Model-Based Security Analysis of a Water Treatment System

Eunsuk Kang
Massachusetts Institute of Technology
eskang@csail.mit.edu

Sridhar Adepu
Singapore University of Technology and Design
sridhar_adepu@sutd.edu.sg

Daniel Jackson
Massachusetts Institute of Technology
dnj@csail.mit.edu

Aditya P. Mathur
Singapore University of Technology and Design
aditya_mathur@sutd.edu.sg

ABSTRACT

An approach to analyzing the security of a cyber-physical system (CPS) is proposed, where the behavior of a physical plant and its controller are captured in approximate models, and their interaction is rigorously checked to discover potential attacks that involve a varying number of compromised sensors and actuators. As a preliminary study, this approach has been applied to a fully functional water treatment testbed constructed at the Singapore University of Technology and Design. The analysis revealed previously unknown attacks that were confirmed to pose serious threats to the safety of the testbed, and suggests a number of research challenges and opportunities for applying a similar type of formal analysis to cyber-physical security.

Keywords: Cyber-physical system, security, model checking, attack generation.

1. INTRODUCTION

Cyber-physical systems (CPSs) are the next frontier for security engineering, as malicious actors will seek opportunities to wreak havoc in our civic infrastructure for water, electricity, transportation, etc. Most research in CPS focuses on their distinguishing qualities. For example, the physical components exhibit continuous behaviors, which are traditionally modeled by engineers using differential equations, and are thus hard to integrate with the discrete methods familiar to computer scientists (such as static analysis and model checking).

Clearly, addressing such qualities will be essential to obtain full analysis. Nevertheless, we believe that much can be achieved by taking a simpler approach, in which conventional discrete methods are applied directly, and continuous behaviors are approximated by discrete transitions. Our aim is to develop simple but effective methods that can be readily applied in practice, and which leverage computation to find security vulnerabilities in such systems.

Figure 1: Secure Water Treament (SWaT) testbed

To explore this idea, we have been experimenting with an initial analysis approach on a particular CPS. This system is a water treatment plant constructed as a testbed at the Singapore University of Technology and Design (shown in Figure 1). It is a complete, working plant [1] with all the elements of a full-blown industrial plant; the only key differences are that an industrial plant would have multiple paths for increased capacity, and larger versions of the individual components.

The analysis approach is a form of parameterized model checking. We extract a model of the software from the code; currently this is done manually, but we believe it would be straightforward to automate it. We then construct a model of the physical plant, as accessed by the software through the sensors and actuators. This model describes, for example, the physical properties of a tank or a UV filtration unit, but in a simple, discrete way. The values of each physical variable are partitioned into ranges and a proposition associated with each variable being within a given range. A safety condition (for example, that a tank does not overflow) is specified, along with an attack model (for example, that an attacker can inject fake sensor readings or send bad commands to actuators).

A model checker is then used to find attack scenarios in

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

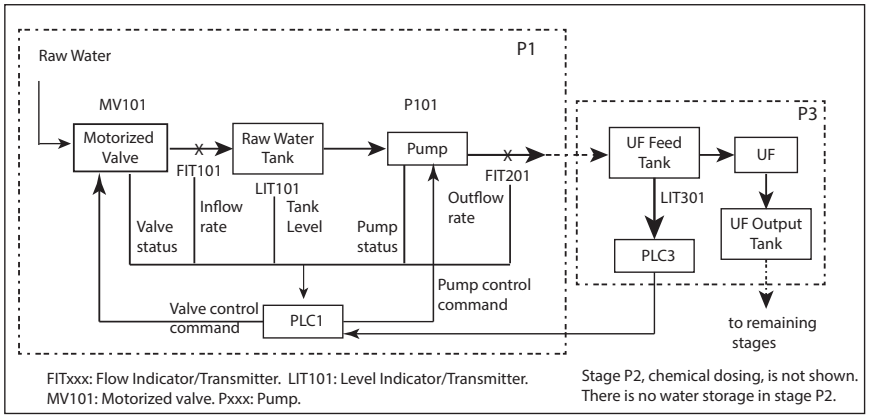

Figure 2: Portion of the SWaT testbed studied in this paper.

which the safety condition is violated on the assumption that some limited number of sensors or actuators is compromised. The resulting counterexamples are then examined. As the compromise number rises, more attacks generally become possible; the most worrying cases are those vulnerabilities that require only very few compromises.

This paper describes a case study in which the above approach was applied. The analysis revealed some serious vulnerabilities that had not been known previously to the engineers who maintain the testbed system nor to the researchers who have been applying various kinds of simulation and analysis to it. These vulnerabilities were validated for feasibility by executing them on the actual system, and were thus found to be real concerns.

We believe that this case study provides some evidence that existing methods when appropriately applied can have real impact in this domain. The model was written in the Alloy language, which was chosen for ease of expression; since the model is finite state, the analysis is actually decidable and could be encoded in the input language of many different analysis tools. On the other hand, the case study highlights a variety of opportunities for improvement, suggesting some interesting research challenges (such as automatic derivation of abstractions).

The reminder of the paper is organized as follows. Section 2 provides the system overview and threat model, and Section 3 presents the proposed analysis approach. Section 4 describes a formal model of the testbed in the Alloy modeling language. Sections 5 and 6 present the analysis and validation of potential security issues on the SWaT system, respectively. Section 7 discusses the benefits and limitations of our approach, and Section 8 presents the related work. The paper concludes with future work in Section 9.

2. SYSTEM OVERVIEW

Secure Water Treatment (SWaT [1]) is a testbed system located at the iTrust Center at the Singapore University of Technology and Design. The testbed is a scaled-down but fully functional version of a modern water treatment plant found in cities, and is used to investigate responses to cyber attacks and experiment with novel defense mechanisms.

The treatment process in SWaT consists of six different stages, starting with raw water and undergoing various chemical treatments to output filtered water in the last stage. For our preliminary experiment in this paper, we focused on the first three stages, which are outlined in Figure 2. In each stage, the system employs various *sensors* and *actuators* that monitor and manipulate the state of a *physical process* (e.g., a water tank). A *programmable logic controller* (PLC) sits between these sensors and actuators, and has an ability to activate or deactivate an actuator depending on the value received from its sensors. For instance, if the value from the sensor LIT101 indicates that the water level in the Raw Water Tank is above a desirable threshold, PLC1 sends a command to deactivate the valve MV101, stopping the flow of water into the tank.

Sensor values and actuator commands are communicated to and from a PLC via network links. PLCs are located in a separate location from the physical processes, and so these communication links are established through a wireless network. This network is password-protected, but due to the limited computational resources on sensors and actuators, data packets are unencrypted; as a result, the network may be vulnerable to eavesdropping and packet injection attacks.

The system also contains one or more *monitors* that are used to ensure that the states of the physical processes remain within an acceptable *operational boundary* (e.g., the level of a water tank does not overflow). If a monitor detects that a physical process has crossed this boundary, it will trigger an *alarm* and alert the human operator.

Threat Model.

We assume that the attacker has an ability to compromise a communication link between a PLC and a sensor or an actuator. When a link to a link is compromised, the attacker may inject an arbitrary packet to represent the sensor value, so the value read by PLC may differ from the actual state of the associated physical process. If an actuator link is compromised, the attacker may inject an arbitrary command into this link, gaining an ability to activate or deactivate the actuator independently of the PLC; in addition, the state of the actuator as detected by the PLC may be different from its actual state.

The system is assumed to be designed with a reasonable amount of physical protection; that is, the attacker cannot arbitrarily tamper with physical processes in the system.

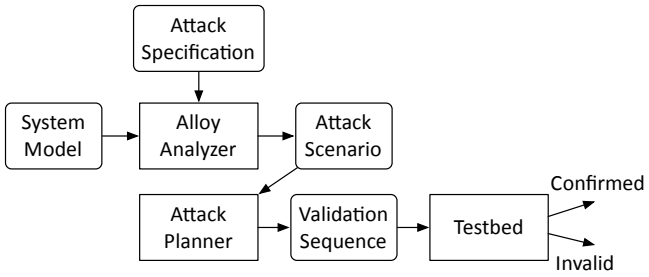

Figure 3: Overview of the proposed approach.

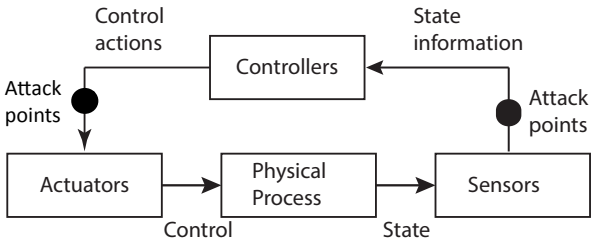

Figure 4: Relationship between major types of components in a system.

In addition, PLCs are also considered trusted, in that they always send an appropriate command to an actuator given an input sensor value (i.e., the PLC code cannot be modified by the attacker).

3. APPROACH

Figure 3 shows an overview of our approach to the security analysis of a CPS. The engineer begins by building a model of the system, and providing an *attack specification* that determines the overall capabilities of the attacker. Given these two inputs, an analysis tool is used to automatically generate an *attack scenario* that describes how the system may end up in an unsafe state. The attack planner then converts this scenario into a *validation sequence*, which is a series of control inputs that is intended to simulate the effects of the generated attack; currently, this step is performed manually by an engineer of the system. The validation sequence is then performed on the actual testbed, confirming that the attack is indeed feasible, or that it describes an invalid behavior of the system; in the latter case, the engineer may refine the system model in order to rule out the spurious scenario. The entire process may be carried out repeatedly until the analyzer fails to detect any further attacks on the system.

System Model.

In our approach, a system is modeled as consisting of four types of components, as shown in Figure 4: Physical processes, sensors, actuators, and controllers. A *physical process* represents a mechanical or physical entity that the system is designed to control (e.g., water tank). Each physical process is connected to a *sensor*, which periodically performs a reading of its state, and one or more *actuators*, which directly manipulate the physical process to alter its state. A *controller* performs actions to activate or deactivate an actuator, depending on the information received from its sensors.

Analysis.

A system is said to be in an *unsafe* state when one of its physical processes exceeds its operational boundary. The goal of our analysis is to check whether there exists a system trace that results in an unsafe state, given that some subset of components have been compromised by the attacker; this subset is called an *attack configuration*.

A distinguished feature of our analysis is in the parametricity of attack configurations. Instead of requiring the user to explicitly specify which components are compromised, our analysis will automatically explore all possible attack configurations and enumerate ones that lead the system into an unsafe state. If desired, the user can also tune possible attack configurations by providing an optional specification that limits the number of components in a configuration, or requiring a particular component to be included in or excluded from a configuration.

4. MODEL OF THE SWAT TESTBED

We constructed a model of SWaT in *Alloy*, a modeling language based on first-order relational logic [2, 3]. We chose Alloy as our modeling notation for two reasons: (1) its expressiveness, which allowed us to capture both structural (architectural connections between components) and behavioral (state transition rules) aspects of the system within a single model, and (2) its backend tool, the Alloy Analyzer, which supports an automated simulation and assertion checking of an input model.

Figure 5 shows a fragment of the SWaT model in Alloy. The Alloy keyword `sig` introduces a *signature*, which defines a set of elements in the universe. A signature may contain one or more *fields*, each introducing a relation that maps the elements of the signature to the field expression; for example, field `process` in `PhysicalProcess` is a binary relation that maps each `Sensor` object to the physical process that it monitors (line 5). The keyword `extends` creates a subtyping relationship between two signatures; an `abstract` signature has no elements except those belonging to its extensions, and `one sig` introduces a signature that contains only one element. The first part of the model contains a generic definition of system components and their relationships (lines 1-11), and the rest describes the SWaT-specific components.

The dynamic aspect of the system is modeled using a standard Alloy idiom in which a system trace is represented as a sequence of global ticks, and each dynamic entity is associated with exactly one state at each tick [2]. To do this, we introduce a set of totally ordered elements called `Tick`, and append it to the last column of relations that are considered mutable; for example, the field `state` in `PhysicalProcess` (line 4) keeps track of states of the physical process at different points in the system execution.

At each tick, a PLC reads state information from every one of its sensors and actuators (line 10). The accuracy of the readings, however, depends on whether or not the respective sensor or actuator has been compromised by the attacker. To model this attacker behavior, we first designate some subset of sensors and actuators to be compromised (line 11). Then, we introduce an Alloy fact– a constraint

```
1  open util/ordering[Tick]  // Create a total ordering of global ticks
2  sig Tick {}
3  /** Generic part of a system model **/
4  abstract sig PhysicalProcess { state : PhysicalState one -> Tick }
5  abstract sig Sensor { process : PhysicalProcess }
6  abstract sig Actuator { state : ActuatorState one -> Tick }
7  abstract sig PLC {
8    sensors : set Sensor, actuators : set Actuator, processes : set PhysicalProcess,
9    // At every tick, PLC reads state information from its sensors and actuators
10   reads : (Sensor + Actuator) -> (State one -> Tick) }
11 sig Compromised in Sensor + Actuator {} // Some of the sensors and actuators are compromised
12
13 /** SWaT-specific part of the model **/
14 one sig On, Off extends ActuatorState {}
15 abstract sig LevelState extends PhysicalState {}  // State of water level in a tank
16 one sig UF, LL, L2, L1, L, H, H1, H2, HH, OF extends LevelState {} // UF = underflow, OF = overflow
17 // SWaT components for Stage P1
18 one sig RawWaterTank extends PhysicalProcess {}
19 one sig LIT101 extends Sensor {}{ process = RawWaterTank }
20 one sig P101, MV101 extends Actuator {}
21 one sig PLC1 extends PLC {}{
22   sensors = LIT101 + FIT101 and actuators = P101 + MV101 and processes = RawWaterTank }
23 // Describes how the state of the water tank changes given the status of actuators
24 pred transition_RawWaterTank[t, t': Tick] {
25   let pre = state.t, post = state.t' {
26     MV101.pre = On and P101.pre = On implies this.post = this.pre
27     MV101.pre = On and P101.pre = Off implies this.post = increaseLevel[this.pre]
28     MV101.pre = Off and P101.pre = On implies this.post = decreaseLevel[this.pre]
29     MV101.pre = Off and P101.pre = Off implies this.post = this.pre } }
30 // Describes how the state of MV101 changes given PLC1 sensor readings
31 pred transition_MV101[t, t': Tick] {
32   let plc1readings = PLC1.reads.t, pre = state.t, post = state.t' {
33     plc1reading[LIT101] = H implies MV101.post = Off
34     plc1readings[LIT101] = L implies MV101.post = On
35     else MV101.post = MV101.pre } }
36 fact CompromisedBehavior {
37   // Sensor reading matches the actual state of the physical process if sensor is not compromised
38   all s : Sensor, p : PLC, t : Tick | s not in Compromised implies (p.reads.t)[s] = s.process.(state.t)
39   // Reading from the actuator matches its actual state if it is not compromised
40   all a : Actuator, p : PLC, t : Tick | a not in Compromised implies (p.reads.t)[a] = a.state.t
41   // MV101 follows expected state changes if it is not compromised
42   all t : Tick - last | let t' = t.next |  MV101 not in Compromised implies transition_MV101[t, t'] }
43
44 /** Safety property **/
45 // True iff system detects overflow or unexpected level changes in the raw water tank
46 pred alarm_tankMonitor[t : Tick] {
47   let t0 = t.prev, readings = (PLC1.reads.t0), level = readings[LIT101], level' = (PLC1.reads.t)[LIT101] |
48   (level' = HH) or  // reading indicates that water is about to overflow
49   (readings[MV101] = On and readings[P101] = On and level' != level) or
50   (readings[MV101] = On and readings[P101] = Off and level' != increaseLevel[level]) or
51   (readings[MV101] = Off and readings[P101] = On and level' != decreaseLevel[level]) or
52   (readings[MV101] = Off and readings[P101] = Off and level' != level) }
53 // True iff water overflows and no alarm is triggered
54 pred unsafe[t : Tick] { RawWaterTank.state.t = OF and no alarm_tankMonitor[t.prevs] }
55 // System should never reach an unsafe state
56 assert SystemIsSafe { no t : Tick | unsafe[t] }
```

Figure 5: A fragment of the SWaT model in Alloy.

that holds for every satisfying instance of the model–to state that a sensor or actuator reading reflects the actual state if it has not been compromised (lines 37-40); in other words, once compromised, the device may output an arbitrary state value (as selected by the attacker).

The state of the raw water tank in Stage 1 is modeled as being in one of 10 different levels of water (line 16). The state of the tank evolves over time depending on the status of valve MV101 (which, when turned on, allows water to flow into the tank) and P101 (which is used to pump water out of the tank). This behavior is captured by the transition predicate (line 24): When both MV101 and P101 are turned on or off at the same time, the water level in the tank remains stable; if only the valve is in the active status, it will cause the water level to rise an increment in the next tick (e.g., from `H` to `H1`); similarly, if only the pump is active, the water level in the tank will decrease by a set amount (e.g, from `H1` to `H`).

During every tick, a PLC may alter the state of its actuators, given the latest readings from its sensors. For example, the transition predicate for MV101 describes how the valve may be closed or opened by PLC1 depending on the current reading from level sensor LIT101 (line 31). However, if the actuator has been compromised, the attacker may inject an arbitrary command to change the state of the actuator, meaning it will no longer follow the normal transition rules; this is described by the last constraint in the fact `CompromisedBehavior` (line 42).

Transition rules that describe the behavior of a PLC (i.e., how it controls the state of an actuator given an input sensor value) were directly transliterated from the actual PLC code running on the system. A more challenging task was

	LIT101		MV101	
Tick	State	Reading	State	Reading
0	L	L	Off	Off
1	L	L	On	Off
2	H	L	On	Off
3	H1	L	On	Off
4	H2	L	On	Off
5	HH	L	On	Off
6	OF	L	On	Off

Figure 6: A potential attack scenario on the SWaT.

capturing the dynamics of the physical processes; that is, how the state of a process evolves over time depending on the status of its actuators. For this task, we consulted the engineer and operator of the system, and constructed an approximate, discrete model that describes how the water level of the tank rises or falls depending on whether MV101 pump P101 is activated.

5. ANALYSIS

A safety property can be stated by specifying what it means for a system to be in an unsafe state, and then asserting that the system never reaches that state. In Figure 5, the SWaT system is considered to be in an unsafe state (line 54) if the state of the raw water tank reaches an overflow at a certain point (tick `t`), and the system fails to trigger an alarm during the previous executions steps (represented by `t.prevs`). The conditions for raising an alarm is defined in terms of the sensor and actuator readings by PLC1 (line 46). In particular, PLC1 will raise an alarm if the LIT101 reading indicates that the water is about to overflow (line 48), or the change in the water level does not follow the expected pattern (e.g., it does not rise when the valve is on and the pump is off, as described on line 50).

The Alloy Analyzer is a general-purpose constraint solver; given a set of relations and constraints over them, it attempts to find a satisfying assignment of tuples to those relations[1]. Assertion checking, for example, can be phrased as the problem of satisfying the conjunction of the constraints describing the system behavior and the negation of a given assertion ($S \wedge \neg P$); a satisfying instance, if exists, would correspond to a counterexample to the assertion.

The constraint-based analysis can be further exploited to not only generate an unsafe trace, but also to find a particular subset of sensors and actuators that would need to be compromised in order for that trace to be a feasible behavior of the system (i.e., an attack configuration). More precisely, the analysis problem can be phrased as finding a satisfying instance to $S \wedge C \wedge \neg P$, where C is a partial specification of attack configurations, written as a constraint over the set `Compromised` (line 12, Figure 5).

By default, when no attack specification is provided (i.e., $C =$ `true`), the analyzer will explore all possible attack configurations. In general, however, the system engineer may make certain assumptions about the extent of an attack, and design an appropriate defense mechanism based on them (after all, if every component in the system is compromised, then no defense mechanism is likely to be sufficient). The specification can be used to state such an assumption and tune the analysis accordingly by, for example, limiting the number of compromised components (#`Compromised` <= 2) or requiring a particular component to be included in or excluded from this set (`MV101` $\in$ `Compromised` or `P201` $\notin$ `Compromised`).

[1]The analyzer itself relies on a first-order model finder called Kodkod [4], which, in turn, uses an off-the-shelf SAT solver to generate an instance.

Compromised	No. Ticks	Duration (min)
LIT101	10	42
MV101, LIT101	7	35
P101, LIT101	11	39

Figure 7: Attack scenarios validated on the SWaT.

For our preliminary analysis of the SWaT model, we started by generating single-point attacks (#`Compromised` = 1), and then moved onto finding attacks that involve compromising multiple components (#`Compromised` = 2). The analyzer discovered 4 distinct attack scenarios (2 single-point, and 2 multi-points), describing how the attacker may be able to cause the tank to overflow without triggering the alarm in the system. On average, the analyzer took around 6 seconds to generate an attack scenario.

One of the potential attack scenarios, involving MV101 and LIT101 as compromised components, is shown in Figure 6. It begins with the water level being low (`L`), and both MV101 and P101 being turned off. In the next step, the attacker injects a command to activate MV101, and at the same time, prevents PLC1 from being aware that the valve is active by displaying a false reading of `Off`. In the following steps, the actual level of water in the tank continues to rise, but the attacker keeps these state changes unknown to PLC1 by continually injecting a false reading of `L` into LIT101. PLC1 fails to trigger an alarm during this execution, since based on its readings, it observes that the valve has been deactivated (thus, no flow into the tank), and the water level remains stable as expected. Eventually, the tank overflows, leading to a violation of the safety property.

6. VALIDATION

An attack scenario generated by the Alloy Analyzer is a sequence of component states and readings at the abstract *modeling* level, as shown in Figure 6. The process of validating an attack then involves (1) converting each abstract concept (e.g, `L` for the water level) into an appropriate *concrete* value, and (2) replaying the concretized version of the attack scenario on the actual system, eventually resulting in an unsafe state as defined in the model. If step (2) is not possible, then the attack scenario is considered *infeasible*, likely hinting at an inaccuracy in the system model.

In Step (1), converting an abstract state of a component into its actual value involves devising a *concretization relation* that relates values of abstract and concrete domains [5]. In our case, coming up with such a relation was straightforward, because the testbed engineers had already been working with their own discretization of continuous physical domains. For example, in the SWaT operational manual, water levels are partitioned into a finite number of intervals and assigned discrete labels (`L` for 500mm to 800mm); these are then used to refer to different stages of the water tank throughout the manual, instead of the actual level values.

For our modeling purpose, we adapted the same discretization used by the engineers.

To support research and experiments on the SWaT, the engineers have developed a flexible scripting tool (called SWaTAssault [6]) that allows one to programmatically override and manipulate control signals between PLCs, sensors, and actuators. For Step (2) of the validation process, we used this tool to simulate the behavior of a compromised sensor or actuator. The most challenging part of this step was determining the duration of elapsed time for each pair of adjacent ticks in the model; this was done manually, by altering the state of an actuator (e.g., turning on a valve) and observing the changes in a physical plant (e.g., gradual rise of water level) until it reaches the state that is depicted in the attack scenario. This meant that a human engineer had to be physically present at the scene to observe the system during the validation process. We believe that a worthwhile research problem would be to automate (partially) this part of the validation, possibly by augmenting the concretization step with information about the expected duration of state changes of the plant.

Out of the 4 attack scenarios generated by the Alloy Analyzer, we were able to validate 3 of them. Figure 7 shows the minimum number of ticks required to represent an attack in the model, and the total duration elapsed to successfully validate the attack; for instance, the attack involving MV101 and LIT101 took around 35 minutes from the initial tank state (L) to overflow. One of these attacks (involving P101 and LIT101) had been manually discovered by the SWaT engineers prior to our analysis; the other two were previously unknown attacks.

The 4th scenario, a single-point attack involving LIT101, was found to be infeasible, because the model used for analysis did not, at the time, include information about the water tank monitor (lines 46-51 in Figure 5). As a result, during the validation process, the PLC detected an unusual change in the water level, raising an alarm and preventing the system from ending up in an unsafe state. Based on this outcome, we refined the model by adding the behavior of the monitor, after which an analysis no longer produced the same attack scenario.

7. DISCUSSION

Our preliminary study has shown that the proposed approach is promising in helping engineers automatically discover and explore attacks on a CPS. However, based on our experience, we believe that there are a number of research challenges that merit further investigations in order to establish an analysis framework that is truly effective and applicable to a wider range of CPSs.

One current limitation of our approach is the approximate nature of the model of physical processes, whose behaviors are often continuous in nature. To our surprise, however, we found that a discrete model can still be quite effective in discovering realistic attacks. The SWaT engineers found our model intuitive to understand, as they themselves already work with a discrete abstraction of the system (this is, in part, because the continuous dynamics of water flow and its chemical properties is considerably complex; in fact, the engineers are yet to come up with an accurate nonlinear model of the system dynamics).

Even then, choosing an appropriate discrete abstraction for continuous behavior is still largely an ad-hoc, manual process. If the chosen abstraction is too coarse, then the analysis may produce infeasible attack scenarios; on the other hand, if the abstraction is unsound, then it may fail to detect actual attacks on the system. Various techniques for constructing suitable abstractions have been successfully developed for program verification, but it is not clear whether the same types of techniques are applicable to a CPS, where coming up with a "ground truth" model of the continuous behavior itself is a challenging task.

Another approach would be to construct a hybrid model, where some parts of the system are described in a continuous manner, while the rest remains discrete. While this approach has potential to provide a more accurate analysis, it has its own set of challenges, especially in (1) scaling the computationally intensive analysis of continuous dynamics, and (2) accurately capturing the transition from a continuous to a discrete part of the system (and vice-versa). Hybrid system analysis is an active area of research [7, 8, 9], and as techniques in this field continue to improve, we believe that it will play a critical role in cyber-physical security.

A different aspect of the proposed approach that deserves further attention is the automation of the validation process. As discussed in Section 6, validating an attack scenario requires a considerable amount of human intervention, to ensure that the actual system behavior matches the behavior depicted in the scenario. It is conceivable that a significant part of this process may be automated, by, for example, augmenting the model with information about timing, and allowing the SWaTAssault script to use this information to drive the replay of the attack. Another area for improvement is achieving a tighter interplay between the modeling and validation steps: If an attack scenario is found to be infeasible, observations made during the validation may be leveraged to (semi-)automatically refine the model to rule out the scenario.

8. RELATED WORK

Any complex CPS is likely to contain a mix of components built by software and system engineers, but the tools that they use for modeling and analysis are significantly different. System engineers have long been constructing a mathematical model of a physical system and analyzing its properties using computer-aided tools such as MATLAB [10] and Simulink [11]. Applying the same types of tools to the security of a CPS faces challenges in two dimensions: (1) the "cyber" aspect of the system, which is more amenable to discrete analysis typically not supported by these tools, and (2) incorporating the behavior of an attacker, who may operate across multiple layers of abstraction (e.g., software architecture and network, instead of just the physical layer). Our aim in this project is to develop and experiment with an approach that provides complementary strengths to these existing methods, ultimately arriving at a design framework that can be used by both groups of engineers.

An increasing body of work exist on techniques for verification of CPSs, including ones that employ model checking or constraint solvers [12, 13, 14]. Most of these techniques focus on the challenge of scaling the state-based analysis to a hybrid model. On the other hand, relatively little work has been done on modeling malicious behavior, and allowing the engineer to explore the consequences of different assumptions on the attacker's capabilities. In this respect, our work is more closely related to traditional safety engineering

techniques such as FMEA [15] and fault tree analysis [16], where some subset of system components are assumed to be faulty, and an analysis is performed to assess their impact on the overal safety of the system. On a different note, Zheng and his colleagues [17] provide an extensive survey of tools and techniques for CPS verification, suggesting various research challenges (some of which we have also discussed in this paper).

Two authors of this paper have previously explored various attacks on the SWaT testbed [18, 19]. However, in these efforts, the attacks were discovered *manually* through an informal though-process; as the authors realized, manual exploration of all attack possibilities proved to be tedious and error-prone, thus motivating a more automated approach like the one proposed in this paper. Indeed, two of the attacks that were discovered by our analysis in this paper were previously unknown to the engineer and researchers working on SWaT.

9. CONCLUSION

This paper proposed an approach to analyzing the security of a CPS, where the behaviors of a physical plant and its controller is captured in approximate, discrete models, and their interaction is rigorously analyzed to discover attacks that involve a varying number of compromised sensors and actuators. The paper also described our experience applying this approach to a fully functional water treatment system, and suggested new research challenges and opportunities for developing tools for cyber-physical security.

The analysis of the SWaT system is an actively ongoing project. As a next step, we plan to extend our model to include the rest of the water treatment process (6 PLCs in total), and consider other types of attacks beside the ones described in this paper, such as compromising a communication link between PLCs. In addition, we have yet to touch upon other significant aspects of the system: (1) SCADA and potential vulnerabilities that arise from its connection to the web, and (2) safe distribution of water to households and other critical endpoints. We plan to investigate how our approach can be extended to model and analyze these types of issues as well.

10. REFERENCES

[1] SWaT: Secure Water Treatment Testbed, 2015. https://itrust.sutd.edu.sg/wp-content/uploads/sites/3/2015/11/Brief-Introduction-to-SWaT_181115.pdf.

[2] Alloy language and analyzer. http://alloy.mit.edu.

[3] Daniel Jackson. *Software Abstractions: logic, language, and analysis*. MIT Press, Second edition, 2012.

[4] Emina Torlak and Daniel Jackson. Kodkod: A relational model finder. In *Tools and Algorithms for the Construction and Analysis of Systems TACAS Portugal, 2007.*, pages 632–647, 2007.

[5] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

[6] David. Urbina, Jairo. Giraldo, Nils Ole. Tippenhauer, and Alvaro Cardenas. Attacking fieldbus communications in ics: Applications to the swat testbed. In *Proc. Singapore Cyber-Security Conference (SG-CRC)*, pages 75 – 89, 2016.

[7] André Platzer. *Logical analysis of hybrid systems: proving theorems for complex dynamics*. Springer Science & Business Media, 2010.

[8] Sicun Gao, Soonho Kong, and Edmund M Clarke. dreal: An smt solver for nonlinear theories over the reals. In *Automated Deduction–CADE-24*, pages 208–214. Springer, 2013.

[9] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. Smt-based verification of hybrid systems. In *AAAI*, 2012.

[10] Mathworks. Matlab. http://www.mathworks.com/products/matlab/.

[11] Mathworks. Simulink. http://www.mathworks.com/products/simulink/.

[12] Ravi Akella and Bruce M McMillin. Model-checking bndc properties in cyber-physical systems. In *Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International*, volume 1, pages 660–663. IEEE, 2009.

[13] Chih-Hong Cheng, Natarajan Shankar, Harald Ruess, and Saddek Bensalem. Efsmt: A logical framework for cyber-physical systems. *arXiv preprint arXiv:1306.3456*, 2013.

[14] Edmund M Clarke and Paolo Zuliani. Statistical model checking for cyber-physical systems. In *Automated Technology for Verification and Analysis*, pages 1–12. Springer, 2011.

[15] Dean H Stamatis. *Failure mode and effect analysis: FMEA from theory to execution*. ASQ Quality Press, 2003.

[16] William E Vesely, Francine F Goldberg, Norman H Roberts, and David F Haasl. Fault tree handbook. Technical report, DTIC Document, 1981.

[17] X. Zheng, C. Julien, M. Kim, and S. Khurshid. Perceptions on the state of the art in verification and validation in cyber-physical systems. *Systems Journal, IEEE*, PP(99):1–14, 2015.

[18] Sridhar Adepu, Aditya Mathur, Jagadeesh Gunda, and Sasa Djokic. An agent-based framework for simulating and analysing attacks on cyber physical systems. In *Algorithms and Architectures for Parallel Processing*, pages 785–798. Springer, 2015.

[19] S. Adepu and A. Mathur. An investigation into the response of a water treatment system to cyber attacks. In *Proceedings of the 17th IEEE High Assurance Systems Engineering Symposium, Orlando*, January 2016.