanning its tag, and the EDS screening the bag ($a_{2_b}$) and ensuring the BHS screening service. After the two parallel activities have been performed, the sortation unit proceeds to route the bag to the expected bag destination by issuing movement orders to control units operating the conveyors. This is reflected in the loop gateway ($\circlearrowleft$) 'Bag reached destination?' and its inner activity 'Bag Sortation' ($a_3$). When sortation is finished, bags are routed to one of two possible locations depending on their screening results. This choice between activities is modeled using the exclusive gateway ($\times$). Bags with unclear screening results go to the unclear chute ($a_{4a}$). Bags with clear results go to the flight ($a_{4b}$).

Formally, a process is modeled in the language $\mathcal{L}_p : a, \rightarrow, \times, +, \circlearrowleft$. The diagram of Figure 3 can be represented using the process notation [29]: $\rightarrow (a_1, +(a_{2a}, a_{2b}), \circlearrowleft (a_{3a}), \times(a_{4a}, a_{4b}))$. The sequences of activities (arrows in the

figure) are represented as $\rightarrow$, and the gateways (diamonds in the figure) are described as parallel $+$, exclusive $\times$, and loop $\circlearrowleft$. This work uses the $\mathcal{L}_p$ notation to model specifications of industrial processes extracted from network traffic.

### *3.3. Stage 1: Operation Registry*

The first stage aims to identify the CI operations and store them in a registry, $R_o$, used in the subsequent stages to build the specification. During the first stage, FingerCI functions as a network analyzer tool that *dissects CI communications*, i.e., captures packets and extracts $R_o$ entries according to predetermined acceptance criteria. These acceptance criteria are specified by experts and consist of *dissection rules* $\gamma$ that extract *events* ($e$) from packets. $R_o$ is then composed of a series of steps that reference: CI *operations data* ($e_o$), *industrial process data* related to the operation ($e_p$), and the operation *CI behavior* ($e_b$).

Other approaches require the participation of persons who are experts in the specific CI where they are deployed [32, 31, 40, 41, 39, 9, 26, 21, 25, 60], whereas ours only requires experts in the protocols, independently of the CI. Depending only on the protocol reduces manual effort because once $\gamma$ is defined for a given network protocol, it can be shared and reused in multiple CIs, thus reducing the amount of redundant effort each CI needs to do to construct their specification. This idea of supporting a $\gamma$ sharing community for CIs is realistic as it is similar to that in place in the TShark tool [11], which, for example, supports the MODBUS protocol through a plugin that is used by many energy CIs.

#### *3.3.1. CI network packet structure*

A network packet ($n$) encapsulates different protocols, containing several protocol fields ($n : n_{f1}, n_{f2}...n_{fn}$). Most CI-specific communication protocols, like Profinet [17], are application layer protocols encapsulated on top of TCP or UDP. Therefore, FingerCI first identifies the packet fields $n_f$ using the published protocol specification, e.g., IETF RFCs (Request For Comments) for IT network protocols and International Air Transport Association Resolution (IATA) manuals for airport-specific network protocols. The $n_f$ fields are thus identifiers of portions of the packet. They may refer to operation data $e_o$, namely, identifiers of the CI devices involved in the operations, (e.g., IP addresses, MAC addresses), services (e.g., network ports), and operation identifiers. The operation identifiers are typically found on CI application

network protocols and can be found, for example, on IATA baggage handling messages. According to IATA RP1745 [24], such messages can be of two types: Baggage Source Messages (BSM) that describe baggage check-in operations; or Baggage Process Messages (BPM) that describe sortation and screening operations. Since multiple airport systems can send BSM and BPM, it is also necessary to include the source and destination of the operation. For example, IATA BSMs are typically sent from AODB (source IP address, $n_{f_{src}}$) to BHS sortation units (destination IP address, $n_{f_{dst}}$). FingerCI uses $e_o$ data for building the network model during the second stage.

The $n_f$ protocol fields may also refer to the industrial process to which the operation belongs ($e_p$). Such fields are typically found as transaction identifiers in the CI network application protocol. The intuition behind the use of these fields is that most CI equipment interacts through transactions, where a set of operations are performed to meet a given goal. Given the uniqueness of the transaction identifiers used in CI systems, they are good candidates for inferring process instances during process discovery. For example, in BHSs, operations reference a bag tag, which can be a way to sequence the industrial process operations. Other CIs also have transaction codes for industrial processes. For example, in rail transportation, train movement operations can also be sequenced using the train's ID [31]. FingerCI uses $e_p$ during the third stage to create the process model.

$n_f$ fields may also refer to the physical behavior of the CI when the operation is executed ($e_b$). Such fields are typically found as values in the CI network application protocol. The intuition behind using these fields is that some CI operations occur based on physical conditions. For example, recalling the BHS example from Section 3.2, the decision gateway $\times(a_{4a}, a_{4b})$ to send bags to the unclear chute ($a_{4a}$) or load them to a plane ($a_{4b}$) depends on a condition. This condition results from the bag screening, which can be observed on a baggage process message (BPM) with a location code found in the $n_{f_{bpm}}$ field and with a screening message code $n_{f_{scrcode}}$ [24]. Behavior data can thus be divided into two categories: identifiers of CI physical assets ($e_{b_{id}}$) and the physical state value of those assets ($e_{b_{value}}$). In this example, the $n_f$ field that references the bag tag should be used as an $e_{b_{id}}$ identifier, while the $e_{b_{value}}$ values should be the fields indicating the status of the bag (i.e., checked in, cleared, or uncleared). FingerCI uses $e_b$ data during the fourth stage to construct the behavior model.

*3.3.2. Extracting the registry of operations*

FingerCI uses the dissection rules $\gamma$ that correlate the network packet fields for identifying $R_o$. Dissection rules receive a network packet $n$ containing a collection of packet fields $n_f$ and produce an event $e$ containing operation data $e_o$, process data $e_p$, and behavior data (identifier $e_{b_{id}}$ and value $e_{b_{value}}$). The correlation is given by the intersection ($\wedge$) for describing data that should be extracted from multiple fields and by the disjunction ($\vee$) to express multiple ways to obtain data. Dissection rules can be written in negative form ($\neg$). FingerCI dissection rules are expressed according to the language $\mathcal{L}_n :\rightarrow, \neg, \wedge, \vee, e_o, e_p, e_b, e_f, \gamma$. A dissection rule using $\mathcal{L}_n$ for Figure 3 $a_3$ is expressed as:

$$\gamma(n) : \begin{cases} n_{f_{src}} \wedge n_{f_{dst}} \wedge n_{f_{bpm}} \wedge n_{f_{scrcode}} \rightarrow e_o \\ n_{f_{tag}} \rightarrow e_p \\ n_{f_{tag}} \rightarrow e_{b_{id}} \\ n_{f_{clear}} \vee n_{f_{unclear}} \rightarrow e_{b_{value}} \end{cases} \tag{1}$$

This formula for $\gamma(n)$ means that a given network packet $n$ will be considered a well-formed screening packet (according to IATA RP1745) (1) if it contains the fields $\vec{n_f}$ $(n_{f_{src}}, n_{f_{dst}}, \dots)$, and (2) if they allow extracting the data items $e_o, e_p, e_{b_{id}}, e_{b_{value}}$. These items are: a) operation data $e_o$ (a pattern that identifies the screening operation, in this case, the IPs and BPM message ID); b) process data $e_p$ (an identifier of the process the message belongs to, e.g., a bag tag in this scenario); c) behavior data $e_{b_*}$, composed of c1) an identifier $e_{b_{id}}$ (the object used to validate the operation, in this case, the bag tag), and c2) a value $e_{b_{value}}$ (the value used for that validation, in this case, the screening status, cleared or not).

Dissection rules are not CI-dependent, but vendor/version-dependent. That is, most CIs use contractor systems that are installed and maintained by the manufacturer. To maximize revenue, the same system is usually found in similar CIs. In this case, dissection rules applied to a particular CI system (e.g., BHS) can be applied to other CIs where the system is deployed (different airports or even transports, like rail or boats). As such, if the manufacturer constructs the dissector for the communications sent/received from their devices, multiple CIs that use the devices can have the FingerCI fully automated in their infrastructure. An example of a successful dissection rule-sharing community is Wireshark, where a dissection rule created for a SCADA system (which commonly uses S7 protocol) is useful for CIs, and research communities, among other interested parties.

Therefore, at this stage, FingerCI makes use of a collection of expert-

defined dissection rules $\gamma : \gamma_1, \gamma_2, ..., \gamma_n$ for classifying packets. Packets that match any of the $\gamma$ functions are considered relevant to the other stages, while non-matching packets are discarded. $e_o$ and $e_p$ should be specific enough not to have collisions between operations and processes described by other dissection rules, but should also cover the complete execution of the operations described in the network capture. Given this selection's complexity, FingerCI delegates this decision to the expert.

The first stage of FingerCI improves the current specification inference methods [7, 57]. The first improvement is the abstraction of network fields that allow $\gamma$ functions to be compatible with any network packet regardless of the protocols underlying them. This design allows compatibility with several protocols for which there is a standard to extract the $n_f$ fields. Notorious examples of supported CI protocols include MODBUS, Profinet, or IEC 61850[5]. The second improvement is the abstraction found on $\gamma$ functions. It allows correlation rules not to be focused on specific network protocols and uses fields present on the complete stack of protocols. For example, a $\gamma$ function can identify BHS operations by correlating the IP address fields found on the Ethernet protocol with operation codes found in IATA application protocols. The third improvement is that fingerprints can be created passively using network captures, thus complying with current CI security practices. Combining these three improvements offers an innovative way of obtaining the necessary event data for fingerprinting CI processes.

### *3.4. Stage 2: Network Model*

After the registry of operations $R_o$ has been identified, the second stage can be executed. In this stage, FingerCI constructs the network model ($M_n$) and comprises a set of identification rules based on REGEXP that uniquely identify the CI operations reported on $R_o$. FingerCI traverses the registry of operations to construct the model and produces the identification rules ($\beta$) of CI operations. These rules are created based on network packets identified by the dissection rules $\gamma$ during the first stage.

The network model is described by the grammar in Table 2. Following the grammar structure, $M_n$ is composed of one $\beta$ rule for each operation. Each operation representation is then constructed using four main elements:

[5]The abstraction of $n_f$ offers compatibility with the listing of protocols presented in `https://wiki.wireshark.org/ProtocolReference`, and others since, for example, IATA protocols are not currently supported by Wireshark.

<β> ::= <γ> | <γ> ; <β>
<γ> ::= <$e_n$> , <$e_o$> , <$e_p$> , <$e_b$>
<$e_b$> ::= ( <$b_i$> ; <$b_v$> )
<$e_n$> ::= <*src*> - <*dst*> - <*protocols*>
<$e_o$,$e_p$,$b_i$,$b_v$> ::= <*statement*>
<*src*,*dst*> ::= <*value*>

<*protocols*> ::= <*value*> | ( <*value*> <*condition*> <*protocols*> )
<*statement*> ::= <$n_f$> | ( <$n_f$> <*condition*> <*statement*> )
<*condition*> ::= ∧ | ∨
<$n_f$> ::= [A-Za-z0-9]+ : <*value*> | ¬ <$n_f$>
<*value*> ::= [A-Za-z0-9]+ | ( [A-Za-z0-9]+ <*condition*> <*value*> )

**Table 2:** Network model grammar definition.

network data $e_n$, operation data $e_o$, process data $e_p$, and behavior data $e_b$. Each element is composed of a *statement* that includes fields $n_f$ described on the $\gamma$ dissection rules, and the *value* observed during the first stage of the fingerprinting. For example, the identification pattern for the screening operation is as follows: $\beta \rightarrow IP_A - IP_S - IATA, (n_{f_{bpm}} : BPM \wedge n_{f_{scrcode}} : C \vee U), n_{f_{tag}} : BAG, (n_{f_{tag}} : BAG; (n_{f_{bpm}} : BPM \wedge n_{f_{scrcode}} : (C \vee U)))$. $e_n$ contains $IP_A$ and $IP_S$ addresses and the *IATA* protocol identifier. $e_o$ includes the $n_{f_{bpm}}$ and $n_{f_{scrcode}}$ screening message identifiers. $e_p$ is the $n_{f_{tag}}$ identifier for the bag. $e_b$ is comprised of the $n_{f_{tag}}$ and $n_{f_{scrcode}}$ that contains the screening result, clear or unclear. The resulting identification patterns $\beta$ allow sensors to perform data collection and identify: packets to monitor using the $e_n$ data; CI operations to analyze using $e_o$ data; Industrial processes using $e_p$ data; and CI state during the operation using $e_b$ data. Using this specification language, we were able to integrate our specification into two SBSs without modifications to their software. One, M-PE [36], is widely used in process management, which shows the compatibility of FingerCI with current SBSs.

### *3.5. Stage 3: Process Model*

$R_o$ also enables FingerCI to build the process model $M_p$ using existing process discovery techniques. Although process discovery has been applied to specific network protocols [57], FingerCI adds novelty by supporting multiple network protocols simultaneously. It solely relies on the operation data $e_o$ and process data $e_p$ present in the $M_n$ model.

#### *3.5.1. Preprocessing $R_o$*

To create the process model, FingerCI traverses the registry of CI operations. It uses operation data $e_o$ and process data $e_p$ to perform activity extraction. The output generated by this step contains all activities identified, i.e., executions of operations reported in each event, grouped by the process instance they belong to, and in chronological order. Table 3 shows some process instances of the CI process example described throughout the

document (Figure 3). The process instances (lines - PI) are organized by the positions each activity is executed (columns - AP).

| Process/Activity | $AP_1$ | $AP_2$ | $AP_3$ | $AP_4$ | $AP_5$ | $AP_6$ |
|---|---|---|---|---|---|---|
| $PI_1$ | $a_1$ | $a_{2_b}$ | $a_{2_a}$ | $a_3$ | $a_3$ | $a_{4_a}$ |
| $PI_2$ | $a_1$ | $a_{2_a}$ | $a_{2_b}$ | $a_3$ | $a_3$ | $a_{4_b}$ |
| $PI_3$ | $a_1$ | $a_{2_b}$ | $a_{2_a}$ | $a_3$ | $a_3$ | $a_{4_b}$ |

**Table 3:** Process instances (PI) for the BHS example of Figure 3. Lines represent PI, and columns are activities positions (AP).

FingerCI then proceeds to create a *directly-follows graph* [54]. This type of graph (exemplified in Figure 4) represents all process instances in the log by describing all possible transitions (edges) between activities. The graph is constructed by parsing the process instances previously identified and drawing arrows for each transition verified in the graph, starting from the first activity in the process instance. For example, Figure 4 is the directly-follows graph that results from the process instances described in Table 3. The graph starts in $a_1$, since all process instances start on that activity, and, from that, identifies the transitions. FingerCI then constructs from the graph a process specification.

*3.5.2. Extracting $M_p$*

FingerCI inspects activities and corresponding process instances to create a *process model* ($M_p$) [54], a generalized specification of a log that summarizes the order in which activities should be executed to achieve process correctness. This generalization into a process model is achieved using the *inductive miner* process discovery technique that studies the relationships between activities according to the order of occurrence. It constructs the process model in language $\mathcal{L}_p : a, \rightarrow, \times, +, \circlearrowleft$.

The *inductive miner* technique [29] creates process models using a divide-and-conquer approach. It sections the graph into smaller regions according to patterns. The algorithm is recursively executed on the smaller regions until sectioning is no longer possible. There is a sectioning pattern for each $\mathcal{L}_p$ operator, $\rightarrow$, $\times$, $+$ and $\circlearrowleft$. As can be seen in the example graph in Figure 4, the inductive miner divides the graph into four sections using the sequence ($\rightarrow$) sectioning pattern (blue lines). The algorithm selects this division since

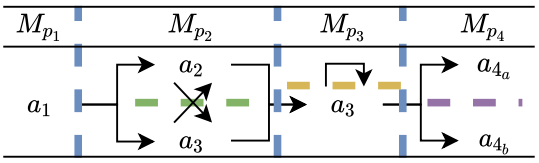


**Figure 4:** Directly-follows graph (arrows) and inductive miner execution ($M_p$ regions and colored lines) for Table 3.

all edges passing the blue lines go from a node on the left (predecessor) to a node on the right (successor). Thus, the result of the main run is $M_p :\rightarrow (a_1, M_{p_2}, M_{p_3}, M_{p_4})$, which represents the sequence of operations involved in the process. The inductive miner analyzes the $M_{p_2}, M_{p_3}, M_{p_4}$ regions separately. $M_{p_2}$ is obtained using the parallel ($+$) sectioning pattern which implies that edges cross the green line. $M_{p_3}$ is obtained using the $\circlearrowleft$ sectioning pattern since the edges move to already visited activities. $M_{p_4}$ is obtained using the exclusive ($\times$) sectioning pattern since the edges do not cross the purple line. As a result, the following process model is obtained $M_p :\rightarrow (a_1, +(a_{2a}, a_{2b}), \circlearrowleft (a_{3a}), \times(a_{4a}, a_{4b}))$. This process model represents the BHS routing bags to their destination (Figure 3).

### *3.6. Stage 4: Behavior Model*

Process models provide a sequential ordering of activities in a process. These models act as *causal rules* for activities (e.g., bag sortation must happen after bag recognition and screening have finished). However, they do not provide monitoring rules for an activity to be considered a *legitimate action* (e.g., bags can only be routed to the destination flight if they have a clear screening result). To identify those rules, FingerCI extends process models with behavior models $M_b$. Behavior models are novel because they provide the necessary CI state conditions to validate the correctness of industrial processes, which is something previous process discovery methods did not [54, 58, 29, 53, 40, 41, 39]. Figure 5 illustrates the $M_b$ behavior model of the BHS example being given throughout this article. As the figure shows, $M_b$ state conditions include the physical state of the equipment ($e_{b_{value}}$) before and after executing activities of the process model (i.e., *precondition* and *postconditions*). Furthermore, $M_b$ includes the state conditions for gateway executions based on the behavior of the inner activities and frequency counting for loop gateway stopping conditions. The resulting conditions remove

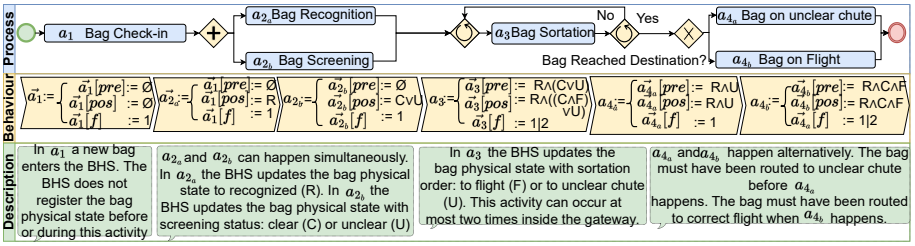


**Figure 5:** FINGERCI (top-down) process $M_p$, behavior $M_b$ , description.

the ambiguity of the process model and can be used, for example, by IDSs as detection rules.

### *3.6.1. Constructing $M_b$*

The $M_b$ behavior model is obtained based on the registry of operations $R_o$ that is traversed, and all activity state conditions are registered based on the physical status of the equipment reported on $e_{b_{value}}$. Activity *preconditions* are the intersection of all the physical equipment states before the activity occurs (obtained from the $e_{b_{value}}$ of the previously inspected activities). Activity *postconditions* are the intersection of the physical equipment states after the activity occurs (obtained from the $e_{b_{value}}$ fields inside the activity). Given the activities in different processes, $M_b$ includes as activity state conditions the disjunction of all preconditions and postconditions seen during activity executions. For example, in Figure 5, different process executions may refer to bags with different statuses. Thus, a precondition for $a_3$ is the bag being clear or unclear.

Aside from inferring activity state conditions, the $M_b$ model also provides gateway validation conditions. Such conditions include activity frequency counting and state conditions for validation.

*Frequency counting* measures the occurrence of all elements inside a process. This is important to determine the number of times an activity or region occurs inside a loop gateway (↻). The frequency of the loop gateway inner element is given by the disjunction of the number of times they appear in each process execution. For example, the sortation activity ($a_3$) of Figure 5 occurs once or twice in a process.

*State conditions* differ by gateway type. Exclusive gateway (×) conditions are given by the disjunction of their inner element conditions since their inner regions are executed alternately. For instance, the exclusive gate-

way in Figure 5 has the precondition of the bag being cleared or uncleared. This precondition covers $a_{4_a}$ (if the bag is cleared) and $a_{4_b}$ (if it is unclear). Parallel gateways (+) preconditions must ensure all inner regions can be executed. Thus, they are given by the intersection ($\wedge$) of the inner element conditions. For example, the parallel gateway in Figure 5 has the postcondition, $R \wedge (C \vee U)$. It states that the bag must be recognized by the ATR ($R$) and screened (clear or unclear, $C \vee U$) after the gateway is completed. This postcondition derives from $a_1$ postcondition ($R$) and $a_2$ postcondition ($C \vee U$). Loop gateways ($\circlearrowleft$) can be limited by the frequency of the inner region or may have the physical state of the equipment as a stopping condition. The stopping physical conditions for a loop gateway are given by the inner element postconditions that stop the loop, i.e., the $e_{b_value}$ contained in the postcondition that is not listed as a precondition. For instance, the loop gateway in Figure 5 has the stopping condition of routing bags to flight. This condition is listed as an $a_3$ postcondition and not as a precondition. This gateway is also limited by $a_3$ frequency, once or twice, even if no stopping condition is reached. For example, in the case that bags go to unclear chute, $a_3$ is executed twice.

### *3.6.2. Structuring $M_b$*

| | |
|---|---|
| <$M_b$> ::= <*section*> ; <$e_p$> | <$e_b$> ::= (<$b_i$>-<$b_v$>) |
| <*section*> ::= <*expression*> : <*frequency*> | <$b_v$> ::= <*pre − condition*>,<*post − condition*> |
| <*expression*> ::= <*operator*> ( <branches> ) \| <*activity*> | <$e_p$,$e_o$,$b_i$,*pre − condition*,*post − condition*> ::= <*statement*> |
| <*operator*> ::= → \| ↺ \| × \| + | <*statement*> ::= <$n_f$> \| ( <$n_f$> ∨ <*statement*> ) \| ( <$n_f$> ∧ <*statement*> ) |
| <*branches*> ::= <*section*>;<*branches*> \| <*section*> | <$n_f$> ::= [A-Za-z0-9]+ : REGEXP \| ¬ <$n_f$> |
| <*activity*> ::= (<$e_o$>:<$e_b$>) | <*frequency*> ::= [0-9]+ |

**Table 4:** behavior model is described by the following grammar

The behavior model is given by the grammar in Table 4. The grammar structure, $M_b$ of an $e_p$ process comprises nested <*section*> rules. Each rule corresponds to $M_p$ sectioning and is constructed using two elements, expression and frequency. The *expression* is built over the $M_p$ process model and corresponds to the inner elements of the sectioning. The *frequency* is the number of times that a section can occur. The <*expression*> has an $L_p$ operator ($\rightarrow, \circlearrowleft, \times, +$) and has either inner sections or activities. Activities are identified by $e_o$ data and respect rules described on $e_b$ data. The $e_o$ is the same as the $M_n$ and $M_p$, but $e_b$ is different. The $M_b$ model $e_b$ data contains preconditions and postconditions REGEXP for activity validation. For example, the partial $M_b$ for the sectioning $+(a_{2a}, a_{2b})$ is given as follows: $M_b :\rightarrow +((a_{2a} : ((\emptyset - n_{f_{tag}} : R)) : 1) : 1; (a_{2b} : ((\emptyset - n_{f_{scrcode}} : C|U)) : 1)) : 1.$ $a_{2a}$

and $a_{2b}$ are simplified $e_o$ data (described in Section 3.4)[6]. $n_{f_{tag}}$ is the bag tag and $n_{f_{scrcode}}$ is the screening status. Integer values in each section/activity represent frequency. $\emptyset$ represents the absence of a pre-condition, R and C|U represent the REGEXP for the activity post-conditions. These preconditions are also expressed in the yellow boxes in Figure 5. The resulting $M_b$ model allows process fault detection and, in conjunction with $M_n$ and $M_p$ models, can build an SBS that is configured automatically, such as an IDS. This automatic procedure does not require human experts to write the specifications, but they may review and modify them.

## 4. Behavior Model and Conformity Checking

FingerCI monitoring rules are expressed as sequence diagrams that can be easily audited by CI experts and can be used to configure existing specification-based systems. FingerCI can help configure SBSs, as their detection validates the conformity between the data collected from the sensors and the specification. This conformity check can be defined [35] as the ability to attest whether a log *aligns* with a specification. A log is said to be aligned when the sequence of operations reported in each transaction respects the specification $M$. Moreover, in the case where behavior detection rules are included in $M$, a log aligns only with $M$ if both the sequence and the behavior rules are respected. The log alignment between the transaction $t$ reported in log $\mathbb{L}$ and the specification $M$ is verified by mapping $t$ operations to $M$. The alignment attesting between a transaction $t : o'_{i=1}, ..., o'_{i=|T|}$ and the specification $M : o_{z=1}, ..., o_{z=|M|}$ is as follows:

1. $(o'_i, \rightarrow)$ is a transaction valid move (◖) iff $o_i \in t$
2. $(\rightarrow, o_z)$ is a specification valid move (◗) iff $o_z \in M$
3. $(o'_i, o_z)$ is a valid move (●) iff $o_z \in M \wedge o'_i \in t \wedge o_z = o'_i \wedge b_z = o_{b_i}$
4. $(o'_i, o_z)$ is a invalid move ($\otimes$) iff $o_z \in M \wedge o_i \in T \wedge o_z \neq o_i \vee b_z \neq b_i$

For example, transaction $t_1 : a_1, a_{2_a} = F1, a_{2_b} = F1, U, a_3 = F1, U, a_{4_b} = F1, U$ does not align with the specification of Figure 5. All $t_1$ operations

[6]These activities correspond to the ATR recognizing the bag ($a_{2a}$) by scanning its tag, and the EDS screening the bag ($a_{2_b}$).

match the specification, except $a_{4_b}$. This non-alignment can be considered an anomaly (since it violates the safety conditions imposed by the specification). To validate the utility of FingerCI specifications, this work created a prototype of an IDS. This IDS (Figure 1) has sensors installed on the CI to identify operations using $M_n$ specifications. The operations are then validated by the IDS using process and behavior specifications ($M_p$ and $M_n$ specification).

## 5. Implementation

The implementation of FingerCI aims to confirm that it is possible to extract CI process specifications based on network traffic. It also aims to confirm that its use requires little human assistance and that it produces specifications that are detailed enough to configure an SBS. The implemented prototype integrates FingerCI with BP-IDS [32, 31], an SBIDS that identifies anomalies in CIs. The prototype also integrates FingerCI with a process management system called *Multi-Perspective Process Explorer* [36] (M-PE), which verifies compliance between logs and specifications.

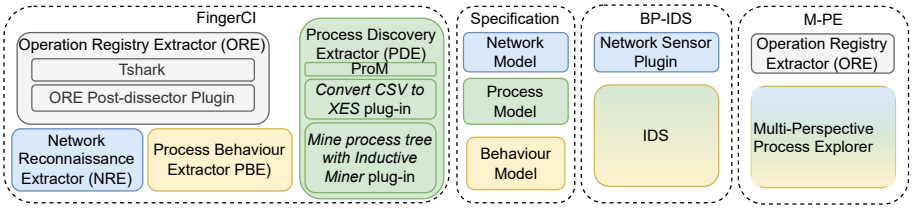


**Figure 6:** Architecture of FingerCI integrated with BP-IDS and M-PE.

The architecture of the prototype is represented in Figure 6. The figure shows the internal structure of FingerCI and its integration with BP-IDS and M-PE. FingerCI is implemented has four modules. Each module corresponds to a fingerprinting stage (as described in Section 3). *Stage 1* involves obtaining the $R_o$ registry of CI operations, and is implemented by the *Operation Registry Extractor (ORE)* module. ORE dissects CI communications, i.e., PCAPs, and extracts $R_o$ according to predetermined acceptance criteria. ORE is developed as a Lua post-dissector plugin[7] for the TShark network

[7] `https://wiki.wireshark.org/Lua/Examples#editing_columns`

monitoring tool [11]. The plugin first parses the network packet attributes ($n_f$) identified by the TShark protocol dissection plugins, and decides based on the $n_f$ fields, identified by those plugins, if the network packet constitutes a relevant CI operation to include in the specification. This decision is made based on ontology $\gamma$ dissection rules that correlate specific $n_f$s for identifying the registry of operations. *Stage 2* constructs the network model $M_n$ and is implemented by the Network Reconnaissance Extractor module (NRE). This module uses $R_o$, to produce the $M_n$ network model in Backus–Naur Form grammar format (as presented in Section 3.4). NRE is comprised of *awk* scripts that filter the $R_o$ to generate the model. *Stage 3* involves creating the $M_p$ process model and is handled by the Process Discovery Extractor (PDE) component. PDE is developed as a wrapper to the process mining library (ProM) [55] that parses the $R_o$, and selects the process and operation identifiers required for ProM to produce the process model (based on $e_p$ and $e_o$ respectively). ProM provides multiple plugins to parse logs extract process models, and analyze their performance. PDE uses the *Mine process tree with Inductive Miner* to construct the process in the $L_p$ (as described in Section 3)[8]. PDE interacts with the ProM tool using Java provided as an argument to the ProM command line interface. This method fully automates the construction of the process model without requiring human-in-the-loop (which would be the case if PDE used the ProM user interface). Finally, *Stage 4* is implemented by the Process Behavior Extractor (PBE) component, a Java-based library that parses the registry of operations and produces the $M_b$ behavior model.

FingerCI was designed to be easily integrated with systems that rely on specifications [32, 31, 10, 14]. In this prototype, FingerCI was integrated with BP-IDS and M-PE as a proof-of-concept (Figure 6). BP-IDS uses a business process specification to model the accepted behavior of a CI. BP-IDS detects, in real-time, abnormal machine operations on CI services by comparing the behavior observed by network sensors with specifications. M-PE also performs comparisons to find noncompliance but does not work in real-time. BP-IDS has been previously validated on a railway transportation CI [31]. The $M_n$ serves as the specification for the BP-IDS sensor. This

[8]PDE uses the default configuration of the plugin but disables the noise threshold of the inductive miner algorithm to zero, since it allows the process model to fit 100% to the $R_o$. This noise threshold was disabled since it excludes infrequent operations from the resulting $M_p$ which could influence the precision and accuracy of the SBSs.

specification provides the REGEXP for parsing network traffic and identifying CI processes executed. The $e_n$ collected the packets that contained relevant CI process data. $e_o$ pinpointed the operation identifiers referenced in the collected packets. $e_p$ captured the process identifiers referenced by the operations. $e_b$ extracted the conditions to which the operation was executed. These REGEXP allowed the BP-IDS sensor to collect the data for anomaly detection. In the case of M-PE integration, we used ORE to parse network traffic since M-PE does not have network sensors. The $M_p$ process model specifies the expected sequence of operations for BP-IDS and M-PE verification. With the sequence of operations, they can verify the ordering of the operations reported by its sensors and detect anomalous process executions (deviations from $M_p$). The $M_b$ behavior model offers the behavior rules that BP-IDS and M-PE can use to validate processes. Each $<activity>$ rule of the $M_b$ allows them to validate the $e_b$ data collected from the sensors, and detect repeating activities thanks to the $<activity>$ frequency threshold. Gateway rules ($\times$ and $\circlearrowleft$) of the $M_b$ allow them to validate gateways. The validation is done by checking whether the processes are compliant with the gateway rules. Thus, using the $M_b$ model, BP-IDS and M-PE can detect anomalous operations that compromise the CI process by altering the physical equipment to an invalid state. The FingerCI prototype reduced the effort required to write BP-IDS and M-PE specifications while retaining the main functionalities.

## 6. Evaluation

The objective of the evaluation of FingerCI is to answer the following research questions (RQ):

RQ1 How long does it take to produce a fingerprint (Section 6.1)?

RQ2 Are models reliable for intrusion detection (Section 6.2)?

RQ3 Do models correctly describe the CI (Section 6.3)?

The evaluation was based on the Airbus simulation platform[9]. This platform offered a digital twin of an airport network infrastructure, with VLAN-connected virtual machines. The platform mimicked a BHS manufactured

[9] `https://www.cyber.airbus.com/cyberrange/`

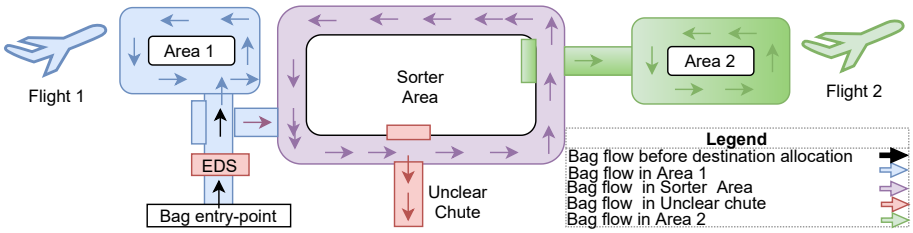


**Figure 7:** Testbed BHS organization.

by the Alstef Group[10]. The platform included a virtual AODB that provided the BHS sortation unit with fictitious identifiers of bags and flights assigned to BHS locations. The simulation also included virtual physical equipment (EDS, ATR, and conveyors) managed by PLCs connected to the BHS sortation unit in the simulation platform. The simulation used Emulate3D[11] to mimic the physical equipment. Emulate3D is a high-fidelity emulator used by BHS providers to test their systems against contractual conditions before deployment in actual airports.

The experiments used network packet captures (PCAPs) recorded on the Airbus simulation platform operating continuously for 25 hours. The PCAPs were recorded in promiscuous mode through a port mirror on the simulation platform router. As shown in Figure 7, the BHS sends bags to three locations: two flights and one unclear bag *chute U* (in red). *Flight 1* was assigned to *chute 1* (in blue, located in the same area 1 of the check-in and EDS equipment). *Flight 2* was assigned to *chute 2* (in green, located in area 2, accessible through the BHS sorter area). This architecture is found in civil airports, which differ mostly in terms of the number of flight chutes. Emulate3D spawned one bag every 30 seconds, with bags being routed from check-in counters to *chute 1* in $1m30s$, to *chute U* in $2m05s$, and to *chute 2* in $2m35s$. AODB sent, to the BHS sortation unit, the check-in information of the bags of 200 bags (100 bags per flight) which would then be introduced repeatedly by random order[12] on the BHS through the baggage check-in area.

[10] `https://alstefgroup.com/baggage-handling/`

[11] `https://www.demo3d.com/Baggage-Handling/`

[12] Reintroduction of bags can often happen at airports due to logistic reasons. Bags can be stored on early bag store (EBS) systems and reintroduced into the sorting system when required to be delivered to baggage handlers. In this evaluation, bags are introduced in

The automatic *reintroduction of bags* made by Emulate3D allowed us to have a total of 2045 bags during 25 hours. FingerCI inspected 89 bags to generate a fingerprint. 1956 bags were used to validate the accuracy and correctness of the fingerprint.

### *6.1. Fingerprint Performance*

To answer RQ1, the performance of FingerCI was measured and compared with the performance of TShark. For this experiment, 8 different network traffic datasets were created, with durations of 1 to 8 hours. The experiments measured the variation between FingerCI and TShark processing times on a 64-bit Ubuntu VM with 2048 MB RAM and a 1 CPU processor i7-8565U (1.8GHz up to 1.99GHz). For each execution, it was measured (Figure 8) the average time (based on 30 repetitions) taken for FingerCI as a whole, its components (ORE, NRE, PDE, and PBE), and TShark. As seen

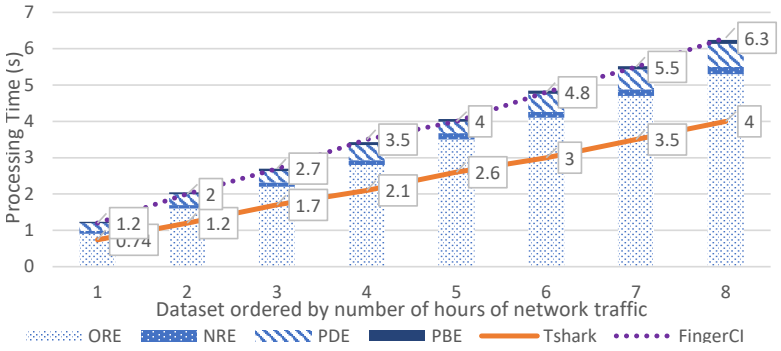


**Figure 8:** Average execution times (YY) of FingerCI (line), its components (bars), and TShark (line) for 8 datasets (XX).

in Figure 8, for all datasets, in terms of computational costs, FingerCI requires, on average, 62% more than the time required for TShark to process network traffic when used in stand-alone mode. The computational costs required by the PDE to generate the process diagram, NRE to generate the network model, and the PBE to generate the behavior model represent in

random order to simulate the EBS reintroduction of bags. For more information about EBS, refer to the following video (not created by authors): `https://www.youtube.com/watch?v=d6_0eC0qZPE`

total only 17% of total processing time (11%, 2%, and 4%, respectively); ORE consumes most of the processing time (83%). ORE introduces a Wireshark Lua post-dissector plugin that intercepts all network traffic analyzed and extracts the fingerprint fields. Although the Lua plugin is an easy way to extend TShark/Wireshark, its performance limitations are well-known. Replacing Lua with a C-based plugin would improve performance. Despite the increase in computational time, fingerprint generation takes at most 12 seconds based on eight hours of network traffic. Considering that configuring a specification-based solution often takes a high number of hours[13], FingerCI can be used as a first approach for generating the fingerprint and then placing a human-in-the-loop to perform the necessary corrections to get an accurate fingerprint. Therefore, the experiments conducted show FingerCI can save hours of expert work.

### *6.2. Intrusion Detection*

To answer RQ2 and validate the accuracy of an SBIDS, BP-IDS was configured with the fingerprint generated by FingerCI. The fingerprint included AODB, control unit, and sortation unit interactions during one hour. During this hour, FingerCI inspected flight information messages, bag check-in, screening, and sortation operations. BP-IDS monitored the remaining network traffic (24 hours), summing up a total of 1956 bags monitored. During two hours, the simulation platform forced the BHS into two abnormal situations. For the first one, *screening anomaly*, the EDS screening results of 42 bags were changed by the simulation platform to route unclear bags to flight instead of the unclear chute (7 unclear bags) and clear bags to the unclear chute (33 clear bags). The second abnormal situation, the *sortation anomaly*, overwrote the BHS sortation messages to falsely route bags to different destinations. In this abnormal situation, the simulation platform changed sortation orders for 36 bags. Bags expected for *flight 1* went to *flight 2* (20) and vice-versa ( 16). As shown in Table 5, the fingerprints generated by FingerCI to profile BHS behavior were very accurate when detecting the two abnormal situations, with an accuracy of 99.89%. This good result in terms of accuracy is due to the combination of process mining with the behavior model created by FingerCI that makes the verification deterministic. The deterministic verification is confirmed by the fact that BP-IDS does not

[13]Typically experts take at least 6 hours to write a specification [51].

| Abnormal Situation | Valid bags | Anomalous bags | FN | FP | Accuracy | FPR |
|---|---|---|---|---|---|---|
| Screening | 1914 | 42 | 0 | 0 | 100% | 0% |
| Sortation | 1920 | 36 | 0 | 2 | 99.89% | 6% |
| Both | 1878 | 78 | 0 | 2 | 99.89% | 3% |

**Table 5:** Accuracy of the SBIDS (BP-IDS) with FingerCI.

have false negatives (FN) regarding the sortation and screening anomalies, assuring a high fidelity result. There were two false positives (FP) from the 1956 bags inspected in the 24 hours of BHS functioning. The FP, in this case, was due to the reintroduction of anomalous bags in the system. In this case, BP-IDS had information cached from previous inspections of the sortation anomaly and considered the bags to be anomalous when no anomaly was taking place. BP-IDS achieved a 0% false positive rate (FPR) for the screening anomaly and 6% FPR for the sortation anomaly, resulting in a total of 3% FPR for the whole abnormal period. These results show that using the FingerCI process and behavior models allows SBIDSs to achieve the high fidelity required to identify anomalies that may affect CIs.

We have also compared our detection results with a previous application of ComSEC in the same dataset [5]. ComSEC is a bump-in-the-wire technology for detecting integrity and replay attacks. ComSEC does not resort to specifications and uses digital signatures and timestamp validations to detect attacks. ComSEC case study is adequate for some CIs which look to upgrade network traffic connections with security assurances. For validation of BP-IDS results, we have compared ComSEC with BP-IDS (equipped with FingerCI specifications) and used the same conditions previously described. Since ComSEC works at the network packet level, ComSEC produced 220 alarms (one per network packet) while BP-IDS had 78 alarms (one per bag, an aggregation of network packets). ComSEC had 99.99(9)% accuracy, with 3.2% (similar to BP-IDS in Table 5), with 7 false alarms in contrast with 2 BP-IDS false alarms. Based on these results, ComSEC could provide an early warning of packet integrity violations, but it is more difficult for experts to understand based on network packet payload what caused the alarm and find the appropriate measures. FingerCI specifications, included in BP-IDS, provide alarms that are better to understand the reason for an alarm to be triggered (non-compliance to specification) and also understand

the appropriate measures based on the physical impact of the anomaly. In this scenario, those decisions would be to remove the abnormal bags before destination flights take off.

### *6.3. Process Model Validation*

To answer RQ3 and validate the correctness of the process models, we conducted a *conformance analysis* [45, 56] of the specification created by FingerCI. A conformance analysis measures the correctness of a process model based on how well the model represents the activities observed on a given activity log. The specification from RQ2 was reused to perform this analysis. The ProM plugin used for conformance checking [56] compared the process model with the BHS complete network traffic (24 hours), excluding anomalous bags. The objective of excluding these bags was to measure how well the process model represents the correct behavior of the BHS (malicious activity is covered in RQ2). The experiments followed the conformance analysis approach proposed by Rozinat and van der Aalst [45], which measures log correctness in two dimensions: model fitness and appropriateness.

The *model fitness* dimension compares the execution of the process instance (in this case sortation of bag) $k$ described on a log with the process model. The process $i$ activity execution described on a log is quantified with produced tokens $p_i$, while executed activities in the model are quantified as consumed tokens $c_i$. A perfect model and log fitness will mean the number of tokens produced $\sum_{i=1}^{k} p_i$ and consumed $\sum_{i=1}^{k} c_i$ are the same. However, an unfitted log can have missing tokens $m_i$ when there are missing conditions in the model to satisfy process execution or classified as remaining tokens $r_i$ when the log lacks activities that satisfy the process model conditions. The occurrence of $m_i$ tokens increases $c_i$ while $r_i$ increases $p_i$. Model fitness is given by:

$$f = \frac{1}{2}(1 - \frac{\sum_{i=1}^{k} m_i}{\sum_{i=1}^{k} c_i}) + \frac{1}{2}(1 - \frac{\sum_{i=1}^{k} r_i}{\sum_{i=1}^{k} p_i}) \tag{2}$$

The tests made in this experiment with $k = 1878$ process instances show the log execution resulted in a total of $\sum_{i=1}^{k} p_i = 48788$ produced, $\sum_{i=1}^{k} c_i = 48788$ consumed, $\sum_{i=1}^{k} m_i = 8$ missing, and $\sum_{i=1}^{k} r_i = 10$ remaining tokens. The 18 unfitted tokens (total of remaining and missing tokens) occurred on two process instances that were executed during a temporary delay in communications exchanged between sortation units and control units that forced message retransmissions of sortation decisions. The retransmissions were classified as misplaced activities in the process model. The experiment shows FingerCI process model obtained a fitness measurement of 99.98% by applying the fit-

ness formula. The FingerCI process models have shown high fidelity in specifying CI operations.

*Model appropriateness* measures the model's comprehensibility to humans and the precision in describing only the log's observed processes. The comprehensibility $c$ is measured by relating the observable and non-observable transitions. The observable transitions $|T|$ are the movement between one activity of the process model to another. The non-observable transactions can be either: redundant invisible $|T_{RI}|$ or alternative duplicate $|T_{AD}|$. $|T_{RI}|$ transitions are the ones that were not executed by any process instance. $|T_{AD}|$ are the repeated transitions that form an alternative repeated path to one activity. Compreehensibility is measured by:

$$c = (|T| - |T_{AD}| + |T_{RI}|)/|T| \tag{3}$$

The tests made in this experience with $k = 1878$ process instances show the log execution resulted on $|T_{AD}| = 0$ alternative duplicate transitions, $|T_{RI}| = 5$ of the total of $|T| = 36$ transitions represented on the process model. The FingerCI process model obtained a comprehensibility score of 86%. This level of comprehensibility allows experts to understand the process model and pinpoint the overall execution. As seen in the validation reasoning described in the previous section, despite the invisible tasks, the FingerCI process and behavior model offered an understandable explanation of the overall functioning of the BHS and can be interpreted by experts.

Process model precision is calculated by relating the model to the log executions and determining whether the observed executions represent the complete process model or partially. Process model precision $P$ is measured by relating the number of transitions $x_i$ with the number of visible transitions $T_V$ represented on the process model. The precision measurement can be calculated using the formula:

$$P = (T_V - \frac{\sum_{i=1}^{k} x_i}{k})/(T_V - 1) \tag{4}$$

The tests made in this experience with $k = 5987$ process instances show the log execution resulted in a total of $\sum_{i=1}^{k} x_i = 33431$ enabled transitions of the process model $T_V = 26$ visible transitions. By applying the precision formula, the experiment shows FingerCI process model obtained a precision appropriateness measurement of 82%. FingerCI displays high fidelity to represent the CI specifications.

Furthermore, we have also studied the impact of varying the amount of bags to infer FingerCI specifications. As can be seen in Figure 9, we have used M-PE to measure model fitness (blue) and non-compliant operations (orange) for specifications created with 1 to 89 bags (the amount used in

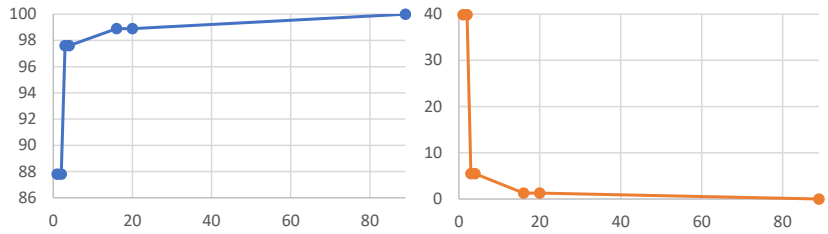


**Figure 9:** M-PE fitness (blue) and non-compliant operations (orange) (1878 bags, y-axis) as more bags are fed into a FingerCI specification (1-89, x-axis).

previous evaluations). It can be seen in this evaluation, results vary from 88% to 99.98% model fitness. And errors from 40% to 0%. Showing FingerCI just needs to inspect 1 to 4 bags to infer specifications that fit the CI observations, but requires more bags 89 to handle the less frequent BHS process variations. We have also tested FingerCI integrated with M-PE and BP-IDS by altering the BHS communication protocol without updating the specification. We changed the reporting messages of 37 bags. In this case, BP-IDS had a 100% FN rate, since the FingerCI specification did not capture the changed messages. M-PE however, was able to highlight the differences in the messages. This shows complementary: BP-IDS detects in real-time messages that deviate from the specification; M-PE detects when FingerCI specification needs to be updated.

## 7. Discussion and lessons learned

The results of the evaluation allow for the conclusion of three key points: FingerCI was fast to generate CI fingerprint specifications (taking at most 12 seconds to analyze eight network traffic hours); FingerCI integrated with SBSs automates the work required from experts on writing specifications for the CI, with accurate detection results and a low false-positive rate; and FingerCI provided correct and comprehensive specifications with a process model that has: 99.98% fitness to the observed behavior, 86% comprehensibility score, and a precision of 82%.

### *7.1. Lessons learned*

Several lessons can be learned: (1) dissection rules identified by the experts influence the meaningfulness of the produced specification; (2) abstraction of network fields in dissection rules allows $\gamma$ functions to be compatible

with any network packet regardless of the protocols underlying them. This design allows compatibility with several protocols for which there is a standard to extract the $n_f$ fields; (3) FingerCI process model indicates it is understandable based on Rozinat and van der Aalst's evaluation method [45], but with some room for improvement in terms of readability.

Regarding point one, the meaningfulness of the produced specifications depends on the dissection rules produced by the experts. They need to cherry-pick the fields found on a network packet that identify critical operations ($e_o$), processes ($e_p$) and behavior data ($e_b$). To configure a dissection rule, experts need to know the process they want to specify, which includes how to identify the main operations and physical state conditions in the network traffic. The ideal expert to write the dissection rules would be an expert in the network communications used by the CI system (e.g., the manufacturer of the BHS simulation platform Alstef, or the airports that use the BHS services). A good dissection rule allows specifications to portray the sequential ordering of activities in a process and identify the conditions for legitimate action. They affect the efficacy of the SBS in portraying the real behavior of the CI. For example, the dissection rules used on the evaluation allowed BP-IDS to use the FingerCI specification to distinguish between the good behavior of the BHS and anomalies. If the wrong fields were selected on the dissection rule, BP-IDS could experience false positives (raise alarms for normal behavior), or false negatives (would not detect anomalies). Such choices do not affect the correctness and conformance of the FingerCI system, since the specifications produced would still reflect the registry of operations to which the network packets are portrayed in the PCAP. It is thus independent of the techniques employed by FingerCI for process and behavior discovery and affects only the support and confidence that the dissection rule has to represent the real CI system.

Regarding point two, the implementation of the ORE post-dissector plugin shows that the design abstraction presented in Section 3.3.1, of network fields that allow $\gamma$ to be compatible with any network packet regardless of the protocols underlying them, holds in the proof-of-concept implemented (Section 5). In TShark, dissection protocols are processed before the post-dissection field. This allowed ORE to have available all the network fields previously processed by the specific dissection protocols at the moment of the post-dissection analysis. With this, $\gamma$ functions that are written for ORE can simply refer to the identifiers of the TShark protocol dissector. As in the above, writing $\gamma$ is as simple as writing a TShark filter. The implementation

of ORE validates the design and allows compatibility with several protocols for which there is a standard (or a Wireshark protocol dissector) to extract the $n_f$ fields. Dissection rules enable a sharing community of airports, technology manufacturers, law enforcement, and research partners to increase their knowledge of underlying CI network protocols.

Regarding point three, FingerCI process models include three types of gateway choices: parallel, exclusive, and loop. This allows the process model to be portrayed as a state machine, and displayed in multiple formats such as process trees, process diagrams written in BPMN language, or Petri-Nets. BPMN also has other elements that could be used besides those already used by FingerCI. Such elements are specific to that modeling language such as pools, lanes, event-triggering gateways, or message flows. The elements already supported are sufficient to portray a valid process model and are typically those provided by process discovery approaches [54, 58, 29, 53, 40, 41, 39].

### *7.2. Generalizability and Limitations*

FingerCI specification language can be represented as sequence diagrams (e.g., BPMN, PetriNet) that are easily auditable by CI experts and can be used to configure existing specification-based systems. The specification language employed in FingerCI showed good results for BHSs, and its applicability is theoretically generalizable to other cyber-physical systems. Sequence diagrams have been largely used in cyber-physical systems to model operation safety conditions. Examples include unmanned aircraft systems [37], safety-critical medical systems [38], power grid systems [6], railways [32, 31], and airports [4]. The specification language could help configure the aforementioned solutions, since their detection validates the conformity between data collected from sensors, with the specification.

The FingerCI current method has limitations regarding its applicability to other CI domains. The limitations include: (1) the necessity to create dissection rules specific to the case study; and (2) the necessity for more tailored behavior models.

Regarding limitation one, the system requires the construction of dissection rules for interpreting the CI network protocol and identifying processes, operations, and behavior. Some protocols are shared across sectors (e.g., MODBUS) other protocols are specific to the airport sector (IATA RP1745 [24]). To employ FingerCI in a different sector, new dissection rules need

to be specified. For example, BP-IDS [43], was previously tested in the railway sector. Namely, BP-IDS monitored the Passenger Information System (PIS), and the Train Movement Management System (TMM) to ensure the schedules provided at the station were coherent with the train position and velocity. To also deploy FingerCI under this scenario would involve the construction of dissection rules that identify process, operation and behavioral data on the PIS and TMM messages.

Regarding limitation two, the current system behavioral model $M_b$ is adequate for monitoring changes in physical variables that are expressed in categorical values. The BHS categorical values are simplified in Figure 5 as ($\emptyset$,$R$,$C$,$F$,$U$). On other CI applications, physical variables assume continuous values such as speed and coordinates which need to be converted to categorical on FingerCI dissection rules.

### *7.3. Future Work*

Dissection rules are a continuous improvement for FingerCI to be generally applied to more CI sectors. Future work could apply algorithms that convert the continuous values into categorical values. Previous works have made significant progress in inferring such categorical values from continuous data. According to previous work [59, 46] on the cybersecurity domain, categorical values can be inferred by finding minimum and maximum points from physical sensor variables, and by measuring gradient and periodicity between changes in the sensor variables. Both articles have applied this approach to the railway and energy sectors. Becoming a good candidate for future work to apply them in dissection rules.

In recent years, the use of CI specifications has allowed the deployment of digital twins [3, 30], and the investigation of cascading effects [42, 27] between CIs in supply chains. FingerCI enables future work to use semi-automatic dissection rules and specifications to reduce effort while retaining the necessary context required for configuring and maintaining these SBSs up-to-date. FingerCI opens the way for automated specification writing. Experts steer the specification writing using dissection rules. Future work could study how to automate dissection rules, which was beyond the scope of this work since the main goal of FingerCI is automating the writing of a CI specification while retaining expert decisions on what constitutes a critical operation and procedure (which is what makes SBS opposed to anomaly-based, where the expert is not involved at all).

## 8. Conclusion

This paper presented FingerCI, a solution that fingerprints a Critical Infrastructure (CI) and collects the network, process, and behavior models required by Specification-Based Systems (SBSs) [14, 32, 31, 10]. We argue that automating the modeling of such infrastructures is crucial, given that they frequently contain outdated legacy code and systems for which specifications are no longer accessible. Additionally, finding experts to develop these specifications can be challenging. We show the solution proposed is capable of automating process discovery solely based on CI network traffic, looking beyond the causal relations between activities, and providing detection rules based on behavior analysis. The specifications are compatible with existing SBSs. Moreover, the evaluation conducted shows that FingerCI contributes to an increase in the overall security of critical infrastructures and the timely detection of malfunctions.

## References


[1] 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 '18, page 566–581, New York, NY, USA, 2018. Association for Computing Machinery.

[2] M. R. Aliabadi, H. Haghighi, M. V. Asl, and R. G. Meybodi. Challenges of specification mining-based test oracle for cyber-physical systems. In *2020 11th International Conference on Information and Knowledge Technology (IKT)*, pages 1–7, 2020.

[3] D. Allison, P. Smith, and K. Mclaughlin. Digital twin-enhanced incident response for cyber-physical systems. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ARES '23, 2023.

[4] F. Apolinário, N. Escravana, E. Hervé, M. L. Pardal, and M. Correia. Fingerci: generating specifications for critical infrastructures. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, SAC '22, page 183–186, New York, NY, USA, 2022. Association for Computing Machinery.

[5] F. Apolinário, J. Guiomar, É. Hervé, S. Hrastnik, N. Escravana, M. L. Pardal, and M. Correia. Comsec: Secure communications for baggage handling systems. In *European Symposium on Research in Computer Security*, pages 329–345. Springer, 2022.

[6] R. Berthier and W. H. Sanders. Specification-based intrusion detection for advanced metering infrastructures. In *2011 IEEE 17th Pacific rim international symposium on dependable computing*, pages 184–193. IEEE, 2011.

[7] 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*, pages 791–806, 2016.

[8] Y. Chen, C. M. Poskitt, and J. Sun. Learning from mutants: using code mutation to learn and monitor invariants of a cyber-physical system. In *IEEE Symposium on Security and Privacy*, pages 648–660, 2018.

[9] K.-T. Cho and K. G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX Security Symposium*, pages 911–927, 2016.

[10] A. Choquehuanca, D. Rondon, K. Quiñones, and R. León. Formal specification and validation of a gas detection system in the industrial sector. In *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6. IEEE, 2020.

[11] G. Combs. Tshark: dump and analyze network traffic, 2012.

[12] G. Dan and H. Sandberg. Stealth attacks and protection schemes for state estimators in power systems. In *1st IEEE International Conference on Smart Grid Communications*, pages 214–219, 2010.

[13] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers. *Fundamentals of business process management*, volume 1. Springer, 2017.

[14] M. Eckhart and A. Ekelhart. A specification-based state replication approach for digital twins. In *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and Privacy*, pages 36–47, 2018.

[15] European Commission. Guidelines for the classification of information in research projects. *H2020 Programme*, 2020.

[16] N. Falliere, L. O. Murchu, and E. Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5(6):29, 2011.

[17] J. Feld. PROFINET: Scalable factory communication for all applications. In *IEEE International Workshop on Factory Communication Systems, 2004. Proceedings*, pages 33–38, 2004.

[18] C. Feng, V. R. Palleti, A. Mathur, and D. Chana. A Systematic Framework to Generate Invariants for Anomaly Detection in Industrial Control Systems. In *Proceedings 2019 Network and Distributed System Security Symposium*, San Diego, CA, 2019. Internet Society.

[19] D. Formby, P. Srinivasan, A. Leonard, J. Rogers, and R. A. Beyah. Who's in control of your control system? device fingerprinting for cyber-physical systems. In *Network and Distributed System Security Symposium (NDSS)*, 2016.

[20] I. N. Fovino, A. Carcano, T. D. L. Murel, A. Trombetta, and M. Masera. Modbus/DNP3 state-based intrusion detection system. In *24th IEEE International Conference on Advanced Information Networking and Applications*, pages 729–736, 2010.

[21] N. Goldenberg and A. Wool. Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems. *International Journal of Critical Infrastructure Protection*, 6(2):63–75, 2013.

[22] K. E. Hemsley, E. Fisher, et al. History of industrial control system cyber incidents. Technical report, Idaho National Lab (INL), 2018.

[23] W. Hurst and N. Shone. *Critical infrastructure security: Cyber-threats, legacy systems and weakening segmentation*, pages 265–286. Academic Press, United States, Sept. 2023.

[24] International Air Transport Association. Recommended practice 1745. *IATA*, 2013.

[25] A. Kleinmann and A. Wool. A statechart-based anomaly detection model for multi-threaded SCADA systems. In *International Conference on Critical Information Infrastructures Security*, pages 132–144. Springer, 2015.

[26] M. Kneib and C. Huth. Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 787–800, 2018.

[27] S. König, A. M. Shaaban, T. Hadjina, K. Gregorc, and A. Kutej. Identification and evaluation of cyber-physical threats on interdependent critical infrastructures. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ARES '23, 2023.

[28] R. M. Lee, M. Assante, and T. Conway. Crashoverride: Analysis of the threat to electric grid operations. *Dragos Inc., March*, 2017.

[29] S. J. Leemans, D. Fahland, and W. M. van der Aalst. Discovering block-structured process models from event logs - a constructive approach. In *International conference on applications and theory of Petri nets and concurrency*, pages 311–329. Springer, 2013.

[30] T. Lieskovan and J. Hajny. Security of smart grid networks in the cyber ranges. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, ARES '22, 2022.

[31] J. Lima, F. Apolinário, N. Escravana, and C. Ribeiro. BP-IDS: Using business process specification to leverage intrusion detection in critical infrastructures. In *31st IEEE International Symposium on Software Reliability Engineering (ISSRE 2020)*, 2020.

[32] J. Lima, N. Escravana, and C. Ribeiro. BPIDS-using business model specification in intrusion detection. In *Research in Attacks, Intrusions and Defenses: 17th International Symposium, RAID 2014*, volume 8688, page 479. Springer, 2014.

[33] H. Lin, A. Slagell, Z. T. Kalbarczyk, P. W. Sauer, and R. K. Iyer. Runtime semantic security analysis to detect and mitigate control-related attacks in power grids. *IEEE Transactions on Smart Grid*, 9(1):163–178, 2016.

[34] M. V. Mahoney and P. K. Chan. Learning rules for anomaly detection of hostile network traffic. In *3rd IEEE International Conference on Data Mining*, pages 601–604, 2003.

[35] F. Mannhardt. Multi-perspective process mining. In *BPM (Dissertation/Demos/Industry)*, pages 41–45, 2018.

[36] F. Mannhardt, M. De Leoni, and H. A. Reijers. The multi-perspective process explorer. In *13th International Workshops on Business Process Management Workshops (BPM 2015)*, pages 130–134, 2015.

[37] R. Mitchell and I.-R. Chen. Specification based intrusion detection for unmanned aircraft systems. In *Proceedings of the first ACM MobiHoc workshop on Airborne Networks and Communications*, pages 31–36, 2012.

[38] R. Mitchell and R. Chen. Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems. *IEEE Transactions on Dependable and Secure Computing*, 12(1):16–30, 2014.

[39] D. Myers. *Detecting cyber attacks on industrial control systems using process mining*. PhD thesis, Queensland University of Technology, 2019.

[40] D. Myers, K. Radke, S. Suriadi, and E. Foo. Process discovery for industrial control system cyber attack detection. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 61–75. Springer, 2017.

[41] D. Myers, S. Suriadi, K. Radke, and E. Foo. Anomaly detection for industrial control systems using process mining. *Computers & Security*, 78:103–125, 2018.

[42] L. Papadopoulos, A. Karteris, D. Soudris, E. Muñoz Navarro, J. J. Hernandez-Montesinos, S. Paul, N. Museux, S. Kuenig, M. Egger, S. Schauer, J. Hingant Gómez, and T. Hadjina. Praetorian: A framework for the protection of critical infrastructures from advanced combined cyber and physical threats. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ARES '23, 2023.

[43] F. Reuschling, N. Carstengerdes, T. H. Stelkens-Kobsch, K. Burke, T. Oudin, M. Schaper, F. Apolinário, I. Praça, and L. Perlepes. Toolkit to enhance cyber-physical security of critical infrastructures in air transport. *Cyber-Physical Threat Intelligence for Critical Infrastructures Security*, pages 254–287, 2021.

[44] L. Rosa, M. Freitas, S. Mazo, E. Monteiro, T. Cruz, and P. Simões. A comprehensive security analysis of a SCADA protocol: From OSINT to mitigation. *IEEE Access*, 7:42156–42168, 2019.

[45] A. Rozinat and W. M. van der Aalst. Conformance testing: Measuring the fit and appropriateness of event logs and process models. In *International Conference on Business Process Management*, pages 163–176. Springer, 2005.

[46] G. Saraiva, F. Apolinário, and M. L. Pardal. Im-disco: Invariant mining for detecting intrusions in critical operations. In *European Symposium on Research in Computer Security*, pages 42–58. Springer, 2023.

[47] K. Stouffer, S. Lightman, V. Pillitteri, M. Abrams, and A. Hahn. Guide to industrial control systems (ICS) security. Technical Report NIST Special Publication (SP) 800-82 Rev. 2, National Institute of Standards and Technology, June 2015.

[48] S. D. Sudarsan, D. Mohan, and S. Rohit. Industrial control systems-legacy system documentation and augmentation. In *IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*, pages 167–170, 2018.

[49] R. Tan, H. H. Nguyen, E. Y. Foo, D. K. Yau, Z. Kalbarczyk, R. K. Iyer, and H. B. Gooi. Modeling and mitigating impact of false data injection attacks on automatic generation control. *IEEE Transactions on Information Forensics and Security*, 12(7):1609–1624, 2017.

[50] The European Commission. Council directive 2008/114/EC L 345/75/82 of 8 december 2008 on the identification and designation of european critical infrastructures and the assessment of the need to improve their protection. *Official Journal of the European Union*, 2008.

[51] P. Uppuluri and R. Sekar. Experiences with specification-based intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 172–189. Springer, 2001.

[52] D. I. Urbina, J. A. Giraldo, A. A. Cardenas, N. O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, and H. Sandberg. Limiting the impact of stealthy attacks on industrial control systems. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1092–1105, 2016.

[53] W. van der Aalst. *Process mining: discovery, conformance and enhancement of business processes*, volume 2. Springer, Heidelberg, 2011.

[54] W. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[55] B. F. van Dongen, A. K. A. de Medeiros, H. Verbeek, A. Weijters, and W. M. van der Aalst. The ProM framework: A new era in process mining tool support. In *International Conference on Application and Theory of Petri Nets*, pages 444–454. Springer, 2005.

[56] S. K. vanden Broucke, J. De Weerdt, J. Vanthienen, and B. Baesens. A comprehensive benchmarking framework (CoBeFra) for conformance analysis between procedural process models and event logs in ProM. In *2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 254–261, 2013.

[57] C. Wakup and J. Desel. Analyzing a TCP/IP-protocol with process mining techniques. In *International Conference on Business Process Management*, pages 353–364. Springer, 2014.

[58] A. J. M. M. Weijters, W. M. P. van der Aalst, and A. K. A. de Medeiros. Process mining with the heuristics miner-algorithm. Technical report, Technische Universiteit Eindhoven, 2006.

[59] K. Wolsing, L. Thiemt, C. v. Sloun, E. Wagner, K. Wehrle, and M. Henze. Can industrial intrusion detection be simple? In *Computer Security – ESORICS 2022: 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26–30,*

*2022, Proceedings, Part III*, page 574–594, Berlin, Heidelberg, 2022. Springer-Verlag.

[60] 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.