Lightweight Misbehavior Detection Management of Embedded IoT Devices in Medical Cyber Physical Systems

Gaurav Choudhary, Philip Virgil Astillo, Ilsun You*, *Senior Member, IEEE*, Kangbin Yim, Ing-Ray Chen, *Member, IEEE*, and Jin-Hee Cho, *Senior Member, IEEE*

***Abstract*—We propose a lightweight specification-based misbehavior detection management technique to efficiently and effectively detect misbehavior of an IoT device embedded in a medical cyber physical system through automatic model checking and formal verification. We verify our specification-based misbehavior detection technique with a patient-controlled analgesia (PCA) device embedded in a medical health monitoring system. Through extensive ns3 simulation, we verify its superior performance over popular machine learning anomaly detection methods based on support vector machine (SVM) and k-nearest neighbors (KNN) techniques in both effectiveness and efficiency performance metrics.**

***Index Terms*—Medical cyber physical systems, IoT, misbehavior detection, behavior rules, zero-day attacks, false positives, false negatives.**

I. Introduction

In a large-scale cyber physical system (CPS), there will be a huge number of embedded Internet of Things (IoT) devices and it is neither scalable nor practical to rely on a central entity such as a cloud to perform misbehavior detection. Since the central entity cannot physically perform misbehavior detection itself, it needs to collect misbehavior reports/logs from IoT devices. The amount of traffic generated will not only consume IoT energy but also cripple the CPS communication network. Hence, distributed misbehavior detection emerges as a feasible way for a large-scale CPS.

To-date, there are three types of misbehavior detection techniques for IoT devices embedded in a medical CPS: signature-based, anomaly-based, and specification-based techniques [12]. We dispose signature-based detection as it cannot deal with zero-day attacks. Our method is based on specification-based detection by specifying the intended behaviors of a medical device. Thus, it can deal with zero-day attacks by detecting device misbehaviors manifested due to the IoT device experiencing attacks, rather than detecting attacker patterns/tactics (which would not be known for zero-day attacks). Detecting misbehaviors rather than attack patterns can also be done by anomaly-based detection methods [2, 6-7, 10-11, 14-15, 24, 25] based on profiling and machine learning through correlation and statistical analysis of a large amount of data or logs for classifying misbehavior. While both specification-based and anomaly-based detection methods can cope with zero-day attacks by detecting misbehavior, our proposed specification-based misbehavior detection method is lightweight because it uses less code space and does not have to first learn misbehavior patterns in the training phase and then detect misbehavior in the operational phase. Consequently, we also dispose anomaly-based detection for distributed misbehavior detection because many embedded IoT devices are severely resource-constrained and do not have enough computational or storage power to store classification patterns or execute computationally expensive classification algorithms. The only viable method to perform distributed misbehavior detection for resource-constrained IoT devices is specification-based detection.

The novelty of our work is that we pioneer the use of lightweight behavior rule specification-based misbehavior detection for lightweight IoT devices embedded in a medical CPS (we call our misbehavior detection technique MedIoT for short) with memory, run time, communication, and computational overhead considerations. Our work is novel relative to existing specification-based intrusion detection techniques (see Section 2 Related Work for details) as follows:

- We propose a methodology for deriving the behavior rules of an IoT device embedded in a medical CPS, when given the embedded IoT device's operational profile [16] as input for specifying the IoT device's security requirements.
- We conduct model checking and formal verification of the correctness and completeness of the generated behavior rules such that the embedded IoT device in a medical CPS will not violate the security requirements if it does not violate the behavior rules.
- We develop a methodology of transforming the derived

This work is partially supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2017-0-00664, Rule Specification-based Misbehavior Detection for IoT-Embedded Cyber Physical Systems). This work is also supported in part by the U.S. AFOSR under grant number FA2386-17-1-4076. (*Corresponding author: Ilsun You)

G. Choudhary, P.V. Astillo, I. You, and K. Yim are with the Department of Information Security Engineering, Soonchunhyang University, South Korea (e-mail:gauravchoudhary7777@gmail.com;pvbastillo@gmail.com;ilsunu@gmail.com; yim@sch.ac.kr).

I.R. Chen, and J.H. Cho are with the Department of Computer Science, Virginia Tech USA (e-mail: irchen@vt.edu; jicho@vt.edu).

behavior rules into a state machine for lightweight misbehavior detection at runtime.
- We develop a lightweight data collection module for collecting compliance degree data from runtime monitoring of an IoT device based on its derived state machine.
- We develop a lightweight statistical analysis module for misbehavior detection based on experimentally collected misbehavior data at runtime.
- We verify the validity of our approach with a patient-controlled analgesia (PCA) device embedded in a medical health monitoring system with a comparative performance analysis against popular machine learning anomaly detection methods based on support vector machine (SVM) [28] and k-nearest neighbors (KNN) [34] techniques in both effectiveness and efficiency performance metrics.

The rest of the paper is organized as follows. In Section II, we survey existing work on misbehavior detection of IoT devices and compare as well as contrast our work with existing work. In Section III, we discuss the system model. In Section IV, we describe our MedIoT design in detail and apply MedIoT to a patient-controlled analgesia (PCA) device embedded in a medical health monitoring system. In Section V, we evaluate the performance of MedIoT and conduct a comparative performance analysis with contemporary machine learning misbehavior detection techniques in both effectiveness and efficiency metrics. In Section 6, we conclude the paper and outline future research areas.

II. Related Work

Specification-based misbehavior detection has been mostly applied to communication networks [4, 8, 21] and CPS security [1, 9, 17, 18, 26]. To the best of our knowledge, we are the first to consider specification-based detection for distributed misbehavior detection specifically for resource-constrained IoT devices embedded in a medical CPS.

An important aspect of specification-based detection is to verify if the specifications cover all the threats and satisfy the security requirements. Existing work [1, 21] focused on the construction of a formal framework utilizing ACL [27], a theorem prover, to first define security requirements and behavior specifications as ACL function, and then complete formal verification by defining a theorem (also an ACL function) that is evaluated to be true, proving that the behavior specification will not violate the security requirements. However, one risk is that the behavior specifications may be incomplete or even incorrect, leading to missing cases and high false positives. Relative to [1, 21], our contribution is to formally verify the completeness and correctness of behavior rule specifications following the design concept of Software Engineering research, i.e., proving that a piece of software is correct and complete with respect to the field expert specifications. We start with the "operational profile" [16] of an embedded IoT that defines the operational specification of an embedded IoT device to derive the security requirements of the embedded and hence the threats of the embedded IoT device. Then from the threats identified, behavior rules are generated to fully specify the intended behavior. Utilizing Hierarchical Context-Aware Aspect-Oriented Petri Net (HCAPN) [32, 33], a model checking tool, we formally verify that the generated behavior rules are complete and correct and cover all the threats and thus satisfy the security requirements derived from the operational profile.

Anomaly-based detection methods have been studied extensively for misbehavior classification for IoT-embedded CPSs [2, 6-7, 10-11, 14-15, 24, 25]. The bulk of research lies in applying profiling and machine learning through correlation and statistical analysis of a large amount of data or logs for classifying misbehavior. Recently, Artificial Neural Network (ANN) [10], Support Vector Machine (SVM) [28], and K-Nearest Neighbors (KNN) [34] have emerged as the leading machine learning techniques for misbehavior classification. We dispose ANN because of its huge memory and computational requirement which hinders its application to distributed IoT device misbehavior classification. We adopt SVM and KNN as baseline schemes against which our proposed specification-based detection method (MedIoT) is compared due to their relatively smaller memory and computation requirements, with the intent to demonstrate the efficiency and effectiveness of our proposed detection method against contemporary anomaly-based detection methods.

III. System Model

We refer the readers to [12, 13, 24, 25] for attacker behaviors and intrusion detection mechanisms available for IoT-embedded CPSs. Medical IoT devices in the IoT operational environment communicate with each other based on IoT machine-to-machine (M2M) wireless communication protocols such as MQTT [40] and LWM2M [41] without the need to connect to the broader Internet. Our behavior-rule based intrusion detection system (IDS) approach relies on the concept of monitoring. A monitor PCA is assigned to monitor a target PCA and the monitor code, i.e., the misbehavior detection algorithm (MedIoT, SVM or KNN), runs on the monitor PCA. To address the issue of the monitor node itself already compromised, the monitor code can be put in a secure computational space (e.g., [5, 37, 38]) such that each monitor node can execute misbehavior detection code in its secure computation space, even if the operating kernel has been compromised. For CPSs that do not have many redundant nodes, we advocate the concept of self-monitoring, i.e., each IoT device can execute misbehavior detection code in its own secure computation space and self-monitors itself. The monitoring process is lightweight and will not interfere with the normal operations of the monitor IoT device or the target IoT device (see Section IV-D for detail).

Note that our design can be extended into a fault tolerant structure based on the concept of recovery block [39]. Using a patient-controlled analgesia (PCA) device as an example, we can set up a recovery block structure consisting of two "functional" modules, e.g., a target PCA and a monitor PCA, along with an "acceptance" module corresponding to the monitor code preloaded into the monitor PCA's secure

computation space such that the acceptance module can securely execute misbehavior detection code, even if its operating kernel has been compromised. When the target PCA is deemed misbehaved by the acceptance module, the target PCA is taken offline and replaced by the monitor PCA for continued operations. The above design can also be applied to the case of self-monitoring, i.e., every PCA executes misbehavior detection code preloaded into its own secure computation space [5, 37, 38] upon bootstrapping and self-monitors itself. In this case, the two functional modules can adopt two distinct software implementations to reduce correlated software faults with module 1 as the main module and module 2 as the recovery module that will switch in to take over the PCA function once the acceptance module (i.e., the monitor code executed in the secure computational space) decides that module 1 is misbehaved or faulty.

TABLE 1
PCA SECURITY REQUIREMENTS.

ID	Security Requirement
SR 1	The PCA must raise alert to designated personnel and hold analgesic injection if the patient's condition is unfit for analgesic injection
SR2	The PCA must raise alert to designated personnel and hold analgesic injection if the PCA is not ready for analgesic injection
SR 3	The PCA must change its injection rate and medicine dosage upon authorized commands only
SR 4	The PCA must perform correct IDS functions when serving as a monitor node, i.e., providing true recommendations
SR 5	The PCA must perform analgesic injection at the specified dosage without exceeding the allowable injection rate

TABLE 2
PCA THREATS.

ID	Threat
THREAT 1	The PCA is not able to raise alert and hold analgesic injection when patient is unfit
THREAT 2	The PCA is not able to raise alert and hold analgesic injection when PCA is not ready
THREAT 3	The PCA is not able to follow authorized commands
THREAT 4	The PCA is not able to perform correct IDS functions, i.e., not able to provide true IDS recommendations
THREAT 5	The PCA's analgesic injection rate is above the specified injection rate
THREAT 6	The PCA is not injecting the specified dosage

IV. MEDIOT DESIGN FOR MISBEHAVIOR DETECTION OF PCA IN A MEDICAL CPS

In this section, we provide the detail of our MedIoT design and exemplify MedIoT with patient-controlled analgesia (PCA) devices embedded in a health monitoring medical CPS [17]. We choose PCA as our example medical IoT device because it is one of the most referenced medical IoT devices in the literature. We consider a PCA device as illustrated in Figure 1 that is programmed to perform analgesic injection in response to the injection button being pressed by a patient, with the injection period and dosage controlled by authority. Figure 1 shows that the monitor code in the monitor PCA executes in secure computational space, detects misbehavior of the target PCA based on behavior rules, and performs control strategies. When the target PCA does not conform to behavior specifications as defined by behavior rules in Table 3 (discussed later) derived

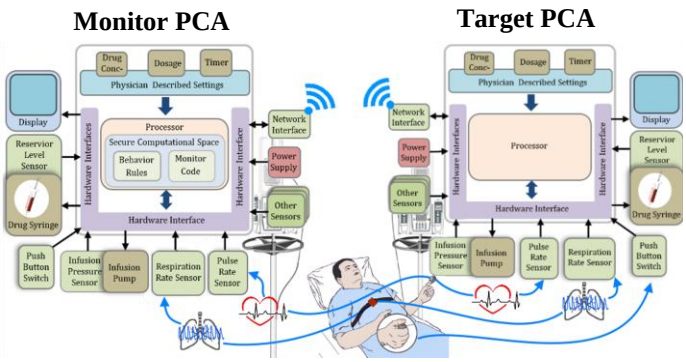

Fig. 1. A patient-controlled analgesia (PCA) device embedded in a health monitoring medical CPS [31]. The "monitor code" module in the monitor PCA is executed in secure computational space; it detects misbehavior of the target PCA based on behavior rules and performs control strategies.

TABLE 3
PCA BEHAVIOR RULES.

ID	Behavior Rule	Security Aspect
BR 1	Raise alert to designated personnel and hold analgesic injection if patient pulse rate is not normal	Integrity, confidentiality, availability
BR 2	Raise alert to designated personnel and hold analgesic injection if patient respiration rate is not normal	Integrity, confidentiality, availability
BR 3	Raise alert to designated personnel and hold analgesic injection if patient status is defibrillation	Integrity, confidentiality, availability
BR 4	Raise alert to designated personnel and hold analgesic injection if drug reservoir is empty	Integrity, confidentiality, availability
BR 5	Raise alert to designated personnel and hold analgesic injection if infusion pressure is not normal	Integrity, confidentiality, availability
BR 6	Accept authorized commands	Integrity, confidentiality, availability
BR 7	Provide true recommendations	Integrity
BR 8	Perform analgesic injection without exceeding the specified rate	Integrity
BR 9	Perform analgesic injection at the specified dosage	Integrity

from the PCA's operational profile, the monitor code will detect such misbehaviors regardless whether it is caused by malicious attacks or environment uncertainties/disturbances. For example, it may be the button that the patient must press to release drug is jammed, so no drug dosage is injected. Although it is not caused by an attack (rather, it is caused by a hardware fault of the button device), this misbehavior would be detected by ABI 9 in Table 4 (discussed later) which says "(Dosage ≠ Specified Dosage) ∧ (Action = Inject)" because upon the patient's pressing the button, no dosage is injected. As another example, it may be the life vital sign measurement device that measures the patient respiration rate is malfunctioning, so the patient respiration rate is uncertain. Although it is not caused by an attack (rather, it is caused by a hardware fault of the life vital sign measurement device), this misbehavior would nevertheless be detected by ABI 2 in Table 4 (discussed later) which says (Patient Respiration Rate ≠ Normal) and (Action ≠ Alert-and-Hold) because the patient respiration rate is uncertain and thus is not normal and the action is to alert the hospital personnel and hold drug injection. Since the behavior rules completely

and correctly cover all misbehaviors that will violate the security requirements of the PCA device regardless of the source of failure (by attacks or just environment uncertainties/disturbances), it ensures stability of the closed loop system of drug injection in presence of uncertainties and disturbances.

Here we note that if we enumerate all possible threats to a PCA, then very likely there will be missing threats and the PCA will not be considered safe. To address this missing-threat problem, we follow the design concept of Software Engineering research, i.e., proving that a piece of software is correct and complete with respect to the user specifications in the form of "operational profile" [16] that defines the operational specification and security requirements of a PCA. Since the security requirements are fully defined by the operational profile, we can mechanically derive a full set of threats that could violate the security requirements. This process is illustrated below as we convert the security requirement table (Table 1 PCA Security Requirements) to the threat table (Table 2 PCA Threats). Hence, there is no missing threat with respect to the security requirements defined in the operational profile. A similar idea is applied as we convert the threat table (Table 2) to the behavior rule table (Table 3) and subsequently to the attack indicator table (Table 4). As a result, the conversion is correct and complete with respect to the user specifications in terms of the full set of Boolean conditions as well as the full set of physical variables whose runtime values determine the truth or false of a Boolean condition.

A. Behavior Rule Specification of a PCA

We use the design concept of "operational profile" [16] during the testing and debugging phase of an embedded IoT device when the IoT software is built to identify the complete set of behavior rules. An IoT device's operational profile essentially is a mission assignment during the operational phase of the IoT device. A mission assignment in an embedded IoT device's operational profile explicitly defines a set of security requirements for the mission to be successful, from which a set of threats as well as a set of behavior rules to cope with the threats may be automatically derived.

We consider a PCA in a medical CPS with the following operational profile:

> *Raise an alert to designated personnel and halt analgesic injection if the patient's medical condition is unfit for analgesic injection; raise an alert to designated personnel and halt analgesic injection if the PCA is not ready for analgesic injection; communicate with authorized personnel only regarding the injection rate and dosage of medicine; perform correct IDS functions; when the injection button is pressed, if the patient controlled injection rate is less than or equal to the specified injection rate then inject a specified dose of medicine.*

Given this operational profile as input, the security requirements of this PCA may be derived as listed in Table 1.

With the system requirements defined, it is relatively straightforward to identify the threats that will keep this PCA from accomplishing its mission, as listed in Table 2.

Next, we derive the behavior rule set for this PCA. Deriving behavior rules from a threat will require field expert knowledge because only a field expert can properly identify the physical cause or source of a threat. Table 3 lists the behavior set without priority order for simplicity. It also lists the security aspect (integrity, confidentiality, or availability) associated with each behavior rule. A threat that has more than one cause or source for the negative event can require multiple behavior rules. For example, in THREAT 1 there are 3 causes for defining "patient is unfit," thus requiring three behavior rules (BR 1, BR 2, and BR 3) for handling THREAT 1. Similarly in THREAT 2 there are 2 causes for defining "PCA is not ready." Thus, two behavior rules (BR 4 and BR 5) are created for handling THREAT 2. The remaining threats each have a single cause and therefore they each map to a single behavior rule, i.e., THREAT 3 maps to BR 6, THREAT 4 maps to BR 7, THREAT 5 maps to BR 8, and THREAT 6 maps to BR 9. Note that as illustrated in Figure 1, a PCA can obtain a patient's vital sign and status measurements (such as pause rate, respiratory rate, defibrillation status, etc.) via communicating with various sensing and measuring devices mounted on the patient's body for collecting the patient's vital sign and status measurements.

TABLE 4
PCA ATTACK BEHAVIOR INDICATORS IN CNF.

ID	Attack Behavior Indicator
ABI 1	(Patient Pulse Rate ≠ Normal) ∧ (Action ≠ Alert-and-Hold)
ABI 2	(Patient Respiration Rate ≠ Normal) ∧ (Action ≠ Alert-and-Hold)
ABI 3	(Patient Status = Defibrillation) ∧ (Action ≠ Alert-and-Hold)
ABI 4	(Drug Reservoir = Empty) ∧ (Action ≠ Alert-and-Hold)
ABI 5	(Infusion Pressure ≠ Normal) ∧ (Action ≠ Alert-and-Hold)
ABI 6	(Command = AUTHORIZED) ∧ (Action ≠ Accept)
ABI 7	(Audit = ON) ∧ (Action = Report False Audit)
ABI 8	(Injection Rate > Specified Injection Rate) ∧ (Action = Inject)
ABI 9	(Dosage ≠ Specified Dosage) ∧ (Action = Inject)

B. Transforming the Behavior Rules to a State Machine for Misbehavior Detection

After the behavior rule set is identified, we transform it to a state machine for lightweight misbehavior detection. The behavior-rule-to-state-machine transformation process is automatic. First, one "attack behavior indicator" (ABI) for each behavior rule is derived. Then, each ABI is expressed as a conjunctive normal form (CNF) predicate to be evaluated to true or false indicating whether the corresponding behavior rule is violated or not. Then, all ABIs are combined altogether into a disjunctive normal form (DNF) predicate. Lastly the state machine is formed with all ABIs being the state components, each taking the value of 1 (true) or 0 (false). When all ABIs take the value of 0, it means that none of the behavior rules is violated and hence the system is in a safe state. Conversely, when an ABI takes the value of 1, it means that the corresponding behavior rule is violated. We describe the behavior rules to the state machine transformation process in the following subsections.

1) Attack Behavior Indicators Expressed as CNF Predicates

Table 4 lists 9 ABIs, each to be evaluated to 1 (true) or 0 (false) at runtime through monitoring, indicating whether the

corresponding behavior rule is violated or not. When an ABI is evaluated to true, the PCA is detected as misbehaving against the corresponding behavior rule.

The transformation of each behavior rule to an ABI involves the following two steps:

a) For each behavior rule, identify a set of Boolean conditions connected in CNF, such that if all Boolean conditions are evaluated true, then the ABI is evaluated true and the corresponding behavior rule is violated.
b) For each Boolean condition, identify a set of physical variables whose runtime values determine the truth or false of the Boolean condition.

The completeness and correctness of steps (a) and (b) will be discussed further in Section IV-C below. As an example, BR 1 in Table 3 has two Boolean conditions: "Is patient pulse rate not normal?" and "Is the action not alerting and holding?" For the first Boolean condition, the physical variable is *Patient Pulse Rate* and for the second Boolean condition the physical variable is *Action*. When the patient's pulse rate is not normal and the action is not alert-and-hold, it is a violation of BR 1. The first part of ABI 1 (the event part) specifies the event condition under which the 2nd part (the action part) is to be evaluated true or false.

Below we explain in detail how BR *i* in Table 3 (*i* from 1 to 9) is transformed into ABI *i* in Table 4 for the PCA device.

The 1st ABI (ABI 1 in Table 4) is that this PCA does not alert personnel and hold analgesic injection when the patient's pulse rate is not normal. A normal pulse rate for adults is 60-100 beats per minute. The CNF of the Boolean expression is (Patient Pulse Rate ≠ Normal) ∧ (Action ≠ Alert-and-Hold).

The 2nd ABI (ABI 2 in Table 4) is that this PCA still injects analgesic when the patient's respiration rate is not normal. The normal respiratory rate for adults is 12–20 breaths per minute. The CNF of the Boolean expression is (Patient Respiration Rate ≠ Normal) ∧ (Action ≠ Alert-and-Hold).

The 3rd ABI (ABI 3 in Table 4) is that this PCA still injects analgesic when the patient is being treated with defibrillation. The CNF of the Boolean expression is (Patient Status = Defibrillation) ∧ (Action ≠ Alert-and-Hold).

The 4th ABI (ABI 4 in Table 4) is that this PCA does not alert personnel and hold analgesic injection when the drug reservoir is empty. The CNF of the Boolean expression is (Drug Reservoir = Empty) ∧ (Action ≠ Alert-and-Hold).

The 5th ABI (ABI 5 in Table 4) is that this PCA does not alert personnel and hold analgesic injection when the infusion site is incorrect, e.g., the injection is pulled of the patient's body or the injection is not at the patient's correct infusion point. This is indicated by measuring the infusion pressure being normal or not. The CNF of the Boolean expression is (Infusion Pressure ≠ Normal) ∧ (Action ≠ Alert-and-Hold). This ABI has a local variable called *Infusion Pressure* for measuring the infusion pressure to detect if the infusion site is correct. If image sensors are built inside the PCA, image-sensing the infusion site may directly detect if the infusion site is at the right place.

The 6th ABI (ABI 6 in Table 4) is that a PCA does not accept authorized commands to update its injection rate and medicine dosage. The CNF is (Command = AUTHORIZED) ∧ (Action ≠ Accept).

The 7th ABI (ABI 7 in Table 4) is that a PCA acting as a monitor PCA provides false recommendations toward a behaving target PCA (called bad-mouthing attacks), and good recommendations toward a misbehaving target PCA (called ballot-stuffing attacks). This may be detected by detecting recommendation discrepancies among multiple monitor PCAs. The CNF is (Audit = ON) ∧ (Action = Report False Audit).

The 8th ABI (ABI 8 in Table 4) is that this PCA injects analgesic at a rate exceeding the specified injection rate. The CNF is (Injection Rate > Specified Injection Rate) ∧ (Action = Inject). Finally, the 9th ABI (ABI 9 in Table 4) is that this PCA does not inject analgesic at the right dosage. The CNF is (Dosage ≠ Specified Dosage) ∧ (Action = Inject).

Here we note that in Table 4 under ABI 1 - ABI 5, "Alert-and-Hold" refers to two separate actions because alerting the personnel of the misbehavior of the PCA device and holding the PCA injection would trigger different parts of the system. Hence the Boolean expression "Action ≠ Alert-and-Hold" is false when both alert personnel and hold dosage are done. We also note that an attacker can possibly block alert and infect a PCA to hold dosage. However this misbehavior would be detected by ABI 9 in Table 4 which says "(Dosage ≠ Specified Dosage) ∧ (Action = Inject)" because upon the patient's pressing the button, no dosage is injected.

2) All ABIs are combined into a DNF Predicate

All 9 ABIs in Table 4 are combined into a DNF predicate (ABI 1 ∨ ABI 2 ∨ ABI 3 ∨ ABI 4 ∨ ABI 5 ∨ ABI 6 ∨ ABI 7 ∨ ABI 8 ∨ ABI 9) because every ABI if evaluated to true is an indication of misbehavior.

3) Generated State Machine for Misbehavior Detection

For the PCA state machine, there are 9 Boolean variables (each taking the value of either 1 or 0) in the state representation, resulting in the total number of states being 2^9= 512, out of which only one is a safe state (when all 9 Boolean variables are false or take the value of 0) and all other 511 states are unsafe states. For a target PCA, we label its 512 states in the state machine as states 0, 1, 2, ..., 511 with state 0 represented by (0, 0, 0, 0, 0, 0, 0, 0, 0) as the only safe state in which all 9 Boolean variables (ABI 1 – ABI 9) take the value of 0 or false. Note that there are many variables in these 9 ABIs. However, these variables are internal variables maintained by a monitor PCA who updates these internal variable values at monitoring intervals to determine the true/false (or 1/0) of the 9 Boolean variables for a target PCA that is being monitored on. We note that it is possible all event conditions can occur simultaneously. Therefore it is possible for the target PCA to go from state (0, 0, 0, 0, 0, 0, 0, 0, 0) to state (1, 1, 1, 1, 1, 1, 1, 1, 1) when all 9 event conditions are true and the target PCA is a reckless attacker that attacks all the time whenever it has a chance.

C. Model Checking and Formal Verification

While generating an ABI (in Table 4) from the corresponding behavior rule (in Table 3), it requires that (a) a complete set of Boolean conditions be specified for each behavior rule; and (b) a complete set of physical variables be specified for each Boolean condition. If step (a) or (b) is incorrect or incomplete, it could lead to high false positives.

Our method to deal with the above problem resorts to what *Software Engineering* research does, i.e., proving that a piece of software is correct and complete with respect to the user specifications. In our design, we prove that both steps (a) and (b) are correct and complete with respect to user-specified security requirements (in Table 1 as derived from the device's operational profile) and user-defined Boolean conditions. Specifically, for step (a) it would require the field expert to specify or define a set of Boolean conditions for each behavior rule. For example, for BR 1 "Raise alert to designated personnel and hold analgesic injection if patient pulse rate is not normal," a field expert would specify if (Patient Pulse Rate ≠ Normal) and (Action ≠ Alert-and-Hold) would be the correct and complete Boolean conditions for BR 1. For step (b), it would require the field expert to specify or define the set of physical variables for each Boolean condition based on the field expert's knowledge about the domain. For example, the field expert would identify that "Patient Pulse Rate" and "Action" are two physical variables for BR 1. To prove that the transformation from a behavior rule to an ABI is complete and correct, we would encode the field expert knowledge as "auxiliary" rules with which we formally prove the each ABI is derived correctly such that it can correctly and completely cover the corresponding threat and thus satisfy the corresponding security requirement based on the user-specified Boolean conditions for each ABI.

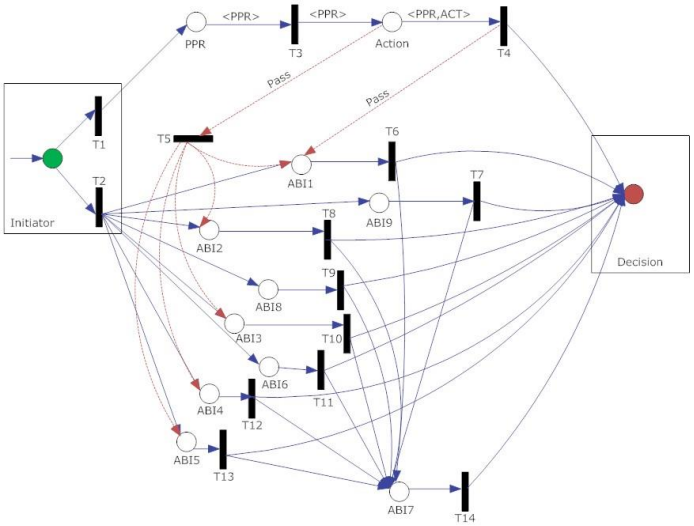

Fig. 2: A 2-layer HCAPN model for formal verification of correctness and completeness. For illustration, the upper layer contains only places and transitions related to ABI 1.

We conduct automatic model verification of the behavior rules and the corresponding ABIs expressed in XML format by verifying if the behavior rules generated are correct and complete with respect to user specifications and cover all the threats and thus satisfy the security requirements. We formulate a Hierarchical Context-Aware Aspect-Oriented Petri Net (HCAPN) [32, 33] model implemented through a Petri Net simulator [36] for model checking and formal verification. In the HCAPN model, we define security requirements (in Table 1) and ABIs (in Table 4) by (1) places each holding an ABI, a physical variable, or a security requirement; (2) transitions each with a set of Boolean conditions specifying the firing conditions; and (3) arcs connecting places with transitions. We complete formal verification by defining a HCAPN model that is evaluated to be true, proving that this PCA device will not violate the security requirements if it does not violate the behavior rules.

Figure 2 illustrates the resulting HCAPN model with places modeling the physical variables in the upper layer and ABIs in the lower layer. For illustration purposes, we only show two places "Patient Pulse Rate (PPR)" and "Action" (physical variables of ABI 1) and the associated transitions in the upper layer for formal verification of behavior rule 1. The lower layer consists of behavior rules (in the form of ABIs) each connecting to their physical variables by arcs. A transition is fired if and only if each of the input places contains a token and the Boolean condition associated with the transition is evaluated true. The "Initiator" place has a role of initiating token firing for model checking. This is done by generating tokens representing physical variables to flow through the HCAPN and verifying if a transition firing condition is true causing a transition to be fired. The "Decision" place holds the security requirements for decision making of correctness verification. The verification of ABI 1, for example, starts with the "Initiator" place generating a token to put in place "PPR." Transition "T3" checks if the first Boolean condition of ABI 1, i.e., (Patient Pulse Rate ≠ Normal), is true and if yes will fire and propagate the token to place "Action." Transition "T4" will check if the second Boolean condition of ABI 1, i.e., (Action ≠ Alert-and-Hold), is true and if yes will fire the transition and generate a token to flow to place "Decision" for decision making, as well as a token to flow to place "ABI 1" for Boolean condition verification of behavior rule 1 (in the form of ABI 1). Finally transition "T6" checks the Boolean verification of behavior rule 1 and if true will fire the transition and generate a token to place "Decision" to complete the correctness verification. In summary, we verify the completeness and correctness properties as follows:

- *Completeness*: The HCAPN model built is complete only if all physical variables, Boolean conditions, and ABIs are fully defined in the HCAPN model. All entities are fully connected by arcs in one HCAPN model and no isolated subnet exists in the HCAPN model. This is verified by conducting a token governance analysis such that all places and transitions defined are utilized at least once during HCAPN model execution. This verifies that the behavior rules specified are complete with respect to the field expert's specifications of Boolean conditions (in each behavior rule) and physical variables (in each Boolean condition).
- *Correctness*: The HCAPN model built is correct only if the reachability graph is correct, i.e., the tokens are flowing properly through the model and collected at place "Decision" (holding the security requirements) and the property of boundness is fully observed. We conduct a token reachability and boundness analysis by checking if tokens can reach all places and are bounded in all places. The token reachability defines the completeness and correctness of behavior rules and verifies that the behavior rules specified are correct with respect to the field expert's specifications of Boolean conditions (in each behavior rule) and physical variables (in each Boolean conditions). Also as tokens always flow to place "Decision" holding the security requirements, it verifies that the behavior rules satisfy the security requirements.

D. Runtime Collection of Compliance Degree Data

Unlike anomaly detection which frequently requires heavy resources to profile/learn anomaly patterns, our behavior rule specification-based data collection process is lightweight. By using the transformed state machine, a monitor device only needs to periodically monitor if a target IoT device is in safe or unsafe states without interfering with the normal operation of either the monitor device or the target device. Hence the monitor node can simply collect an instance of the compliance degree of the target node (to be monitored on) by measuring the proportion of time the target PCA node is in state 0. This collection process is repeated periodically. By the end of the *n*th monitoring periods, the monitor node would collect the compliance degree history $c_1, c_2, \dots, c_n$ of the target PCA. As the state machine has incorporated the knowledge of safe vs. unsafe states, this data collection process is extremely lightweight. The monitor node just needs to check which states the target node is in during the i^{th} monitoring interval and measures c_i as the proportional of time the target node is in safe states in the i^{th} monitoring interval. To save energy, this monitoring process can be done in discrete time space involving probing the states of the target node at discrete time points. Then an instance of the compliance degree can be measured as the ratio of the number of times in which the target node is found to be in safe states over the total number of times the monitor node probes the status of the target node.

E. Statistical Analysis for Misbehavior Detection

Our lightweight statistical analysis does not involve training, that is, we do not partition the "compliance degree" history $c_1, c_2, \dots, c_n$ collected (see Section IV-C) into the training set and the data set for testing because such heavy profiling and learning at run time is impractical for resource-constrained IoT devices. Rather, we simply model an IoT device's "compliance degree" by a random variable C following a probability distribution function $G(.)$ with the value of 0 indicating zero compliance and 1 indicating perfect compliance. Once we know the target node's compliance degree distribution function, we can compute the expected value of C to know the average compliance degree of the target node over a time period. This information will allow us to decide if the target node is considered "malicious" based on a binary grading criterion, i.e., if the target node's average compliance degree is less than or just equal to a minimum threshold C_T, we consider the target node as malicious.

In this work we consider $G(.) = Beta(\alpha, \beta)$ as the probability distribution function. The α and β parameters can be parameterized using the target node's compliance degree history $c_1, c_2, \dots, c_n$ collected during runtime. The computation overhead would be manageable because the monitor node just needs to solve the maximum likelihood equations to parameterize the α and β parameters. In this case, the run time complexity is $O(nlogn)$. If we use a one-parameter $Beta(\beta)$ distribution with α fixed at the value of 1 as the probability distribution function, the maximum likelihood estimate of β is given by a simple analytical expression: $\beta = n/\sum_{i=1}^{n} \log(1/1 - c_i)$ and the run time complexity to parameterize β using the target node's compliance degree history is only $O(n)$ which is extremely lightweight compared with contemporary anomaly based detection methods that would incur the run time complexity of $O(n^p)$ to $O(p^n)$, $p > 1$, where n is the number of data samples because of the need to profile or learn anomaly patterns.

The effectiveness of our lightweight statistical analysis method described above can be measured by the false negative probability P_{fn} and false positive probability P_{fp}. During an experimental run if a seeded "good" node's compliance degree is lower than or just equal to the minimum threshold C_T, we incur a false positive, i.e., treating a good node as a bad node. On the other hand, if during an experimental run a seeded "bad" node's compliance degree is higher than the minimum threshold C_T, we incur a false negative, i.e., treating a bad node as a good node. For the target PCA, since we know the target PCA's compliance degree distribution function $G(.)$ after applying our lightweight statistical analysis method, we can easily compute $P_{fn} = \Pr(C > C_T) = 1 - G(C_T)$ given that the PCA device is "bad" and $P_{fp} = \Pr(C \leq C_T) = G(C_T)$ given that the PCA device is "good" during experimental runs.

V. Performance Evaluation

In this section, we evaluate the performance of MedIoT in terms of "effectiveness" measured by detection rate, false positive probability, false negative probability, and area under the receiver operating characteristic (AUROC) curve, as well as in terms of "efficiency" measured by memory utilization, communication overhead, and computation overhead. We analyze the tradeoff between effectiveness and efficiency by performing a sensitivity analysis of effectiveness/efficiency performance results with respect to the monitoring rate with which compliance data are collected for misbehavior detection. We also conduct a comparative analysis with contemporary anomaly-based misbehavior detection techniques including SVM [28] and KNN [34] in both "effectiveness" and "efficiency" metrics.

A. Experiment Setup

We setup separate ns3 processes to simulate various entities as illustrated in Figure 1 to reflect reality. These ns3 processes model (1) a monitor PCA, (2) a target PCA, (3) a patient pressing/releasing a button on the target PCA for drug injection, (4) hospital personnel for defining dosage/injection rate and receiving alerts, (5) an event condition generator checking if an event condition (e.g., PCA button hold/release, Patient Pulse Rate, Patient Respiration Rate, Patient Status, Drug Reservoir, Infusion Pressure, Injection Rate, or Dosage) is triggered or not, (6) an action condition verifier checking if the target PCA is well behaved or misbehaved, and (7) a M2M communication system. Our ns3 simulation is verified to reflect reality since it covers all event conditions defined in the ABI Table (Table 4) and covers all security threats (Table 2) derived from the target PCA's operational profile.

We consider a noisy environment resulting in imperfect monitoring. That is, the monitor PCA may mis-detect the state a target PCA is in with a mis-monitoring probability due to environment noises. We model the mis-monitoring probability P_e as a random variable of a continuous uniform distribution in the interval [a, b] where a and b are both real numbers in the range of [0, 1] with $a \leq b$. Typically, we test three cases: [a, b] = [0, 10%] (i.e., low noises), [a, b] = [0, 30%] (i.e., medium noises), and [a, b] = [0, 50%] (i.e., high noises). Furthermore, we consider the following attacker types for a malicious target PCA:

1. Reckless: the malicious PCA attacks all the time whenever it has a chance, i.e., when any of the 9 ABI event conditions listed in Table 4 is true.
2. *Random*: the malicious PCA attacks randomly or on-and-off to avoid detection with an attack probability of P_a. That is, when an ABI event condition (as listed in Table 4) is true, it decides to attack with probability P_a. In the simulation, we randomly generate a number in [0, 1] and if this number is less than P_a, then the malicious PCA attacks.
3. Opportunistic: the malicious PCA attacks only when the mis-monitoring probability is high. That is, when an ABI event condition (as listed in Table 4) is true, it decides to attack only if the current mis-monitoring probability is high. In the simulation, we randomly generate a number in [a, b] (which is the range of mis-monitoring probability) and if this number is equal to or greater than (a+b)/2 then the malicious PCA attacks.
4. Insidious: the malicious PCA is hidden and attacks only under certain ABI event conditions in which the gain (causing damage) outweighs the loss (being detected). Out of the 9 ABIs listed in Table 4, ABI 8 (analgesic injection rate is above the specified injection rate) and ABI 9 (not injecting the specified dosage) will cause the most damage among all. In the simulation, the malicious PCA in insidious mode will attack only when the event conditions of ABI 8 or ABI 9 are true.

We conduct a discrete event simulation. There are 9 events as listed in Table 4, i.e., (Patient Pulse Rate ≠ Normal), (Patient Respiration Rate ≠ Normal), (Patient Status = Defibrillation), (Drug Reservoir = Empty), (Infusion Pressure ≠ Normal), (Command = AUTHORIZED), (Audit = ON), (Injection Rate > Specified Injection Rate), and (Dosage ≠ Specified Dosage). In the simulation, we simulate the arrivals of these 9 events by Poisson arrival processes with rates λ_{ABI1}- λ_{ABI9}, respectively. For example, to simulate the next ABI 1 event occurrence time, we randomly generate a number following exponential distribution with parameter λ_{ABI1} and the next ABI 1 event would occur at now plus this number. Initially, we generate the event occurrence times of the 9 ABI events each and put them in the event queue sorted by event occurrence time. These events are processed one at a time based on their event occurrence times. After an event is processed, the simulation clock is advanced to the event occurrence time in the discrete time space. If multiple ABI events occur at the same time, they are also being processed one at a time, but the simulation clock will advance only after all are processed. After an ABI event is processed, the next event occurrence time for the same ABI event is generated again and placed in the event queue, so the event-driven simulation continues running until a stopping criterion is met (discussed below).

The monitor PCA monitors (what it perceives) the state machine status of a target PCA for the purpose of misbehavior detection. It applies the same mechanical way of updating the state machine of the target PCA regardless of whether the target PCA is malicious or not (which it does not know). The start state at t=0 initially is (0, 0, 0, 0, 0, 0, 0, 0, 0). When an event occurs, the monitor PCA updates a target PCA's state machine. Below we separate the discussion for the malicious target PCA and the good target PCA because the resulting state machine as perceived by the monitor PCA will be different.

The good target PCA would not attack when an ABI event occurs. Therefore, the good target PCA would stay in state (0, 0, 0, 0, 0, 0, 0, 0, 0) forever. However, the monitor PCA can mis-detect because of the mis-monitoring probability P_e. In the simulation we first randomly generate a number in [a, b] for the current mis-monitoring probability. Then we randomly generate another number in [0, 1]. If this number is larger than the current mis-monitoring probability, then it means that there is no mis-monitoring and the monitor PCA will put a "0" in the state component position corresponding to the ABI event. Otherwise, the monitor PCA puts a "1" in the state component position corresponding to the ABI event as the good target PCA is mis-identified as performing attack by the monitor PCA.

The malicious target PCA may or may not attack when an ABI event occurs, depending on its attacker type. For example, a reckless attacker always attacks whenever an event occurs. However, an opportunistic attacker attacks only when the environment noise is high. There are two cases:

- If the malicious target PCA does not attack, then the monitor PCA will update the state machine of the malicious target PCA in the same way it updates the state machine of a good target PCA (who does not attack) as described earlier.
- If the malicious target PCA does attack, the monitor PCA can detect the attack subject to mis-monitoring probability P_e under which the malicious target PCA's attack is not detected. In the simulation we again first randomly generate a number in [a, b] for generating the current mis-monitoring probability P_e. Then we randomly generate another number in [0, 1]. If this number is larger than the current mis-monitoring probability, then it means that there is no mis-monitoring and the attack is detected by the monitor PCA which will update the state machine of the malicious target PCA by putting a "1" in the state component position corresponding to the ABI event. For example if ABI 1 event occurs and the malicious target PCA decides to attack and the attack is detected, then the malicious target PCA's state machine is updated from (0, 0, 0, 0, 0, 0, 0, 0, 0) to (1, 0, 0, 0, 0, 0, 0, 0, 0). Otherwise, the monitor PCA puts a "0" in the state component position corresponding to the ABI event since the attack is mis-detected by the monitor PCA.

The monitor PCA updates the state machine of a target PCA it is assigned to monitor at every ABI event occurrence time in the event-driven simulation. While updating the state machine,

the monitor PCA keeps track of the state transitions that have occurred and collects the target PCA's compliance degree history $c_1, c_2, \ldots, c_n$ as discussed in Section IV-D. The target PCA's compliance degree history $c_1, c_2, \ldots, c_n$ is then used to parameterize the β parameter from the one-parameter $Beta(\beta)$ distribution with α being fixed at the value of 1. We stop the simulation when the following stopping criterion is met: when the β parameter value converges based on 95% confidence level and 10% confidence accuracy. That is, with 95% confidence, the mean β value obtained is within the true mean by 10%. Once the β value is obtained, we obtain the target PCA's compliance degree distribution function, based on which the monitor PCA computes the false negative probability (P_{fn}) and false positive probability(P_{fp}) by adjusting the minimum compliance threshold C_T as described in Section IV-E.

We verify that MedIoT performs better than or comparably against contemporary anomaly-based misbehavior detection techniques including Support Vector Machine (SVM) [28] and k-nearest neighbors (KNN) [34] in "effectiveness" metrics. Moreover, we verify that MedIoT can dominantly outperform SVM and KNN in "efficiency" metrics, thus supporting our claim that MedIoT is the only feasible misbehavior detection scheme applicable to resource-constrained IoT devices.

SVM [28] is a popular data classification technique for classifying data into groups and has been well adopted for misbehavior detection for CPSs [29, 30]. We use it to classify a target IoT device into "well-behaved" or "misbehaved" groups based on the data collected by the monitor device on the target device. Applying SVM is a two-stage process: in the "training" stage, we train a SVM classifier with examples of "well-behaved" or "misbehaved" data collected from good and bad target IoT devices, respectively, so the SVM classifier can learn their data patterns and classify the target IoT device into the right group. In the "application" stage, the trained SVM classifier takes new data collected at run time and classifies the responsible target device into the "well-behaved" or "misbehaved" group. Typically, when new data arrives, the SVM classifier should be retrained so it can learn from experiences dynamically. SVM involves identifying the main features of a device so that it can effectively differentiate "well-behaved" from "misbehaved" data patterns. These key features are variables whose values are collected at runtime and are put into a *p*-dimensional support vector in the form of $(X_1, X_2, \ldots, X_p)$ where X_i is the i^{th} variable. For fair comparison, we use the same physical variables identified in our ABI table (Table 4) as SVM variables. That is, $(X_1, X_2, \ldots, X_p)$ = (Patient Pulse Rate , Patient Respiration Rate, Patient Status, Drug Reservoir, Infusion Pressure, Command, Audit, Injection Rate – Specified Injection Rate, Dosage – Specified Dosage) with p=9. Note that the last two variables are represented as the differences between the current measured quantity and the specified quantity to allow effective data pattern learning. During the training stage, a data sample (containing p support vector variable values measured when an ABI event occurs) is in the form of $(X_1, X_2, \ldots, X_p, y)$ where y is 1 for "well-behaved" and −1 for "misbehaved." During the application stage, y is the output of SVM based on which the target node is classified into "well-behaved" or "misbehaved." The output produced by SVM is collected as an instance of the target device's compliance degree c_i.

K-nearest neighbors (KNN) [34] is also a popular classification scheme which we take as a 2nd baseline machine learning algorithm for performance comparison. For fair comparison, we also use the same feature vector as in SVM. That is, the feature vector is $(X_1, X_2, \ldots, X_p)$ = (Patient Pulse Rate , Patient Respiration Rate, Patient Status, Drug Reservoir, Infusion Pressure, Command, Audit, Injection Rate – Specified Injection Rate, Dosage – Specified Dosage) with p=9. Different from SVM, KNN simply stores the feature vectors and the corresponding class labels of training samples in the form of $(X_1, X_2, \ldots, X_p, y)$ where y is 1 for the "well-behaved" class and −1 for the "misbehaved" class. An automatic hyperparameter optimization method [35] is used to generate an optimized model which suggests the best k neighbors and distance algorithm for classification. During the application stage when being presented with a feature vector of a target node, KNN assigns a label to the target node which is the most frequent among the k training samples nearest to the feature vector of the target node to classify the target node as "well-behaved" or "misbehaved." The output produced by KNN is then collected as an instance of the target device's compliance degree c_i.

In our experiment, MedIoT, SVM and KNN are tested under the same operational and environment settings as described above, including the mis-monitoring probability P_e. SVM and KNN are trained with 6000 training events (or cases) including both well-behaved and misbehaved events. Then all schemes are tested with 3000 events in a simulation run, using the same data input for analysis when an event occurs. When an event occurs, all schemes acquire the event input data in the form of (Patient Pulse Rate, Patient Respiration Rate, Patient Status, Drug Reservoir, Infusion Pressure, Command, Audit, Injection Rate - Specified Injection Rate, Dosage - Specified Dosage) via data communication from various sensing or measuring devices in exactly the same way. When the *i*th event occurs, the target device's compliance degree c_i is collected as follows: MedIoT checks if the end state is a safe or unsafe state by checking against its state machine, while SVM and KNN classify if the target device is well-behaved or misbehaved by searching through the misbehavior space learned during the training phase. After the target PCA's compliance degree history $c_1, c_2, \ldots, c_n$ is collected, we apply the lightweight statistical analysis method as described in Section IV-E for computing the false negative rate (P_{fn}) and false positive rate (P_{fp}) for all schemes.

B. Effectiveness Performance Evaluation

We measure *effectiveness* by the following performance metrics: (a) detection rate: probability of correctly identifying a malicious node (goal: above 99% for reckless attackers); (b) false negative probability (P_{fn}): it is equal to 1 – detection rate (goal: below 1% for reckless attackers); (c) false positive probability (P_{fp}): probability of misidentifying a good node as a malicious node (goal: below 3%); (d) AUROC: area under a receiver operating characteristic (ROC) curve with detection rate vs. false positive probability (goal: above 97% for reckless attackers). In particular, the AUROC curve with the detection

rate $(1 - P_{fn})$ on the Y coordinate and the false positive rate (P_{fp}) on the X coordinate is especially a well-adopted metric for performance comparison of misbehavior detection methods because it can properly reflect the tradeoff between false negative rate (P_{fn}) and false positive rate (P_{fp}).

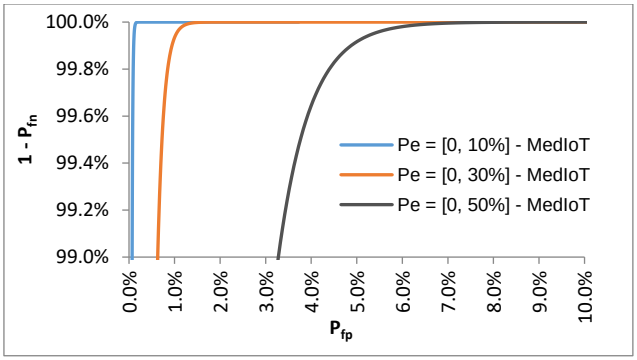

Fig. 3. ROC Curve for MedIoT under various noisy environments. The Y coordinate is the detection rate $(1 - P_{fn})$ and the X coordinate is the false positive rate(P_{fp}).

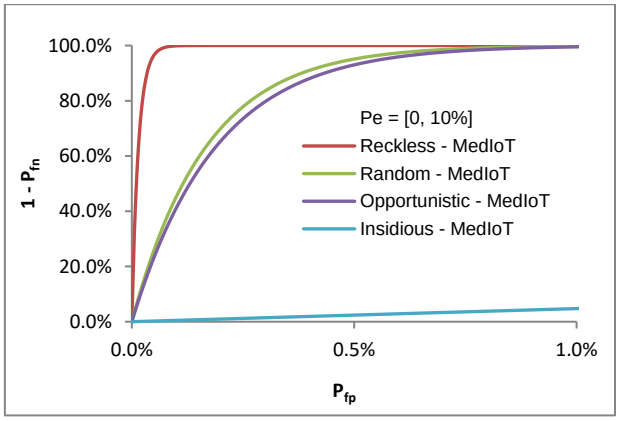

Fig. 4. Effect of attacker type on ROC of MedIoT.

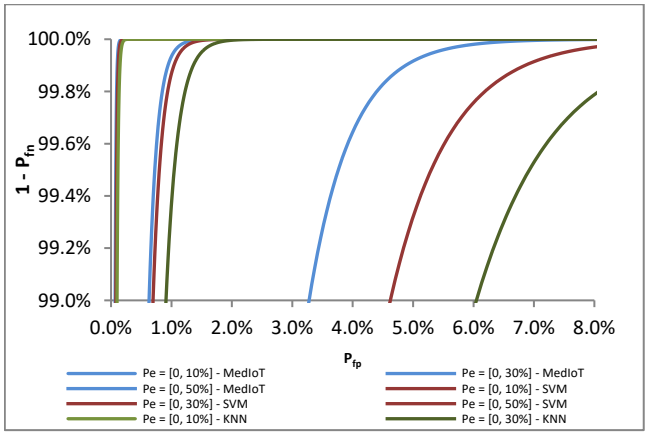

Fig. 5. Comparing AUROC of MedIoT against SVM and KNN for the case in which the attacker type is reckless under various noisy environments.

Figure 3 displays the ROC curves for the case in which the malicious PCA is a reckless attacker. Each ROC curve has the mis-monitoring probability P_e in a different $[a, b]$ range. The top curve is for $[a, b] = [0, 10\%]$ (i.e., low noises). The middle curve is for $[a, b] = [0, 30\%]$ (i.e., medium noises). The bottom curve is for $[a, b] = [0, 50\%]$ (i.e., high noises). We see that under low noises, the AUROC is close to 1 because both the false negative rate (P_{fn}) and false positive rate (P_{fp}) are very close to zero and therefore AUROC is close to 100% This means that MedIoT can achieve almost perfect detection under low noises. As the environment noise degree increases, the AUROC is not 1 anymore and there is a tradeoff between P_{fn} and P_{fp}. That is, if we increase the minimum compliance threshold C_T, we can reduce P_{fn} but increase P_{fp}. Conversely, if we decrease the minimum compliance threshold C_T, we can reduce P_{fp} but increase P_{fn}. We see that under medium noises, MedIoT can achieve $P_{fn} < 1\%$, $P_{fp} < 0.5\%$, and AUROC > 99%. Even under high noises MedIoT can still achieve the goal of $P_{fn} < 1\%$, $P_{fp} < 3\%$, and AUROC > 97% until it breaks when P_e is above [0, 45%] because of the very high level of noise causing it not able to perform accurate misbehavior detection. Overall we see that MedIoT can achieve the goal of $P_{fn} < 1\%$, $P_{fp} < 3\%$, and AUROC > 97% over a wide range of noise levels (up to [0, 45%] as observed from our experiment) by adjusting the value of the minimum compliance threshold C_T in response to the environment noise level sensed at runtime. When the noise level is high, MedIoT sets the minimum compliance threshold C_T to a high bar, so that it can achieve above 99% of detection rate for detecting the malicious target PCA, while limiting the false positive probability P_{fp} to under 3% for misjudging the good target PCA. To preserve memory space, we use a very simple table look-up method to match the noise level detected at runtime to the best C_T level at which the goal of $P_{fn} < 1\%$, $P_{fp} < 3\%$, and AUROC > 97% can be optimized in the priority order of P_{fn}, P_{fp}, and AUROC.

Figure 4 analyzes the effect of attacker type on detection accuracy of MedIoT. Figure 4 displays 4 ROC curves, each for a distinct attacker type (reckless, random with $P_a = 0.5$, opportunistic, or insidious) for the case in which $[a, b] = [0, 10\%]$ (i.e., low noises) to reveal interesting performance characteristics. The top curve is when the malicious target PCA is a reckless attacker for which the AUROC is nearly 1 because the attacker exposes itself by reckless attack behavior and can be easily detected by MedIoT. The 2nd highest curve is when the malicious target PCA is a 50% random attacker for which the AUROC is not as close to 1 because the attacker is hidden 50% of the time. The third curve from the top is when the malicious target PCA is an opportunistic attacker who will attack when it detects the current mis-monitoring probability is above the average. The bottom curve is when the malicious target PCA is an insidious attacker for which the AUROC is the lowest because an insidious attacker attacks only when ABI 8 or ABI 9 event conditions are true while it behaves under all other ABI event conditions. The malicious PCA in this case is mostly hidden and does not attack to avoid detection unless it sees ABI 8 or ABI 9 conditions are true. As a result, it limits its chance of exposure and thus decreases the detection accuracy of MedIoT.

Figure 4 indicates that MedIoT can easily satisfy the security goals when the attacker type is reckless, random (with $P_a = 0.5$) or opportunistic but MedIoT can fail miserably when the attacker type is insidious even under low noises. As we claim MedIoT performs better than contemporary anomaly detection methods such as SVM (to be shown later) in AUROC, we believe that to-date there is no single misbehavior detection method that can effectively counter insidious attackers. One

way to counteract insidious attack behavior is to severely penalize a target PCA that violates critical ABIs (that will cause severe damage to the system) such as ABI 8 and ABI 9. Specifically, instead of assigning equal weights to all ABIs and consequently measuring c_i as the proportion of time the target PCA is in safe states in the ith monitoring interval, we assign heavier weights to critical ABIs and consequently measuring $1 - c_i$ (i.e., non-compliance) as the proportion of weighted-time the target PCA is in unsafe states in the ith monitoring interval. Since we put higher weights to critical unsafe states, this in effect penalizes a target PCA that violates critical ABIs by assigning it with a high non-compliance degree (or a low compliance degree). This is to be investigated in future work.

Figure 5 compares the ROC curves for MedIoT (blue lines) against SVM (red lines) and KNN (green lines) for the case in which the malicious PCA is a reckless attacker. We can see clearly that the AUROC of MedIoT dominates that of SVM and KNN as MedIoT always has a better detection rate given the same P_{fp} and always has a better P_{fp} given the same detection rate, given the same operational and environmental characteristics as input. While MedIoT breaks when the mis-monitoring probability P_e is above the range of [0, 45%], SVM breaks (i.e., not able to satisfy the goal of $P_{fn} < 1\%, P_{fp} < 3\%$, and AUROC > 97%) when P_e is above the range of [0, 33%] and KNN breaks when P_e is above the range of [0, 31%]. We attribute the superior performance of MedIoT by the unique design that safe and unsafe-state information is already summarized in the transformed state machine, based on which a monitor node just needs to measure the proportion of time a target node (being monitored on) is in safe states for effective misbehavior detection regardless of the attacker type of the malicious PCA (reckless, random, opportunistic, or insidious). In contrast, SVM must be trained with well-behavior/misbehavior examples in the training phase to learn a hyperplane that can include well-behavior data and exclude misbehavior data and then in the application phase use the hyperplane to classify misbehavior data. There is inherently some imprecision in learning and profiling the "ideal" hyperplane especially when the environment noises are high. Consequently, SVM performs worse than MedIoT. Similarly there is some imprecision for KNN to find k training samples nearest to the feature vector of the target node to classify the target node as "well-behaved" or "misbehaved."

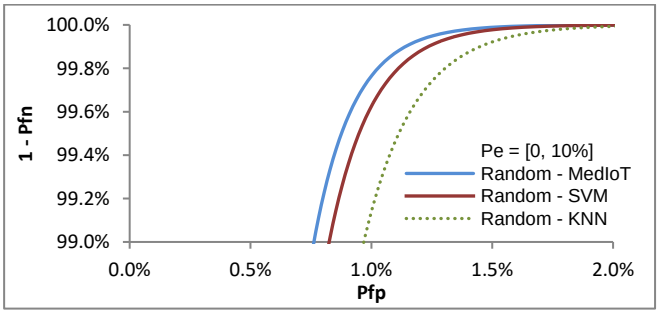

Fig. 6. Comparing AUROC of MedIoT against SVM and KNN for the case in which the attacker type is random.

Figures 6, 7, and 8 compare the ROC curves for MedIoT (blue lines) against SVM (red lines) and KNN (green lines) for the cases in which the malicious PCA's attacker types are random (with $P_a = 0.5$), opportunistic, and insidious, respectively. A similar trend is observed compared with the reckless attacker type case (as shown in Figure 5). That is, the AUROC of MedIoT dominates that of SVM and KNN. Here we note that despite the attacker is more hidden, especially in the case of insidious attackers, MedIoT is still more effective than SVM and KNN because of the use of state-machine-based monitoring which greatly improves runtime detection accuracy.

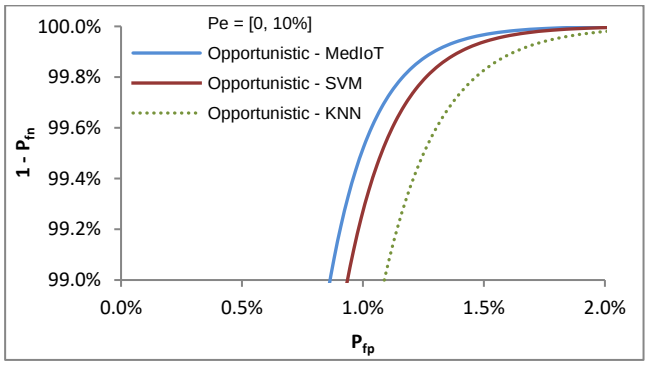

Fig. 7. Comparing AUROC of MedIoT against SVM and KNN for the case in which the attacker type is opportunistic.

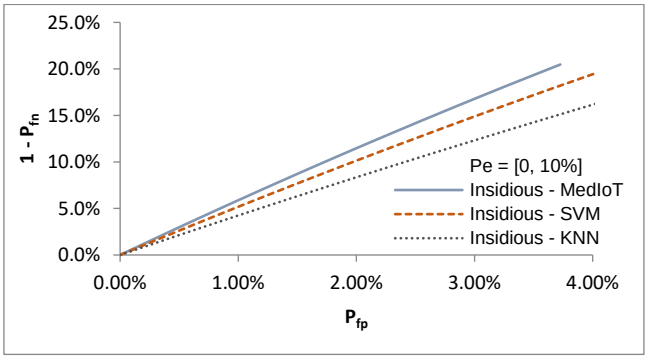

Fig. 8. Comparing AUROC of MedIoT against SVM and KNN for the case in which the attacker type is insidious.

C. *Efficiency Performance Evaluation*

We evaluate the efficiency aspect of MedIoT by the following metrics: (a) the memory utilization: the amount of memory utilized (goal: less than 100MB); (b) the computation overhead: the amount of computation time needed for misbehavior detection (goal: below 5 milliseconds or 0.005 seconds); and (c) the time delay for reaching a decision about whether the target PCA is well-behaved or misbehaved when an event occurs (goal: below 10 milliseconds or 0.01 seconds). The time delay for reaching a decision when an event occurs is the sum of the time taken for the monitor PCA to obtain event input data in the form of (Patient Pulse Rate, Patient Respiration Rate, Patient Status, Drug Reservoir, Infusion Pressure, Command, Audit, Injection Rate - Specified Injection Rate, Dosage - Specified Dosage) from various sensing or measuring devices via data communication plus the time for each scheme (MedIoT, SVM, or KNN) to analyze event input data to reach a decision. The first part of the delay is the same for all schemes because all schemes use the same event input when an event occurs. The second part of the delay is the computation time for reaching a decision, given that the event data is in hand. For MedIoT, it is the computation time to decide if the target PCA is in an unsafe state in the state machine which it updates on an

event by event basis, and for SVM or KNN it is the learning algorithm computation time to classify if the target PCA is in the "misbehaved" class.

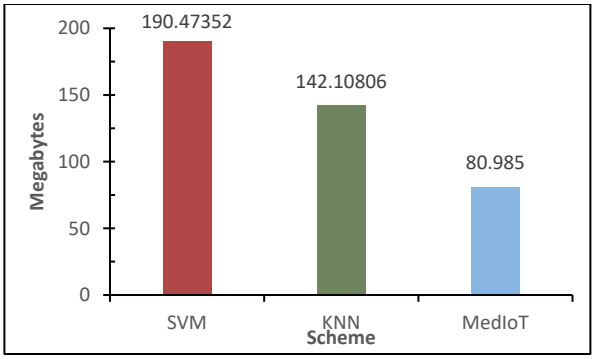

Fig. 9a. Memory Utilization of MedIoT against SVM and KNN.

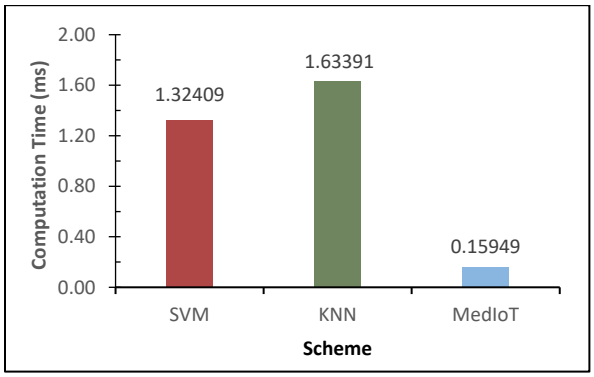

Fig. 9b. Computation Time of MedIoT against SVM and KNN.

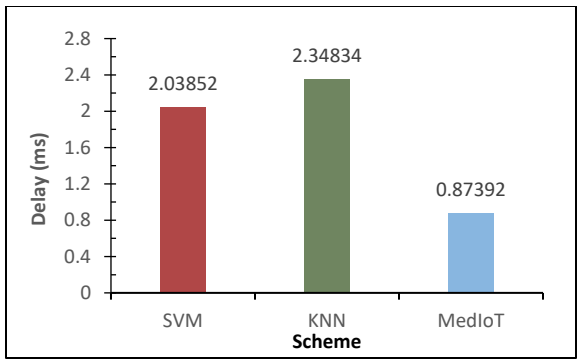

Fig. 9c. Time delay in acquiring and analyzing data to reach a decision of MedIoT against SVM and KNN when an event occurs.

Figures 9a, 9b, and 9c compare the memory size, computation time, and time delay for reaching a decision, respectively, of MedIoT against SVM and KNN for the case in which the malicious PCA is reckless and [*a*, *b*] = [0, 10%] (i.e., low noises). Figure 9a shows that the memory usage of MedIoT (80 MB) is substantially smaller than that of SVM (190MB) and KNN (142MB). This means that for a resource constrained PCA device with a memory size as small as 100MB [31], only MedIoT (our proposed scheme) can satisfy this memory usage goal. This is mainly because MedIoT only needs to store a relatively small state machine generated at static time for identifying safe/unsafe states while SVM/KNN need to store a large amount of trained patterns for classifying misbehavior. Figure 9b compares MedIoT against SVM and KNN in terms of the computation time (excluding the learning time) needed to reach a decision after data are in place for all methods. Figure 9b shows that MedIoT has substantially lower computational time to perform misbehavior detection compared with SVM and KNN. The reason is that MedIoT already preloads the state machine into memory for runtime misbehavior detection. On the contrary, SVM needs to search through the misbehavior space represented by its stored hyperplanes at runtime for misbehavior classification, and KNN needs to search through its stored training samples to find *k* training samples nearest to the feature vector of the target node for group classification. Figure 9c compares MedIoT against SVM and KNN in terms of the time delay in acquiring and analyzing data to reach a decision computed by the sum of the data communication time and the computation time for all three methods. Figure 9c shows that MedIoT incurs a substantially smaller time delay in acquiring and analyzing data to reach a decision whether the target PCA is well-behaved or misbehaved when an event occurs compared with SVM and KNN. We again attribute MedIoT's superior performance to its fast safe/unsafe state look-up operation merely by inspecting the target PCA's state machine as opposed to the relatively slow machine learning operation performed by SVM or KNN to classify the target PCA into the "well-behaved" or "misbehaved" class.

D. Tradeoff Analysis of Effectiveness vs. Efficiency

In this section we analyze the tradeoff between effectiveness vs. efficiency. Specifically, we analyze the effect of the monitoring rate (the frequency at which compliance data are collected) on effectiveness/efficiency performance results. The tradeoff exists because a finer data granularity due to more frequent monitoring will lead to less missing cases and improve the detection rate, but it can exhaust energy of a resource-constrained IoT device due to excessive monitoring. We aim to identify the best monitoring rate that can best balance efficiency and effectiveness.

In the simulation, instead of having the monitor PCA check the state machine of a target PCA at event occurrence times, we create a timer event with T_{IDS} as the monitoring interval so that a data compliance degree sample is collected in each T_{IDS} period after which the monitor PCA updates the target PCA's state machine. In effect, the monitor PCA collects the target PCA's compliance samples periodically with rate $1/T_{IDS}$. Obviously the smaller the T_{IDS} value, the finer the data granularity and hence the better the *effectiveness* performance because of a higher chance of not missing any misbehavior of the target PCA device. However, a smaller T_{IDS} value increases the computation time needed for data collection and statistical analysis because a higher number of samples (*n*) would need to be collected and analyzed. This degrades *efficiency* performance. Thus, we analyze the tradeoff between effectiveness and efficiency by controlling the magnitude of T_{IDS} which essentially translates into the total number of samples (*n*) to be collected for decision making, which is directly related to the computation overhead (for compliance data collection and statistical analysis) since the runtime complexity of MedIoT is O(*n*) as discussed earlier in Section IV-E.

Figure 10 shows AUROC (the most important effectiveness metric) under various T_{IDS} values (each resulting a different *n* value) representing the computation overhead (the most

important efficiency metric) for the case in which the malicious PCA is a reckless attacker and [*a*, *b*] = [0, 10%] (i.e., low noises). We observe that there indeed exists a tradeoff between effectiveness measured by AUROC and efficiency measured by the detection interval and consequently the number of compliance samples (*n*) collected during the simulation run. Specifically, as the detection interval T_{IDS} decreases, the efficiency performance decreases since the computation time increases (due to more data samples being collected during data collection and being analyzed during statistical analysis), while the effectiveness performance increases as AUROC becomes more and more approaching 1 due to more sample data being analyzed. Depending on the effectiveness and efficiency performance requirements of the system, the tradeoff analysis as shown in Figure 10 can help the system designer identify the best monitoring rate to be applied to satisfy both requirements.

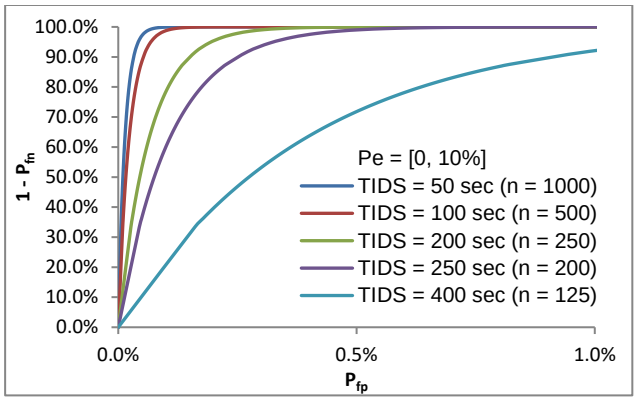

Fig. 10. MedIoT design tradeoff between effectiveness (measured by AUROC) vs. efficiency (measured by monitoring interval during data collection and run time complexity during statistical analysis for misbehavior detection).

VI. Conclusion

The proposed behavior rule specification-based misbehavior detection technique is generic and can be applied to practical IoT-embedded cyber physical systems for which very lightweight embedded IoT devices (e.g., sensors, actuators, or a combination of both) are an integral part of the overall system design. We illustrated the feasibility of our proposed method with a PCA device embedded in a medical CPS where a peer PCA serves the role of a monitor node. We position our behavior rule specification-based misbehavior detection technique as the only feasible solution in terms of low memory, run time, communication, and computation overhead, and high misbehavior detection prediction accuracy to ensure protection of resource-constrained embedded IoT devices against zero-day attacks. In this paper we conducted extensive simulation to verify that MedIoT can outperform SVM-based or KNN-based machine learning methods for misbehavior detection of a PCA device embedded in a medical CPS in both the effectiveness and efficiency performance metrics. In the future, we plan to port the data collection and statistical analysis code to a real PCA device for experimental validation.

References

[1] R. Berthier and W.H. Sanders. 2011. Specification-based Intrusion Detection for Advanced Metering Infrastructures. *17th IEEE Pacific Rim Int. Symp. Dependable Computing*, 184-193.

[2] A. Bezemskij, G. Loukas, R.J. Anthony, and D. Gan. 2016. Behaviour-based anomaly detection of cyber-physical attacks on a robotic vehicle. *IEEE Symposium on Cyberspace and Security*, 1-8.

[3] M. Aldebert, M. Ivaldi and C. Roucolle, "Telecommunications Demand and Pricing Structure: An Econometric Analysis," *Telecommunication Systems*, 25:89–115, 2004.

[4] A. DaSilva et al. 2005. Decentralized intrusion detection in wireless sensor networks. *1st ACM inter. workshop on quality of service & security in wireless and mobile networks*., 16–23.

[5] N. Zhang, K. Sun, W. Lou, and Y.T. Hou. 2016. CaSE: Cache-Assisted Secure Execution on ARM Processors. *IEEE Symposium on Security and Privacy*.

[6] J. Hong, C.C. Liu, and M. Govindarasu. 2014. Integrated Anomaly Detection for Cyber Security of the Substations. *IEEE Trans. Smart Grid*, 5 (4), 1643-1653.

[7] S. Huda, et al. 2017. Defending unknown attacks on cyber-physical systems by semi-supervised approach and available unlabeled data. *Information Sciences*, 379, 211-228.

[8] K. Ioannis, T. Dimitriou, and F. Freiling. 2007. Towards intrusion detection in wireless sensor networks. *13th European Wireless Conference*.

[9] P. Jokar, H. Nicanfar, and V.C.M. Leung. 2011. Specification-based Intrusion Detection for Home Area Networks in Smart Grids. *IEEE Int. Conf. on Smart Grid Communications*.

[10] A.M. Kosek. 2016. Contextual anomaly detection for cyber-physical security in smart grids based on an artificial neural network model. *IEEE Workshop on Cyber-Physical Security and Resilience in Smart Grids*.

[11] C. Kwon, S. Yantek, and I. Hwang. 2016. Real-Time Safety Assessment of Unmanned Aircraft Systems Against Stealthy Cyber Attacks. *Journal of Aerospace Information Systems*, 13 (1), 27-46.

[12] R. Mitchell, and I.R. Chen. 2014. A Survey of Intrusion Detection Techniques in Cyber Physical Systems. *ACM Computing Survey*, 46 (4), article 55.

[13] R. Mitchell and I.R. Chen. 2016. Modeling and Analysis of Attacks and Counter Defense Mechanisms for Cyber Physical Systems. *IEEE Transactions on Reliability*, 65 (1), 350-358.

[14] S. Ntalampiras. 2016. Automatic identification of integrity attacks in cyber-physical systems. *Expert Systems with Applications*, 58, 164-173.

[15] Y. Zhang, L. Wang, W. Sun, R. Green, and M. Alam. 2011. Distributed intrusion detection system in a multi-layer network architecture of smart grids. *IEEE Trans. Smart Grid*, 2 (4), 796–808.

[16] J. Musa. 1993. Operational profiles in software reliability engineering. *IEEE Software*, 14–32.

[17] R. Mitchell and I.R. Chen. 2015. Behavior Rule Specification-based Intrusion Detection for Safety Critical Medical Cyber Physical Systems. *IEEE Transactions on Dependable and Secure Computing*, 12 (1), 16-30.

[18] R. Mitchell and I.R. Chen. 2014. Adaptive Intrusion Detection of Malicious Unmanned Air Vehicles Using Behavior Rule Specifications. *IEEE Transactions on Systems, Man and Cybernetics*, 44 (5), 593-604.

[19] V. Sharma, I. You, R. Kumar, "Energy Efficient Data Dissemination in Multi-UAV Coordinated Wireless Sensor Networks," *Mobile Information Systems*, Volume 2016, Article ID 8475820, June 2016

[20] V. Sharma, I. You, R. Kumar, P. Kim "Computational offloading for efficient trust management in pervasive online social networks using osmotic computing" *IEEE Access*, March 2017.

[21] T. Song, et al. 2006. Formal Reasoning about a Specification-based Intrusion Detection for Dynamic Auto-configuration Protocols in Ad Hoc Networks. *Formal Aspects in Security and Trust*, 16-33.

[22] K. Park, Y. Lin, V. Metsis, Z. Le, and F. Makedon. 2010. Abnormal human behavioral pattern detection in assisted living environments. *3rd ACM Int. Conf. Pervasive Technol. Related Assist. Environments*, 9:1–9:8.

[23] V. Sharma, I. You, R. Kumar, "ISMA: Intelligent sensing model for anomalies detection in cross platform OSNs with a case study on IoT" *IEEE Access*, January 2017.

[24] B.B. Zarpelao, R.S. Miani, C.T. Kawakani, and S.C. de Alvarenga. 2017. A Survey of Intrusion Detection in Internet of Things. *Journal of Network and Computer Architecture*, 84, 25-37.

[25] A. Saeed, A. Ahmadinia, A. Javed, and H. Larikani. 2016. Intelligent Intrusion Detection in Low-Power IoTs. *ACM Trans. Internet Technology*, 16 (4), article 27.

[26] M.T. Khan, D. Serpanos, and H. Shrobe. 2017. ARMET: Behavior-based Secure and Resilient Industrial Control Systems. *Proceedings of The IEEE*.

[27] M. Kaufmann and J.S. Moore. 2017. A Computational Logic for Applicative Common Lisp. http://www.cs.utexas.edu/users/moore/acl2/.

[28] Support Vector Machine. https://en.wikipedia.org/wiki/Support-vector_machine.

[29] H. Sedjelmaci, S.M. Senouci, and N. Ansari. 2018. A Hierarchical Detection and Response System to Enhance Security Against Lethal Cyber-Attacks in UAV Networks. *IEEE Trans. Systems, Man, and Cybernetics: Systems*, 48 (9), 1594-1606.
[30] M. Attia, S.M. Senouci, H. Sedjelmaci, E. Aglzima, and D. Chrenko. 2018. An efficient Intrusion Detection System against cyber-physical attacks in the smart grid. Computers and Electrical Engineering, 68, 499-512.
[31] B. Sherman, I. Enu, and R. Sinatra. 2009. Patient-Controlled Analgesia Devices and Analgesic Infusion Pumps. In H. McQuay (Author) & R. Sinatra, O. De Leon-Cassasola, E. Viscusi, & B. Ginsberg (Eds.), *Acute Pain Management* (pp. 302-322). Cambridge: Cambridge University Press. doi:10.1017/CBO9780511576706.021
[32] V. Sharma, I. You, K. Yim, I.R. Chen, and J.H. Cho. BRIoT: Behavior Rule Specification-based Misbehavior Detection for IoT-Embedded Cyber-Physical Systems. *IEEE Access*, vol. 7, no. 1, 2019, pp. 118556-118580.
[33] V. Sharma, G. Choudhary, Y. Ko, and I. You. Behavior and vulnerability assessment of drones-enabled industrial internet of things (iiot). *IEEE Access*, vol. 6, 2018, pp. 43368-43383.
[34] K-nearest Neighbors Algorithms. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.
[35] MathWorks "fitcknn". https://www.mathworks.com/help/stats/fitcknn.html
[36] Petri.Net Simulator. https://github.com/larics/Petri.Net.
[37] W. Sun, R. Zhang, W. Lou, and Y. T. Hou, "Rearguard: Secure keyword search using trusted hardware," *IEEE Conference on Computer Communications*, 2018.
[38] V. Costan and S. Devadas, "Intel SGX Explained," *IACR Cryptology ePrint Archive*, pp. 1-86, 2016.
[39] T. Anderson and R. Kerr "Recovery Blocks in Action: A System Supporting High Reliability," *Reliable Computer Systems*, Springer, Berlin, Heidelberg, 1985.
[40] MQTT 3.1.1 specification. OASIS. December 10, 2015. Retrieved April 25, 2017.
[41] Lightweight Machine to Machine Requirements: Version 1.1 – 10 Jul 2018 Open Mobile Alliance (OMA-RD-LightweightM2M-V1_1-20180710-A).

Gaurav Choudhary received the B.Tech. degree in Computer Science and Engineering from Rajasthan Technical University in 2014 and the Master Degree in Cyber Security from Sardar Patel University of Police in 2017. He is currently pursuing Ph.D. degree in the Department of Information Security Engineering, Soonchunhyang University, Asan, South Korea. His areas of research and interests are UAVs, IoT security, Network security, and Vulnerability Assessment.

Philip Virgil Astillo received the B.S. degree in computer engineering from University of San Carlos, Cebu, Philippines in 2009 and the M.Eng. Degree in computer engineering from the same university in 2011. He is currently pursuing the Ph.D. degree in Information Security Engineering at Soochunhyang University, South Korea. His research interests include sensor development, embedded system design and development, mobile Internet security, and IoT security.

Ilsun You (M'12-SM'13) received the M.S. and Ph.D. degrees in computer science from Dankook University, Seoul, South Korea, in 1997 and 2002, respectively, and the Ph.D. degree from Kyushu University, Japan, in 2012. He is currently an Associate Professor with the Department of Information Security Engineering, Soonchunhyang University, South Korea. His main research interests include the Internet security, authentication, access control, and formal security analysis. He is a Fellow of the IET. He is the EiC of the Journal of Wireless Mobile Networks, Ubiquitous Computing, and the Dependable Applications (JoWUA), and the Journal of Internet Services and Information Security (JISIS). He is on the Editorial Board of the Information Sciences the Journal of Network and Computer Applications, the International Journal of Ad Hoc and Ubiquitous Computing, Computing and Informatics, the Intelligent Automation and Soft Computing, and so on.

Kangbin Yim received the B.S., M.S., and Ph.D. degrees from the Department of Electronics Engineering, Ajou University, Suwon, South Korea, in 1992, 1994, and 2001, respectively. He is currently a Professor with the Department of Information Security Engineering, Soonchunhyang University. His research interests include vulnerability assessment, code obfuscation, malware analysis, leakage prevention, secure platform architecture, and mobile security. Related to these topics, he has involved in more than 60 research projects and published more than 100 research papers. He has served as an Executive Board Member of the Korea Institute of Information Security and Cryptology, the Korean Society for Internet Information, and The Institute of Electronics Engineers of Korea. He also has served as a Committee Chair of the international conferences and workshops and the Guest Editor of the journals, such as JIT, MIS, JCPS, JISIS, and JoWUA.

Ing-Ray Chen (M'90) received the B.S. degree from National Taiwan University, and the M.S. and Ph.D. degrees in computer science from the University of Houston. He is currently a Professor with the Department of Computer Science, Virginia Tech. His research interests include trust and security, network and service management, and reliability and performance analysis of mobile wireless networks and cyber physical systems. He was a recipient of the IEEE Communications Society William R. Bennett Prize in Communications Networking and of the U.S. Army Research Laboratory (ARL) Publication Award. Dr. Chen currently serves as an associate editor for the IEEE TRANSACTIONS ON SERVICES COMPUTING, the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, and The Computer Journal.

Jin-Hee Cho (SM'14) received the M.S. and Ph.D. degrees in computer science from Virginia Tech, in 2004 and 2008, respectively, where she has been an Associate Professor with the Department of Computer Science, since 2018. She has been a Computer Scientist with the U.S. Army Research Laboratory (USARL), Adelphi, MD, USA, since 2009. She has published more than 100 peer-reviewed technical papers in leading journals and conferences in the areas of trust management, cyber security, metrics and measurements, network performance analysis, resource allocation, agent-based modeling, uncertainty reasoning and analysis, information fusion/credibility, and social network analysis. She is a member of the ACM. She received the Best Paper Awards in the IEEE TrustCom'2009, BRIMS'2013, IEEE GLOBECOM'2017, 2017 ARL's Publication Award, and IEEE CogSima 2018. She was a recipient of the 2015 IEEE Communications Society William R. Bennett Prize in communications networking. In 2016, she was selected for the 2013 Presidential Early Career Award for Scientists and Engineers (PECASE), which is the highest honor bestowed by the U.S. government on outstanding scientists and engineers in the early stages of their independent research careers. Dr Cho id an associate editor for IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT and The Computer Journal.