ARMET: Behavior-Based Secure and Resilient Industrial Control Systems

This paper describes a reliable/secure-by-design methodology for industrial control systems and complementary online monitoring approach.

By Muhammad Taimoor Khan, Dimitrios Serpanos, and Howard Shrobe

ABSTRACT | In this paper, we introduce a design methodology to develop reliable and secure industrial control systems (ICSs) based on the behavior of their computational resources (i.e., process/application) and underlying physical resources (e.g., the controlled plant). The methodology has three independent, but complementary, components that employ novel approaches and techniques in the design of reliable and secure ICSs. First, we introduce reliable-and-secure-by-design development of secure industrial control applications through stepwise sound refinement of an executable specification, employing deductive synthesis to enforce functional and nonfunctional (e.g., security and safety) properties of ICS applications. Second, we present a runtime security monitor at the middleware level of ICSs that protects ICS operation in the field through comparison of the application execution and the application specification execution in real time; the runtime security monitor can be synthesized from the executable specification. Finally, based on the specification, we perform a vulnerability analysis for false data injection (FDI) attacks, which leads to ICS application designs that are resilient to this type of attacks. We demonstrate the methodology through its application to a basic and typical ICS example application, describing all the tools used and ARMET, the middleware monitor that constitutes the core component of the methodology.

KEYWORDS | Computational attacks; cyber–physical systems (CPSs); efficient; false data injection (FDI) attacks; industrial control systems (ICSs); reliable-and-secure-by-design; resilient; runtime monitoring

Manuscript received March 25, 2017; revised June 8, 2017; accepted June 30, 2017. Date of publication August 16, 2017; date of current version December 20, 2017. *(Corresponding author: Muhammad Taimoor Khan.)*

M. T. Khan is with the Institute of Informatics Systems, Alpen-Adria University, 9020 Klagenfurt, Austria (e-mail: muhammad.khan@aau.at).
D. Serpanos is with the Industrial Systems Institute/ATHENA RC, GR-26504 Patras, Greece, and also with the ECE Department, University of Patras, GR-26504 Patras, Greece (e-mail: serpanos@ece.upatras.gr).
H. Shrobe is with the MIT Computer Science and Artificial Intelligence Laboratory (MIT CSAIL), Cambridge, MA 02139 USA (e-mail: hes@csail.mit.edu).

Digital Object Identifier: 10.1109/JPROC.2017.2725642

I. INTRODUCTION

Industrial control systems (ICSs) are a significant class of cyber–physical systems (CPSs) that are increasingly employed in the control and management of critical infrastructure around the world, from transport systems to water management systems and from the manufacturing floor to smartgrids. They constitute typical hybrid systems and their control process models are adopted to several other application domains, especially health systems and the emerging Industrial Internet of Things (IIoT). Importantly, from the computational point of view, ICSs constitute special purpose computing systems with different characteristics from typical information technology (IT) systems, in terms of purpose, ownership, interfaces, functional and nonfunctional requirements, etc. ICSs are considered a different form of computation from IT system and as such are referred to as operational technology (OT) systems [1]. As special purpose systems, ICSs (OT systems) are embedded, cyber–physical systems with restricted functionality, typically, to the application domain targeted by the system.

A typical ICS is shown in Fig. 1. In this typical configuration, measurements of a process' data are collected by the sensors and delivered to the programmable logic controllers/remote terminal units (PLCs/RTUs) through a network, while commands from PLCs/RTUs travel back through the network to actuators. PLCs/RTUs typically implement simple operations and control sensors and actuators, while they are managed and coordinated by the hierarchically higher layer supervisory control and data acquisition (SCADA) system, which implements a control application or plant.

The reliability and security of ICSs is of major importance considering the effects of their compromise or failure. A failed ICS can cause serious damage beyond the ICS itself, for example, through release of toxic chemicals. An attack on the power grid may have major secondary

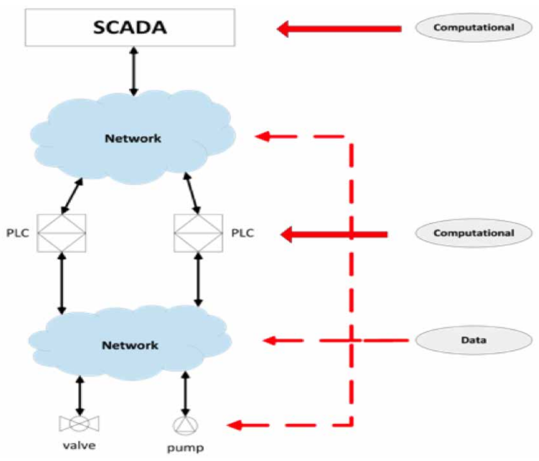


Fig. 1. ***Overview of ICS design.***

effects, disabling transportation, medical systems, and pipelines. The ability to cause such effects through cyber means alone was first demonstrated in the well-known Aurora experiment where a pure cyber attack successfully destroyed a diesel generator [2]; Stuxnet [3] and the recent Ukrainian smartgrid attack [4] as well as several additional less known incidents have demonstrated the potential of cyber attacks to inflict serious societal damage. Finally, ICSs may be compromised to serve as elements of a Botnet as the recent Mirai attack indicates [5].

To address reliability and security in ICSs, one needs to take into account the characteristics of OT technology that differentiates ICSs from typical IT systems, because this influences the security methods and solutions that are appropriate for ICSs. ICSs typically have real-time requirements, often hard ones, whose violation may have significant, or even catastrophic, results. It is important to note that functional and nonfunctional requirements of ICSs, and CPSs in general, are different from the requirements of typical IT systems and, thus, need to be carefully considered in the design and development of security and safety solutions. In fact, requirements for ICSs involve domains for cyber and physical resources that have discrete and continuous operations based on linear and nonlinear constraints with strict time bounds.

Until recently, security of the ICS infrastructure has not been a significant requirement although safety and dependability have been quite strong requirements. The reason is that ICSs and networks have been typically considered as vulnerable mainly to failures and not to attackers who target them on purpose. So, methodologies for dependability and safety have been developed assuming stochastic models of failures of components and subsystems. Although these methodologies have served control processes well until recently, the growth of cyber attacks and the interest of strong actors to damage critical infrastructure lead to the need of developing new methods that protect ICSs and networks against purposeful cyber attacks. The first efforts in this direction have been to harden systems and interconnections, adding security methods and modernizing systems that have been in the field, unprotected for long periods. Operating systems are being increasingly adopted, mostly specialized real-time operating systems (RTOSs) or variations of workstation operating systems such as Windows and Linux, providing access control mechanisms. Intrusion detection mechanisms for the systems themselves or for the networks are becoming common, but most of them are adopting tools and methods from the IT environment, i.e., malware detection and intrusion detection methods and tools that have been developed for general IT environments [6], [7]. Importantly, little effort has been made to include in the security methods the processes/applications that get implemented by the ICSs, although the requirements on the ICSs are placed by them for both functional and nonfunctional ones. Recently, the OT characteristics and differentiators from classical IT systems have been introduced in the efforts to develop effective security and safety solutions, especially for defensing against false data injection (FDI) attacks, which constitute a special class of attacks that has been emerging against CPSs. In these attacks, attackers do not compromise the computational systems or their networks, but introduce false data at points of measurements, i.e., at sensors, introducing false data to the sensors, who, in turn, send them to control centers and lead them to wrong actions [8], [9].

We introduce a design methodology for developing reliable and secure ICSs based on the process (application) behavior rather than the classical intrusion detection mechanisms. We define as "behavior" of an ICS process, in terms of software engineering, the set of functional and nonfunctional (e.g., security, performance) characteristics of both the cyber and the physical resources of the process. Typically, the cyber resources have a discrete behavior, while the physical resources have a continuous one. The complexity of modeling such a process increases when functional and nonfunctional characteristics are a mix of discrete and continuous ones. We manage the modeling complexity by describing the "behavior" of an ICS process

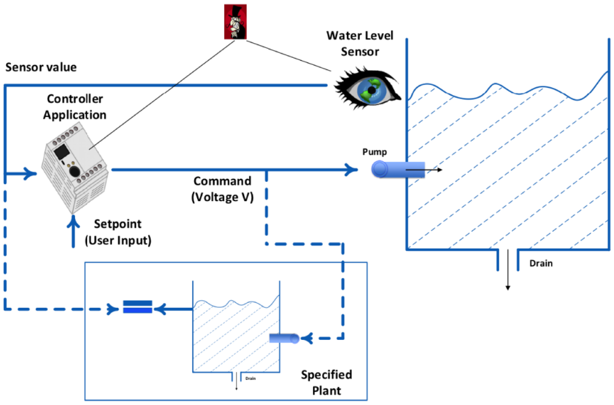


Fig. 2. ***Gravity-draining water tank–an example scenario.***

with an executable specification based on an abstract data type (in Coq). We derive reliable and secure implementations of the ICS process by stepwise refinement of the specification applying deductive synthesis (in Coq); each refinement formally preserves the behavioral semantics. Furthermore, we assure that the derived implementation is reliable in respect to characteristics of functional behavior and is secure and efficient conforming to characteristics of nonfunctional behavior. Importantly, the proof obtained through deductive synthesis assures that only the derived implementation is reliable and secure and not the execution of the implementation. Thus, from the given executable specification, we generate a runtime monitor that assures that the behavior of the implementation execution is also reliable, secure and efficient at runtime by checking consistency between the specification produced predictions and the runtime execution produced observations. The monitor is bound to detect any arising inconsistency under the assumption that the execution of the specification is always correct, i.e., that it cannot be attacked; this can be achieved by executing the specification in a strongly protected environment, such as SGX (Intel) or Trust Zone (ARM). Hence, the runtime monitor detects any computational attack but may not be able to detect FDI attacks. Therefore, we synthesize the executable specification after vulnerability analysis that identifies potential FDI attacks through the application of an SMT solver for nonlinear, real functions. Based on the results of the solver, we either synthesise the specification so that the implementation is not vulnerable to false data attacks, or consider the values of the identified FDI attacks as attack vectors to be monitored and detected at runtime.

Our threat model includes both computational attacks and FDI attacks. As computational, we consider all attacks that lead to the active alteration (injection, deletion, or replacement) of application code and/or system code, or of variable values (data or control) that lead to the incorrect execution of the application code on the ICSs; such attacks include viruses, worms, etc. As FDI attacks, we consider the ones that input (insert) into the ICSs values that are different from the intended ones, i.e., sensor measurements that have been tampered with, for example, between the environment and sensors, sensors and PLC/RTUs, etc. Importantly, our approach leads to systems that defend against the attacks, both computational and data ones, of our threat model because the attacks constitute violations of the security requirements.

In this paper, we demonstrate our approach to secure and safe ICSs by describing the design methodology using a simple example of a control application for a water tank. We describe the methods and the tools that we use as parts of the methodology, focusing on ARMET, the real-time security monitor that constitutes our main research contribution. Although our work is currently progressing the three components of the methodology separately, i.e., the secure-by-design application development, the real-time security monitor, and the vulnerability analysis, our ongoing work focuses on the integration of the three components in a unified environment that will enable synthesis of the application code, the real-time monitor, and the FDI attack monitor automatically.

The paper is organized as follows. Section II presents our threat model. Section III introduces an approach to develop reliable and secure ICS applications by design. Section IV demonstrates our approach for runtime security monitoring, including a proof for system defense against known and unknown computational attacks. Section V presents our approach for vulnerability analysis of FDI attacks in ICSs. We demonstrate each of the aforementioned approaches with the help of an example of a water tank, as the controlled plant, in each corresponding section. In Section VI, we present prior art for each component of our approach separately.

II. THREAT MODEL

A program for an ICS, e.g., a typical controller (PLC) program or a supervisory (SCADA) program, is composed of instructions (also known as source code) operating on some external-input data values. Based on this view, we consider a threat model that includes computational attacks for instructions as well as FDI attacks for external-input values. The program implementation P, its execution P_E, or the program external-input values are considered to have a threat, when some insider or external adversary (e.g., user or other program) with legitimate or illegitimate access attempts to affect their behavior.

In a computational attack, on one hand, the adversary attempts to modify the behavior of the program by: 1) modifying the instructions or internal-data values of the implementation or execution of P; 2) exploiting any vulnerability or a bug in them; or 3) injecting an additional code (e.g., command) into P or P_E. On the other hand, the adversary may attempt to affect the nonfunctional behavior of P_E by running some other program, for instance, running a malware that may affect the performance of P_E. Note that P refers to the program of the application implementation and not the program of the specification implementation. The latter always runs correctly as discussed in Section I.

In an FDI attack , the adversary attempts to change external data values that are input for computation at the controller by attacking sensors or the network between sensors and the computing system. However, as the values belong to data-type "real," therefore, small variation in the values may not violate the system constraints and thus goes undetected leading to some advanced persistent threat.

The specification of functional and nonfunctional behavior of a program allows us to detect computational threats by comparing the behavior of P or P_E with the specified behavior. Such specification also enables identification of legal data values that lead to FDI attacks by the application of the nonlinear verification method.

III. RELIABLE-AND-SECURE-BY-DESIGN APPLICATIONS

Developing reliable and secure ICS applications by design through derivation of the application implementation from a given declarative specification formally assures that the application meets its requirements on one hand and is free from various classes of vulnerabilities and attacks (e.g., FDI, cross-site scripting) on the other hand. Furthermore, runtime monitoring of such an application based on the declarative specification assures that the application execution is reliable and secure at runtime. However, formally deriving such reliable and secure implementations for ICS applications is challenging because it involves reasoning about both discrete and continuous system behavioral models.

Most of the existing approaches have focused on either deriving reliable or secure implementations of ICS applications. Many of these approaches only allow modeling of timed security properties or security policies of ICS applications, and then deriving application implementations that assure enforcing the security properties and policies at runtime [10]–[13]. Quite a few have attempted to derive reliable ICS application implementations, for instance, Soulat [14] derives correct implementation of schedulers of hybrid systems.

Based on Fiat [15], we introduce a different approach that employs deductive synthesis to develop reliable-and-secure-by-design ICS applications through interactively stepwise refinement of declarative specifications, where description of both cyber and physical resources of ICSs are first class models on one hand and nonfunctional properties (i.e., security, performance) are modeled integral to functional properties on the other hand. In this approach, the user starts with an initial nondeterministic program/specification with obscure nonfunctional characteristics, for example, security and efficiency. Then, the specification is synthesized through stepwise refinements; each refinement replaces some statements in the specification with other at least equally deterministic statements such that no extra behavior is introduced that is beyond that of the replaced statements on one hand, and none of the security properties are violated on the other hand, as depicted in Fig. 3. Finally, the specification is refined into a fully deterministic implementation that is not only correct with respect to its specification but is also obviously secure and efficient, respecting security constraints and employing efficient representations and algorithms, respectively. In detail, first the initial program is realized by its high-level declarative specification of its functionality and security. Then, through an iteration of semantically preserved optimizations, an efficient and correct executable implementation is generated that also conforms specified security constraints. The optimizations can be modified through sound refinements to meet nonfunctional requirements, e.g., security and performance.

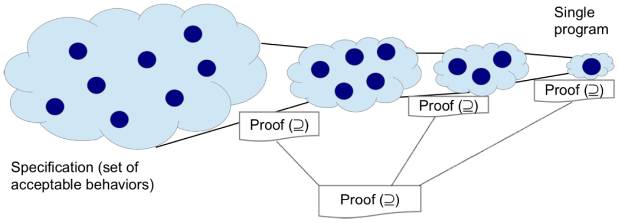


Fig. 3. ***Deductive program synthesis through sound refinements.***

In the reliable-and-secure-by-construction approach, we use the Coq proof assistant to encode an abstract-data-type-based declarative specification of an ICS behavior that includes both functional and nonfunctional properties as its first class elements. The encoding is based on abstraction relation as specified by Hoare [16]. Then, we employ deductive synthesis of the specification through sound refinements. Importantly, all refinement steps are encoded in the proof assistant and thus provide high assurance of reliable resulting implementation that is correct, secure, and efficient. Every refinement corresponds to an optimization script that resolves nondeterminism of the program in a (functional and nonfunctional) behavior-preserving way as shown in Fig. 3.

A. Example

Based on the water tank example (see Fig. 2), we demonstrate our approach in a familiar notation, i.e., Java-like syntax as shown in Listing 1.

For our example specification (Listing 1), in each clock tick, either the pump FILLs the water tank, or does NOTHING. The water tank is continuously DRAINing as enumerated by "Action" [see L(ine).1]. The specification says that the water tank is initially empty (see L.4). When a user issues a command to fill the water tank up to a certain level, if the command (i.e., reading) is in the range of sensor accuracy (see L.7), then we either accept the value or any

```
 1 public enum Action { FILL, DRAIN, NOTHING }
 2
 3 class WaterTankSpec {
 4   private int water_level = 0;
 5
 6   public void newSensorReading(int reading){
 7     if(abs(reading-water_level) > SENSOR_ACCURACY)
 8      water_level = {n | True}; }
 9
10   public Action timestamp(int target_level){
11     Action act =
12      {a| (a = FILL ->
13       water_level + FILL_RATE - GRAVITY_DRAIN <= TANK_MAX)
14       /\ (a = DRAIN ->
15       water_level - GRAVITY_DRAIN >= 0) };
16     if (act == FILL)
17      water_level += FILL_RATE - GRAVITY_DRAIN;
18     else if (act == DRAIN)
19      water_level -= GRAVITY_DRAIN;
20     return act; }    }
```

Listing 1. ***Water tank specification.***

```
 1  public enum Action { FILL, DRAIN, NOTHING }
 2
 3  class WaterTankSpec {
 4    private int water_estimate = 0;
 5
 6    public void newSensorReading(int reading){
 7      water_estimate = reading;    }
 8
 9
10    public Action timestamp(int target_level){
11      if (water_estimate < target_level &&
12          (water_estimate + SENSOR_ACCURACY +
13            FILL_RATE - GRAVITY_DRAIN <= TANK_MAX)){
14          water_estimate += FILL_RATE - GRAVITY_DRAIN;
15          return FILL;
16      }else if (water_level - GRAVITY_DRAIN >= 0) {
17          water_estimate -= GRAVITY_DRAIN;
18          return DRAIN;
19      }else
20          return NOTHING; }    }
```

Listing 2. ***Water tank code.***

value (see L.8). At each time stamp (see L.10), the water tank is either in

- FILLing state, i.e., the pump is pumping in the water such that water tank does not overflow (see L.13-14) and the water tank attains a new water level (see L.19); or
- DRAINing state, i.e., the water is draining at a constant GRAVITY_RATE assuring that the tank 1) does not underflow (see L.15–16) and 2) attains a new water level (see L.20).

Based on design decisions and synthesising the specification (Listing 1), we derive the Java implementation (Listing 2).

Analogous to specification description, water tank implementation (Listing 2) says that initially water tank is empty (see l.4). We accept any value from a user command (see L.7). At each time stamp, we

- FILL the water tank to attain new height of the water (see L.14), if the tank does not overflow (see L.11-13); or
- DRAIN the water tank to attain new height of the water (see L.17), if the tank does not underflow (see L.16); or
- do NOTHING (see L.20), if all the water has been drained.

Although deductive synthesis produces a correct-and-secure-by-construction controller, this by itself is inadequate. The proof is constructed under the assumption that the operating system and runtime environment in which the controller executes are correct. These assumptions are not generally correct; the runtime libraries supporting C (and other language) code are known to contain vulnerabilities and these vulnerabilities can be exploited to change the code that executes in the controller. Furthermore, FDI attacks (i.e., corrupting the sensor data before it reaches the controller) can cause the control algorithm to issue commands that will have disastrous effects, even if the controller is unmodified (for example, corrupting the water level sensor data to be lower than it should be will result in the controller issuing a “FILL” command when it should not). Thus, while it is certainly an advantage to have constructed a provably correct and secure controller, one must also actively monitor the runtime behavior of the system to guarantee that the controller continues to behave correctly.

IV. RUNTIME SECURITY MONITOR

The workflow of our runtime security monitor (RSM) is shown in Fig. 4. The RSM requires both the specification (AppSpec) and implementation (AppImpl) of an ICS application for monitoring [17]. Since the specification and implementation of the application operate at different levels of abstraction, the “Wrapper” wraps the implementation in order to share the observed data of interest with the RSM in a way that is comparable to the executable specification of the application. During monitoring, the RSM checks for the consistency of the runtime behavior (observations generated by the “Wrapper”) of the application and the application’s expected behavior (predictions generated by the “AppSpec”). The RSM raises an alarm if an inconsistency is detected. To support the real-time constraints of ICSs, the alarm can be used by AWDRAT [18], which may first suspend the execution and later resume the execution in a safe state, after diagnosis.

In detail, the “AppSpec” allows us to model the behavior of cyber and physical resources of ICSs as first class models through description of

- the normal behavior (“good behavior”) of the (cyber and physical) resources by decomposing their behavior into various submodules, by encoding precondition and postconditions and invariant(s) for each submodule;
- the flow and control model of values as data-flow and control-flow links connecting the submodules;
- the exceptional behavior of the resources, known attacks, and suspected attack plans to rigorously characterize the misbehavior (“bad behavior”) of a module/submodule.

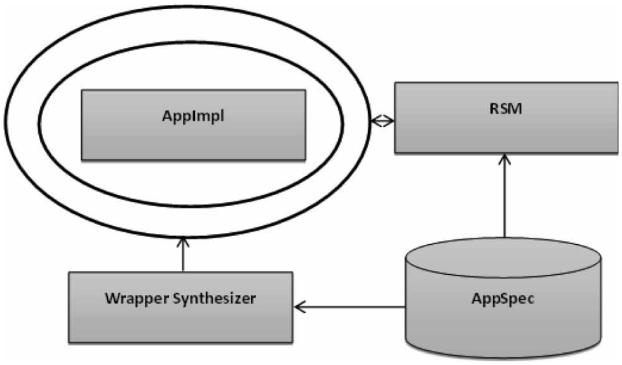


Fig. 4. ***Runtime security monitor (RSM).***

Application Specification	ω ::=	... $\zeta\ \eta\ \epsilon$...
Decomposition	ζ ::=	$\alpha \mid (\alpha)\ \zeta$
Behavioral Model	η ::=	$\beta \mid (\beta)\ \eta$
Attack Plan	ϵ ::=	$\delta\ \rho \mid (\delta\ \rho)\ \epsilon$

Fig. 5. ***Top level syntactic domains.***

Based on the model (AppSpec) of the application implementation (AppImpl), the RSM runs the AppSpec and AppImpl in parallel and checks their consistency by comparing predictions (generated from the model, i.e., "AppSpec") and runtime observations (produced by the implementation, i.e., "AppImpl"). The level of granularity at which the monitoring application operates determines the performance–diagnosis tradeoff. 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.

The novelty of the RSM arises from the specification language (its elements, e.g., attack plans, formalism, and encoding cyber and physical resources as first class models based on their both functional and nonfunctional properties) of the application it monitors, whose high level domains are shown in Fig. 5. In principle, our specification language allows to describe 1) discrete behavior of cyber–physical ICS resources; and 2) continuous behavior of physical ICS resources side by side as first class specifications. Monadic second-order logic and event-calculus-based rich formalism of the specification language enables us to describe system behavior at various levels of abstraction, with higher degree of modularity. Semantically, such logical formalism-based executable specification language can be directly compiled into a machine code, and is thus, inherently efficient for our runtime behavioral comparison. In detail, the formalism of our specification translates into a finite automaton that recognizes only the words that satisfy the specification [19]. "AppSpec" is an active model of normal behavior [18] and consists of a decomposition into submodules as well as in precondition and postconditions and invariant(s) for each submodule. Furthermore, data-flow and control-flow links connect the submodules, specifying the intended control and flow of values. The preconditions and postconditions and invariant(s) are first-order statements about the set of data values (that flow into and out of the submodules) and arbitrary constraints, respectively. Optionally, the model can also specify suspected/known undesired behavior of a resource and associated potential attack plans, allowing diagnostic reasoning to characterize the component's misbehavior. In principle, attack plans are hypothetical attacks based on rules that describe different ways of compromising a component by specifying "bad" behavior of the systems. The monitor exploits the attack plans at runtime to detect any such misbehavior, thus making the monitor more robust. For an example attack plan for our controller, see Listing 3.

```
(define-attack-model code-attack
    :attack-types
        ((hacked-code-file-attack .3))
    :vulnerability-mapping
        ((load-code hacked-code-file-attack)))

(defrule bad-code-file-takeover (:forward)
    if [and [resource ?ensemble ?resource-name ?resource]
            [resource-type-of ?resource controller-file]
            [resource-might-have-been-attacked ?resource
            hacked-code-file-attack]]
    then [and [attack-implies-compromised-mode
            hacked-code-file-attack ?resource hacked .9 ]
            [attack-implies-compromised-mode
            hacked-code-file-attack ?resource normal .1 ]])
```

Listing 3. ***Example specification of code-attack.***

Based on the assumption that the specification runs safely, our security monitor is sound and complete, i.e., there are no false alarms or undetected computational attacks by the monitor. Specifically, we have proved the soundness and completeness of our monitor for some specific constructs, as shown in [17] and [20] to show that our proof method works, in principle.

A. Example

In order to demonstrate our approach, we have defined a specification language that allows to describe the behavior of a simple PID controller (see Listing 4) that manages the level of a water tank through a pump. We have also implemented a working prototype for our security monitor and have applied it to monitor the PID controller. As a starting point, we have modeled the cyber resources (i.e., PID controller) and physical resources (i.e., feed-water subsystem) of a typical ICS. In detail, the application specification ("AppSpec") includes a cyber model that specifies the

```
(define-component-type controller-step
  :entry-events (controller-step)
  :exit-events (controller-step)
  :allowable-events (update-state accumulate-error)
  :inputs (controller observation dt)
  :outputs (command error)
  :behavior-modes (normal)
  :components (
    (estimate-error
        :type estimate-error
        :models (normal))
    (compute-derivative
        :type compute-derivative
        :models (normal))
    (compute-integral
        :type compute-integral
        :models (normal)))
  :dataflows (
    (observation
        controller-step observation estimate-error)
    (the-error
        estimate-error the-error compute-derivative)
    (derivative
        compute-derivative derivative
            compute-derivative-term)
    (weighted-proportional
            compute-proportional-term
                proportional compute-correction)
    (the-error
        estimate-error error controller-step) ) )
    ...
```

Listing 4. ***Example specification of controller-step.***

computations performed by the PID controller, and a physical model that specifies physical characteristics and dynamics of the water tank subsystem as shown in Fig. 2. The latter is employed to detect computational and FDI attacks, by observing the systematic deviation of the sensor's behavior from the state of the physical plant, as predicted by the model. Moreover, based on "AppSpec," bugs in the implementation (AppImpl) can also be detected.

The focus of our prototype development of RSM is to detect "computational" attacks, where an attacker successfully alters the operation or parameter values of the PID controller implementation (AppImpl). For demonstration, we have modeled the PID controller with four parameters: the set point, and three weighting factors K_p, K_i and K_d. It performs the following steps (see :components in Listing 4).

1) Use the sensor values to estimate the state of the system.
2) Compute the difference between the estimated system state and the set point of the controller (error term).
3) Compute the local derivative of the error.
4) Integrate the error.
5) Compute the corresponding correction:
    a) multiply the error by K^P (see Listing 5);
    b) multiply the integral of the error by K^i;
    c) multiply the derivative of the error by K_d.
6) Compute and output the sum of the above three terms as the correction term.

Each component of the PID controller implementation (AppImpl) corresponds to each of the above six algorithmic steps. Similarly, each component has an AppSpec model including preconditions and postconditions. The actual implementation of each of these steps is wrapped, and their input and output are presented to the RSM for their consistency checking.

We have implemented our prototype monitor in Allegro Common Lisp on a MacBook Pro with a 2.8-GHz Intel Core i7 processor. In a real operational environment, the PID algorithm will run as the application in a programmable logic controller (PLC), while our RSM will run as part of the PLC's middleware. Thus, the PID application will be developed through any application development environment, e.g., using ladder logic, while the RSM will be part of the operating system, considering the current trend of PLC environments where real-time operating systems (RTOSs) are continuously adopted, or the middleware that runs on the PLC, if it is not an operating system. We assume that the PLC runs a control cycle on the water tank subsystem every 0.1 s, which is a very low rate of control cycles for such an ICS. Our experiment simulates the subsystem for 100 s, i.e., 1000 control cycles, and we observe that, when we enable fine-grained monitoring, the RSM consumes 8.93×10^{-4} s of CPU time and 9.07×10^{-4} s of real time for each cycle of the PID algorithm, which is very small (less than 2%) compared to the length of the control cycle (0.1 s).

In the evaluation of our prototype implementation, the RSM quickly detects arbitrary modifications (i.e., artificial attack), because of the low abstraction gap between the RSM model (AppSpec) and the actual computation (AppImpl). For instance, if the parameter K_i is altered, then the aforementioned step 5(c) produces a different output that is instantaneously detected as being inconsistent with the predicted value.

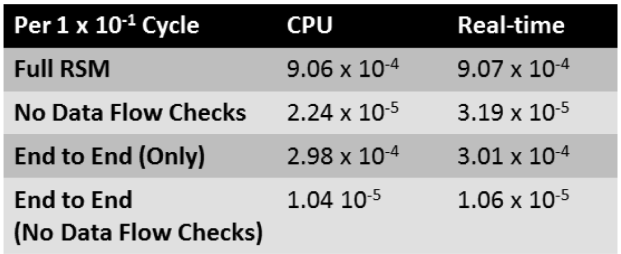
Our fine-grained modeling allows us to classify the nature of the attack immediately through fine-grained diagnostic resolution. For instance, in the above case, we can easily deduce that either K_i or the implementation of step 5(c) was altered. However, fine-grained behavioral monitoring suffers from a computational overhead. Therefore, to achieve high performance, we can simply monitor the input and output of the algorithm and ignore its intermediate steps. Clearly, this reduction in communication overhead would result in a higher diagnostic resolution overhead, because the only information we have is that the algorithm is somehow corrupted. Nevertheless, we can bypass other intermediate monitoring checks, such as the procedure execution order, and instead monitor the correctness of the data and control flows. All the aforementioned variations are evaluated as shown in Fig. 6.

A security monitor is necessary to detect cyber computational attacks at runtime. However, it is important to understand what attacks it is capable of detecting and what remaining vulnerabilities remain. Particularly, in the case of FDI attacks it has been shown that it is possible to construct a "stealthy" attack [21] that evades detection by the runtime monitor. Thus, we also have constructed methods to analyze

```
(define-component-type compute-proportional-term
    :entry-events (compute-proportional-term)
    :exit-events (compute-proportional-term)
    :inputs (error)
    :outputs (weighted-proportional)
    :behavior-modes (normal) )

(defbehavior-model (compute-proportional-term normal)
    :inputs (error)
    :outputs (weighted-proportional)
    :prerequisites
        ([data-type-of ?error number])
    :post-conditions
        ([data-type-of weighted-proportional number]
         [Quotient weighted-proportional Kp error]) )
```

Listing 5. ***Example specification of the compute-proportional-term.***



Per 1 x 10^{-1} Cycle	CPU	Real-time
Full RSM	9.06 x 10^{-4}	9.07 x 10^{-4}
No Data Flow Checks	2.24 x 10^{-5}	3.19 x 10^{-5}
End to End (Only)	2.98 x 10^{-4}	3.01 x 10^{-4}
End to End (No Data Flow Checks)	1.04 10^{-5}	1.06 x 10^{-5}

Fig. 6. ***Application execution performance.***

the behavior of the runtime monitor thereby either proving that the monitor provides complete coverage or identifying types of attacks that can evade the monitor.

V. VULNERABILITY ANALYSIS FOR FDI ATTACKS

FDI attacks against CPSs are becoming increasingly popular among attackers, because they avoid attacking computing systems and their networks and just compromise the input data to control systems. The goal of an FDI attack is to lead the control system to make wrong decisions and take wrong actions based on manipulated input data, rather than compromising the systems themselves; in these attacks, systems operate correctly, but on the wrong input data. A simple example, referring to our water tank case, would be to input to the controller a sensor value for the water level that is close to zero, leading the controller to open the pump, introduce more water in the tank (to reach the targeted water level) and thus, overflow the tank. In industrial environments, cars, medical systems, and similar application domains, such actions can be catastrophic, even fatal.

Recently, FDI attacks have drawn research attention, focusing mostly on power systems [8], [22]–[25]. In these efforts, power systems have been analyzed to identify conditions under which FDI attacks remain undetected and to propose defenses. Most efforts have been following traditional approaches to attack and failure detection, based on network topology characteristics and statistical analysis of data.

We follow a different approach to defend against FDI attacks. We have developed a method for vulnerability analysis that identifies FDI vulnerabilities of a system at the design phase [8]. Based on the identified vulnerabilities, we redesign the system, introducing constraints that lead to elimination of identified FDI attacks. This redesign process is repeated until either FDI vulnerabilities cannot be identified, or the expected FDI attacks have been considered.

The basic concept of our method is to describe the system through a state function, which is analyzed for input combinations that constitute FDI attacks. We consider that a CPS implements a control loop, as shown in Fig. 7, for a process P. At every instant t, the state of the system is described with a function $P(x_t)$, where x_t is the set of input variables to the process at time t. In the implementation of the system, the variables x are measured with sensors, input to the controller, and used to calculate the state of the system as well as the necessary actions, denoted with a_t. We denote with z_t the measurements of variables x_t, i.e., the measurements of variables x at time t. We also consider that, as in typical control systems, the system is monitored by a monitor $\text{mon}(x, z)$, which, at time t, inputs the measurements z_t and evaluates them for acceptance due to potential sensor failures.

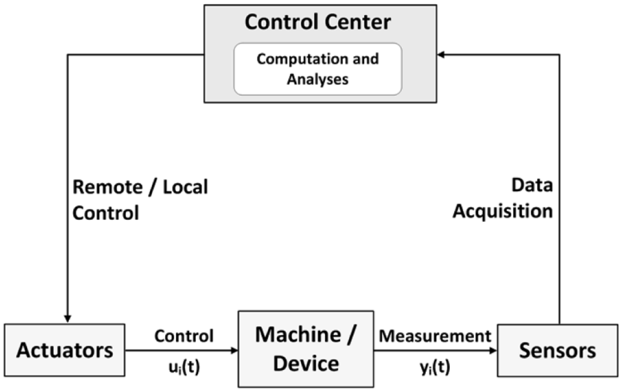


Fig. 7. *ICS control loop.*

Considering a fault-free operation, $\text{mon}(x, z)$ accepts all measurements z and calculates the state $P(x, z)$ of the system using them. A successful FDI can be launched against the system using a set of measurements z', if 1) $\text{mon}(x, z')$ accepts z'; and 2) z' are compromised values of z, i.e., $z' \neq z$. Thus, we can identify the existence of one or more FDI attacks, if we can answer positively the question: Does there exist a z' for the system $P(x, z)$ and $\text{mon}(x, z)$?

Our approach identifies the existence of FDI attacks by answering the above question. To achieve this, we represent the state of the system with a real function $f()$ and we express the above question as an input to an SMT solver for real functions, namely dReal [26]. If the expression that represents the FDI existential question is satisfiable, then our system is vulnerable to FDI attacks; importantly, the SMT solver provides us with one set z' that constitutes such an attack.

When FDI attacks are identified, a method to defend against them is to place constraints on the values of the measured parameters, so that the set of values that satisfy the state function is reduced. As we introduce more constraints, the space of solutions to the state function becomes smaller and thus, reduces—if it does not eliminate—the potential FDI attacks. Based on this, when we identify an FDI vulnerability, we can redesign the system introducing constraints and thus, reducing the attack surface. The new design can be subsequently analyzed for new FDI attacks with the new constraints. If FDI vulnerabilities are identified again, the system can be further constrained and redesigned and so on. The process can be repeated until the realistic FDI attacks are eliminated or reduced to meet the set system specifications.

A. Example

In order to demonstrate our approach, we provide an example using a variation of the water tank that we have analyzed through the paper, as shown in Fig. 2. In this variation, the water tank has two pumps, one with incoming water and one with outgoing water, each pump with one sensor measuring its water rate (r_{in} and r_{out} for the incoming and outgoing flow, respectively) and one actuator, opening it to a specified flow rate. Furthermore, we assume that the system has one sensor measuring the water level. With this model, we analyze a system with three sensors rather than the trivial case of two sensors, as the gravity-draining water tank. For convenience, we consider that the rates r_{in} and r_{out} are real numbers in the range $\{0,1\}$ and the horizontal cross-section area of the tank is 1. These normalized values lead to the property that the volume of water in the tank has the same value as the height of the water in it, making the state function expression clearer.

For the example water tank, we consider that it operates in discrete time and at the beginning of every time unit the state of the pumps may change. The state of system, which is monitored by the monitor $\text{mon}(x,z)$, is expressed with the function $H(t+1) = H(t) + r_{\text{in}}(t+1) - r_{\text{out}}(t+1)$ that represents the height $H(\,)$ of the water at the end of the time unit $t+1$.

Let us consider the following scenario: at time t, $(t) = 5$, and at time $(t+1)$ the system is configured with $r_{\text{in}}(t+1) = 0.5$ and $r_{\text{out}}(t+1) = 0$. Clearly, $H(t+1) = 5.5$. If the sensors of the system provide to the monitor the correct values $r_{\text{in}}(t+1) = 0.5$, $r_{\text{out}}(t+1) = 0$, and $H(t+1) = 5.5$, the monitor will accept the state of the system, based on $H(t) = 5$. However, an attacker could compromise the values for $r_{\text{in}}(t+1)$, $r_{\text{out}}(t+1)$, and $H(t+1)$ and provide fake values to the monitor. The monitor will accept them, if they are consistent with the function for $H(t+1)$, and reject them, i.e., detect the attack, if the fake values are not consistent with the function $H(t+1)$. For example, if an attacker provides as input the values $r_{\text{in}}(t+1) = 1$, $r_{\text{out}}(t+1) = 0$, and $H(t+1) = 8$, then the monitor will detect an inconsistency since $H(t+1) \neq H(t) + r_{\text{in}} - r_{\text{out}} \Leftrightarrow 8 \neq 5+1-0$. However, if the attacker provides the values $r_{\text{in}}(t+1) = 1$, $r_{\text{out}}(t+1) = 0$, and $H(t+1) = 6$, the monitor will accept them because they satisfy $H(t+1) = H(t) + r_{\text{in}} - r_{\text{out}} \Leftrightarrow 6 = 5+1-0$ although they are not the real values; this demonstrates that the system is vulnerable to an FDI attack when (sensors) values r_{in}, r_{out}, and H are compromised. We can identify such an FDI attack with our method through the use of the SMT solver, providing the monitoring function $H(\,)$ as an input and asking if there is a solution to this equation with parameter values r_{in}, r_{out}, and H in the defined ranges, which is different from the real action. Listing 6 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 7. Analyzing the system, one can easily realize that the space of existent FDI attacks is large, considering all the combinations of values that satisfy $H(\,)$ and are different from the specific real values of the example.

```
(set-logic QF_NRA)
(declare-fun ht () Int)
(declare-fun ht1 () Int)
(declare-fun a () Int)
(declare-fun rin () Real)
(declare-fun rout () Real)
(assert (and (and (= ht 5) (= ht1 6)) (= a 1)))
(assert (and (<= 0 rin) (<= rin 1)))
(assert (and (<= 0 rout) (<= rout 1)))
(assert (= ht1 (+ ht (- (/ rin (^ a 2)) (/ rout (^ a 2))))))
(check-sat)
(exit)
```

Listing 6. ***Example water tank specification.***

Based on the above and considering the large space of potential attacks, a designer can introduce constraints on the vulnerable parameters (r_{in}, r_{out} and H), in order to reduce the attack surface. For example, one can decide that r_{in} should be an integer, i.e., 0 or 1, rather than a real number, reducing the attack surface significantly. A similar constraint on the value of r_{out} will make the potential attacks quite few, leading to a new design and a significantly more robust system. It should be noted that the gravity-draining water tank that we have analyzed in this paper is a constrained system (r_{out} is not a parameter), which provides a smaller attack surface than the two-pump tank we analyze here; this is the reason why it constitutes a trivial example for this analysis.

Clearly, this methodology can be applied to any system with a monitoring function that can be input to an SMT solver like dReal. We have applied our method to more complex systems, specifically alternating current (ac) state estimators for smartgrids [27], with successful results. We have analyzed benchmark power distribution networks, based on the IEEE 14-Bus, 30-Bus, 157-Bus, and 300-Bus benchmark suite, demonstrating not only the effectiveness of the approach, but also its feasibility for realistic and practical systems. As we have demonstrated in our power state analysis, FDI vulnerabilities are identified successfully with analyses that are efficient, independently of the size of realistic networks. Fig. 2 in [27] shows that the time required to analyze benchmark power networks for FDI on state estimation is in the order of seconds or minutes, depending on the exact network configuration and the constraints on node branches and buses. However, for a design cycle of a practical power distribution network, this performance of the analysis makes the approach an indispensable tool for the development of networks that are robust against FDI attacks.

VI. RELATED WORK

This section includes state of the art for each component of our design methodology, i.e., secure-by-design ICS applications, runtime security monitoring, and vulnerability analysis for FDI attacks, respectively.

```
a : [ ENTIRE ] = [1, 1]
ht : [ ENTIRE ] = [5, 5]
ht1 : [ ENTIRE ] = [6, 6]
rin : [ ENTIRE ] = [1, 1]
rout : [ ENTIRE ] = [0, 0]
delta-sat with delta = 0.00100000000000000
```

Listing 7. ***Example FDI detection.***

A. Reliable-and-Secure-by-Design ICS Applications

Based on the principle of correct-by-construction approach [28], we have devised an approach to develop secure-by-design ICS applications. Here, the challenge is to model behavior of both cyber and physical as first class models, though they possess fundamentally different semantics (i.e., discrete and continuous) and characteristics (i.e., linear and nonlinear). Yet another challenge here is to adequately model uncertain physical environmental variations that may influence behavior of physical resources of an ICS on one hand, and operate on variable levels of information abstraction about physical environment on the other hand. Based on Fiat [15], secure-by-design approach allows to derive correct and secure implementation from specification through stepwise refinements using Coq. Thus, allowing to construct reliable and secure ICS applications by design, the approach has advanced existing approaches in at least one of the following ways: 1) reliability and security by design; 2) ADT-based CPS modeling; and 3) deductive synthesis of security properties.

1) Reliability and Security by Design: For the last decade, there have been various efforts to develop approaches that allow reliable and secure implementations by design. For instance, Ur/Web [29] is a language for developing reliable and secure web applications by construction/design. For reliability, the language assures that the application will 1) not crash during generating web pages; 2) not return invalid HTML; and 3) not produce dead intra-application links, to name a a few. Furthermore, Ur/Web assures that the application 1) does not suffer from code-injection attacks; and 2) does not attempt invalid SQL queries, to name a few. The language supports a rich-type system based on dependent types that guarantees that the developed application respects the aforementioned features. However, the goal of Ur/Web was to provide a unified web model, where a programmer develops web application in a single programming language that can be compiled to other web standards supporting encapsulation of state and concurrency of multithreaded applications. In [10], Yang *et al.* have developed a language Jeeves whose runtime enforces security policies and guarantees that the programs respects security properties by construction. However, the goal of Jeeves was to enforce security policies at runtime. Recently, there has been some efforts that focused on applying the aforementioned alike approaches in ICS domain. For instance, the ROSCoq framework [30] has been developed in Coq to model cyber and physical resources of robots. The framework has extended logic of events to model the resources of CPS involving CoRN theory of constructive real analysis and then to assure various properties of the model. In [31] and [32], the authors have developed a Coq library "VeriDrone" as a reasoning framework to ensure security of CPS models at different but independent levels, i.e., from high level models to CPS implementation in C.

In contrast to the aforementioned approaches, our approach derives correct and secure implementation from a specification that treats models of cyber and physical resources as first class models with complex behavioral characteristics. The derivation is based on stepwise refinements of the hybrid model using Coq. We encode such complex models (including security and reliability properties) and characteristics (including discrete and continuous) as abstract data types (ADTs) in Coq and then derive implementations through deductive synthesis of ADTs using Coq that is secure and correct by construction with a formal proof. As far as we are aware, no other design methodology allows the derivation of secure, correct, and efficient implementations of ICSs from a specification treating cyber and physical models as first class models.

2) ADT-Based CPS Modeling: There have not been more results in the modeling of cyber–physical resources using ADTs. Recently, in [33], based on VDM, the authors have developed a design methodology that allows comodeling of cyber and physical resources using operations defined over ADTs. The methodology allows to interpret the model as a bond graph, which is a directed graph that compares the comodel at the end.

3) Deductive Synthesis of Security Properties: As far as we are aware, deriving correct and secure implementations from a given ADT-based CPS model through deductive synthesis has not been addressed so far. However, there have been efforts to derive correct implementations for simple domains from a given specification through a deductive synthesis. For instance, Paige and Henglein [34] show the derivation of initial implementations of the ADT-based specification employing fix-point iterations, and later optimizing the implementation using finite differencing. Lately, in [35], Hawkins *et al.* have applied the deductive synthesis to derive ADT-based implementations based on abstract relational descriptions for various database operations, e.g., query and update.

Our approach extends the above works by allowing to encode the specification in a general purpose theorem prover, Coq, which enables us to perform sound synthesis by checking the proof for consistency. However, our goal is to apply the same method to the CPS domain, by algorithmically generating secure-and-reliable implementations from ADT-based specifications through a deductive synthesis using Coq.

B. Runtime Security Monitoring

Runtime security monitoring of ICSs is a complex and challenging task, as it involves physical processes and critical infrastructures. In order to address these challenges, we have developed a runtime security monitor that advances the existing techniques in at least one of the following ways: 1) formalism of our specification language; 2) modeling security properties; 3) monitoring based on executable specification; 4) modular and abstract specification; and 5) performance efficiency.

1) Formalism: Formalism of our specification language consists of monadic second-order logic (MSL) [36] and event calculus [37] operating over algebraic data structures. This formalism is the most appropriate to model cyber and physical resources of ICSs and their security properties, with strict real-time requirements. Based on equivalence results of finite automaton and MSL [38]–[41], any MSL formula can be efficiently translated into a machine code [19], enabling our monitor to perform efficient comparison of predictions and observations respecting real-time constraints of ICSs. In order to handle potential unknown attacks, our language allows to specify various attack plans that are later exploited by the monitor for timely detection of such attacks and threats including advanced persistent threats and zero-day attacks. Crash Hoare-logic [42] is similar to our formalism, as it also allows to specify the unsafe behavior of a file system. In order to specify fundamentally different characteristics (e.g., hybrid systems with a mix of discrete and continuous behaviors) and semantics of physical and cyber models, our formalism allows composition and construction of high-level system behavior (i.e., discrete) from semantically different low-level behavior(s) (i.e., continuous) by employing a method analogous to classical set builder together with closure property of MSL formulas under function composition [43], [44]. The formalism of most of the existing ICS security monitors is based on temporal logic [45], [46], rule systems [47], regular expressions and grammars [48], to name a few. However, the expressive power of these formalisms does not comprehend ICS requirements for modeling physical processes and security properties as first class specifications, integral with functional specification.

2) Modeling Security Properties: Adequate modeling of an ICS application is a complex task, as it involves modeling of both cyber and physical resources that have different characteristics and real-time constraints [49]. Therefore, as a prerequisite of a security model of an ICS application, our language allows to describe physical processes/resources as first class models along with cyber resources. However, with cyber and physical models as first class models, modeling and monitoring security properties become challenging because security properties need to combine the discrete and continuous behaviors of cyber and physical processes. Therefore, we model a security property as a relationship between a set of discrete behavior of cyber processes and a set of continuous behavior of physical processes based on event calculus. In fact, in a typical CPS, cyber processes are realized as discrete controllers that are responsible for making decisions, while the physical processes are realized as continuous controllers that determine physical dynamics and execute the decisions of the discrete controllers. Recently, there has been some effort to model security properties of ICSs. For instance, a language ASLan++ [45] allows to model security properties of ICSs for a watar treatment plant in discrete time. Here, our approach is similar to APEX [50] which employs recent results in reachability analysis [51] to verify hybrid systems. In detail, APEX allows to model discrete and continuous system constraints and then checks whether the system reaches unsafe regions. The tool suffers from scalability issues due to the state explosion problem in model checking. However, the goal of our work is to check the consistency between hybrid system constraints and a "run" of the system at runtime. In fact, we deal with a single instance of execution at a time and thus avoid scalability concerns.

3) Monitoring Based on Executable Specifications: Building runtime security monitors from executable specifications for monitoring real-time systems has recently started getting attention [52]. However, executable specifications are powerful in detecting any violation of real-time constraints while executing the specification in parallel to the application are more suitable for the runtime security monitoring of ICSs. In [53], an executable specification ASML for runtime monitoring has been developed at Microsoft. ASML is developed based on state transition systems whose states are first-order algebras [54]. An executable specification language for runtime monitoring of timed systems has been proposed by Chupilko and Kamkin [55] who use extended time interval as a pair of a time event and a time interval to model properties, which is used by the monitor to check the conformance of an implementation word and the specification trace. Also in [56], Ghezzi *et al.* developed executable specification specification and environment (TRIO/TRIO+) based on events and their relationship, interpreted in first-order temporal logic. However, the language is not suitable for modeling real-time systems, as it does not support modeling hierarchical and modular specifications [57]. Recently, the authors have developed a method for attack detection in ICSs (a water treatment plant) by deriving physical process invariants for each stage of the CPS from its design and then monitoring the invariants at runtime. Like [58], we also specify physical process invariants to detect any security threat to ICSs. Additionally, we also specify other functional and nonfunctional constraints, e.g., performance. In contrast to aforementioned executable specification languages, the set theoretic formalism of our specification language directly supports classification of observed behaviors that belong to different sets of specification. Furthermore, our language enables modeling hierarchical and modular specifications.

4) Modular and Abstract Specification: The ICS physical processes are dynamic, as they depend on physical environmental conditions, and thus, are abstract in nature, subject to evolution. For instance, the filtering membrane of a water desalination plant may start filtering at a different rate with respect to the amount of humidity in the environment.

Thus, in order to support modeling of such systems and their evolving constraints, we have introduced model-based abstract [59] and modular specification, whose syntax (i.e., constructs) and operational semantics (i.e., execution) are not directly dependent on the structure of the ICS application implementation. In contrast to our language, classical-model-based specification languages, for instance [60], allow only a contract-based model, whose execution flow operation (i.e., structure and semantics) representation is dependent on the syntactic structure of the application implementation. Such models 1) do not support the modeling of information that is independent of the application implementation, e.g., nonfunctional constraints; and 2) supports limited modularity, only in the case when the application implementation is modular. Furthermore, our language is highly modular and abstract, e.g., in our case, application specification (AppSpec) is independent of the application's implementation (AppImpl). Hence, our languages allow modeling those behavioral details that operate on top of the implementation, for instance, component specific workflow execution under additional (dynamic) constraints or (evolving) security policies/requirements. The syntax of our language is similar to that of the executable specification language OBJ [61]. Since our language is developed in Lisp, the efficiency and the strength of the abstraction and modularity of our language lie in Lisp's underlying expert system Joshua [62]; Joshua provides deductive facilities to our language, based on forward and backward chaining rules that are realized as generic functions to support arbitrary abstraction. Furthermore, Joshua has builtin support for modularity that is identifiable and reusable. Joshua enables the selection of arbitrary data structures to achieve the desired efficiency.

5) *Efficiency:* Developing runtime security monitors to meet real-time ICS constraints is a tradeoff between security and efficiency of the ICS application. Thus, in order to meet the strict real-time ICS constraints, we have developed a tunable runtime security monitor that monitors adequate behavior (i.e., all preconditions, postconditions, and invariant) at the time of high threat, and monitors partial (i.e., any combination of preconditions, postconditions, and invariant) behavior otherwise. There are no fixed performance metrics for runtime security monitoring of ICS applications [63]. However, an evaluation of such monitors is required against the real-time constraints of ICSs. In principle, the real-time constraints 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. Furthermore, our results (see Section IV-A) show that, even in highly threatened scenarios, our monitor executes in less than 1 ms, well below the real-time constraints of ICSs for various application domains. For example, water or power management systems have a desired response delay of a few milliseconds.

Additionally, we will develop a mechanized proof showing that the monitor only alarms if there is an attack, and *vice versa*, using some proof assistant, e.g., Coq. We have already shown in [17] and [20] that our proof method works in principle.

C. Vulnerability Analysis for FDI Attacks

FDI attack modifies the measurement values that are exchanged among various ICS subsystems. Such values eventually mislead the controller application to conclude undesired results [8]. Most of the existing approaches have attempted to model linear state estimates of power grids [23], [64]–[67]. In fact, analysis of realistic nonlinear state estimation models is much harder [22], [24], [66]. 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.

In contrast to the aforementioned approaches, we aim to develop FDI free models by design based on recent results in delta-decision procedures [68]. Our approach allows to specify nonlinear ICS models in dReal [26], which then searches those input values of the variables for which the monitor does not alarm by reasoning about nonlinear logical constraints over real numbers.

VII. CONCLUSION AND FUTURE WORK

We have introduced a complete behavior-based approach to the design of secure ICSs. Our method targets to produce an industrial control application implementation that satisfies desired security properties (secure-by-design) as well as a runtime security monitor that identifies runtime attacks by comparing the expected application behavior with the behavior of the executed code. The method is feasible and practical because it originates from an executable specification of the application. Importantly, our approach includes a method to develop systems that are resistant not only to computational attacks but also FDI attacks, through the use of a vulnerability analysis technique that leads to secure application designs or identification of FDI attack values that can be monitored at runtime. We have demonstrated the effectiveness and practicality of our method through detailed descriptions of the development of a secure application for a simple water tank management ICS.

Our approach leads to a proposed design methodology that is composed of three components: secure-by-design application development, production of a security runtime monitor, and production of an FDI attack monitor through vulnerability analysis. In our experiments, up to date, we have been using Coq and dReal for the secure-by-design component and the vulnerability analysis component, while we have developed the security runtime monitor, ARMET.

In future work, we will be working on the automation of all required procedures to develop a unified design methodology, envisioning the ability to produce reliable and secure application code, runtime security monitors, and runtime FDI attack value monitors automatically from a single executable specification of the ICS application.

Furthermore, with the recent great developments in formal verification of software applications and operating systems [69], [70], our monitoring component will run without any performance and scalability issues in real-time ICSs. Such developments will allow our RSM and other associated components (e.g., control application, operating system) to run at the same level of abstraction (i.e., specification language), thus ensuring computation overhead to be negligible on one hand, and no scalability issues on the other hand. Furthermore, to achieve even higher performance, our RSM allows to monitor desired ICS behavior on demand, e.g., to monitor, any or all, among preconditions, postconditions, and invariant. ■

Acknowledgment

The authors would like to thank Prof. A. Chlipala, Prof. A. Solar-Lezama, and Dr. S. Gao for their indispensable collaboration and support of this project.

REFERENCES

[1] W. A. Conklin, "IT vs. OT security: A time to consider a change in CIA to include resilienc," in *Proc. 49th Hawaii Int. Conf. Syst. Sci. (HICSS)*, Jan. 2016, pp. 2642–2647.

[2] M. Zeller, "Myth or reality—Does the aurora vulnerability pose a risk to my generator?" in *Proc. 64th Annu. Conf. Protective Relay Eng.*, Apr. 2011, pp. 130–136.

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

[4] B. Kang, K. McLaughlin, and S. Sezer, "Towards a stateful analysis framework for smart grid network intrusion detection," in *Proc. 4th Int. Symp. ICS SCADA Cyber Secur. Res.*, 2016, pp. 1–8.

[5] U. Lindqvist and P. G. Neumann, "The future of the Internet of Things," *Commun. ACM*, vol. 60, no. 2, pp. 26–30, Jan. 2017.

[6] R. Mitchell and I.-R. Chen, "A survey of intrusion detection techniques for cyber-physical systems," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 55-1–55-29, Mar. 2014.

[7] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes, "Using model-based intrusion detection for scada networks," in *Proc. SCADA Secur. Sci. Symp.*, Jan. 2007, pp. 1–12.

[8] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 1, pp. 13-1–13-33, Jun. 2011.

[9] A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry, "Attacks against process control systems: Risk assessment, detection, and response," in *Proc. 6th ACM Symp. Inf., Comput. Commun. Secur.*, 2011, pp. 355–366.

[10] J. Yang, K. Yessenov, and A. Solar-Lezama, "A language for automatically enforcing privacy policies," in *Proc. 39th Annu. ACM SIGPLAN-SIGACT Symp. Principles Programm. Lang.*, New York, NY, USA, 2012, pp. 85–96.

[11] M. Zhang, Y. Duan, Q. Feng, and H. Yin, "Towards automatic generation of security-centric descriptions for android apps," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2015, pp. 518–529.

[12] F. Martinelli and I. Matteucci, "An approach for the specification, verification and synthesis of secure systems," *Electron. Notes Theor. Comput. Sci.*, vol. 168, pp. 29–43, Feb. 2007.

[13] I. Matteucci, "Automated synthesis of enforcing mechanisms for security properties in a timed setting," *Electron. Notes Theor. Comput. Sci.*, vol. 186, pp. 101–120, Jul. 2007.

[14] R. Soulat, "Synthesis of correct-by-design schedulers for hybrid systems," Ph.D. dissertation, École Normale Supérieure Paris-Saclay—ENS Cachan, Cachan, France, Feb. 2014.

[15] B. Delaware, C. Pit-Claudel, J. Gross, and A. Chlipala, "Fiat: Deductive synthesis of abstract data types in a proof assistant," in *Proc. 42nd Annu. ACM SIGPLAN-SIGACT Symp. Principles Programm. Lang. (POPL)*, Mumbai, India, Jan. 2015, pp. 689–700.

[16] C. A. R. Hoare, "Proof of correctness of data representations," *Acta Inf.*, vol. 1, no. 4, pp. 271–281, 1972.

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

[18] H. Shrobe, "AWDRAT: A cognitive middleware system for information survivability," in *Proc. 18th Conf. Innovative Appl. Artif. Intell.*, vol. 2. 2006, pp. 1836–1843.

[19] B. Courcelle and J. Engelfriet, *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Cambridge, MA, USA: Cambridge Univ. Press, 2012.

[20] M. T. Khan, D. Serpanos, and H. Shrobe, "Sound and complete runtime security monitor for application software," Dept. Comput. Sci. Artif. Intell. Lab., MIT, Cambridge, MA, USA, Tech. Rep. MIT-CSAIL-TR-2016-017, Dec. 2016.

[21] C. Kwon, W. Liu, and I. Hwang, "Analysis and design of stealthy cyber attacks on unmanned aerial systems," *J. Aerosp. Inf. Syst.*, vol. 11, no. 8, pp. 525–539, 2014.

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

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

[24] M. A. Rahman and H. Mohsenian-Rad, "False data injection attacks with incomplete information against smart power grids," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2012, pp. 3153–3158.

[25] M. A. Rahman, E. Al-Shaer, and M. A. Rahman, "A formal model for verifying stealthy attacks on state estimation in power grids," in *Proc. IEEE 4th Int. Conf. Smart Grid Commun. (SmartGridComm)*, Vancouver, BC, Canada, Oct. 2013, pp. 414–419.

[26] S. Gao, S. Kong, and E. M. Clarke, "dReal: An SMT solver for nonlinear theories over the reals," in *Proc. 24th Int. Conf. Autom. Deduction*, 2013, pp. 208–214.

[27] S. Gao, L. Xie, A. Solar-Lezama, D. N. Serpanos, and H. E. Shrobe, "Automated vulnerability analysis of AC state estimation under constrained false data injection in electric power systems," in *Proc. 54th IEEE Conf. Decision Control (CDC)*, Dec. 2015, pp. 2613–2620.

[28] E. W. Dijkstra, "A constructive approach to the problem of program correctness," *Circulated Privately, Tech. Rep.*, Aug. 1967.

[29] A. Chlipala, "Ur/Web: A simple model for programming the Web," *Commun. ACM*, vol. 59, no. 8, pp. 153–165, Aug. 2016.

[30] A. Anand and R. Knepper, *ROSCoq: Robots Powered by Constructive Reals*. Berlin, Germany: Springer-Verlag, 2015, pp. 34–50.

[31] G. Malecha, D. Ricketts, M. M. Alvarez, and S. Lerner, "Towards foundational verification of cyber-physical systems," in *Proc. Sci. Secur. Cyber-Phys. Syst. Workshop (SOSCYPS)*, Apr. 2016, pp. 1–5.

[32] M. Chan, D. Ricketts, S. Lerner, and G. Malecha, "Formal verification of stability properties of cyber-physical systems," in *Proc. CoqPL*, Jan. 2016.

[33] J. Fitzgerald, K. Pierce, and C. Gamble, "A rigorous approach to the design of resilient cyber-physical systems through co-simulation," in *Proc. IEEE/IFIP 42nd Int. Conf. Depend. Syst. Netw. Workshops (DSN-W)*, Jun. 2012, pp. 1–6.

[34] R. Paige and F. Henglein, "Mechanical translation of set theoretic problem specifications into efficient RAM code—A case study," *J. Symbolic Comput.*, vol. 4, no. 2, pp. 207–232, Aug. 1987.

[35] P. Hawkins, M. Rinard, A. Aiken, M. Sagiv, and K. Fisher, "An introduction to data representation synthesis," *Commun. ACM* , vol. 55, no. 12, pp. 91–99, Dec. 2012.

[36] J. Henriksen, "Mona: Monadic second-order logic in practice," in *Tools and Algorithms for the Construction and Analysis of Systems*

(Lecture Notes in Computer Science), vol. 1019. 1995, pp. 89–110.

[37] G. C. Borchardt, "Event calculus," in *Proc. 9th Int. Joint Conf. Artif. Intell.*, vol. 1. 1985, pp. 524–527.

[38] R. J. Büchi, "Weak second-order arithmetic and finite automata," *Zeitschrift Mathe. Logik Grundlagen Math.*, vol. 6, nos. 1–6, pp. 66–92, 1960.

[39] J. R. Buechi, "On a decision method in restricted second-order arithmetic," in *Proc. Int. Congr. Logic, Methodol. Philos. Sci.*, 1962, pp. 1–11.

[40] M. O. Rabin, "Decidability of second-order theories and automata on infinite trees," *Trans. Amer. Math. Soc.*, vol. 141, nos. 1–35, p. 4, 1969.

[41] C. C. Elgot, "Decision problems of finite automata design and related arithmetics," *Trans. Amer. Math. Soc.*, vol. 98, no. 1, pp. 21–51, 1961.

[42] H. Chen, D. Ziegler, T. Chajed, A. Chlipala, M. F. Kaashoek, and N. Zeldovich, "Using crash hoare logic for certifying the FSCQ file system," in *Proc. 25th Symp. Oper. Syst. Principles*, 2015, pp. 18–37.

[43] R. Alur, A. Durand-Gasselin, and A. Trivedi, "From monadic second-order definable string transformations to transducers," in *Proc. 28th Annu. ACM/IEEE Symp. Logic Comput. Sci. (LICS)*, Mar. 2013, pp. 458–467.

[44] B. Courcelle, "Monadic second-order definable graph transductions: A survey," *Theor. Comput. Sci.*, vol. 126, no. 1, pp. 53–75, 1994.

[45] M. Rocchetto and N. O. Tippenhauer, "Towards formal security analysis of industrial control systems," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 114–126.

[46] A. Bauer, M. Leucker, and C. Schallhart, "Runtime verification for LTL and TLTL," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 4, pp. 1–68, 2011.

[47] H. Barringer, D. E. Rydeheard, and K. Havelund, "Rule systems for run-time monitoring: From eagle to ruler," *J. Logic Comput.*, vol. 20, no. 3, pp. 675–706, 2010.

[48] F. Chen and G. Rosu, "MOP: An efficient and generic runtime verification framework," *ACM Sigplan Notices*, vol. 42, no. 10, pp. 569–588, 2007.

[49] I. Ruchkin, S. Selva, S. Bradley, R. Amanda, and G. David, "Challenges in physical modeling for adaptation of cyber-physical systems," in *Proc. IEEE World Forum Internet Things MARTCPS*, Dec. 2016, pp. 210–215.

[50] M. O'Kelly, H. Abbas, S. Gao, S. Shiraishi, S. Kato, and R. Mangharam, "APEX: Autonomous vehicle plan verification and execution," in *Proc. SAE World Congr.*, vol. 1. Apr. 2016, pp. 1–13.

[51] S. Kong, S. Gao, W. Chen, and E. Clarke, *dReach: δ-Reachability Analysis for Hybrid Systems*. Berlin, Germany: Springer-Verlag, 2015, pp. 200–205.

[52] M. Blum and H. Wasserman, "Software reliability via run-time result-checking," *J. ACM*, vol. 44, pp. 826–849, 1994.

[53] M. Barnett and W. Schulte, "Runtime verification of.NET contracts," *J. Syst. Softw.*, vol. 65, no. 3, pp. 199–208, 2003.

[54] E. Borger and R. F. Stark, *Abstract State Machines: A Method for High-Level System Design and Analysis*. New York, NY, USA: Springer-Verlag, 2003.

[55] M. M. Chupilko and A. S. Kamkin, "Runtime verification based on executable models: On-the-fly matching of timed traces," in *Proc. 8th Workshop Model-Based Test.*, 2013, pp. 67–81.

[56] C. Ghezzi, D. Mandrioli, and A. Morzenti, "Trio: A logic language for executable specifications of real-time systems," *J. Syst. Softw.*, vol. 12, no. 2, pp. 107–123, May 1990.

[57] S. Gérard, H. Espinoza, F. Terrier, and B. Selic, "Modeling languages for real-time and embedded systems: Requirements and standards-based solutions," in *Proc. Int. Dagstuhl Conf. Model-Based Eng. Embedded Real-Time Syst.*, 2010, pp. 129–154.

[58] S. Adepu and A. Mathur, *Using Process Invariants to Detect Cyber Attacks on a Water Treatment System*. Cham, Switzerland: Springer-Verlag, 2016, pp. 91–104.

[59] J. M. Wing, "A specifier's introduction to formal methods," *Computer*, vol. 23, no. 9, pp. 8–23, Sep. 1990.

[60] T. Gary Leavens and Y. Cheon (2006). *Design by Contract With JML*. A Tutorial. [Online]. Available: ftp://ftp.cs.iastate.edu/pub/leavens/JML/jmldbc.pdf

[61] J. A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud, *Introducing OBJ*. Boston, MA, USA: Springer-Verlag, 2000, pp. 3–167.

[62] S. Rowley, H. Shrobe, R. Cassels, and W. Hamscher, "Joshua: Uniform access to heterogeneous knowledge structures or why joshing is better than conniving or planning," in *Proc. AAAI*, Seattle, WA, USA, 1987, pp. 48–52.

[63] A. Kane, "Runtime monitoring for safety-critical embedded systems," Ph.D. dissertation, Dept. Electr. Comput. Eng., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2015.

[64] G. Dán and H. Sandberg, "Stealth attacks and protection schemes for state estimators in power systems," in *Proc. 1st IEEE Int. Conf. Smart Grid Commun. (SmartGridComm)*, Oct. 2010, pp. 214–219.

[65] O. Kosut, L. Jia, R. J. Thomas, and L. Tong, "Malicious data attacks on the smart grid," *IEEE Trans. Smart Grid*, vol. 2, no. 4, pp. 645–658, Dec. 2011.

[66] A. Teixeira, I. Shames, H. Sandberg, and K. H. Johansson, "Revealing stealthy attacks in control systems," in *Proc. 50th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2012, pp. 1806–1813.

[67] 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, Dec. 2011.

[68] S. Gao, J. Avigad, and E. M. Clarke, "δ-Complete decision procedures for satisfiability over the reals," in *Proc. 6th Int. Joint Conf. Autom. Reasoning-(IJCAR)*, Manchester, U.K., Jun. 2012, pp. 286–300.

[69] Deepspec. [Online]. Available: https://deepspec.org/page/Research/

[70] Certikos. [Online]. Available: http://flint.cs.yale.edu/certikos/

ABOUT THE AUTHORS

Muhammad Taimoor Khan graduated in computer science from the Islamia University Bahawalpur, Bahawalpur, Punjab, Pakistan, in 2001, and received the M.S. degree in advanced distributed systems from the University of Leicester, Leicester, U.K., in 2008 and the Ph.D. degree from the Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Austria, in 2014.

He is a Postdoctoral Researcher with the Institute of Informatics, Alpen-Adria University, Klagenfurt, Austria. In the last decade, he has been applying formal methods as a powerful tool to assure reliability and security of various software systems, for instance, industrial control systems, computer mathematics-based systems, to name a few. He has extensive experience in both software industry and research institutes. He has been working as a scientist at various premier international research institutes, including INRIA, France and MIT CSAIL, Cambridge, MA, USA; he is jointly working with these institutes now.

Dr. Khan has won various research and academic awards including best paper award(s).

Dimitrios Serpanos (Senior Member, IEEE) received a Diploma in computer engineering and informatics from the University of Patras, Patras, Greece, in 1985 and the Ph.D. degree in computer science from Princeton University, Princeton, NJ, USA, in 1990.

He is the Director of the Industrial Systems Institute/ATHENA RC and a Professor at the Department of Electrical and Computer Engineering, University of Patras. Before joining the University of Patras, he was a Research Staff Member at IBM, T.J. Watson Research Center, Yorktown Heights, NY, USA (1990–1996) and faculty member at the Department of Computer Science, University of Crete, Crete, Greece (1996–2000), where he also worked at the Institute of Computer Science of the Foundation for Research and Technology-Hellas (ICS-FORTH). He has been Principal Scientist at the Qatar Computing Research Institute (2013–2016). He has served as President of the University of Western Greece (2010–2013) and another term as Director of the Industrial Systems Institute/ATHENA RC (2008–2013), where he has been conducting research since 2000. His research interests include embedded systems, security systems, industrial systems, and computer architecture.

Prof. Serpanos is currently a member of the Board of Governors of the IEEE Computer Society (2017). He is a member of the New York Academy of Sciences, the Association for Computing Machinery (ACM), the American Association for the Advancement of Science (AAAS), and the Technical Chamber of Greece.

Howard Shrobe received the M.S. and Ph.D. degrees from the MIT's Artificial Intelligence Laboratory, Cambridge, MA, USA, in 1975 and 1978, respectively.

He is a Principal Research Scientist at MIT Computer Science and Artificial Intelligence Laboratory (MIT CSAIL), Cambridge, MA, USA. He is a former Associate Director of CSAIL and is the current Director of CSAIL's CyberSecurity@CSAIL initiative. He has served twice as a program manager at DARPA. From 1994 to 1997, he served as Chief Scientist of the Information Technology Office and led the Information Security Initiative; from 2010 to 2013, he served as a Program Manager in TCTO and then I2O, leading the CRASH and MRC programs. While at DARPA, he initiated and led the Evolutionary Design of Complex Software program as well as the Information Survivability program. While at MIT, he has participated in several Cyber Security research projects such as DARPA's Intrusion Tolerant Systems program, the OASIS program, the Self-Regenerative Systems and IARPA's NICECAP program.

Dr. Shrobe was selected Fellow of the American Association for the Advancement of Science (AAAS) in November 2016.