Highly Assured Safety and Security of e-Health Applications

Muhammad Taimoor Khan
Surrey Center for Cyber Security
Department of Computer Science, University of Surrey
Guildford, UK
m.t.khan@surrey.ac.uk

Dimitrios Serpanos
ISI/ATHENA and ECE
University of Patras
Patras, Greece
serpanos@ece.upatras.gr

Howard Shrobe
CSAIL
MIT
Cambridge, USA
hes@csail.mit.edu

***Abstract*—Modern medical devices aim at providing invasive e-health care services to patients with long-term conditions. Typically, these services are implemented as embedded software applications that remotely and automatically control the operations of the devices according to the patient's condition as monitored by the underlying sensors. Such applications are neither safe nor secure mainly because of unreliable sensors, which may provide incorrect input data either due to its malfunctioning or due to some accidental (by privileged user) or intentional (by adversary) interference. Hence, the incorrect sensor data may lead to identification of inaccurate patient condition, which may threaten the patient's life. To ensure safety and security of e-health applications, current approaches employ data analysis techniques to monitor sensor data and alarm when some unusual value is detected and employ access control strategies to ensure that controller decisions are consistent with sensor input data. However, such approaches fail to detect stealthy attacks, e.g. bad data (false data injection) and bad computations because they do not understand what the application or device is trying to do. To this end, we evaluate our existing approach (i.e., ARMET) to assure safety and security of an emerging and critically real-time application domain of e-health. The approach is based on the specification of the application and device, which has a design and a run-time component. Given an application specification, the design component employs logical verification methods to assure that the application design is resilient to some bad data, i.e., there are no sensor input data values with meaningful threshold which are admissible to the specification but are not true. Given the specification, the runtime component monitors application's execution and assures that the execution is consistent with the specification and alarms whenever it detects a violation, i.e., there is a bad computation. We evaluate the methodology through its application to an example medical e-health application that controls and monitors blood glucose through an insulin pump.**

***Index Terms*—Safety, Security, CPS, Medical Devices, Data Integrity Attacks, False Data Injection, Computational Attacks**

I. Introduction

Modern embedded medical devices are (health-care) cyber physical systems (HCPS). HCPS are "engineered systems that are built from, and depend upon, the seamless integration of computational algorithms and physical components" [1]. The operations of the health-care CPS (HCPS) are monitored and controlled by embedded software (aka e-health-care control application) which typically runs on embedded processors, wireless sensors and remote computing infrastructure. HCPS aims to support personalized health-care diagnosis, monitoring, and treatment pervasively. The safety and security of HCPS is critically important, considering the effects of HCPS compromise or failure, which can threaten the patient's life in addition to public expenditures for treatment [2].

HCPS employ e-health control applications to support various health-care services that automatically and remotely control the operations of the medical devices according to the patient's condition. For instance, automatic glucose control systems are widely used (as approx. 430 million people are affected by diabetes globally [3]) to (i) measure the patient's glycaemia perpetually and (ii) to control glycaemia by injecting an appropriate amount of insulin, if the glycaemia level is low or high. Modern automatic glucose monitoring and insulin delivery system (whose work-flow is shown by Sommerville [4] in Fig. 3 consists of

1) A blood glucose monitor that measures blood glucose levels continuously based on a special glucose sensor, i.e. every few minutes.
2) An insulin pump (a medical device) that automatically controls insulin through subcutaneous infusion. The pump delivers insulin doses (i.e. bolus and basal) whose rate, time and amount can be programmed.
3) A remote controller device that is used to control the insulin pump with the help of an e-health control application.

The e-health control applications are neither safe not secure because the communication among various components (i.e. glucose monitor, insulin pump and remote control) of the control system is based on wireless sensors, which are vulnerable to several safety and security attacks [2]. Using such applications without proper safety and security measures may threaten the patient's life, by injecting improper insulin accidentally or intentionally.

For our safety and security analysis of the glucose monitoring system, we view this system as a typical cyber physical system (CPS) that receives sensor input *data*, performs *computations* and issues commands to control the glucose. For e-health control applications, it is extremely desirable to efficiently (i.e., multitude of seconds) detect arbitrary bad *computations* and bad input *data* (i.e., data respecting the threshold that has negligible damage) to such systems, achieving highly assure safety and security of the patients health. Current approaches either employ data analysis tech-

niques to detect bad sensor input data [5] or employ access control mechanism to ensure that computations are legal and only allowed commands are issued by the privileged user [6]. However, recent attacks (e.g., Ukrainian Grid [7], Stuxnet [8]) on industrial CPS implies that HCPS may also be vulnerable to these attacks.Former attacks result in modification of the application's execution by a privileged user, while other attacks result in the compromise of input values, which are legal (i.e., satisfy specification) but are not real. Furthermore, it has been demonstrated that erroneous blood glucose measurements (aka *bad data*) may inject fatal insulin doses. Also, computing errors in the insulin control program (aka *bad computations*) may cause hyperglycemia (high blood glucose) or hypoglycemia (low blood glucose) and can threaten the patient's life. We are not aware of any effort that is able to detect arbitrary *bad computation* and *bad data* of e-health control runtime applications in general and insulin pump control applications in particular. Therefore, this work is an effort to apply a design and monitoring method (i.e. ARMET) to assure the safety of the patient's health and life by efficiently (i.e. few seconds) detecting arbitrary *bad computations* and some *bad data* in e-health applications at runtime.

In detail, in this work, we evaluate the ability of an existing design and monitoring methodology ARMET [9] to assure the safety and security of HCPS at run-time using the knowledge of physical process (i.e., blood glucose). We characterize the *behavior* as a set of *functional* and *non-functional* (e.g. safety, security, efficiency) characteristics of both the computational/cyber and physical resources of the process. The cyber and physical resources exhibit discrete and continuous behavior that operate at different granularity levels, making the behavioral description complex. To handle the complexity, ARMET allows one to describe the *behavior* of a HCPS process with an *abstract* executable specification. Importantly, we treat *computational* and *data* behaviors of HCPS separately, since they are susceptible to attacks with different characteristics and thus require different analysis and defenses. From the specified behavior, we generate a monitor that assures the run-time safety and security of HCPS control applications by detecting deviations of the HCPS implementation's behavior from that sanctioned by the specification's behavior. The approach detects both *computational attacks* (aka bad computations in which the adversary may change code or data of the HCPS control application-execution) and some *input-data integrity (FDI) attacks* (aka bad data in which the adversary may change the values such that the application works correctly but with wrong (legal but not real) input values). The run-time monitor guarantees detection of any functional deviation (*computational attack*) but may not detect data-integrity attacks. Therefore, for data-integrity attacks, we perform verification based vulnerability analysis on the HCPS specification to identify potential FDI vulnerabilities i.e., values with reasonably larger threshold. As a result, we obtain the values of the identified integrity attacks. Considering these values as attack-vectors, we either monitor input-data to detect attack-vectors or refine the design specification, by adding constraints, to eliminate such attacks [9].

The rest of the paper is organized as follows. Section II presents evaluation of runtime monitor ARMET to provide defense against computational attacks and assures safety and security of HCPS applications. Section III evaluates the non-linear vulnerability analysis [9] to identify false data injection attacks in e-health control applications. Section IV sketches the prior art to our approach and Section V concludes the paper.

II. Runtime Safety and Security Monitor

The goal of the ARMET [9] is to assure that the operations of an application are safe and secure against the known and a class of unknown threats at run-time. However, in this work, we evaluate the ability of ARMET to model and monitor safety of critical e-health control application and its operations, in addition to the application's security. We call it runtime safety and security monitor (RSSM). As depicted in Fig. 1, the RSSM requires both the specification (AppSpec) and implementation (AppImpl) of a health-care control application [10]. However, as the specification and implementation of the application are described at different levels of abstraction, the "Wrapper" wraps the implementation so as to share the input data of interest with the RSSM such that application implementation execution becomes comparable to the application specification execution. While monitoring, the RSSM checks if run-time behavior (*observations* generated by the "Wrapper") of the application is consistent with the predicted behavior (*predictions* generated by the "AppSpec") of the application. An alarm is raised by the RSSM, if an inconsistency is detected. To support the real-time performance of health-care application, the alarm can be used by AWDRAT [11], which may first suspend the execution and later resume it in a safe-mode, after diagnosis. Furthermore, the RSSM assures to detect any arising inconsistency (i.e., any known or unknown safety and security threat) iff the "AppSpec" executes in a safe environment, as proved in [10].

Algorithm 1 Monitoring safety and security of e-health applications

Input: `AppImpl`, `AppSpec`
Initialization :
1: **RUN** `AppImpl` and `AppSpec`
Monitoring (health-care) application execution :
2: **while** `AppImpl` is executing **do**
3: **for** each executing module `m` in `AppImpl` **do**
4: **LET** `obs` = runtime observations/behavior of `m` **IN**
5: **LET** `pred` = specified predictions/behavior of `m` **IN**
6: **if** `obs` $\neq$ `pred` **then**
7: **ENABLE** safe-mode execution for `m`
8: **RAISE** alarm
9: **end if**
10: **end for**
11: **end while**

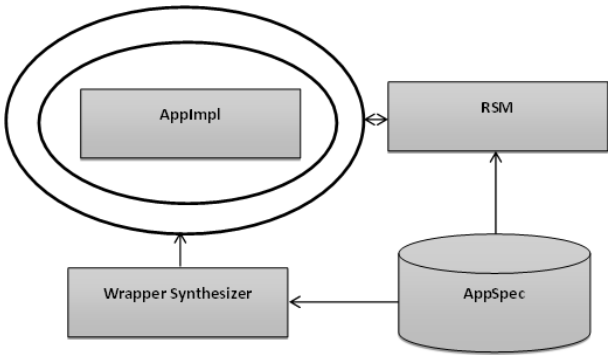

Fig. 1. Run-time Safety and Security Monitor (RSSM)

In detail, the specification ("AppSpec") supports to model behavior of cyber (e.g., controller) and physical (e.g., insulin pump) resources of medical device (resp. application) as first class models through description of

- the *normal behavior* (aka "good behavior") of the resources by
  - decomposing their behavior into several sub-modules and
  - by integrating pre- and post-conditions and invariant to form the decomposed sub-module,
- the *flow and control model* of values as data-flow and control-flow links connecting the sub-modules and
- the *exceptional behavior* of the resources, *known safety and security threats*, and suspected *attack plans* to rigorously characterize the *misbehavior* (aka "bad behavior") of a (sub-)module.

The RSSM rigorously assures the safety and security of the (health-care) device operations by alarming whenever behavior of the application operations and resources deviates from the desired one. Based on the aforementioned model of the control application the RSSM makes the application self-aware, i.e. the application knows what it is exactly trying to do. Thus, the RSSM makes the application highly robust and automatic by determining the application context rigorously. In fact, the RSSM supports detection of both known and unknown (aka hypothetical) safety and security threats by maintaining trace of what has exactly gone wrong.

As sketched in Algorithm 1, based on the above model (AppSpec) of the application implementation (AppImpl), the RSSM runs both the implementation and the specification in parallel and checks their *behavioral consistency* by comparing *predictions* (generated from the model execution, i.e. "AppSpec") and run-time *observations* (produced by executing the implementation, i.e. "AppImpl"). Whenever behavior of the application execution deviates from the desired ones, the RSSM alarms and enables safe-mode of the application execution, after diagnosis. The level of granularity (i.e., behavioral specification of a module) at which the monitoring application operates determines the performance-diagnosis trade-off. Coarse-grained monitoring lessens the execution overhead, but limits the resulting diagnostic information. On the other hand, fine-grained monitoring incurs higher computational overhead, but is able to produce quick and thorough diagnoses.

Application Specification	ω ::=	... ζ η ϵ...
Decomposition	ζ ::=	α | (α) ζ
Behavioral Model	η ::=	β | (β) η
Attack Plan	ϵ ::=	δ ρ | (δ ρ) ϵ

Fig. 2. Top Level Syntactic Domains

The novelty of the safety and security monitor arises from the specification language (its elements, e.g. attack plans, formalism and encoding cyber and physical resources of control systems as first class models based on their both functional and non-functional properties) of the application it monitors, whose high level domains are shown in Fig. 2. In principle, the specification language allows to model (i) discrete behavior of cyber and physical resources and (ii) continuous behavior of physical resources side by side as first class specifications. Monadic second order logic and event calculus based rich formalism of the specification language, enables to describe system behavior at various but practical levels of abstraction, with higher degree of modularity. Semantically, such logical formalism based executable specification language can be directly compiled into machine code, and is thus, inherently efficient for run-time behavioral comparison to meet real-time performance constraints of the health-care applications.

A. Example

In order to evaluate and demonstrate the extended monitor, RSSM, based on ARMET specification language, we have described the behavior of a simple health-care medical device (i.e., insulin pump) controller (see Listing 1) that is responsible of managing the level of glucose in human blood through an insulin pump. We are implementing a working prototype for our runtime monitor and are evaluating it to monitor the insulin pump and glucose controller. As a starting point, we have modeled the cyber resources (i.e., insulin controller) and physical resources (i.e., insulin in blood and its characteristics) of a typical health-care control application. In detail, the application specification ("AppSpec") includes a cyber-model that specifies the computations performed by the controller, and a physical-model that specifies physical characteristics and dynamics of the insulin as shown in Fig. 3 (as shown in [4]). The latter is employed to detect computational and false data injection attacks, by observing the systematic deviation of the sensor's behavior from the state of the physical condition of the patient, as predicted by the model. Moreover, based on "AppSpec", bugs in the implementation (AppImpl) can also be detected.

The focus of our current prototype development of RSSM is to detect "computational" attacks, where an attacker successfully alters the operation or parameter values of the insulin controller implementation (AppImpl). For demonstration, we consider a health-care control application, which has a sensor that monitors glucose, and an insulin (control) pump that receives and issues commands for the management of

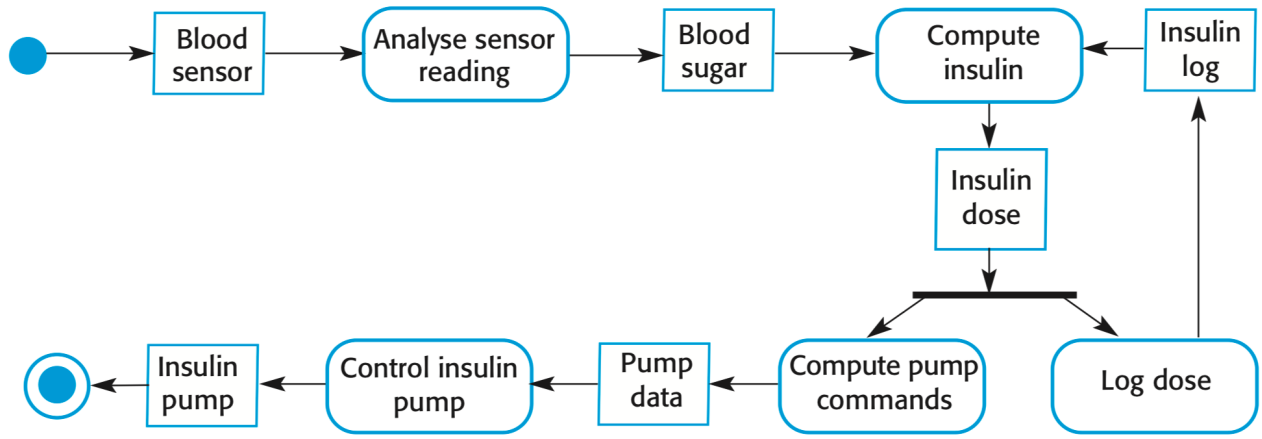

Fig. 3. Workflow of an Insulin Controller [4]

```
(define-component-type insulin-controller-step
  :entry-events (insulin-controller-step)
  :exit-events (insulin-controller-step)
  :allowable-events (update-state)
  :inputs (controller observation obs)
  :outputs (command error)
  :behavior-modes (normal)
  :components (
    (estimate-error
        :type estimate-error
        :models (normal))
    (compute-glucose-rate
        :type compute-glucose-rate
        :models (normal))
    (compute-insulin-dose
        :type compute-insulin-dose
        :models (normal)))
  :dataflows (
    (observation
        insulin-controller-step observation estimate-error
    )
    (the-error
        estimate-error the-error compute-glucose-rate)
    (level
        compute-glucose-rate level compute-insulin-dose)
    (rate
        compute-glucose-rate rate compute-rate-term)
    (dose
        compute-insulin-dose dose insulin-controller-step)
    (the-error
        estimate-error error insulin-controller-step) )
    ... )
```

Listing 1. Example Specification of `insulin-controller-step`

glucose level in blood and an insulin pump. The main task of the insulin pump is to inject an appropriate amount of insulin automatically by performing the following steps (see `:components` in Listing 1)::

1) Read the new glucose level (l_t at time t) through sensor
2) Compute the rate of glucose (r_t) based on
   a) new glucose level (l_t) and
   b) some (say k) previous levels ($L = \{l_{t-k}, l_{t-k-1}, \ldots, l_{t-1}\}$)
3) Compute the new insulin dose (d_t) based on
   a) computed glucose rate (r_t),
   b) current glucose level (l_t) and
   c) some (say k) previously injected insulin doses ($D = \{d_{t-k}, d_{t-k-1}, \ldots, d_{t-1}\}$)

```
(define-component-type compute-insulin-rate-term
   :primitive t
   :entry-events (compute-insulin-rate-term)
   :exit-events (compute-insulin-rate-term)
   :inputs (rate low-rate high-rate)
   :outputs (weighted-rate)
   :behavior-modes (normal)
   )

 (defbehavior-model (compute-insulin-rate-term normal)
 :inputs (rate low-rate high-rate)
 :outputs (weighted-rate)
 :prerequisites ([data-type-of ?rate number])
 :post-conditions
              (and ([data-type-of ?weighted-rate number
    ])
                   [(and (> weighted-rate low-rate)
                         (< weighted-rate high-rate)])))
```

Listing 2. Example Specification of `compute-insulin-rate-term`

4) Inject the computed dose (d_t)

Each step of the algorithm has a corresponding module in the control application (AppImpl), which are individually specified as pre and post-conditions and invariant (AppSpec). The actual implementation of each of these steps is wrapped, and their input and output are presented to the RSSM for their consistency checking.

In principle, analogous to our previous case study of water tank controller [10], the RSSM aims to quickly detect arbitrary modifications (i.e., artificial attack) in e-health controller application, because of the low abstraction gap between the RSSM model (AppSpec) and the actual computation (AppImpl).

Though, the RSSM enables detection of computational attacks at run-time, yet it is critical to understand what attacks it is capable of detecting and what remaining vulnerabilities remain. Particularly, in the case of false data injection attacks it has been shown that it is possible to construct a "stealthy" attack [12] that evades detection by the monitor. To this end, we have developed methods to identify types of attacks that can evade the monitor.

III. False data injection attacks

False data injection (FDI) attacks against critical control systems perform attacks just by compromising the input data to the control systems either by injecting noise into the signal monitored by the sensor or by corrupting the sensor output in it traverses the system's communications network. In fact, an FDI attack manipulates input data without compromising the systems themselves and leads the control system to make wrong decisions and take wrong actions: Importantly, in such attacks, systems operate correctly, but on the wrong input data. A simple example, referring to our insulin controller case, would be when the attacker makes it seem that the glucose level is high, causing the controller to inject large amounts of insulin, resulting in a precipitous drop in blood insulin levels, and thus threatening the patient's health and life.

We are not aware of any work that handles FDI attacks in e-health and medical control applications [2]. However, some approaches [13], [14] have been developed that either apply statistical analysis to detect any sensor data inconsistencies or encrypt sensor data to ensure integrity of the exchanged data. Traditionally, encryption has been difficult to apply in embedded applications due to the lack of computational power in the embedded processor. However, in some cases, the availability of newer encryption schemes for embedded applications and the increase of processor power has made the use of encryption feasible. As a general architectural rule, all data in flight should be encrypted, if feasible.

We evaluate the devised method [9] to defend against FDI attacks in medical control applications. The method employs a verification based vulnerability analysis to identify FDI vulnerabilities of a system at the design phase [15]. Based on the identified vulnerabilities, we refine the system design by adding those constraints that reduces the identified FDI attacks. The refinement is repeated until either FDI vulnerabilities cannot be identified any more, or the expected FDI attacks have been considered for monitoring at run-time. Given the sensor accuracy, the approach can detect only those false data injection attacks that cause physically unreasonable large jumps (i.e., meaningful threshold) in sensor input values. Furthermore, we assume that below the (meaningful) threshold there is no safety problem, i.e. the attack is like a "natural deviation" and has limited effect that does not make much of a difference to the monitored critical process.

In detail, the method requires a description of the system design as a state function, which is then analyzed to identify those input combinations that constitute FDI attacks. Typically every control system implements a control loop, as depicted in Fig. 4, for some control process P, whose specification and implementation are explain in the following:

Specification says that at every time instant t, the state of the system process P is described by a function $P(x_t)$, where x_t is the set of input variables to the process at time t.

Implementation requires the set of variables x_t that are measured with sensors, as input parameters. In fact, these variables are input to the controller and are

```
(set-logic QF_NRA)
(declare-fun gt () Int)
(declare-fun gt1 () Int)
(declare-fun cg () Real)
(declare-fun rin () Real)
(assert (and (and (= gt 3) (= gt1 5)) (= rin 2.0)))
(assert (= cg rin)
(assert (and (<= 3 gt) (<= gt 9)))
(assert (and (<= 1 rin) (<= rin 2)))
(assert (= gt1 (+ gt cg))
(check-sat)
(exit)
```

Listing 3. Example Insulin Pump Controller Specification

1) used to estimate the state of the system and
2) the necessary actions that are denoted by a_t.

These measurements are denoted by z_t, which represent the measurements of variables x at time t. Furthermore, implementation of a control system typically has an evaluation function $eval(x, z)$, which at time t, gets measurements z_t as an input and evaluates them for acceptance due to potential sensor failures.

During safe and error-and-fault-free operation, $eval(x, z)$ accepts all measurements z and estimate the state $P(x, z)$ of the system based on the measurements. However, during a successful launch of a FDI attack, $eval(x, z')$ accepts measurements z', which are compromised values of z, s.t. $z' \neq z$. To detect such values, we require to positively answer the question: does there exist z' for the system design $P(x, z)$ while evaluating $eval(x, z)$?.

As part of our approach we rely on the work of our former colleague Sicun Gao, whose work formalizes state of the system with a real function $f()$ and expresses the above question as an input to an SMT solver for real functions, namely dReal [15]. If the expression that represents the FDI existential question is satisfiable, then the system is vulnerable to FDI attacks; importantly, the SMT solver provides with a set z' of values that constitutes such attack.

Once FDI attacks are identified, a method to defend against them is to refine the system design by adding constraints on the values of the measured parameters, so that the set of undesired values that satisfy the state function is reduced. By refining the system design, the space of solutions to the state function becomes much smaller and thus, reduces –if it does not eliminate- the possible FDI attacks. The new refined design can be subsequently analyzed for new FDI attacks. If FDI vulnerabilities are identified again, the system can be further constrained and refined and so on. The process can be repeated until the realistic FDI attacks are eliminated or reduced to meet the required system specification.

A. Example

We evaluate our vulnerability analysis by its application to a variation of the running example of an insulin controller, as shown in Fig. 3. In this variation, the insulin controller has a pump which has one sensor measuring its insulin rate (r_{in} for incoming flow) and one actuator, opening it to a specified flow rate. Furthermore, the controller has one sensor measuring the glucose level. For convenience, we consider that the rate r_{in} is a real number in the range $\{1, 2\}$.

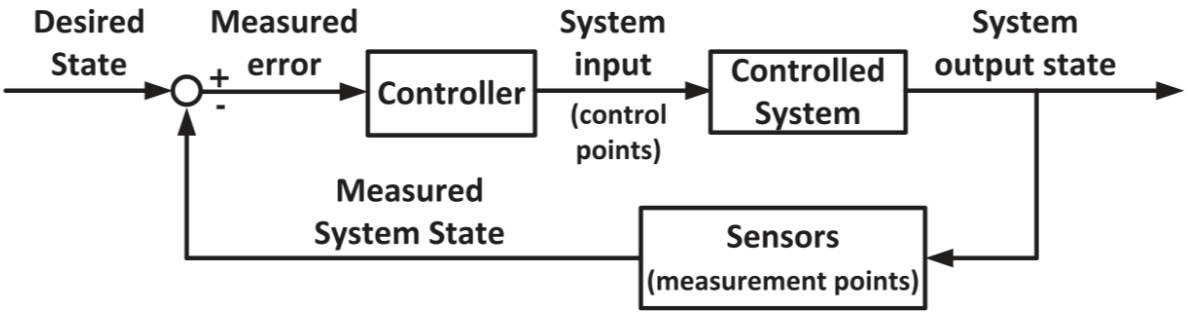

Fig. 4. Typical Control Loop for Control Systems

For the sake of example, we consider that the pump operates in discrete time and at the beginning of every time unit the state of the pump may change. Therefore, the estimated state of system, which is evaluated by $eval(x, z)$, is expressed with the function: $G(t+1) = G(t) + CG(r_{in}(t+1))$ that represents the glucose level $G()$ in the blood at the end of the time unit $t+1$. Please note that CG computes the concentration of glucose for the given rate r_{in} of insulin at time $t+1$. For simplicity, we consider that the concentration CG equals the rate, i.e. $CG(r_{in}(t+1)) = r_{in}(t+1)$.

Consider the following scenario, where at time t, $G(t) = 3.0$, and at time $(t+1)$ the insulin control system is configured with $r_{in}(t+1) = 1.0$, which gives $CG(r_{in}(t+1)) = 1.0$. If the system sensors provide the correct values of $r_{in}(t+1) = 1.0$ and $G(t+1) = 4.0$ to the monitor, the monitor will accept the state of the system, based on $G(t) = 3.0$. However, an attacker could compromise the sensor values of $r_{in}(t+1)$ and $G(t+1)$ and provide fake values to the monitor. The monitor will accept them, if the values are consistent with the function for $G(t+1)$, and will reject them, i.e. detect the attack, if the (fake) values are not consistent with the function $G(t+1)$. For instance, if the attacker can provide

1) either input values $r_{in}(t+1) = 0.5$ and $G(t+1) = 3.8$, then the monitor will be able to detect an inconsistency because $G(t+1) \neq G(t) + CG(r_{in}(t+1)) \Leftrightarrow 3.8 \neq 3.0 + 0.5$
2) or provides the values $r_{in}(t+1) = 2.0$ and $G(t+1) = 4.0$, the monitor will accept them because they satisfy $G(t+1) = G(t) + CG(r_{in}(t+1)) \Leftrightarrow 4.0 = 3.0 + 2.0$ although the values are not the real values, which demonstrates that the system is vulnerable to FDI attack when (sensors) values r_{in} and G are compromised.

To identify such FDI attacks, we use the SMT solver, providing the monitoring function $G()$ as an input and asking if there is a solution to this equation with parameter values r_{in} and G in the defined ranges, which is different from the real action. Listing 3 shows the input to dReal for our described example scenario as well as the result, which is a successful FDI attack to the system as depicted by results in Listing 4. Considering all the combinations of values that satisfy $G()$, it clear that the space of FDI attacks is large as

```
gt : [ ENTIRE ] = [3, 3]
gt1 : [ ENTIRE ] = [5, 5]
rin : [ ENTIRE ] = [2, 2]
cg : [ ENTIRE ] = [2, 2]
delta-sat with delta = 0.00100000000000000
```

Listing 4. Example FDI Detection

the values are different from the specific real values of the example.

To remove such a vulnerability or to build a defense against such an attack, a designer can introduce constraints on the vulnerable parameters (r_{in} and G), such that the attack surface reduces, e.g.

- one can decide that r_{in} should be integer, i.e. 1 or 2, rather than a real number, reducing the attack surface significantly
- a similar constraint on the value of G will make the potential attacks quite few and
- finally adding multiple sensors to monitor glucose level further reduces the chances of attacks and eventually will lead to a new design that is significantly more robust.

Importantly, we recognize that FDI attack detection is a particularly difficult problem, because the only knowledge the controller has of the system under control is through it's sensors. A typical way to handle FDI attacks is to monitor physical laws of the system under control, for example, monitoring (by Kalman filter) whether successive sensor reports indicate a physically un-realizable jump.A complicating factor in detecting FDI is that even a good sensor introduces some noise; such noise is normally Gaussian. The tests tease out whether the difference between expected sensor readings and actual ones (residual) fails to match this Gaussian assumption; i.e. they test whether the residuals exhibit a systematic distortion.

IV. Related Work

In this section, we review state of the art for each component of our existing design methodology, i.e. run-time safety and security monitoring of medical control applications and vulnerability analysis for false data injection attacks, respectively.

A. Run-time Safety and Security Monitor

Run-time safety and security monitoring of medical control systems is a challenging task, as it involves monitoring of physical processes (i.e., glucose and insulin) to handle critical health conditions of the patient. In order to address these challenges, we have developed a run-time safety and security monitor that advances the existing techniques in at-least one of the following ways: (i) handling safety and security threats and (ii) performance efficiency.

Handling Safety and Security Threats. Most of the existing approaches consider insulin control system as a system of connected devices and handle only those safety and security threats that either originate from connectivity, e.g. protocol, access control or originate from failures of devices, e.g. sensor, insulin pump. For instance, in [16], authors have developed a network of connected devices and have analyzed access control of the network, i.e. confidentiality, integrity and availability. In another effort [5], a supervised learning based scheme has been developed that learns safe insulin dose ranges for a patient and can detect attacks that result in overdose of insulin or acute dose. In [14], [6], authors have developed a system that can detect failure of sensor and pump by comparing the sensor values to Kalmen estimated values. However, since these approaches employ statistical techniques to handle threats, they fail to detect any stealthy attacks (i.e., input values are legal but not real), computational attacks (i.e. a person with right privileges executes legal commands with harmful values) or advanced persistent threats (i.e. in which values are observed for a long period of time and then attacks are launched). Therefore, the goal of our monitor is to detect not only known safety and security threats but also detects unknown threats and errors. Since, our monitor maintains the current and legal state (as described in the specification) of the system, we are able to detect accidental or intentional malicious activity by comparing the runtime behavioral state with expected behavioral state (i.e. specified behavior). Our approach is similar to the one developed in [17][18]; However, these efforts could only assure six temporal properties of the closed loop and failed to detect any prolonged bad computations.

Efficiency. Developing efficient run-time safety and security monitors that meet real-time constraints of medical control applications is a trade-off between safety and security, and efficiency of the application. Thus, in order to meet the strict real-time constraints of the application, we have developed a tunable safety and security monitor. This implies that we monitor adequate behavior (i.e. all pre-, post conditions and invariant) at the time of high threat, and monitor partial (i.e. any combination of pre-, post conditions and invariant) behavior otherwise. There are no fixed performance metrics for run-time safety and security monitoring of medical control applications [19]. However, there are general guidelines available that help to characterize a critical situation of patient which requires an immediate action. For instance, the very low level or very high level of glucose is considered a critical condition, which requires immediate treatment. Our specification language allows to model such conditions, which can be detected within desired time helping patient to recover from her critical condition. Our monitor can raise the alarm in such situations calling for a doctor and (if possible, i.e. modeled) can provide a quick treatment to sustain the patient for a while until doctor arrives. In fact, the real-time constraints of such applications are periodic, i.e. the response of a certain component or a certain decision is expected to be completed within a certain time period, say T . Therefore, we ensure that the longest execution of our monitor's implementation completes in time T, thus respecting the real-time constraints of the associated component of the monitored application. In a different but complex case study of water management system, our monitor executed in less than $1ms$, well below the real-time constraints of control application belonging to various application domains. For example, insulin controllers have a desired response/decision delay of a few ms.

B. False Data Injection Attacks

False data injection (FDI) attack modifies the (sensors) measurement values that are exchanged among various components of control systems. Such values eventually mislead the controller application to conclude undesired results [20]. Such attacks have been variously studied in large scaled critical application domains, e.g. power grids. Most of the existing approaches have attempted to model linear state estimates of power grids [21], [22], [23]. In fact, analysis of realistic nonlinear state estimation models is much harder [24], [25]. Many of the existing nonlinear models bypass solving complete nonlinear constraints involved in state estimation, for instance, flow equations in power grid. Thus such solutions only offer analysis of system topology or data based on statistical techniques.

More recently, a cryptographic protocol [13] has been developed to build defense against attacks that involve forging incorrect sensor readings of glucose and involve tracking user performing malicious activity for insulin monitoring. However, such protocols are not practical because sensors are resource limited and encryption of data requires more computing and battery power on one hand and being low powered may hinder performance of insulin injection on the other hand.

In contrast to aforementioned approaches, our evaluated approach focused on developing those models that are free of some FDI by design. Based on recent results in delta-decision procedures [26], the evaluated approach allows to encode control system models in dReal [15], then the tool searches for input values to the variables for which monitor does not alarm by reasoning about nonlinear logical constraints over real numbers. The detection of arbitrary FDI attacks is a very challenging task as current approaches fail to differentiate between a noisy data and sensor compromise [27]. We have also recognised the same difficulty as mentioned earlier in Section III-A.

V. Conclusion

We have evaluated our design and monitoring method (based on ARMET) to develop safe and secure e-health con-

trol systems based on the behavioral characteristics of the cyber and physical components of the systems. The evaluation shows that the method is practically feasible and suffers from no scalability and performance issues because the monitor is based on the executable specification of the application. The methodology makes the medical control system resistant not only to computational attacks but also to detect false data injection attacks, by employing a vulnerability analysis technique that leads to application designs that are free of some false data injection attacks. Furthermore, the evaluation demonstrates the effectiveness and usability of the method for e-health control applications through its application to an automatic glucose control and insulin delivery system. The e-health applications are more challenging because the need to understand the real-time and physical situation of the patient, which depends of several factors. For instance, blood glucose depends on the food eaten by the patient, the amount of injected insulin and the rate of glucose.

REFERENCES

[1] NSF, "Cyber Physical Systems," https://www.nsf.gov/pubs/2014/nsf14542/nsf14542.htm, 2018.

[2] N. K. J. Chunxiao Li, A. Raghunathan, "Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system," in *IEEE 13th International Conference on e-Health Networking, Applications and Services*, June 2011, pp. 150–156.

[3] WHO (WORLD HEALTH ORGANISATION), "Global report on diabetes," http://apps.who.int/iris/bitstream/10665/204874/1/WHO_NMH_NVI_16.3_eng.pdf?ua=1, 2016.

[4] I. Sommerville, *Software Engineering*, 9th ed. USA: Addison-Wesley Publishing Company, 2010.

[5] X. Hei, X. Du, S. Lin, I. Lee, and O. Sokolsky, "Patient infusion pattern based access control schemes for wireless insulin pump system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 11, pp. 3108–3121, Nov 2015.

[6] A. Facchinetti, S. Favero, G. Sparacino, and C. Cobelli, "An online failure detection method of the glucose sensor-insulin pump system: Improved overnight safety of type-1 diabetic subjects," *IEEE Transactions on Biomedical Engineering*, vol. 60, no. 2, pp. 406–416, Feb 2013.

[7] B. Kang, K. McLaughlin, and S. Sezer, "Towards a stateful analysis framework for smart grid network intrusion detection," in *Proceedings of the 4th International Symposium for ICS & SCADA Cyber Security Research 2016*, ser. ICS-CSR '16, 2016, pp. 1–8.

[8] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security and Privacy*, vol. 9, no. 3, pp. 49–51, May 2011.

[9] M. T. Khan, D. Serpanos, and H. Shrobe, "Armet: Behavior-based secure and resilient industrial control systems," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 129–143, Jan 2018.

[10] M. T. Khan and D. Serpanos and H. Shrobe, "A rigorous and efficient run-time security monitor for real-time critical embedded system applications," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Dec 2016, pp. 100–105.

[11] H. Shrobe, R. Laddaga, B. Balzer, N. Goldman, D. Wile, M. Tallis, T. Hollebeek, and A. Egyed, "AWDRAT: A Cognitive Middleware System for Information Survivability," in *Proceedings of the 18th Conference on Innovative Applications of Artificial Intelligence - Volume 2*, ser. IAAI'06. AAAI Press, 2006, pp. 1836–1843.

[12] C. Kwon, W. Liu, and I. Hwang, "Analysis and design of stealthy cyber attacks on unmanned aerial systems," *Journal of Aerospace Information Systems*, vol. 11, no. 8, pp. 525–539, 2014.

[13] L. Reverberi and D. Oswald, "Breaking (and fixing) a widely used continuous glucose monitoring system," in *Proceedings of the 11th USENIX Conference on Offensive Technologies*, ser. WOOT'17. Berkeley, CA, USA: USENIX Association, 2017, pp. 18–18. [Online]. Available: http://dl.acm.org/citation.cfm?id=3154768.3154786

[14] A. Facchinetti, S. D. Favero, G. Sparacino, and C. Cobelli, "Detecting failures of the glucose sensor-insulin pump system: Improved overnight safety monitoring for type-1 diabetes," in *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug 2011, pp. 4947–4950.

[15] S. Gao, S. Kong, and E. M. Clarke, "dreal: An smt solver for nonlinear theories over the reals," in *Proceedings of the 24th International Conference on Automated Deduction*, ser. CADE'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 208–214.

[16] D. C. Klonoff, "Cybersecurity for connected diabetes devices," *Journal of Diabetes Science and Technology*, vol. 9, no. 5, pp. 1143–1147, 2015.

[17] F. Cameron, G. Fainekos, D. M. Maahs, and S. Sankaranarayanan, "Towards a verified artificial pancreas: Challenges and solutions for runtime verification," in *Runtime Verification*, E. Bartocci and R. Majumdar, Eds. Cham: Springer International Publishing, 2015, pp. 3–17.

[18] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, *Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications*. Cham: Springer International Publishing, 2018, pp. 135–175.

[19] A. Kane, "Runtime monitoring for safety-critical embedded systems," Ph.D. dissertation, ECE, CMU, USA, 2015.

[20] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," *ACM Transactions of Information Systems Security*, vol. 14, no. 1, pp. 13:1–13:33, Jun. 2011.

[21] A. Teixeira, I. Sha'mes, H. Sandberg, and K. H. Johansson, "Revealing stealthy attacks in control systems," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Oct 2012, pp. 1806–1813.

[22] A. Teixeira, S. Amin, H. Sandberg, K. H. Johansson, and S. S. Sastry, "Cyber security analysis of state estimators in electric power systems," in *49th IEEE Conference on Decision and Control (CDC)*, Dec 2010, pp. 5991–5998.

[23] L. Xie, Y. Mo, and B. Sinopoli, "Integrity data attacks in power market operations," *IEEE Trans. Smart Grid*, vol. 2, no. 4, pp. 659–666, 2011.

[24] G. Hug and J. A. Giampapa, "Vulnerability assessment of ac state estimation with respect to false data injection cyber-attacks," *IEEE Transactions on Smart Grid*, vol. 3, no. 3, pp. 1362–1370, Sept 2012.

[25] M. A. Rahman and A. H. M. Rad, "False data injection attacks with incomplete information against smart power grids," in *IEEE Global Communications Conference, GLOBECOM*, December 2012, pp. 3153–3158.

[26] S. Gao, J. Avigad, and E. M. Clarke, "δ-complete decision procedures for satisfiability over the reals," in *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, 2012, pp. 286–300.

[27] S. Sankaranarayanan, S. A. Kumar, F. Cameron, B. W. Bequette, G. Fainekos, and D. M. Maahs, "Model-based falsification of an artificial pancreas control system," *SIGBED Rev.*, vol. 14, no. 2, pp. 24–33, Mar. 2017. [Online]. Available: http://doi.acm.org/10.1145/3076125.3076128