July 19, 2020
Virtual Event, USA

Association for Computing Machinery

Advancing Computing as a Science & Profession

TAV-CPS/IoT ’20

Proceedings of the 4th ACM SIGSOFT International Workshop on

Testing, Analysis, and Verification of Cyber-Physical Systems and Internet of Things

Edited by:
Yan Cai and Tingting Yu

Sponsored by:
ACM SIGSOFT

Co-located with:
ISSTA ’20

Association for Computing Machinery, Inc.
1601 Broadway, 10th Floor
New York, NY 10019-7434
USA

ACM ISBN: 978-1-4503-8032-4

Additional copies may be ordered prepaid from:

	Phone: 1-800-342-6626
ACM	(USA and Canada)
General Post Office	+1-212-626-0500
P.O. Box 30777	(All other countries)
New York, NY 10087-0777	Fax: +1-212-944-1318
	E-mail: acmhelp@acm.org

Cover photo “Daytime view of Downtown L.A., as seen from Hollywood” by Thomas Pintaric / Licensed under CC BY-SA 3.0 / Cropped from original at https://commons.wikimedia.org/wiki/File:LosAngeles05.jpg

Production: Conference Publishing Consulting
D-94034 Passau, Germany, info@conference-publishing.com

Welcome from the Organizers

Welcome to the Workshop on Testing, Analysis, and Verification of Cyber-Physical Systems and Internet of Things (TAV-CPS/IoT 2020) held in USA jointly with the ACM Sigsoft International Conference on Software Testing and Analysis ISSTA 2020. The workshop follows the successful editions co-located with ISSTA in 2017, ECOOP/ISSTA in 2018, and ISSTA 2019.

TAV-CPS/IoT aims to provide a forum for researchers and practitioners to discuss and exchange ideas on problems and challenges in testing the Cyber-physical Systems and the Internet of Things from different areas (software, system, testing and analysis), to identify promising research directions, and to ultimately create a special interest community.

TAV-CPS/IoT privileges discussion over the program built around invited presentations and Panels. This year, TAV-CPS/IoT is to be held online, by following virtual conference arrangement of ISSTA 2020. We are pleased to have one keynote and three talks, followed by one panel. These are all on the topics of testing and verifying CPS/IoP.

We are grateful to the speakers and the PC members who contribute to this workshop greatly.

Tingting Yu and Yan Cai
Organizing Committee Chairs

Committee Listings

Contents

ObjSim: Efficient Testing of Cyber-Physical Systems

Jun Sun
junsun@smu.edu.sg
Singapore Management University
Singapore

Zijiang Yang
yang@guardstrike.com
GuardStrike Inc.
China

ABSTRACT

Cyber-physical systems (CPSs) play a critical role in automating public infrastructure and thus attract wide range of attacks. Assessing the effectiveness of defense mechanisms is challenging as realistic sets of attacks to test them against are not always available. In this short paper, we briefly describe smart fuzzing, an automated, machine learning guided technique for systematically producing test suites of CPS network attacks. Our approach uses predictive machine learning models and meta-heuristic search algorithms to guide the fuzzing of actuators so as to drive the CPS into different unsafe physical states. The approach has been proven effective on two real-world CPS testbeds.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

cyber-physical system, network, fuzzing, machine learning, testing

ACM Reference Format:
Jun Sun and Zijiang Yang. 2020. ObjSim: Efficient Testing of Cyber-Physical Systems. In *Proceedings of the 4th ACM SIGSOFT International Workshop on Testing, Analysis, and Verification of Cyber-Physical Systems and Internet of Things (TAV-CPS/IoT '20), July 19, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3402842.3407158

1 INTRODUCTION

Cyber-physical systems (CPSs) are characterized by computational elements and physical processes that are deeply intertwined, each potentially involving different spatial and temporal scales, modalities, and interactions. We define CPSs as systems in which algorithmic control and physical processes are tightly integrated. Concretely, we assume that they consist of computational elements such as programmable logic controllers (PLCs), distributed over a network, and interacting with their processes via sensors and actuators. The operation of a CPS is controlled by its PLCs, which receive readings from sensors that observe the physical state, and then compute appropriate commands to send along the network to the relevant actuators. In our work we assume that the sensors read continuous data and that the states of the actuators are discrete.

CPSs are commonly used to automate aspects of critical civil infrastructure, such as water treatment or the management of electricity demand [7]. Given the potential to cause massive disruption, such systems have become prime targets for cyber attackers, with a number of successful cases reported in recent years [4, 6]. However, CPSs are very difficult to reason about: while individual control components (e.g. PLC programs) may be simple in isolation, reasoning about the behaviour of the whole system can only be done with consideration of how its physical processes evolve and interact. This often requires considerable domain-specific expertise beyond the knowledge of a typical computer scientist, which is one of our principal motivations for achieving full automation.

2 OUR APPROACH

Fuzzing, which plays a key role in our solution, is in general an automated testing technique that attempts to identify potential crashes or assertion violations by generating diverse and unexpected inputs for a given system [8]. Most well-known tools perform fuzzing on programs, but in the context of CPSs, we consider fuzzing at the network level. Furthermore, the goal of our fuzzing differs in that we are trying to drive physical sensors out of their safe ranges, using an underlying method that is ML-guided.

Our technique uses predictive machine learning models and meta-heuristic search to intelligently fuzz actuator commands, and systematically drive the system into different categories of unsafe physical states. Smart fuzzing consists of two broad steps. First, we learn a model of the CPS by training ML algorithms on physical data logs that characterize its normal behaviour. The learnt model can be used to predict how the current physical state will evolve with respect to different actuator configurations. Second, we fuzz the actuators over the network to find attack sequences that drive the system into a targeted unsafe state. This fuzzing is guided by the learnt model: potential manipulations of the actuators are searched for, and then the model predicts which of them would drive the CPS closest to the unsafe state.

Our design for smart fuzzing was driven by four key requirements. First, that it should be general, in the sense that it can be implemented for different CPSs and a variety of sensors and actuators. Second, that the approach should be comprehensive, in that the suites of attacks it constructs should systematically cover different categories of sensed physical properties, rather than just a select few. Third, that it should be efficient, with each attack achieving its goal quickly, posing additional challenge for countermeasures. Finally, that it should be practically useful, in that it is straightforward to implement for real CPSs without any formal specification or specific technical expertise, and that the 'test suites' of attacks are of comparable quality to expert-crafted benchmarks, thus a reasonable basis for assessing attack defense mechanisms.

Our approach for automatically finding network attacks on CPSs consists of two broad steps in turn: learning and fuzzing. In the first step, we learn a model of the CPS that can predict the effects of actuator configurations on the physical state. The model takes as input the current readings of all sensors and a proposed configuration of the actuators, returning as output a prediction of the sensor readings that would result from adopting that configuration for a fixed time interval. The idea is that this model can later be used to analyze different potential actuator configurations, and help inform which of them is likely to drive the system closer to a targeted unsafe state. To learn this model, we extract a time series of sensor and actuator data from the system logs and train a suitable machine learning algorithm.

The second step of our approach searches for commands to fuzz the actuators with that will drive the CPS into an unsafe physical state. To find the right commands, our approach applies a search algorithm over the space of actuator configurations, returning the configuration that is predicted by the model (of the first step) to drive the CPS the closest to an unsafe state. We explore different search algorithms for this task, including random, but also meta-heuristic (e.g. genetic algorithms) given that the state space of actuators can grow quite large (e.g. 2^{26} possible configurations in SWaT). We use fitness functions to evaluate predicted sensor states with respect to the attack goal.

3 EVALUATION

To evaluate our approach against these requirements, we implemented it for two CPS testbeds. First, the Secure Water Treatment (SWaT) testbed [1], a fully operational water treatment plant consisting of 42 sensors and actuators, able to produce five gallons of drinking water per minute. Second, the Water Distribution (WADI) testbed [3], a scaled-down version of a typical water distribution network for a city, built for investigating attacks on consumer water supplies with respect to patterns of peak and off-peak demand. The designs of these testbeds were based on real-world industrial purification plants and distribution networks, and thus reflect many of their complexities. We found that smart fuzzing could automatically identify suites of attacks that drove these CPSs into 27 different unsafe states involving water flow, pressure, tank levels, and consumer supply. Furthermore, it covered six unsafe states beyond those in an established expert-crafted benchmark [5]. Finally, we evaluated the utility of smart fuzzing for testing attack defense mechanisms by launching it with SWaT's invariant-based monitoring system enabled [2]. Our approach was able to identify two attacks that evaded detection by its physical invariant checks, highlighting a potential weakness that could be exploited by attackers with the capabilities to bypass its other conditions.

Our evaluation addresses five research questions based on our original design requirements for smart fuzzing (Section I): RQ1 (Efficiency): How quickly is smart fuzzing able to find a targeted attack? RQ2 (Comprehensiveness): How many unsafe states can the attacks of smart fuzzing cover? RQ3 (Setup): Which combinations of model and search algorithm are most effective? RQ4 (Comparisons): How do the attacks compare against those of other approaches or those in benchmarks? RQ5 (Utility): Are the discovered attacks useful for testing CPS attack detection mechanisms? RQs 1-2 consider whether smart fuzzing achieves its principal goal of finding network attacks. We assess this from two different angles: first, in terms of how quickly it is able to drive the CPS into a particular unsafe state; and second, in terms of how many different unsafe states the attacks can cover. RQ 3 considers how different setups of smart fuzzing (i.e. different models or search algorithms) impact its ability to find attacks. RQ 4 compares the effectiveness of smart fuzzing against other approaches: first, the baseline of randomly mutating actuator states without reference to a model of the system; and second, an established, manually constructed benchmark of attacks [5]. Finally, RQ 5 investigates whether the attacks found by smart fuzzing are useful for testing existing cyber-security defense mechanisms. Our empirical study shows promising results on all five research questions.

We remark on some threats to the validity of our evaluation. First, our approach was implemented for CPS testbeds: while they are real, fully operational plants based on the designs of industrial ones, they are still smaller, and our results may therefore not scale-up (this is difficult to test due to the confidentiality surrounding plants in cities). Second, the initial states of the testbeds were not controlled, other than to be within their normal ranges, meaning that our performance results may vary slightly. Finally, for testing CPS attack detection mechanisms, we only studied an invariant based solution, meaning that our conclusions may not hold for other types of defenses.

4 CONCLUSION

Active fuzzing, a black-box approach for automatically building test suites of packet-level CPS network attacks, overcomes the enormous search spaces and resource costs of such systems. Key to achieving this was our use of online active learning, which reduced the amount of training data needed by sampling examples that were estimated to maximally improve the model.

REFERENCES

[1] [n.d.]. iTrust Labs_SWaT - iTrust. https://itrust.sutd.edu.sg/itrust-labs-home/itrust-labs_swat/. (Accessed on 09/26/2019).

[2] Sridhar Adepu and Aditya Mathur. 2018. Distributed attack detection in a water treatment plant: Method and case study. *IEEE Transactions on Dependable and Secure Computing* (2018).

[3] Chuadhry Mujeeb Ahmed, Venkata Reddy Palleti, and Aditya P Mathur. 2017. WADI: a water distribution testbed for research in the design of secure cyber physical systems. In *Proceedings of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks*. 25–28.

[4] ICS-CERT Alert. 2016. Cyber-attack against ukrainian critical infrastructure. *Cybersecurity Infrastruct. Secur. Agency, Washington, DC, USA, Tech. Rep. ICS Alert (IR-ALERT-H-16-056-01)* (2016).

[5] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. 2016. A dataset to support research in the design of secure water treatment systems. In *International Conference on Critical Information Infrastructures Security*. Springer, 88–99.

[6] John Leyden. 2016. Water treatment plant hacked, chemical mix changed for tap supplies. *The Register* (2016).

[7] Ragunathan Rajkumar, Insup Lee, Lui Sha, and John Stankovic. 2010. Cyber-physical systems: the next computing revolution. In *Design automation conference*. IEEE, 731–736.

[8] Ari Takanen, Jared D Demott, Charles Miller, and Atte Kettunen. 2018. *Fuzzing for software security testing and quality assurance.* Artech House.

Formal Verification of Discrete Event Model*

Zhihao Lu
Information Engineering College
Capital Normal University
Beijing China
2252029657@qq.com

Rui Wang
Information Engineering College
Capital Normal University
Beijing China
rwang04@cnu.edu.cn

Yong Guan
Information Engineering College
Capital Normal University
Beijing China

ABSTRACT

Ptolemy is a modeling and simulation toolkit widely used in cyber physical systems. Formal verification is a very important method to guarantee the correctness of systems. However, the proposed verification approach in Ptolemy is inconvenient and only supports a few actors. In this paper, we present a model translation based approach to verify the Ptolemy Discrete-Event model and implement a plug-in tool in Ptolemy. A set of mapping rules are designed to translate the Ptolemy Discrete-Event model into a network of timed automata. A template library that translates from actors in Ptolemy Discrete-Event model is also built in advance. A case study of traffic lights control system has been successfully translated and verified. The verification results are reliable and valid. The experimental results confirm our approach is useful to verify the Ptolemy Discrete-Event model.

CCS CONCEPTS

• **Security and privacy → Formal security models; Logic and verification**

KEYWORDS

Formal verification, Ptolemy DE model, Automatic model translation, Timed automata

ACM Reference format:

Zhihao Lu, Rui Wang, and Yong Guan. 2020. Formal Verification of Discrete Event Model. In *Proceedings of the 4th ACM SIGSOFT International Workshop on Testing, Analysis, and Verification of Cyber-Physical Systems and Internet of Things (TAV-CPS/IoT'20). July 19, 2020, Virtual Event, USA. ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/3402842.3407159

*This work was supported by NSFC 61877040

1 INTRODUCTION

Ptolemy II[1] is an open-source software package for modeling and simulating concurrent, realtime and embedded systems. It facilitates the design process with a component assembly framework and a graphical user interface[2]. The provided actor library, which is extendible, allows the designers to construct models quickly. Systems in Ptolemy II focus on assembly of concurrent components which are called actors. The key principle of Ptolemy model is the use of well-defined models of computation that govern the interaction between actors. Ptolemy II supports many models of computation and discrete event(DE) is one of them.

However, Ptolemy can't verify model[3]. Now Ptolemy DE model is used widely, especially in aerospace, military, medical and autopilot. More and more practitioners choose Ptolemy DE model of computation. In these highly reliable areas, it will be dangerous if the model is not verified. In order to solve the predicament, we present an automated model translation approach for verifying Ptolemy DE model in this paper. We choose the Uppaal[4][5] tool to verify the translation model because of the advantage of Uppaal in modeling, simulation and verification. It can analyze the timing behavior of a system, and methods for checking both safety and liveness properties of time automata are well developed and intensively studied in Uppaal. The tool contains the editor, simulator and verifier, which can parse the translated model file, simulate the model to check equivalence with source model and verify properties of model. So, Uppaal is well suited to the validation of Ptolemy DE model. Mapping rules are designed to translate the Ptolemy DE model into a network of timed automata in Uppaal. A template library that translates actors in Ptolemy DE is also built in advance. The approach first parses Ptolemy DE model file and generates an XML file that can be parsed in Uppaal. We also develop a plug-in tool based on this approach and integrate it into Ptolemy environment. The main contributions of this paper are as follows:(1) we propose a new model translation based approach to verify Ptolemy DE model. A translation algorithm from Ptolemy DE model to Uppaal timed automata is defined that keeps the architecture consistency of the two models. The concrete workflow consists of loading defined mapping rules and template library, parsing Ptolemy DE model, automatically generating Uppaal model code, verifying properties and experimental implementation. (2) We develop a plug-in and integrate it into the Ptolemy tool to support the verification of DE model. The model can be translated and verified in Ptolemy.

2 MAPPING RULES

An actor consists of some built-in parameters, several input or output ports and special function in Ptolemy DE model. In our approach, each actor is translated into an automaton depending on an automaton template. These templates in template library are essentially timed automata and we design at least a timed automaton template for every actor according to the its function. The concrete contents about ports, parameters of actor are mapping into the template by mapping rules and make templates to be real timed automata. A DE model consists of a lot of actors so that there are many timed automata are generated. These timed automata communicate with each other by synchronization.

2.1 Mapping Rule Formulation

In our approach, the key is the the mapping rules formulation.

2.1.1 Mapping Rule of DiscreteClock Actor: DiscreteClock is a versatile event generator. It has three noticeable parameters: *period, offsets and values.* And its simplest use is to generate a sequence of events that are regularly spaced in time. Its default parameters cause the actor to produce tokens with value 1 (an int) spaced one time unit apart, starting at execution start time (typically time zero). But DiscreteClock can generate much more complex event patterns that cycle through a finite sequence of values and have periodically repeating time offsets from the start of each period.

$$s0 \xrightarrow{getI\_update} s1 \xrightarrow{\substack{guard \\ sync! \\ po\_update}} s0$$

$$s1 \xrightarrow{\substack{!guard \\ iclock\_update}} s1$$

The first edge *s0->s1* is used to get values of parameter *period, offsets and values.* We define a data typed int to simulate clock, the second edge *s1->s1* is used to the clock accumulation, the third edge *s1->s0* is used to send a synchronization and update the parameters and output port value if the *guard* is true. The *guard* guarantees that the automaton will send a synchronization at a particular clock.

2.1.2 Mapping Rule of FSMActor: FSMActor is an actor that can create a finite state machine (FSM). The FSM is built using states and relations rather than actors and connections. A relation represents the act of moving from one state to another, it can be triggered by a guard(g), which specifies the conditions under which the transition is taken. It is also possible to specify outputActions (o, actions that produce outputs when the relation is taken) and setActions (s, actions that set parameters when the relation is taken). FSMActor consists of multiple transitions, but the mapping rules of transitions are the same. i.e. $s0 \xrightarrow{g,o,s} s1$,the mapping rule is shown below:

$$s0 \xrightarrow{\substack{synci? \\ ip\_update}} s0p \xrightarrow{\substack{guard \\ po\_update}} s0o \xrightarrow{synco!} s1$$

The first edge *s0->s0p* is used to receive synchronization, and reset the value of *ip_isPresent* is true. The second edge *s0p->s0o* is used to update parameters and the values of output ports if the *guard* is true. The third edge *s0o->s1* is used to send synchronization. If the transition *s0->s1* in FSMActor doesn't need an input event, *s0* goes straight *s0o* with *guard* and *po_update.* And if the transition products more than one output event, an another edge *s0o->s0o* is needed to send an another synchronization.

2.1.3 Mapping Rule of TimedDelay Actor: The actor increases the model time of the input event by a specified amount. The increment is parameter of actor and its amount is displayed in the icon and defaults to 1.0. This actor delays an event in model time.

$$s0 \xrightarrow{\substack{synci? \\ update}} s1 \xrightarrow{\substack{guard \\ update}} s0$$

The first edge *s0->s1* is used to receive synchronization and reset the local clock to 0, the second edge *s1->s0* presents it sends a synchronization update the output port value when the local clock arrivals the delay clock.

2.2 Translation Correctness

We define a DE model and its associated timed automata are equivalent if their observable execution are equivalent, i.e. (1)the order that actor fires is the same as the order that automaton runs, (2)the two model have the same execution path if the input settings are the same, (3)all variable values at each execution step of the two model are equal.

3 CONCLUSION

In this paper, we proposed a model translation based approach to automatically translate and verify the Ptolemy DE model. Since Uppaal tool has a verifier and can effectively express discrete events, we translated a Ptolemy DE model to a network of timed automata in Uppaal and verify them using Uppaal tool. We implemented the approach and developed a plug-in in Ptolemy. In the future, we will extend the verification application area by enriching template library and design a method to automatically detect the consistency of the Ptolemy DE model and the network.

REFERENCES

[1] Lee, Edward A., 2006, Modeling, Simulation, and Design of Concurrent Real-Time Embedded Systems Using Ptolemy.
[2] Ht, L Ig , 2001, Overview Of The Ptolemy Project, Kidney International.
[3] Chihhong, 2008, Applied verification: The ptolemy approach.
[4] Bengtsson, 2006, Timed Automata: Semantics, Algorithms and Tools,
[5] Kim G, 2005, Tool Environment for Validation and Verification of Real-Time System.

Author Index