International Journal of Critical Infrastructure Protection 33 (2021) 100430

Contents lists available at ScienceDirect

International Journal of Critical Infrastructure Protection

journal homepage: www.elsevier.com/locate/ijcip

ARTINALI#: An Efficient Intrusion Detection Technique for Resource-Constrained Cyber-Physical Systems

Maryam Raiyat Aliabadi[a], Margo Seltzer[b], Mojtaba Vahidi Asl[a,*], Ramak Ghavamizadeh[a]

[a] Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran
[b] Faculty of Computer Science, University of British Columbia, Vancouver, Canada

ARTICLE INFO

Article history:
Received 12 October 2020
Revised 8 January 2021
Accepted 27 February 2021
Available online 17 March 2021

Keywords:
Intrusion detection system
Cyber-physical system
Security
Accuracy
Efficiency
Feature selection

ABSTRACT

Cyber-Physical Systems (CPSes) are integrated into security-critical infrastructures such as medical devices, autonomous vehicles and smart grids. Unfortunately, the pervasiveness and network accessibility of these systems and their relative lack of security measures make them attractive targets for attacks. This makes building Intrusion Detection System (IDS) for CPSes a necessity. However, detecting intrusions requires collecting information about a system's internal workings; this can be expensive both in runtime and memory consumption. According to prior research, fine-grain monitoring of a CPS maximizes the chance of intrusion detection but incurs overhead that can exceed the resource constraints of these systems. The objective of this study is to propose a solution for adapting IDSes for deployment on resource-limited CPSes without losing detection accuracy.

We propose ARTINALI#; a Bayesian-based search and score technique that identifies the critical points at which to instrument a CPS. Given a set of security monitors that observe run-time behavior of the system, a set of specifications that verify the correct behavior of the system, and statistics gathered from fault injection, ARTINALI# discovers a small set of locations and a rich set of specifications that yield full attack coverage with low (memory and time) overhead. We deploy ARTINALI# to construct an IDS for two CPSes: a smart meter and a smart artificial pancreas. We demonstrate that our technique reduces the number of security monitors by 64% on average, leading to 52% and 69% reductions in memory and runtime overhead respectively, while still detecting over 98% of emulated attacks, on average. ARTINALI# enables the IDSes to be applicable to a wide range of CPS systems with different resource capacities. In addition, it accelerates the attack detection process which is significantly essential for safety-critical systems.

1. Introduction

A CPS is the key element of the Internet of Things (IoT). It is composed of a cyber system, a physical system, sensors, actuators, and networking components, by which it integrates computations and physical environment. The cyber system (aka control program) can control physical environment via actuators, and can receive feedback from physical environment via sensors in real-time. As the interaction between the physical domain and the cyber domain increases, the physical system becomes more susceptible to the security vulnerabilities that might exist in the control program [1,2]. Therefore, the security of the entire system strongly depends on the security of the CPS' control program. Recently, CPSes have been widely deployed in critical infrastructure such as smart medical devices [3], robots [4], smart grids [5], and Autonomous Vehicles [6–8]. These systems perform sensitive tasks and are therefore potential targets for cyberattack. However, the rapid growth of IoT has led to deployment of CPSes without support for enforcing important security properties

Intrusion Detection Systems (IDSes) are used to monitor computer systems and detect security attacks. Typical IDSes fall into two major categories: Signature-based, and behavior-based. Signature-based IDSes compare the real-time behavior of the system against known security attacks. As they rely on known attack models (signatures) they cannot detect unknown attacks [9]. This is significantly important for CPSes since they are working autonomously for long periods of time, and hence are difficult to

* Corresponding author.
E-mail addresses: m_raiyataliabadi@sbu.ac.ir (M. Raiyat Aliabadi), mseltzer@cs.ubc.ca (M. Seltzer), mo_vahidi@sbu.ac.ir (M. Vahidi Asl), r-ghavami@sbu.ac.ir (R. Ghavamizadeh).

https://doi.org/10.1016/j.ijcip.2021.100430
1874-5482/

be interrupted for frequently patching or upgrading in the field. In contrast, behavior-based systems detect intrusions by watching a system's dynamic execution to identify suspect behavior and are able to detect both known and unknown attacks. We can further divide behavior-based systems into anomaly-based and specification-based according to how suspect behavior is defined. In an anomaly-based IDS, we build a model of normal behavior and flag deviations from that model as intrusions; in a specification-based IDS, we assume that we have the correct specifications and we look for violations of those specifications.

Specification based IDSes are proposed as the best fit for CPS security [10–13]. A specification-based IDS implements two core functions: *data monitoring* and *data analysis*. Data monitoring is the process by which an IDS observes system behaviour and accumulates data logs. Data analysis is the process by which an IDS periodically analyzes the collected logs and checks them against the specifications derived from the CPS's correct behavior. Data monitoring can be performed at host level (host-based IDS) or network level (network-based IDS). Host-based IDS is tailored to the CPS system and monitors operations of CPS application, and application and operating system. Network-based IDS; however, is attached to the network, and monitors all the incoming and outgoing traffic. Host-based IDS provide more visibility than network-based IDS into the individual CPS applications, thus is able to quickly detect CPS misbehavior. Another important advantage of using host-based IDS is distributed control over attack detection; this is especially the case for high-volume configurations like smart grids.

The locations in the system where data monitoring happens are called security monitors. IDSes depend on the information collected by the security monitors, so it is important that they capture adequate information about the run-time behaviour of the system. However, smart security solutions for CPSes need to support light-weight intelligence. On one hand, deployment of security monitors using complete information maximizes the chance of attack detection at the cost of memory usage and performance overhead, which may limit scalability. On the other hand, CPSes have specific constraints that make the current IDSes challenging for them to deploy. These constraints are:

- **Limited memory:** An IDS that is tailored to a CPS system should satisfy resource constraints. For instance, an essential module of an IDS is a pre-trained model (i.e., a set of mined specifications) that represents the correct behavior of the CPS. In some cases, the available memory is not even sufficient to hold this model; further, a large number of security monitors create large log files, that, in turn, may make the system run out of memory. These scenarios make many existing intrusion detection techniques inapplicable to CPSes [5].
- **Real-time requirements:** Real-time CPS applications place strict constraints on processing and reaction time. For example, self-driving cars need to quickly detect objects and make decisions on lane or speed changes or detecting pedestrians. From a security point of view, taking the real-time requirements into account is vital as the IDS performance overhead must not delay the expected response time of the system; particularly, in decision-making scenarios. Hence, the performance overhead must be small enough to address the CPS real-time constraints.

Due to these constraints, existing IDSes are not a good fit for CPS platforms. There has been a lot of research on intrusion detection techniques using static analysis [14–17], dynamic analysis [18–21], artificial intelligence [22–24], and provenance [25–27]. The static analysis-based specification mining techniques build a model of a system based on code analysis. These techniques are inherently conservative and produce few false positives. However, static analysis alone does not provide enough information about the run-time behavior of the system, which in turn, produces a lot of false negatives. Furthermore, these techniques generate large models, leading to high overheads, often exceeding the resource constraints of a CPS. Dynamic analysis-based specification mining techniques; however, observe the run-time behavior. They log the key points of the program to infer a set of likely invariants (aka specifications). Dynamic analysis-based techniques follow the assumption that common behavior is correct behavior, and hence their mined specifications reflect the common behavior rather than potential behavior like what is identified in static analysis. As most software systems are not provided with adequate test-cases, there is a chance that some execution paths are not seen when mining specifications. The result is a high false positive rate [28,29], which makes them challenging for mission-critical CPSes. Provenance-based approaches [27] are a particular instance of a dynamic-analysis-based approach. However, collecting data provenance in a fine-grained setting imposes excessive runtime overhead [26]. Deep Learning (DL) algorithms are accurate for modeling the behavior of complex systems and detecting unknown attacks, but they consume a lot of memory, posing a problem for resource-constrained environments.

While there has been a significant amount of work on CPS security [10,30–32], these techniques offer no systematic way to find the best trade off between accuracy and efficiency. They reduce the size of the model through coarse data monitoring, or they detect only certain categories of attacks at run-time. As a result, their models do not guarantee full attack coverage of the intrusion detection technique.

We formulate the problem of constructing an intrusion detection technique for CPSes as an optimization problem. Given a set of specifications defining correct behaviour of the CPS, a set of fine-grained security monitors that observe the run-time behavior, and statistics gathered from fault injection, we discover a sparse subset of security monitors that achieve full attack coverage. We present ARTINALI#: a greedy technique based on a feature selection algorithm that uses a Bayesian network to predict the probability of full attack coverage given information from partial attack coverage of security monitors. The use of Bayesian network along with feature selection has been shown to be effective to reduce dimensionality in a variety of applications, including data set creation [33], post-silicon validation [34], and fault diagnosis [35]. We use Bayesian-based feature selection to build an efficient IDS for CPSes. We use Bayesian inference as a scoring function in a feature selection algorithm to select a small subset of security monitors whose attack coverage exceeds a user-provided lower bound. Then, we capture data from only these monitors to evaluate whether the IDS run-time achieves high detection accuracy. *To the best of our knowledge, we are the first to design an intrusion detection technique for CPS systems with preserved detection accuracy under resource constraints*. We make the following contributions:

- We present ARTINALI#, which discovers the core set of security monitors and a rich set of specifications that yield high accuracy with low overhead.
- We deploy ARTINALI# in the context of ARTINALI, which is an intrusion detection technique designed for CPS systems.
- We build an IDS prototype for two CPS systems, an advanced metering infrastructure and a smart artificial pancreas.
- We evaluate our IDS on the two systems using arbitrary attacks emulated by fault injection. We find that ARTINALI# exhibits 64% and 23% reduction in the number of security monitors and specifications, respectively, which in turn, leads to 69% and 52% decrease in IDS runtime and space consumption, respectively, while preserving 98% detection accuracy against arbitrary attacks.

We organized the rest of the paper as follows: Section 2 presents the intrusion detection techniques and their current shortcomings for CPS systems. Section 3 presents background material, including an overview of ARTINALI and Bayesian networks. In Section 4, we present ARTINALI#. Section 5 introduces our case studies and explains how to build an IDS using ARTINALI# and then outlines our experimental procedure. Finally, we present an evaluation of our technique in the face of arbitrary attacks in Section 6.

2. Previous work

Prior research explored different intrusion detection techniques. However, these approaches have some the following limitations:

- They often detect only certain categories of known attacks at run-time.
- They analyze only coarse-grain information, which affects detection accuracy, especially for forensic analysis.
- They do not consider intrinsic code properties for attack detection.
- They are not designed to take into account the resource constraints of the underlying platform.

We explore related work in three broad areas: intrusion detection systems, specification mining, and CPS security.

2.1. Network-based IDS

Thakore et al. proposed a quantitative methodology to determine maximum-utility, cost-optimal deployments of monitors in networked distributed systems [36]. This work introduces three metrics; cost, coverage and redundancy and uses them to quantify the richness of monitor data with respect to intrusion detection and the associated cost with deployment. Their exact solutions are, in general, computationally expensive. Grant et al. introduced Dinv for capturing invariants between variables at various distributed nodes and for checking them at real-time using real-time snapshots [37]. However, their empirical evaluation indicated that Dinv's runtime grows exponentially with the number of nodes. Genge et al. developed a heuristic approach for intrusion detection in a smart grid with respect to the available budget and bandwidth [38]. They designed two intrusion detection systems, where the first one optimally places IDS modules on communication paths, and the second one provides resilient communications. Although these techniques are useful for efficient monitoring of network traffic, they are not suitable for deployment in host-based IDSes, because they are all too resource intensive.

2.2. Host-based IDS

Murtaza et al. discover the use of trace abstraction techniques for decreasing the execution time of IDS without losing accuracy [39]. They consider system call traces as traces of kernel module interactions and use the resulting abstract traces as input to current intrusion detection techniques, such as Sequence Time-Delay Embedding and Hidden Markov Models. Farooqui et al. combine symbolic execution and dynamic instrumentation to decrease the instrumentation points and thus performance overheads [40]. Using this approach, they show a decrease in kernel runtime overheads in GPUes. Farid et al. presented a host-based intrusion detection technique for small IoT devices respecting their memory constraints [5]. They built an IDS that maximizes coverage of the security properties using two techniques. The first technique performs security analysis at the design level based on model checking. The second technique includes security analysis at the code level using symbolic ececusion. However, their technique is not scalable to complex CPSes with large code bases. An Inline Automation Model (IAM) approach for efficient intrusion detection in host-based IDSes was presented by Gopalakrishna et al. [41]. IAM is a static analysis-based intrusion detection technique that constructs a control flow graph (CFG) for each user function in a program. The run-time monitor is implemented as a library interposition method, that intercepts library calls and checks them against the model. This approach incurs low overhead, however, monitoring only the library interface does not produce high accuracy.

2.3. Artificial intelligence-based IDS

Recently, Machine Learning (ML), particularly, Deep Learning (DL) techniques, have been introduced to detect and classify cyber attacks [22–24,42,43]. For example, Yang et al. present a deep reinforcement learning scheme to defend a smart grid against data integrity attacks [43]. Chen et al. present HeNet, a DL technique to classify fine-grained control flow traces for malware detection [24]. HeNet achieves high accuracy on detecting Return Oriented Programming (ROP) attacks against AdobeR Reader. A LSTM-based technique for anomaly detection was introduced by Kim et al. [22]. They capture the semantic meaning of each system call and its relation to other system calls. Moreover, they proposed an ensemble method that can better fit IDS design by focusing on lowering false alarm rates. DL techniques are highly accurate for modeling the software and detecting unknown attacks. However, as shown in prior work [44–46], the adversarial characteristics of CPS systems and DL resource requirement make their integration challenging.

2.4. Provenance-based IDS

Provenance is a metadata describing the complete journey of data and processes [47]. Provenance is useful in auditing, debugging, and forensics investigation. There is a considerable amount of work on provenance-based intrusion detection techniques [25–27,47,48]. Nonetheless, the fusion of provenance with CPS security has not yet been explored [47]. Han et al. present UNICORN, a host-based anomaly detection system that uses whole-system data provenance to detect advanced persistent threats [25]. UNICORN discovers provenance graphs that provide adequate historical information to detect malicious activities with high accuracy. FRAPpuccino [48] is another provenance-based approach for detecting unusual behavior in programs running on PaaS clouds. Palyvos et al. [26] present GeneaLog, a highly granular data provenance technique that takes advantage of cross-layer properties of the software stack, that incurs a minimal overhead. To keep track of data in CPS systems, provenance can play a critical role. However, collecting fine-grained data provenance is an expensive operation that imposes a significant time and space overheads [26,49,50]. It might be applicable for servers but is prohibitive for resource-constrained CPSes.

2.5. Static specification mining

Static specification mining includes a family of techniques that analyze source code to mine specifications and deduce correct behavior of the system [14–17,51]. For example, Karim et al. [14] present synchronized pushdown systems (SPDS), and show how SPDS discovers security vulnerabilities due to misusing Crypto APIs in Android apps and Maven Central repositories. Wagner and Dean [51] also extract an automaton model from source code for their FSM-based intrusion detection system. They introduced a non-deterministic pushdown automaton (NDPDA), which builds an extensive model of the software system based on system calls. Although NDPDA show good accuracy, they are slow due to high memory overhead. Giffen et al. [17] propose the Dyck model, based

on static analysis of source code. Although they could remove some of the false negatives they increased the size of the model noticeably.

2.6. Dynamic specification mining

Dynamic specification mining techniques exploit various system execution traces to discover implicit program rules [18,19,21,52–55]. For example, Daikon is a dynamic analysis-based technique that derives (likely) invariants representing constraints on data value relations [28]. DIDUCE [56] combines data invariant inference and checking in a single tool for fault diagnosis purposes. It is not only able to dynamically monitor and check software, but also scales dynamic invariant detection to large programs. DySy [57] combines the advantages of dynamic invariant inference using Daikon and static analysis using symbolic execution, resulting in inferring a more accurate invariant set than that of Daikon. Texada [29] and Perracotta [58] derive temporal logic proposition, and captures sequences of events via tracking dynamic traces. The GK-Tail algorithm combines data invariants and temporal specifications and represents sequences of method calls that are annotated with data [59]. Perfume models system properties based on resource consumption [60]. It integrates event relations and their time constraints in a model. While the current techniques are useful for modeling one or two important dimensions of a system behavior, Aliabadi et al. [10] introduce a dynamic specification mining technique that mines specifications along the three dimensions of data, event, and time, and generates a 3D model for the system. This technique is highly accurate for security attack detection in CPSes, but incurs noticeable overheads for complex CPS platforms.

2.7. CPS security solutions

During recent years, CPS security has received a lot of attention due to the ubiquity and criticality of these systems [10,30–32]. Carreon et al. [31] present a probabilistic formulation for detecting malware by monitoring the internal timing of the different components of the system. They evaluate their model on a pacemaker using three known attacks [61]. Bezemskij et al. present an approach based on Bayesian networks that processes data received from robot sensors to detect attacks on cyber or the physical systems [62]. Zimmer et al. [32] develop an IDS based on analysis of the execution time of tasks in real-time CPSes and use it for detecting code injection attacks. They estimate worst-case execution time (WCET) using static analysis of portions of the source code. The study discusses three instrumentation strategies that allow monitoring of the execution time of tasks without compromising real-time performance. Deng et al. present an anomaly-based IDS for IoT networks using fuzzy c-means clustering (FCM) and PCA [30]. To reduce memory usage, they use PCA for feature extraction and reduction. Overall, the way that these techniques approach the resource limitation challenge in CPSes reduces their detection accuracy. They either do not fully utilize intrinsic code properties of CPS systems for attack detection or are designed to detect only a subset of known attacks. Moreover, none of these techniques propose a systematic way to minimize overheads, while obtaining high detection accuracy against targeted and arbitrary attacks.

3. Background

3.1. ARTINALI

ARTINALI[1] is a dynamic specification mining technique that generates models of CPS correct behavior for specification-based IDSes [10]. ARTINALI models the security policy of a system by defining the set of specifications that must hold true during run time. A specification, or interchangeably an invariant, is a logical condition that is preserved at a particular set of program points, e,g., the insulin dosage taken for a diabetic patient by a smart artificial pancreas never exceeds 3.5 units ($Basel \leq 3.5$).

ARTINALI identifies six major categories of invariants that are needed to verify the fundamental security properties of a CPS:

- *Data invariant:* captures the expected distribution of values of data variables during correct execution [10].
- *Event invariant:* captures the correct sequence of the events' occurrence [10].
- *Time Invariant:* captures the correct time boundaries of an event [10].
- *Data per Event (D|E) Invariant:* captures the data invariant that is preserved within specific event [10].
- *Event per Time (E|T) Invariant:* captures the constraints over event and time. It represents the boundaries of transition times from one event to another in an event sequence [10].
- *Data per Time (D|T) Invariant:* captures the constraints over data and time. It represents the data invariant as a function of time [10].

ARTINALI develops a model of a system's behavior by analyzing the rich data captured from full instrumentation of the CPS code. Unfortunately, this full instrumentation leads to high overhead (quantified in Section 6).

3.2. Bayesian network

We present a brief overview of Bayesian Networks (BNs), which will be used in Section 4. BNs model the joint distribution of statistically dependent random variables. Learning and inference are the two main characteristics associated with BNs [63]. The former involves the ability to capture the causal dependence among the available information (e.g., a set of discrete measurements), while the latter allows one to extract hidden information without knowing the parametric form of the model. A BN is a directed acyclic graph (DAG) with two types of components: A) nodes, which represent random variables, and edges, representing the conditional dependencies between nodes. B) The conditional probability $p(y_i|pa(y_i))$ of each node y_i given its parents $pa(y_i)$, which is described by a conditional probability table (CPT)[61]. These two components are essential to compute the joint probability distribution over all nodes of the network. Given a network with nodes $y = y_1, y_2, \ldots, y_n$, we compute the joint probability distribution of y using the following equation:

$$p(y) = p(y_1, y_2, \ldots, y_n) = \Pi p(y_i|pa(y_i))$$

Fig. 1 shows a Bayesian network. It represents a joint probability distribution over the random variables Age(A), Gender(G), Food intake(F), and Diabetes(D), broken down into distributions $P(A)$, $P(G)$, $P(F|A,G)$ and $P(D|F)$. Having the DAG and CPTs, we are able to compute any user-specified queries. e.g., $P(D = True|G = Female)$.

ARTINALI# uses Bayesian networks both for learning and inference. It first learns the correlations among security monitor measurements in the form of a *DAG* and *CPTs*. Next, it computes probabilistic answers to user specified queries about IDS attack coverage. For example, a user may seek the joint distribution of the subsets of security monitors based on their prediction power in attack detection.

4. Methodology

We present a systematic way to find a good trade-off between IDS *attack coverage* and CPS *resource constraints*. First, we present

[1] A Real-Time-specific Invariant iNference ALgorIthm.

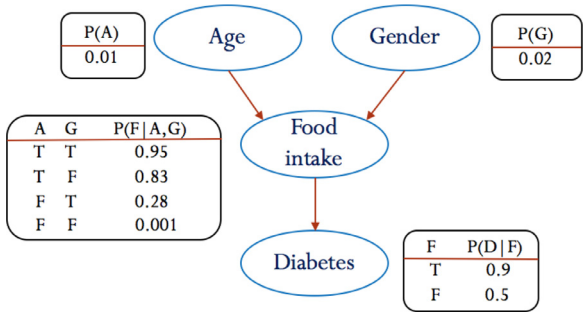


Fig. 1. A Bayesian network. It is a directed acyclic graph; a directed edge from a parent to a child identifies that the parent (source of the edge) is a cause for the child (destination of the edge). For example, *food intake* is a parent for *diabetes*. Having the above DAG and CPTs, we are able to compute any user-specified queries. e.g., $P(D = True|G = Female)$.

our problem formulation. Second, we introduce the ARTINALI# algorithm as our solution.

4.1. Problem formulation

We formulate the problem of constructing an IDS for a CPS as an optimization problem. We define the following terms in the formulation:

- We let $Z(\Phi, \Psi)$ to be an IDS defined over a set of security monitors Ψ and a set of invariants Φ for *monitoring* and *detection* purposes, respectively. We denote the (memory and time) overhead of IDS $Z(\Phi, \Psi)$ by $|Z(\Phi, \Psi)|$.
- We further let E, D, and T denote the subsets of security monitors corresponding to events, data variables and timestamps, respectively. Thus, $\Psi = E \bigcup D \bigcup T$.
- We define Φ as the set of m invariants (specifications) that must hold to verify the correctness of the CPS program. Each of the invariants $\phi_i \in \Phi$ is defined over some of E, D and T.
- We define λ_{ϕ_i} to be a set of security monitors whose values are necessary to evaluate the invariant ϕ_i.
- We define $u(\lambda_{\phi_i})$ to be the attack coverage provided by the invariant ϕ_i and thus by the security monitors λ_{ϕ_i}.
- $U : 2^n \longmapsto R$ is an attack coverage function defined over the security monitors. Attack coverage is defined as the detection accuracy of the IDS over a set of attacks. Overall, we have n security monitors and m invariants. Thus, the real value $U(\Psi) = \bigcup_{i=1}^{m} u(\lambda_{\phi_i})$, denotes the IDS full attack coverage, where the set $\Psi = \bigcup_{i=1}^{m} \lambda_{\phi_i}$, denotes all security monitors that must be observed so that full attack coverage is achieved.
- We let μ denote the lower bound attack coverage, required by the user.
- We want to construct an IDS based on the subset of security monitors $\Psi' \subseteq \Psi$ and a subset of invariants $\Phi' \subseteq \Phi$, so that the i) IDS attack coverage is greater or equal to the lower bound μ, and ii) the overhead of the IDS is significantly lowered subject to the constraint imposed by μ.
  More formally, we define our goal as the following:
- We find $Z(\Phi', \Psi')$, $\Phi' \subseteq \Phi$ and $\Psi' \subseteq \Psi$ so that

$$U(\Psi') \geq \mu \Rightarrow |Z(\Phi', \Psi')| \ll |Z(\Phi, \Psi)| \quad (1)$$

4.2. ARTINALI#

ARTINALI# is a greedy technique based on a feature selection algorithm that utilizes a Bayesian Network to score the subsets of security monitors based on their prediction power. Our Bayesian network quantifies the richness of the security monitor data and evaluates the prediction power of every subset of monitors jointly. ARTINALI# runs as an optimizer between the model training phase and deployment phase of an intrusion detection system. It helps find the core set of security monitors that will produce high detection accuracy with low overhead.

ARTINALI# requires a training data set containing information about security monitors and their contribution to detecting every arbitrary attack. We build a training data set by enabling all security monitors and collecting the information about the extent to which each contributes to attack detection. We use code mutation-based fault injection to emulate attacks. We further assume that our training test cases are rich enough to include all possible attacks on a CPS platform. Our objective then is to identify a sparse subset of security monitors $\Psi' \subseteq \Psi$ that likely provide full attack coverage.

We define a set of security monitor weights SM consisting of n Boolean variables $s_1, s_2, \ldots s_i, \ldots, s_n$ that indicate whether the ith security monitor (ψ_i) contributes to attack detection (i.e., $s_i = 1$ means ψ_i's data is to detect an attack). We show this relationship as $\Psi = SM * \overline{\Psi}$, where $\overline{\Psi}$ is the transpose of Ψ. We aim to find a smaller subset $SM' \subseteq SM$ such that the following equation holds.

$$P(s_i = 1, \forall s_i \in SM | s_j = 1, \forall s_j \in SM') \geq \mu \quad (2)$$

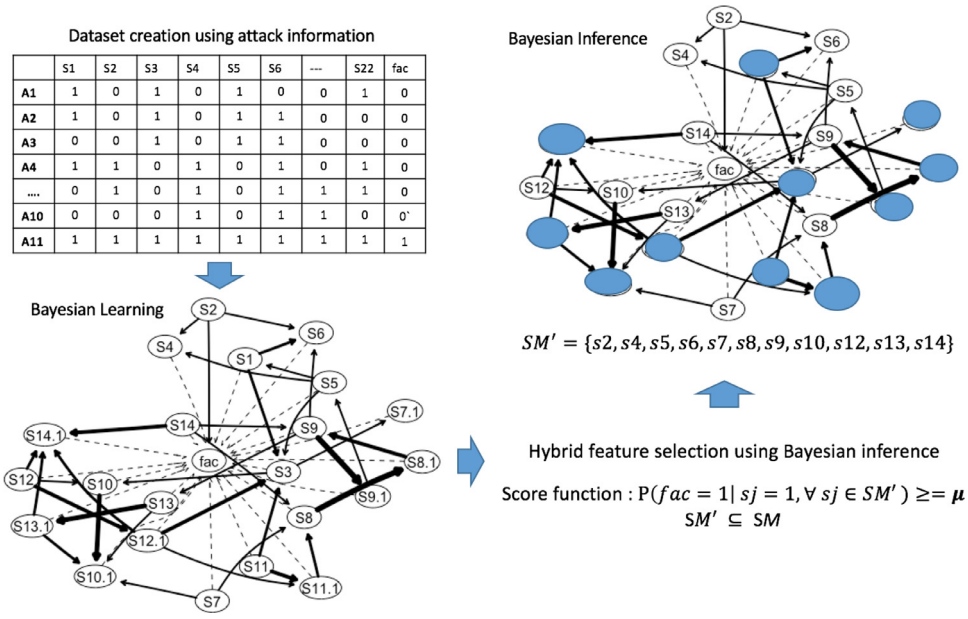
4.2.1. Bayesian network

We use a Bayesian network learning algorithm to construct a BN that infers the probability of full coverage given information from partial coverage of security monitors on our crafted attacks. To compute the conditional probability in Eq. 2, we need to compute the joint probability distribution over $s_1, s_2, \ldots., s_n$. We sample the joint probability distribution over security monitors by running attacks on the CPS code and observing which specification (and hence which security monitor(s)) contribute to attack detection (as shown in Fig. 2). After running attacks, we collect the frequencies with which each security monitor contributes to attack detection, and the actual combination of all security monitors in each attack detection, i.e., the group of security monitors whose data is required for detecting the same attack. It allows us to compute the conditional probabilities (e.g., the probability of the attack coverage of a monitor given that of the other monitors, $P(s_i = 1 | s_j = 1 \wedge s_t = 1)$) and capture dependence information among monitors. This data is required for learning the structure and conditional probability tables (CPTs) of a Bayesian Network that is able to approximate the joint probability distribution over security monitors. Since we are interested only in full attack coverage, we add a *full attack coverage* node (*fac*) to the BN, which is related to all security monitors and indicates whether their weights are all set to 1, i.e., $P(fac{=}1) = P(s_i = 1, \forall s_i \in SM)$. It allows us to compute the probability of full attack coverage *fac=1* given a subset of monitors. This equation is used as a score function in our hybrid feature selection algorithm. The goal is to compute a sparse subset of security monitors to make the attack coverage at the *fac* node equal or greater than μ.

4.2.2. Hybrid feature selection

Feature selection in machine learning is useful for finding the best feature set within high dimensional data sets [64]. The best feature set contains the fewest number of dimensions that most contribute to predicting accuracy. ARTINALI# selects features to transform the original feature set into a smaller one to reduce the amount of resources required. It seeks a small subset of features (security monitors) to build a model to use as a classifier.

Relevance of each feature for prediction alone does not imply a relationship among features. According to previous work, a deep connection between feature selection and causal mechanisms like Bayesian learning and inference leads to a more effective subset of features in practice [65,66]. Furthermore,



	S1	S2	S3	S4	S5	S6	---	S22	fac
A1	1	0	1	0	1	0	0	1	0
A2	1	0	1	0	1	1	0	0	0
A3	0	0	1	0	1	1	0	0	0
A4	1	1	0	1	0	1	0	1	0
....	0	1	0	1	0	1	1	1	0
A10	0	0	0	1	0	1	1	0	0`
A11	1	1	1	1	1	1	1	1	1

Fig. 2. Data set creation for Bayesian learning using 11 attacks ($A_1 - A_{11}$) and 22 security monitors ($s_1 - s_{22}$).

feature selection methods such as Ridge/LASSO regression essentially aim to find the relevant features that maximize a posterior distribution, but do not allow one to actually compute posterior probabilities (E.g., attack detection probability). In contrast, Bayesian network learns the correlations among security monitor measurements in the form of a DAG and CPTs. Next, it computes probabilistic answers to user specified queries. So, it not only finds the minimal relevant features, it allows one to actually compute posterior probabilities (here, attack detection probability). For these reasons, we use Bayesian learning and inference in conjunction with feature selection to score the subsets of security monitors based on their attack detection power.

We design a hybrid approach using a standard sequential Forward Feature Selection (FFS) algorithm to select a small set of rich security monitors (Algorithm 1). We chose FFS due to its simplicity and speed [64]. Using the hybrid algorithm, ARTINALI# greedily adds security monitors sequentially to an empty set until the addition of more features does not increase the IDS attack coverage. In each step, we calculate the attack coverage probability formula in Eq. 2 using BN for adding a feature (s_x). If the probability increases, we keep the added feature in the target set $SM' = \{s_1, s_2, \ldots . s_x\}$; otherwise, we start a backward pass on the current set SM'. First, we add the feature s_x to an empty temporary set ($SM'' = \{s_x\}$) and calculate the attack coverage probability. Second, we add each member of the current subset SM' one by one to the temporary set SM'' and compute the attack coverage probability. In each step, if probability increases we keep the added feature; otherwise, we drop it. Third, we choose the set with smallest size between SM' and SM'' and continue the hybrid algorithm with un-visited security monitors. Once the probability exceeds the lower bound coverage μ, a small set of security monitors with a good coverage had been identified.

4.2.3. Configure the IDS

Once the small subset of security monitor weights SM' is identified, we are able to compute the target subset of security monitors Ψ' using Eq. 3. Recall that the weights have binary values, where $s_i = 1$ means that the monitor ψ_i should be instrumented on the CPS program and $s_i = 0$ denotes that ψ_i should not be instrumented. We also map the target set of security monitors back to the specification set to shrink the specification space and to achieve the target set of specifications Φ' using Eq. 4.

$$\Psi' = SM' * \overline{\Psi} \tag{3}$$

$$\Phi' = \{\phi_i | \bigcup_{i=1}^{m} \lambda_{\phi_i} = \Psi'\} \tag{4}$$

We then configure the IDS based on such target sets, i.e., we instrument only the locations defined by target security monitors Ψ' to collect logs and monitor the CPS behavior. Then we use the target specification set Φ' as a base model in our IDS. ARTINALI# not only reduces memory usage, but speeds up the intrusion detection process. Furthermore, it preserves full attack coverage of the IDS $Z(\Psi', \Phi')$ with high probability, while lowering runtime overhead.

5. Experimental setup

As a proof of concept, we built an IDS based on ARTINALI# and deploy it in the context of two CPSes: an advanced metering infrastructure and an a smart artificial pancreas. We begin by stating research questions (RQs) we address and introducing the two CPS platforms. Then, we detail the procedure that we follow to build the IDS. Finally, we introduce the metrics we use to evaluate the IDS.

5.1. Research questions (RQs)

RQ1. What fraction of attacks can be detected by an IDS using a complete set of security monitors?
RQ2. How much reduction in the number of security monitors is achievable using ARTINALI#?
RQ3. How much reduction in the number of specifications is achievable using ARTINALI#?
RQ4. How much does ARTINALI# decrease memory overhead of the IDS?
RQ5. How much does ARTINALI# improve runtime performance of the IDS?

Algorithm 1 ARTINALI# computes a small subset of security monitors for coverage constraint μ using a hybrid feature selection algorithm and a Bayesian network that calculates the probability of having full attack coverage provided partial coverage of different subsets of security monitors. We use fault injection to emulate arbitrary attacks and measure the attack coverage of the IDS.

// Input: Ψ Security monitors, Φ Invariants, μ Lower bound coverage, *FI* Fault injection statistics
// Output: IDS $Z(\Phi', \Psi')$
$SM \leftarrow s_1, s_2, \ldots s_n$ // The full set of monitors
$SM' \leftarrow \emptyset$ // The target set of monitors
$SM'' \leftarrow \emptyset$ // A temporary set of monitors
Passive_set $\leftarrow \emptyset$ // A subset of monitors that are not present in any attack detection.
Redundant_set $\leftarrow \emptyset$ // A subset of monitors that don't add to the coverage.
$itr \leftarrow zero$
$U_{\max\ [itr]} \leftarrow zero$
Learn Bayesian Network structure from *FI*, Ψ and Φ
for all $\psi_i \in \Psi$ **do**
  if $\psi_i \notin \bigcup_{j=1}^{m} \lambda_{\phi_j}$ **then**
    Add ψ_i to *Passive_set*
    Remove s_i from SM
while $P(fac{=}1|x_j = 1, \forall x_j \in SM') \leq \mu$ **do**
  $itr++$
  for all $s_i \in (SM - SM')$ **do** // Forward pass
    $U_i \leftarrow P(fac{=}1|s_i = 1, x_j = 1, \forall x_j \in SM')$
    Let $s_{\max}$ be the s_i producing the maximum U_i
    Let $U_{\max\ [itr]}$ be the maximum U_i
  if $U_{\max\ [itr]} > U_{\max\ [itr-1]}$ **then**
    Add $s_{\max}$ to SM'
  if $U_{\max\ [itr]} == U_{\max\ [itr-1]}$ **then**
    Add $s_{\max}$ to SM''
    for all $s_k \in (SM' - SM'')$ **do** // Backward pass
      $W_k \leftarrow P(fac{=}1|s_k, s_p = 1, s_p \in SM'')$
      if W_k is increasing **then**
        Add s_k to SM''
  $SM' \leftarrow$ Min (SM' and SM'')
$\Psi' \leftarrow SM' * \overline{\Psi}$
Redundant_set $\leftarrow \Psi - \Psi'$−Passive_set
$\Phi' \leftarrow \{\phi_i | \bigcup_{i=1}^{m} \lambda_{\phi_i} = \Psi'\}$
Return $Z(\Phi', \Psi')$

5.2. Case studies

We use two following CPS platforms as our case studies to evaluate ARTINALI#.

5.2.1. Smart meters

Smart meters are key building blocks of an Advanced Metering Infrastructure (AMI). AMI provides two-way communication with a power provider [67]. The large scale deployment of smart meters and the investigation of many security vulnerabilities in AMI make them good candidates on which to evaluate ARTINALI# [10].

Generally, a smart meter is composed of a *meter* and a *controller* (also called gateway). The meter receives energy usage through analog front end sensors and accumulates them in a buffer. The gateway is a communication interface between the meter and the power provider's server. It sends server commands to the meter, and sends energy usage back to the server at specific time intervals. We use SEGMeter [68], an open source smart meter, as our case study. SEGMeter is implemented in 2500 lines of Lua, excluding libraries [10].

5.2.2. Smart Artificial Pancreas (SAP)

Traditional glucose measurement and manual insulin injection has been substituted to continuous glucose monitoring and autonomous insulin delivery devices. This automated approach is referred to as a *Smart Artificial Pancreas (SAP)*. A SAP is highly security-sensitive as attacks on this system compromise the patients' life [69]. Generally, a SAP is composed of a Continuous Glucose Monitor (CGM), an insulin pump, and a controller, commonly connected through a wireless network [70]. The CGM and insulin pump are wearable medical devices, in which the former regularly measures the patient's blood glucose (BG) level and sends the measurements to the controller, and the latter automatically injects insulin via subcutaneous infusion. A SAP delivers insulin in two doses: *bolus* and *basal*. Each type has a specific injection time, rate, and dosage based on the patient's needs. The controller receives the measured BG from CGM and sends an actuation command to the pump for correcting the BG level. We use OpenAPS [71], an open source SAP, as a second use case to evaluate our IDS. OpenAPS implements the controller component of an SAP in 2000 lines of JavaScript, excluding libraries.

5.3. Build an IDS based on ARTINALI#

Building an IDS based on ARTINALI# requires two steps: In *Step A*, we build the IDS base model, $Z(\Phi, \Psi)$, which uses all security monitors to fully instrument the CPS code. In *Step B*, using ARTINALI#, we identify those security monitors that are most influential in attack detection.

We begin with an IDS using full instrumentation so that we can use it as a baseline to determine to what extent coverage of a subset of security monitors, Ψ', implies coverage of all invariants (and hence full set of security monitors, Ψ). Thus, the goal of *Step A* is to capture a comprehensive CPS model achieving full attack coverage ($U(\Psi) \cong 100\%$), while the goal of *Step B* is to build an efficient IDS with full detection power.

We evaluate the resulting IDS using different types of targeted and arbitrary attacks. In *Step A*, we collect attack coverage information, such as violated specifications and a mapping between security monitors and the attack(s) they help detect. We then use this data as training data in *Step B*. Fig. 3 illustrates the overall experimental procedure that we follow for building an IDS based on ARTINALI#. The procedure consists of the following components: i) *Specification mining*, ii) *Data monitoring*, iii) *Data analysis*, iv) *Fault injection*, and v) *Optimization*.

5.3.1. Specification mining

Specification mining is the process of capturing a model that represents the correct behavior of the CPS system. A CPS model includes a set of specifications that lead to a decision regarding whether a given pattern of activity is suspicious. To build the CPS model, one can use any specification-mining technique; we use ARTINALI [10]. To mine specifications, we first fully instrument the CPS code with three types of security monitors: event monitors, data monitors, and time monitors, which collect high dimensional data for both training and test purposes. To place security monitors into the code, we insert calls to *ARTINALI API functions* that were developed for collecting traces before and after events. Table 1 shows the number and type of security monitors that we instrumented on both platforms. Overall, we employed 216 and 157 security monitors on SEGMeter and OpenAPS, respectively. We collect execution traces from the two CPSes under correct operation. Then, we randomly divide them into a set of training traces (consisting of 3/5 of traces) and testing traces (including 2/5 of traces). We use the training set as input to ARTINALI to mine specifications Φ and to build the CPS model. Table 1 presents the type and the number of invariants that ARTINALI generates for the

Table 1
Security monitor and invariant distributions for full coverage IDS across platforms.

CPS	Monitors				Invariants				
	Event	Data	Time	*Overall*	*Time*	*D|E*	*E|T*	*D|T*	*Overall*
SEGMeter	75	47	94	216	12	24	37	24	97
OpenAPS	69	28	60	157	4	22	18	7	51

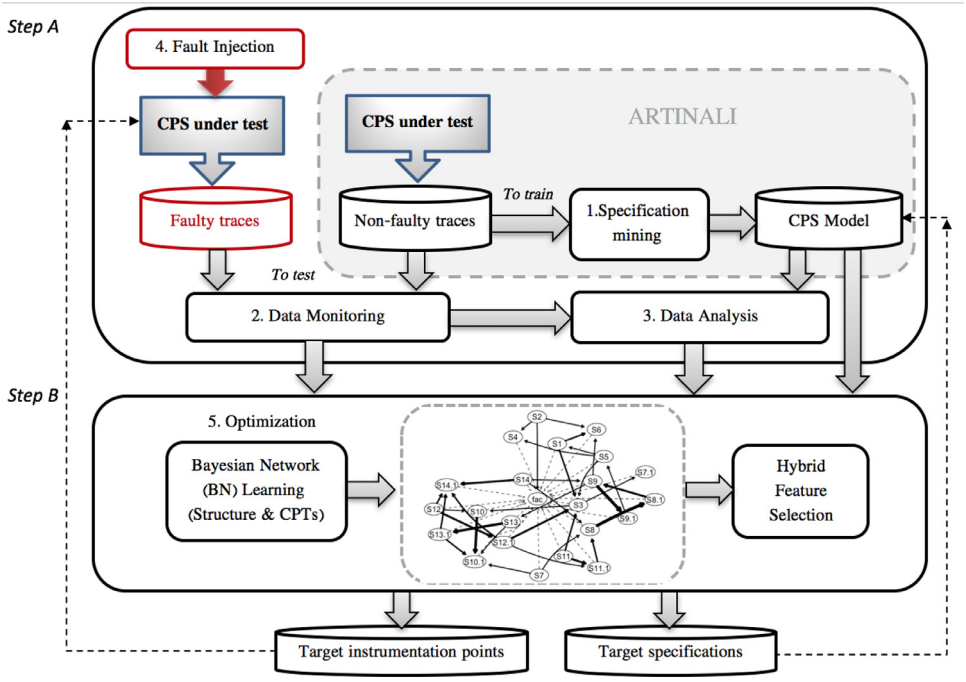


Fig. 3. Overall experimental procedure of building an IDS based on ARTINALI#.

SEGMeter and OpenAPS platforms: 97 for SEGMeter and 51 for OpenAPS, broken down into the *Time*, $D|E$, $E|T$, and $D|T$ categories (defined in Section 3.1).

5.3.2. Data monitoring

The *data monitoring* component observes the run-time behavior of the CPS, collecting the security monitor data from the program's execution and recording it. Before optimization, this component includes all security monitors, Ψ; after optimization, it uses the reduced number of monitors, Ψ'.

5.3.3. Data analysis

The information collected by *data monitoring* is input to the *data analysis* component, which periodically processes the monitoring data and evaluates it against the specifications derived from the CPS model. The detection accuracy of the *data analysis* (and of the IDS) is determined by the security monitor data and the comprehensiveness of the invariants, both of which are resource-intensive. To optimize IDS efficiency, while preserving its accuracy, we first need to collect attack coverage information and build our training test suite. Attack coverage information includes faulty and non-faulty traces collected by security monitors, violated specifications associated with each attack, and detected attacks by data analysis component, that can be achieved after evaluating the IDS against various types of attacks. We use fault injection to emulate attacks and to build our training test suite for the *optimization* step.

5.3.4. Fault injection

A small number of hand-crafted attacks are not sufficient to evaluate an IDS for CPS systems as these systems need protection against unknown attacks. This is especially important for security-critical CPSes such as smart medical devices [10]. Furthermore, unlike general-purpose computer systems, there is no known set of vulnerabilities and attack vectors for these systems. CPSes have different hardware, software and OS stack. Therefore, an attack discovered against a common Linux kernel applicable to many general-purpose servers, may not be applicable to a CPS system [5]. Consequently, the existing attack vectors for computer systems cannot cover new attacks against CPSes.

Previous work has used fault injection (code mutation) to study the effects of unknown attacks in CPS systems [5,72]. The main challenge is that the state space of fault injection may grow exponentially [73]. Moreover, not all the injected faults are manifested as real bugs/attacks. However, we use three types of model-based fault injection to emulate the behavior of security attacks and create more meaningful results:

- *Data mutations*: modify the run-time values of data variables.
- *Branch flipping*: modify the correct control flow of the program by inverting branch conditions.
- *Artificial delay insertions*: change the correct timing behavior of a system.

As stated in [10], data mutation attacks can change the important data in the program (e.g., race condition or memory corruption attacks). Similarly, branch flipping can lead to illegitimate execution flows (e.g., buffer overflow or Return-Oriented Programming attacks). Ultimately, artificial delays prevent essential functions to be completed at expected time, or may cause other important functionalities to be suppressed [10]. Using the above

Table 2
The number of mutations in each attack category for SEGMeter and OpenAPS.

CPS	Attack category		
	Data mutation	Branch flipping	Artificial delay
SEGMeter	175	226	45
OpenAPS	151	170	71

code mutations, we are able to emulate a wide range of attacks without biasing to the specific vulnerabilities. We assume our code mutation attacks are rich enough to include all possible attacks on the CPS platforms.

Table 2 presents the number of mutations performed in each attack category for our CPS platforms. Overall, we performed 447 and 392 fault injections for SEGMeter and OpenAPS, respectively. We injected each of these faults in the control program of the respective CPS platforms. After fault injection, we observe one of four outcomes: a) *Crash*, b) *Hang*,c) *SDC (Silent Data Corruption)*, and d) *No corruption*. We are interested only in *SDC* and *No corruption* outcomes (which comprise about 78% of the outcomes, on average), as the *Crash* and *Hang* outcomes do not need an IDS to be detected.

5.3.5. Optimization

ARTINALI# runs as an optimizer within our intrusion detection framework. It reduces the computational resources in high dimensional data sets to maintain scalability and accuracy of the IDS. As illustrated in Fig. 3, ARTINALI# uses attack coverage information as its training data set to train a Bayesian network and to run a greedy feature selection algorithm. The structure-learning algorithm identifies considerable dependence among the security monitors. We add the full coverage (*fac*) feature to our training data set, representing a logical AND of all the security monitors. The corresponding *fac* node in the network allows us to compute the probability of full attack coverage given a subset of monitors. Once BN construction is complete, ARTINALI# performs the search and score feature selection to compute a sparse subset of security monitors to make the attack coverage at the *fac* node equal or greater than μ. Figs. 5 and 6 show the structure of the learnt BN for the *serial_talker* and *determine_basel* functions of SEGMeter and OpenAPS platforms, respectively. The *serial_talker* function on the controller program of smart meter is in charge of receiving power consumption data at specific time intervals and buffering them for billing calculation purposes. The *determine_basal* on the controller program of OpenAPS is in charge of receiving blood glucose measurements and identifying the insulin (basel) dosage and other important parameters such as injection rate and duration.

When learning the Bayesian network, it is essential to ask how many security monitors to consider in the analysis. The larger the subset of security monitors analyzed together, the more accurate the relationships acquired in the network. However, as the number of relationships between security monitors grows, the complexity of learning a BN increases.

Training is an offline, one-time activity. As such, it does not impact runtime performance of the IDS. Nonetheless, it is important that its training time is practical. We empirically analyzed how ARTINALI# performance changes as the number of security monitors increases on both CPS systems. Fig. 4 illustrates the time spent in BN learning as a function of the number of security monitors analyzed. We started with 15 security monitors and added 15 at a time until we had included all the monitors on each platform. As can be seen, learning is fast ($\leq$ 1 minute) when the number of security monitors is small (less than 75 and 45 on SEGMeter and OpenAPS, respectively). Then, both curves grow quadratically, while the learning time for OpenAPS shows a sharper increase than that

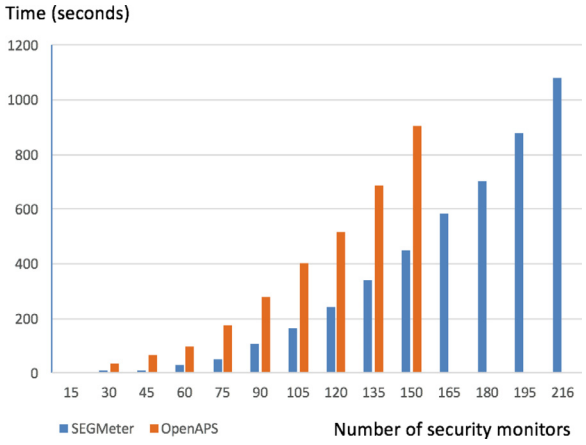


Fig. 4. Time taken (in seconds) to build a BN as a function of the number of security monitors on CPS platforms.

of SEGMeter. Overall, it takes 17.2 and 15.3 min to build a BN including 216 and 157 nodes (security monitors) for SEGMeter and OpenAPS, respectively, which is completely practical.

We implemented ARTINALI# in Python. We used the *bnlearn* [74] and *pgmpy* [75] libraries to learn the Bayesian parameters and structure. We used the *HC (HillClimbSearch*[2]*)* algorithm [76] for structure learning, and *BDeu (Bayesian Dirichlet equivalent uniform prior*[3]*)* [77] as the initialization of prior distributions.

5.4. Evaluation metrics

We chose the following metrics similar to what is considered in the previous work [5,10,37,40] for IDS evaluation purposes.

Accuracy: We use the attack coverage ratio to measure the effective accuracy of an IDS.

- *Attack Coverage ratio (AC)* is the fraction of injected attacks that are successfully detected by the IDS.

Overhead: We also measure the memory and performance overheads of the IDS.

- *Memory overhead* is defined as the total memory used by the IDS. It is a function of the size of IDS, the number of specifications in the CPS model, and the size of the log files produced by *security monitors*.
- *Performance overhead* is the increase in execution time that results from executing the IDS on the target CPS. This metric is dependent on both *Data Monitoring* and *Data Analysis*. As stated in [10], we measure the performance overhead per cycle, where a cycle refers to one complete execution of the main loop of the CPS. (Both the SEGmeter and OpenAPS consist of a repeating single loop).

6. Evaluation

We evaluate the IDS on our CPS platforms before and after optimization using the evaluation metrics presented in Section 5.4, addressing each research questions in its own sub-section.

[2] HillClimbSearch implements a greedy local search that starts from the DAG start and proceeds by iteratively performing single-edge manipulations that maximally increase the score.

[3] BDeu provides uniform prior over the parameters of each local distribution in the network, which makes structure learning computationally efficient.

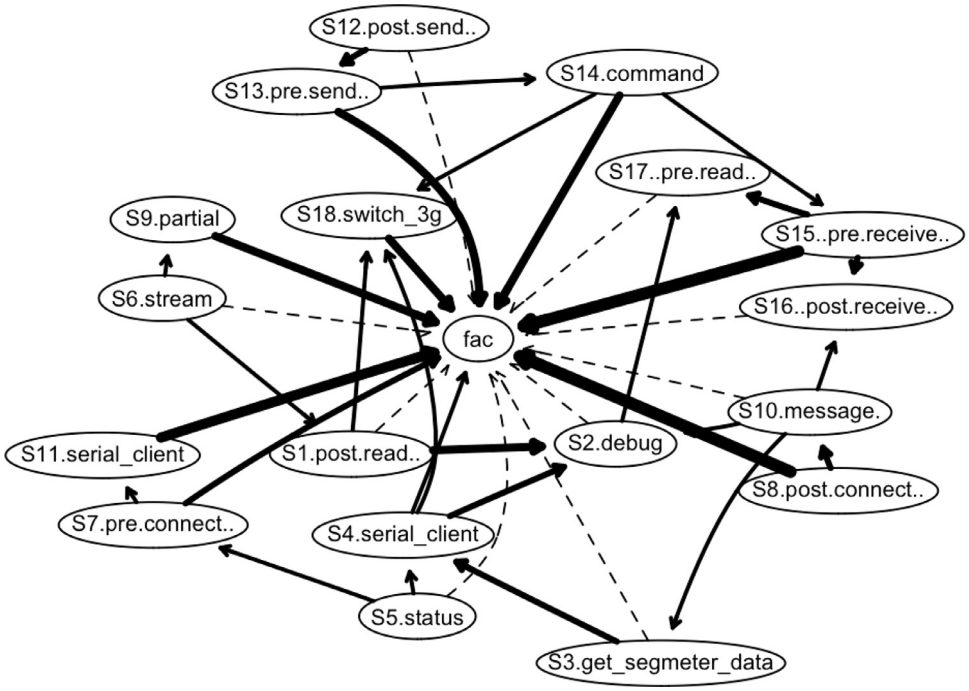


Fig. 5. BN structure created by ARTINALI# for *serial_talker* function on SEGMeter platform. It consists of 20 nodes including 19 security monitor weights and a *fac* node.

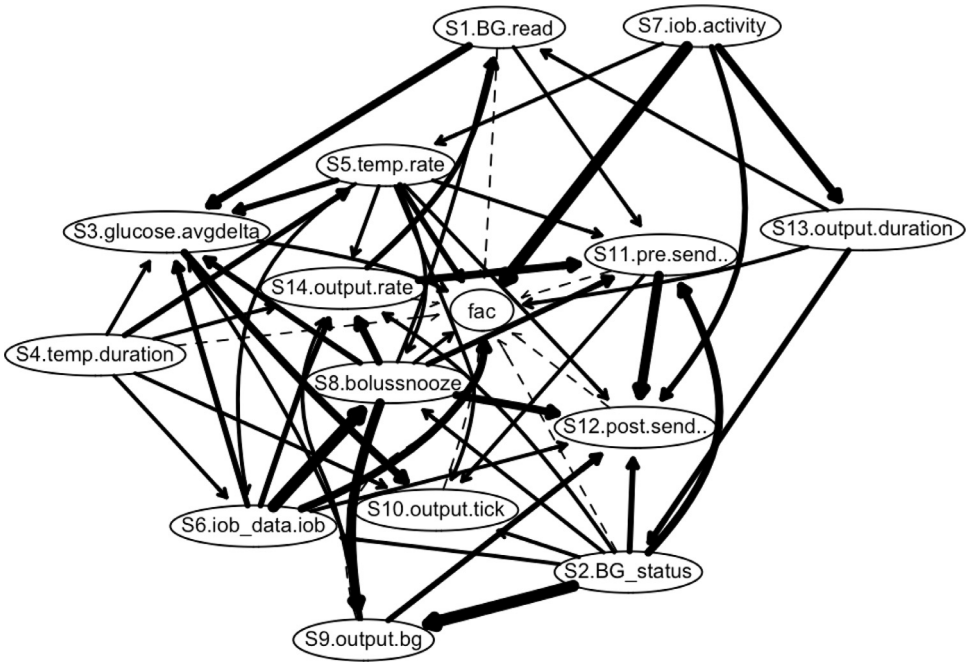


Fig. 6. BN structure created by ARTINALI# for *determine-basal* function on OpenAPS platform. It consists of 15 nodes including 14 security monitor weights and a *fac* node.

6.1. RQ1. Attack coverage

First, we present the attack coverage ratio incurred by the IDS, using the full set of security monitors and the invariants extracted by ARTINALI. Overall, the IDS was able to detect 98.26% and 98.1% of attacks on SEGMeter and OpenAPS, respectively. To better understand the results, we decompose the AC ratio in the different attack categories in Figs. 7 and 8. As illustrated, on average, the IDS exhibits 97.8%, 99% and 97.9% AC ratios against *data mutation, branch flipping* and *artificial delay* attacks, respectively, which result in over 98% aggregated coverage in both platforms. We use these results to establish the lower bound μ (98.26% for SEGMeter and 98.1% for OpenAPS) that we will use to evaluate our optimized system.

6.2. RQ2. Security monitor reduction

Next, we measure the reduction in the number of security monitors achievable under the given attack coverage constraints. We find it useful to categorize the security monitors into three groups: *Active, Passive* and *Redundant*.

Table 3
Security monitor distribution across platforms for full coverage.

CPS	Active			Passive			Redundant			Total monitors
	Event	Data	Time	Event	Data	Time	Event	Data	Time	
SEGMeter	44	11	22	38	39	10	31	7	14	216
OpenAPS	32	9	18	22	5	10	22	13	26	157

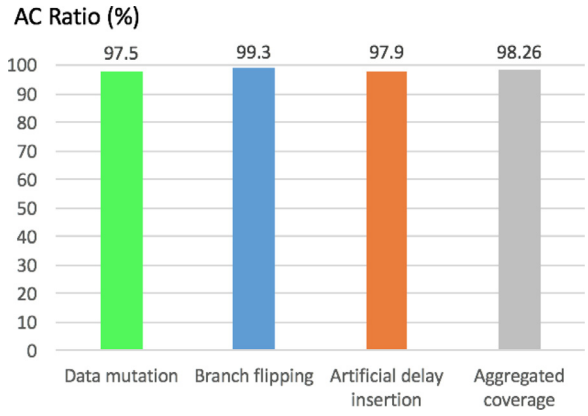


Fig. 7. AC Ratio of IDS for different attack types on SEGMeter platform.

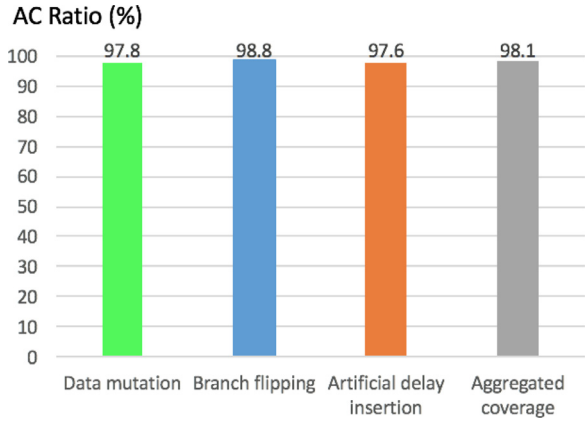


Fig. 8. AC Ratio of IDS for different attack types on OpenAPS platform.

- *Active*: A security monitor is *active*, if the data it collects is used for capturing (or evaluating) an invariant. *Active* monitors include both the monitors quickly selected for inclusion in the minimal set and those monitors whose inclusion is less obvious. In principle, ARTINALI# correctly identifies and prioritizes these security monitors according to the attack coverage constraints on each platform.
- *Passive*: A security monitor is passive if it is not associated with any invariants. Thus, *passive* monitors never contribute to detecting any attack initiated by fault injection and are not selected to be included in the optimized set.
- *Redundant*: An *active* security monitor is *redundant* if it does not add to the current coverage. *Redundant* security monitors are correlated to other *active* monitors, so that if a fault injection test covers an *active* monitor, its correlated monitors will be certainly covered. For example, assume variables x and y are correlated as: $y=x+1$. Any mutation on variable x changes y, so y is *redundant* from an attack coverage perspective and is not selected.

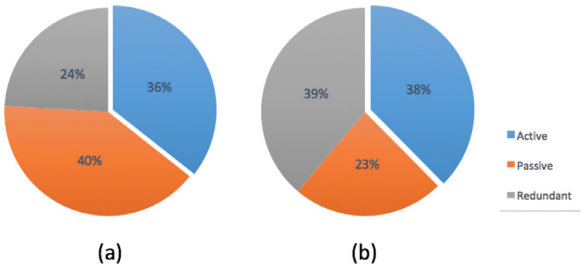


Fig. 9. Security monitor variations in (a) SEGMeter and (b) OpenAPS. Active monitors have 100% contribution for full attack coverage, while constitute less than 40% of security monitors on both platforms.

Table 4
Security monitor distribution on SEGMeter platforms for 98%, 95% and 90% coverage after optimization.

μ	Security monitors			
	Event	Data	Time	Reduction
98%	44	11	22	139 (66%)
95%	39	10	20	147 (68%)
90%	26	8	16	166 (77%)

Table 5
Security monitor distribution on OpenAPS platforms for 98%, 95% and 90% coverage after optimization.

μ	Security monitors			
	Event	Data	Time	Reduction
98%	32	9	18	97 (62%)
95%	20	9	18	109 (69%)
90%	15	8	16	117 (75%)

Table 3 shows how the full set of security monitors are distributed across the Active, Passive, and Redundant categories in both evaluation systems. *Passive* and *redundant* monitors account for 139 (out of 216) and 97 (out of 157) monitors in SEGMeter and OpenAPS, respectively. Only, *Active* monitors, which constitute 36% and 38% of monitors in SEGMeter and OpenAPS, respectively contribute meaningfully to IDS attack coverage (according to Fig. 9). *As a result, ARTINALI# is able to remove 66% and 62% of security monitors in SEGMeter and OpenAPS, respectively, without losing detection accuracy.*

Tables 4 and 5 show how the number of monitors decreases as we relax the coverage constraint from 98% to 95% to 90% on both platforms. The lower bound constraints permits a direct trade-off between detection performance and overhead. For example, reducing the coverage constraint on OpenAPS from 98% to 90% reduces the number of monitors required to only 25% of the original number; SEGMeter achieves a similar reduction. Overall, ARTINALI# decreases the number of security monitors by 77% (166 out of 216) and 75% (117 out of 157) to produce 90% guaranteed coverage on SEGMeter and OpenAPS, respectively.

Using a greedy approach to select the monitors to include in the optimized IDS decreases the computational complexity from exponential to quadratic, but can lead to a sub-optimal solution. According to the feature selection literature [78,79], simple

Table 6
Type and number of invariants on SEGMeter platform for 98%, 95% and 90% IDS coverage after optimization.

μ	Invariants				
	Time	*D|E*	*E|T*	*D|T*	Reduction
98%	10	24	21	17	25 (26%)
95%	8	22	21	16	30 (31%)
90%	8	19	21	13	36 (39%)

Table 7
Type and number of invariants on OpenAPS platform for 98%, 95% and 90% IDS coverage after optimization.

μ	Invariants				
	Time	*D|E*	*E|T*	*D|T*	Reduction
98%	4	16	13	7	11 (20%)
95%	4	15	12	7	13 (25%)
90%	3	15	10	7	16 (32%)

procedures of the floating search, such as feature swapping and backtracking can effectively yield significant improvement in the search performance. To mitigate the possibility of sub-optimal solution, we have included backtracking as a major step in ARTINALI# (Algorithm 1). Moreover, we applied feature swapping through shuffling the feature space. We run the algorithm on 20 shuffled sets of security monitors and select the best results. We find that ARTINALI# selects the same subset of monitors for the shuffled sets on OpenAPS. SEGMeter, however, produces different sets whose sizes vary by only a single monitor. These results suggest that ARTINALI# provides a robust solution on our evaluation platforms.

6.3. RQ3. Specification reduction

We next quantify the optimization effect on the specification space. Having identified the target set of security monitors, we map them back to the system's specifications. We observe that 25 (out of 97) and 11 (out of 51) invariants are excluded from the full set of invariants for SEGMeter and OpenAPS, respectively. Deeper analysis reveals that the excluded invariants fall into three categories: 1) invariants that are never violated during the attack injection tests. Note that attacks are detected only by violated invariants, and we assume that our test suite is complete. Therefore, the invariants that are always true do not detect attacks. E.g., the invariant *Power_consumption* ≥ 0 on SEGMeter always holds true, as there is no meaningful attack that makes the power consumption value negative. 2) Invariants that overlap, e.g., the invariant $b \leq a$ overlaps with the invariants $a \geq 10$ and $0 \leq b \leq 5$; therefore, it does not lead to greater attack coverage. 3) Invariants that are semantically dependent on other invariants, e.g., the following invariants show the logical ordering of three events (including *read(BG), send(BG)*, and *receive(basel)*) in OpenAPS: (i) *read(BG)* $\rightleftharpoons$ *send(BG)*, (ii) *send(BG)* $\rightleftharpoons$ *receive(basel)*, and (iii) *read(BG)* $\rightleftharpoons$ *receive(Basal)*, where invariant $E_i \rightleftharpoons E_j$ means the event E_i always happens before the event E_j. If either of invariants *i* or *ii* are violated, then invariant *iii* will also be violated, thus it does not provide additional attack coverage and can be removed. *Overall, ARTINALI# reduces the number of specifications by* 26% *and* 20% *on SEGMeter and OpenAPS, respectively, while preserving full attack coverage.*

Just as decreasing μ, the attack coverage constraint, reduces the number of security monitors necessary, it also reduces the number of specifications necessary. Tables 6 and 7 show the reduction in specifications, by type, as we reduce μ from 98% to 95% to 90%. probability of full coverage across platforms. While less dramatic than the reduction in required security monitors, the number of invariants also decreases with μ. Overall, ARTINALI# reduces the number of required invariants by $26 - -39\%$ and $20 - -32\%$ as we reduce μ from 98% to 90% for SEGMeter and OpenAPS, respectively.

Table 8
Memory overhead of *full-coverage* IDS, before and after optimization, running on SEGMeter.

	Pre-Optimization	Post-Optimization
Memory (MB)		
Data monitoring	2.96	1.35
Data analysis	1.59	0.87
Total	4.55	2.22

Table 9
Performance overhead of *full-coverage* IDS, before and after optimization, running on SEGMeter.

	Pre-Optimization	Post-Optimization
Time overhead (%)		
Data monitoring	23.3	7.3
Data analysis	0	0
Total	23.3	7.3
Execution Time (sec)		
CPS cycle	60.94	60.94
IDS cycle	4.97	2.93

6.4. RQ4. Memory overhead

We measured the memory usage of our IDS running on the SEGMeter platform. SEGMeter has 16MB of RAM, of which 12MB is utilized by the operating system and the *meter* program. Accordingly, a room of 4MB is available for the IDS (including *data monitoring* and *data analysis* components). Table 8 shows the memory usage of both components, separately, at run-time, before and after optimization. Memory consumption of the *data analysis* component before optimization is 2.96MB. The *data monitoring* component, however, creates log files that consume 1.59MB, on average, for each iteration of CPS execution. Therefore, the full coverage IDS needs 4.55MB memory, which exceeds the available memory on SEGMeter. After optimization, we observe that memory usage is decreased to 1.35 and 0.87MB in the *data monitoring* and *data analysis* components, respectively, for a total of 2.22MB for full coverage IDS. *Overall, ARTINALI# decreases SEGMeter IDS memory consumption sufficiently to make intrusion detection feasible given current resource constraints.*

6.5. RQ5. Performance overhead

Finally, we analyze the performance overhead of the IDS running on the SEGMeter platform, which consists of an embedded micro-controller (Broadcom BCM3302 V2.9 240 MHz CPU and 16 MB RAM) running Linux. More specifically, we measure the overhead imposed by both the *data monitoring* and *data analysis* components. Table 9 (first part) shows the overheads of the two components separately. We repeated these measurements 10 times, and presented the average overhead. We observe that the full coverage IDS incurs 23% runtime overhead before optimization, due to the *data monitoring* component, which runs in the same process as the CPS. The *data analysis* module runs as a separate process and thus adds *zero* overhead. After optimization, the overhead drops from 23% significantly to 7%, a 69% improvement in run-time efficiency.

Moreover, the execution time of the *data analysis* component should be lower than the execution time of the CPS's cycle. This is an important metric that determined if the IDS is able to keep up

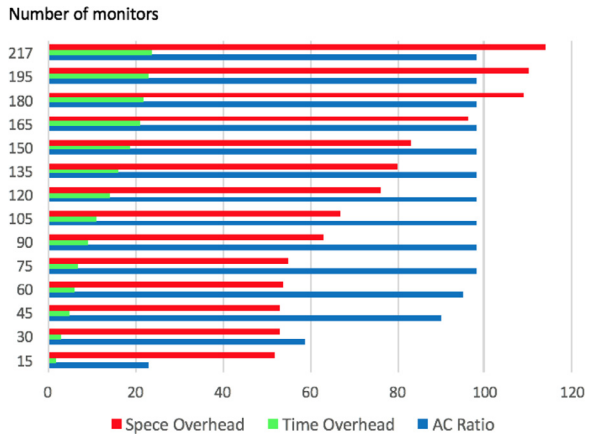


Fig. 10. Variations of (a) AC ratio, (b) run-time overhead and (c) space overhead of the IDS based on the number of security monitors on SEGMeter platform.

with the CPS. Table 9 shows the raw execution time of a CPS cycle as well as an IDS cycle. As shown, the entire CPS cycle takes about 61 seconds, while the IDS cycle before optimization takes 4.97 s, which is less than a sixth of execution time of the CPS cycle. After optimization, however, the IDS execution time reduces to 2.93 s. Thus, ARTINALI# accelerates the attack detection process on SEGMeter by 41%. The IDS is not a bottleneck in either the optimized or unoptimized cases and easily keeps up with the CPS. Furthermore, the *data monitoring* component collects logs as soon as the CPS starts working, while *data analysis* waits until the completion of the first iteration of CPS execution. This leads to a one-iteration delay for attack notification, i.e., 61 s.

Figs. 10 illustrates the relationship between the AC ratio, run-time overhead and space overhead of the IDS running on SEGMeter platform as a function of the number of security monitors. The AC ratio is quite robust to a reduction in the number of security monitors until we have eliminated all but a few tens of monitors. As the number of security monitors increases, all three graphs follow an ascending order. When the number of monitors reaches to 77, AC ratio gets maximized and stabilized at 98.26%. While both space and run-time overhead graphs grow almost linearly, the variations of the former illustrates a sharper increase than the latter. The growth continues until performance and memory overheads reach to 23% and 114%, respectively, when all security monitors are in use.

7. Discussion

We next examine the threats to the validity of our experiments and reflect on ARTINALI#'s generalizability.

7.1. Threats to validity

An external threat to the validity is the small size of CPS codes considered. The time complexity of BN learning varies quadratically with the number of monitors. This could become prohibitive for a large CPS with many hundreds of monitors. One potential solution is to partition the security monitors at the granularity of methods and to learn a separate BN for each partition. This would allow BN learning to scale to arbitrarily large CPSes, but would fail to capture inter-method relationships among security monitors. Nevertheless, BN learning is a one-time process that is done offline, so it does not have a negative impact on scalability of the IDS in run time.

An internal threat to validity is using fault injection to evaluate our IDS. Fault injection does not necessarily represent all real attacks. However, it allows us to emulate the potential attacks without biasing the evaluation towards known exploits. We decreased this threat by using model-based fault injection used for emulating attacks in prior work.

A construct threat to validity is the greedy nature of our technique, in which the order of adding features to the target set may lead to a sub-optimal solution. We minimize this threat in two ways. First, we apply a hybrid feature selection algorithm that adds an additional search step to check whether a feature we may drop encompasses two or more existing features in the target collection. If it does, we retain the new feature and drop the other two. Second, we ran the algorithm for shuffled sets of security monitors and chose the best result.

A second construct threat to validity is that a system developer, aware of a system's security policy, may be able to determine a priori which security monitors were necessary, without using ARTINALI#. While we acknowledge this hypothesis, we suspect that developers will be unable to detect all inter-variable dependencies.

Finally, another construct threat to validity is that the IDS starts data analysis after the first iteration of CPS execution is completed. Such after-the-fact analysis is sufficient to trigger attack mitigation mechanism in many systems. But, it may not be acceptable in security-critical systems that need even more precise real-time attack detection/prevention. To eliminate the notification delay, one solution is to place invariants as assertion constraints into the code. However, solving this problem is out of the scope of this work - which is focused on finding a small set of security monitors - and hence is considered as a direction for the future studies.

7.2. Generalizability

ARTINALI# discovers a small set of security monitors (instrumented locations) for resource-constrained CPS applications, which yields full attack coverage with low overheads. It enables the IDS to be applicable to a wide range of CPS systems with different resource capacities. We hypothesize that CPS code is accessible and can be modified to instrument the code - this is reasonable since we envision this technique to be deployed by CPS developers who wants to enhance their system security (if not, one can use a binary instrumentation engine). Furthermore, ARTINALI# is beneficial to any intrusion detection techniques tailored to *non-CPS* platforms as it significantly lowers the cost of the IDS and accelerates the attack detection process.

8. Conclusion

Resource constraints of cyber-physical systems make tailoring a security solutions to them challenging. We formulated the problem of constructing an intrusion detection system for cyber-physical systems as an optimization problem. We developed ARTINALI#: a greedy technique based on a hybrid feature selection algorithm that deploys the Bayesian network capabilities for approximating the probability of full attack detection given information from partial detection of security monitors. Given a set of security monitors that observe the run-time behavior of the system, a set of specifications that verify the correct behaviour of the system, and statistics gathered from fault injection, ARTINALI# discovers a small set of security monitors and a rich set of specifications that yield full attack coverage with low (memory and time) overheads. We deploy ARTINALI# on two CPS platforms, and show that our technique eliminates 64% of security monitors and 23% of invariants, on average, while preserving over 98% detection accuracy. As a result, ARTINALI# optimizes memory and time overheads by 52% and 69%, respectively, and speeds up the attack detection procedure by 41%.

As future work, we plan to incorporate whole system provenance to the specification mining process and build a multi-layer security model for the system. Another potential direction for future work is to incorporate specifications for dynamic attack diagnosis and mitigation. In particular, we plan to use graphical models for fine-grained attack diagnosis through specifications, and perform system reconfiguration based on the results of the diagnosis.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] A. Cardenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig, S. Sastry, et al., Challenges for securing cyber physical systems, Workshop on Future Directions in Cyber–Physical Systems Security, 5, Citeseer, 2009.
[2] R.C. Machado, D.R. Boccardo, V.G.P. De Sá, J.L. Szwarcfiter, Software control and intellectual property protection in cyber-physical systems, EURASIP J. Inf. Secur. 2016 (1) (2016) 1–14.
[3] N. Leavitt, Researchers fight to keep implanted medical devices safe from hackers, Computer 43 (8) (2010) 11–14.
[4] A. Khalid, P. Kirisci, Z.H. Khan, Z. Ghrairi, K.-D. Thoben, J. Pannek, Security framework for industrial collaborative robotic cyber-physical systems, Comput. Ind. 97 (2018) 132–145.
[5] F.M. Tabrizi, K. Pattabiraman, Design-level and code-level security analysis of IoT devices, ACM Trans. Embedded Comput. Syst. (TECS) 18 (3) (2019) 1–25.
[6] P. Dash, M. Karimibiuki, K. Pattabiraman, Out of control: stealthy attacks against robotic vehicles protected by control-based techniques, in: Proceedings of the 35th Annual Computer Security Applications Conference, 2019, pp. 660–672.
[7] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, et al., Comprehensive experimental analyses of automotive attack surfaces., USENIX Security Symposium, San Francisco, 2011.
[8] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, et al., Experimental security analysis of a modern automobile, in: 2010 IEEE Symposium on Security and Privacy, IEEE, 2010, pp. 447–462.
[9] R. Mitchell, I.-R. Chen, A survey of intrusion detection techniques for cyber–physical systems, ACM Comput. Surv. (CSUR) 46 (4) (2014) 1–29.
[10] M.R. Aliabadi, A.A. Kamath, J. Gascon-Samson, K. Pattabiraman, Artinali: dynamic invariant detection for cyber-physical system security, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ACM, 2017, pp. 349–361.
[11] R. Berthier, W.H. Sanders, H. Khurana, Intrusion detection for advanced metering infrastructures: requirements and architectural directions, in: 2010 First IEEE International Conference on Smart Grid Communications (SmartGridComm), IEEE, 2010, pp. 350–355.
[12] J. Goh, S. Adepu, M. Tan, Z.S. Lee, Anomaly detection in cyber physical systems using recurrent neural networks, in: 2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE), IEEE, 2017, pp. 140–145.
[13] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, S. Sankaranarayanan, Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications, in: Lectures on Runtime Verification, Springer, 2018, pp. 135–175.
[14] J. Späth, K. Ali, E. Bodden, Context-, flow-, and field-sensitive data-flow analysis using synchronized pushdown systems, Proc. ACM Program. Lang. 3 (POPL) (2019) 1–29.
[15] S. Shoham, E. Yahav, S.J. Fink, M. Pistoia, Static specification mining using automata-based abstractions, IEEE Trans. Softw. Eng. 34 (5) (2008) 651–666.
[16] M. Gabel, Z. Su, Symbolic mining of temporal specifications, in: Proceedings of the 30th International Conference on Software Engineering, ACM, 2008, pp. 51–60.
[17] J.T. Giffin, S. Jha, B.P. Miller, Efficient context-sensitive intrusion detection., NDSS, 2004.
[18] P. Bian, B. Liang, W. Shi, J. Huang, Y. Cai, Nar-miner: discovering negative association rules from code for bug detection, in: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ACM, 2018, pp. 411–422.
[19] P. Bian, B. Liang, Y. Zhang, C. Yang, W. Shi, Y. Cai, Detecting bugs by discovering expectations and their violations, IEEE Trans. Softw. Eng. (2018).
[20] R.-Y. Chang, A. Podgurski, J. Yang, Finding what's not there: a new approach to revealing neglected conditions in software, in: Proceedings of the 2007 International Symposium on Software Testing and Analysis, ACM, 2007, pp. 163–173.
[21] B. Liang, P. Bian, Y. Zhang, W. Shi, W. You, Y. Cai, Antminer: mining more bugs by reducing noise interference, in: Proceedings of the 38th International Conference on Software Engineering, ACM, 2016, pp. 333–344.
[22] G. Kim, H. Yi, J. Lee, Y. Paek, S. Yoon, Lstm-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems. arXiv preprint arXiv:1611.01726.
[23] A. Chawla, B. Lee, S. Fallon, P. Jacob, Host based intrusion detection system with combined cnn/rnn model, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2018, pp. 149–158.
[24] L. Chen, S. Sultana, R. Sahita, Henet: A deep learning approach on intel® processor trace for effective exploit detection, in: 2018 IEEE Security and Privacy Workshops (SPW), IEEE, 2018, pp. 109–115.
[25] X. Han, T. Pasquier, A. Bates, J. Mickens, M. Seltzer, Unicorn: runtime provenance-based detector for advanced persistent threats. arXiv preprint arXiv:2001.01525.
[26] D. Palyvos-Giannas, V. Gulisano, M. Papatriantafilou, Genealog: fine-grained data streaming provenance at the edge, in: Proceedings of the 19th International Middleware Conference, 2018, pp. 227–238.
[27] T. Pasquier, X. Han, T. Moyer, A. Bates, O. Hermant, D. Eyers, J. Bacon, M. Seltzer, Runtime analysis of whole-system provenance, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 1601–1616.
[28] M.D. Ernst, J. Cockrell, W.G. Griswold, D. Notkin, Dynamically discovering likely program invariants to support program evolution, IEEE Trans. Softw. Eng. 27 (2) (2001) 99–123.
[29] C. Lemieux, D. Park, I. Beschastnikh, General ltl specification mining (t), in: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2015, pp. 81–92.
[30] L. Deng, D. Li, X. Yao, D. Cox, H. Wang, Mobile network intrusion detection for iot system based on transfer learning algorithm, Cluster Comput. 22 (4) (2019) 9889–9904.
[31] N. Carreon, A. Gilbreath, R. Lysecky, Window-based statistical analysis of timing subcomponents for efficient detection of malware in life-critical systems, in: 2019 Spring Simulation Conference (SpringSim), IEEE, 2019, pp. 1–12.
[32] C. Zimmer, B. Bhat, F. Mueller, S. Mohan, Time-based intrusion detection in cyber-physical systems, in: Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems, 2010, pp. 109–118.
[33] M. Prasad, S. Tripathi, K. Dahal, An efficient feature selection based Bayesian and rough set approach for intrusion detection, Appl. Soft Comput. 87 (2020) 105980.
[34] Z. Wang, Z. Wang, X. Gu, S. He, Z. Yan, Feature selection based on Bayesian network for chiller fault diagnosis from the perspective of field applications, Appl. Thermal Eng. 129 (2018) 674–683.
[35] R.O. Gallardo, A.J. Huy, A. Ivanov, M.S. Mirian, Reducing post-silicon coverage monitoring overhead with emulation and bayesian feature selection, in: 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), IEEE, 2015, pp. 816–823.
[36] U. Thakore, G.A. Weaver, W.H. Sanders, A quantitative methodology for security monitor deployment, in: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, 2016, pp. 1–12.
[37] S. Grant, H. Cech, I. Beschastnikh, Inferring and asserting distributed system invariants, in: Proceedings of the 40th International Conference on Software Engineering, 2018, pp. 1149–1159.
[38] B. Genge, P. Haller, C.-D. Dumitru, C. Enăchescu, Designing optimal and resilient intrusion detection architectures for smart grids, IEEE Trans. Smart Grid 8 (5) (2017) 2440–2451.
[39] S.S. Murtaza, W. Khreich, A. Hamou-Lhadj, S. Gagnon, A trace abstraction approach for host-based anomaly detection, in: 2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), IEEE, 2015, pp. 1–8.
[40] N. Farooqui, K. Schwan, S. Yalamanchili, Efficient instrumentation of gpgpu applications using information flow analysis and symbolic execution, in: Proceedings of Workshop on General Purpose Processing Using GPUs, 2014, pp. 19–27.
[41] R. Gopalakrishna, E.H. Spafford, J. Vitek, Efficient intrusion detection using automaton inlining, in: 2005 IEEE Symposium on Security and Privacy (S&P'05), IEEE, 2005, pp. 18–31.
[42] H. Aghakhani, A. Machiry, S. Nilizadeh, C. Kruegel, G. Vigna, Detecting deceptive reviews using generative adversarial networks, in: 2018 IEEE Security and Privacy Workshops (SPW), IEEE, 2018, pp. 89–95.
[43] D. An, Q. Yang, W. Liu, Y. Zhang, Defending against data integrity attacks in smart grid: a deep reinforcement learning-based approach, IEEE Access 7 (2019) 110835–110845.
[44] K. Tange, M. De Donno, X. Fafoutis, N. Dragoni, Towards a systematic survey of industrial IoT security requirements: research method and quantitative analysis, in: Proceedings of the Workshop on Fog Computing and the IoT, 2019, pp. 56–63.
[45] M. Mohammadi, A. Al-Fuqaha, S. Sorour, M. Guizani, Deep learning for iot big data and streaming analytics: a survey, IEEE Commun. Surv. Tutor. 20 (4) (2018) 2923–2960.

[46] R. Chalapathy, S. Chawla, Deep learning for anomaly detection: a survey. arXiv preprint arXiv:1901.03407.
[47] S. Suhail, C.S. Hong, Z.U. Ahmad, F. Zafar, A. Khan, Introducing secure provenance in iot: requirements and challenges, in: 2016 International Workshop on Secure Internet of Things (SIoT), IEEE, 2016, pp. 39–46.
[48] X. Han, T. Pasquier, T. Ranjan, M. Goldstein, M. Seltzer, Frappuccino: fault-detection through runtime analysis of provenance, 9th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 17), 2017.
[49] B. Lerner, E. Boose, Rdatatracker: collecting provenance in an interactive scripting environment, 6th {USENIX} Workshop on the Theory and Practice of Provenance (TaPP 2014), 2014.
[50] J.F. Pimentel, L. Murta, V. Braganholo, J. Freire, noworkflow: a tool for collecting, analyzing, and managing provenance from python scripts, Proc. VLDB Endow. 10 (12) (2017).
[51] D. Wagner, R. Dean, Intrusion detection via static analysis, in: Proceedings. 2001 IEEE Symposium on Security and Privacy, 2001. S&P 2001, IEEE, 2001, pp. 156–168.
[52] C. Lemieux, D. Park, I. Beschastnikh, General ltl specification mining (t), in: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2015, pp. 81–92.
[53] I. Beschastnikh, Y. Brun, J. Abrahamson, M.D. Ernst, A. Krishnamurthy, Using declarative specification to improve the understanding, extensibility, and comparison of model-inference algorithms, IEEE Trans. Softw. Eng. 41 (4) (2015) 408–428.
[54] J. Abrahamson, I. Beschastnikh, Y. Brun, M.D. Ernst, Shedding light on distributed system executions, in: Companion Proceedings of the 36th International Conference on Software Engineering, ACM, 2014, pp. 598–599.
[55] M.D. Ernst, J.H. Perkins, P.J. Guo, S. McCamant, C. Pacheco, M.S. Tschantz, C. Xiao, The daikon system for dynamic detection of likely invariants, Sci. Comput. Programm. 69 (1–3) (2007) 35–45.
[56] S. Hangal, M.S. Lam, Tracking down software bugs using automatic anomaly detection, in: Proceedings of the 24rd International Conference on Software Engineering, 2002. ICSE 2002, IEEE, 2002, pp. 291–301.
[57] C. Csallner, N. Tillmann, Y. Smaragdakis, Dysy: dynamic symbolic execution for invariant inference, in: Proceedings of the 30th International Conference on Software Engineering, ACM, 2008, pp. 281–290.
[58] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, M. Das, Perracotta: mining temporal API rules from imperfect traces, in: Proceedings of the 28th International Conference on Software Engineering, ACM, 2006, pp. 282–291.
[59] D. Lorenzoli, L. Mariani, M. Pezzè, Automatic generation of software behavioral models, in: Proceedings of the 30th International Conference on Software Engineering, ACM, 2008, pp. 501–510.
[60] T. Ohmann, M. Herzberg, S. Fiss, A. Halbert, M. Palyart, I. Beschastnikh, Y. Brun, Behavioral resource-aware model inference, in: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ACM, 2014, pp. 19–30.
[61] K. Huang, C. Zhou, Y.-C. Tian, S. Yang, Y. Qin, Assessing the physical impact of cyberattacks on industrial cyber-physical systems, IEEE Trans. Ind. Electron. 65 (10) (2018) 8153–8162.
[62] A. Bezemskij, G. Loukas, D. Gan, R.J. Anthony, Detecting cyber-physical threats in an autonomous robotic vehicle using Bayesian networks, in: 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), IEEE, 2017, pp. 98–103.
[63] S. Krishnamurthy, S. Sarkar, A. Tewari, Scalable anomaly detection and isolation in cyber-physical systems using Bayesian networks, in: ASME 2014 Dynamic Systems and Control Conference, American Society of Mechanical Engineers Digital Collection, 2014.
[64] L. Ladha, T. Deepa, Feature selection methods and algorithms, Int. J. Comput. Sci. Eng. 3 (5) (2011) 1787–1797.
[65] C.K. Fisher, P. Mehta, Bayesian feature selection for high-dimensional linear regression via the ising approximation with applications to genomics, Bioinformatics 31 (11) (2015) 1754–1761.
[66] I. Tsamardinos, C.F. Aliferis, A.R. Statnikov, E. Statnikov, Algorithms for large scale Markov blanket discovery., in: FLAIRS Conference, 2, 2003, pp. 376–380.
[67] F. Skopik, Z. Ma, T. Bleier, H. Grüneis, A survey on threats and vulnerabilities in smart metering infrastructures, Int. J. Smart Grid Clean Energy 1 (1) (2012) 22–28.
[68] Smart energy groups home page., 2011, (http://smartenergygroups.com).
[69] J. Radcliffe, Hacking medical devices for fun and insulin: breaking the human scada system, in: Black Hat Conference Presentation Slides, 2011, 2011.
[70] C. Li, A. Raghunathan, N.K. Jha, Hijacking an insulin pump: security attacks and defenses for a diabetes therapy system, in: 2011 13th IEEE International Conference on e-Health Networking Applications and Services (Healthcom), IEEE, 2011, pp. 150–156.
[71] D. Lewis, Introducing the# openaps project(2015).
[72] K.-Y. Tseng, D. Chen, Z. Kalbarczyk, R.K. Iyer, Characterization of the error resiliency of power grid substation devices, in: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012), IEEE, 2012, pp. 1–8.
[73] M.R. Aliabadi, K. Pattabiraman, Fidl: a fault injection description language for compiler-based sfi tools, in: International Conference on Computer Safety, Reliability, and Security, Springer, 2016, pp. 12–23.
[74] E. Taskesen, bnlearn, 2019, (https://github.com/erdogant/bnlearn).
[75] A. Ankan, A. Panda, pgmpy: probabilistic graphical models using python, in: Proceedings of the 14th Python in Science Conference (SCIPY 2015), Citeseer, 2015.
[76] J.A. Gámez, J.L. Mateo, J.M. Puerta, Learning bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood, Data Mining Knowl. Discov. 22 (1–2) (2011) 106–148.
[77] D. Heckerman, D. Geiger, D.M. Chickering, Learning bayesian networks: the combination of knowledge and statistical data, Mach. Learn. 20 (3) (1995) 197–243.
[78] F. Hafiz, A. Swain, E.M. Mendes, Orthogonal floating search algorithms: from the perspective of nonlinear system identification, Neurocomputing 350 (2019) 221–236.
[79] J.Q. Gan, B.A.S. Hasan, C.S.L. Tsui, A filter-dominating hybrid sequential forward floating search method for feature subset selection in high-dimensional space, Int. J. Mach. Learn. Cybern. 5 (3) (2014) 413–423.