Constructing Cyber-Physical System Testing Suites using Active Sensor Fuzzing

Fan Zhang, Qianmei Wu, Bohan Xuan, Yuqi Chen, Wei Lin, Christopher M. Poskitt, Jun Sun and Binbin Chen✉

***Abstract*—Cyber-physical systems (CPSs) automating critical public infrastructure face a pervasive threat of attack, motivating research into different types of countermeasures. Assessing the effectiveness of these countermeasures is challenging, however, as benchmarks are difficult to construct manually, existing automated testing solutions often make unrealistic assumptions, and blindly fuzzing is ineffective at finding attacks due to the enormous search spaces and resource requirements. In this work, we propose *active sensor fuzzing*, a fully automated approach for building test suites without requiring any *a prior* knowledge about a CPS. Our approach employs active learning techniques. Applied to a real-world water treatment system, our approach manages to find attacks that drive the system into 15 different unsafe states involving water flow, pressure, and tank levels, including nine that were not covered by an established attack benchmark. Furthermore, we successfully generate targeted multi-point attacks which have been long suspected to be possible. We reveal that active sensor fuzzing successfully extends the attack benchmarks generated by our previous work, an ML-guided fuzzing tool, with two more kinds of attacks. Finally, we investigate the impact of active learning on models and the reason that the model trained with active learning is able to discover more attacks.**

***Index Terms*—Cyber-physical systems, fuzzing, testing, machine learning, metaheuristic optimisation.**

I. Introduction

CYBER-PHYSICAL systems (CPSs) are highly integrated systems where computing, communication and physical process are deeply intertwined, each potentially involving different spatial and temporal scales, modalities, and interactions [1]. By embedding perception, communication and computation elements in physical devices, CPSs manage to realize the distributed perception of external environment, trusted data transmission and processing of intelligent information. With the feedback of the *Cyber* part, CPSs construct a series of real-time control mechanisms of the *Physical* part. These complex systems are now ubiquitous in modern life, with examples found in fields as diverse as aerospace, autonomous vehicles, and medical monitoring. CPSs are also commonly used to automate aspects of critical civil infrastructure, such as water treatment or the management of electricity demand [2]. 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 [3], [4].

This pervasive threat faced by CPSs has motivated research and development into a wide variety of attack defense mechanisms, including techniques based on anomaly detection [5]–[15], fingerprinting [16]–[19], and monitoring conditions or physical invariants [20]–[27]. The practical utility of these different countermeasures ultimately depends on how effective they are at their principal goal: *detecting and/or preventing attacks*. Unfortunately, assessing this experimentally is not always straightforward, even with access to an actual CPS, because of the need for realistic attacks to evaluate them against. Herein lies the problem: *where exactly* should such a set of attacks come from?

A typical solution is to use existing attack benchmarks and datasets, as have been made available by researchers for different CPS testbeds [28], [29], and as have been used in the evaluation of different countermeasures, *e.g.,* [7], [12], [13]. Across these examples, attackers are typically assumed to have compromised the communication links to some extent, and thus can manipulate the sensor readings and actuator commands exchanged across the network. The aforementioned attacks are usually manually generated by engineers with sufficient expertise in the CPS, or through invited hackathons to discover new attacks from those without insider bias [30]. Unfortunately, constructing such benchmarks requires a great deal of time and expertise. And generalizing them from one CPS to another is a forlorn hope due to the distinct structures, components and complexities of different systems [31].

Using symbolic execution and simulation techniques to explore the state space of the system is also a solution to verify hybrid systems. S-TaLiRo [32] is used for falsification of hybrid systems with temporal logic specifications. Breach [33] is capable of reachability analysis to explore the state space of the system and could be integrated with other tools such as dReach and Simulink. In this paper, we propose *active sensor fuzzing*, an automated, machine learning (ML) guided

Manuscript received April 19, 2021; revised August 16, 2021.*(Corresponding author:Binbin Chen)*

Fan Zhang is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China, with the ZJU-Hangzhou Global Scientific and Technological Innovation Center, Hangzhou 311200, China, with the Key Laboratory of Blockchain and Cyberspace Governance of Zhejiang Province, Hangzhou 310027, China, with the Jiaxing Research Institute, Zhejiang University, Jiaxing 314000, China, and also with the Zhengzhou Xinda Institute of Advanced Technology, Zhengzhou 450000, China (e-mail: fanzhang@zju.edu.cn).

Qianmei Wu and Bohan Xuan are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: qianmei@zju.edu.cn; xuanbohan@zju.edu.cn).

Yuqi Chen is with the School of Information Science and Technology at ShanghaiTech University, Shanghai 201210, China (email: chenyq@shanghaitech.edu.cn)

Wei Lin and Binbin Chen are with the the Information Systems Technology and Design pillar, Singapore University of Technology and Design, 8 Somapah Road, Singapore 487372 (email: wei_lin@mymail.sutd.edu.sg;binbin_chen@sutd.edu.sg)

Christopher M. Poskitt and Jun Sun are with the School of Computing and Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902 (email: cposkitt@smu.edu.sg;junsun@smu.edu.sg)

approach for constructing 'test suites' (or benchmarks) of CPS network attacks, without requiring any specific system expertise including the normal operational ranges of sensors. Our approach consists of two broad steps: *learning* and *fuzzing*. In the learning part, we construct a series of classification models for predicting future actuator configurations and regression models for predicting future sensor readings. Instead of exploiting traditional *online learning*, we leverage *active learning* [34], a form of supervised ML that iteratively retrains a model on examples that are estimated to maximally improve it. Since *online learning* requires enormous search space and resource costs for acquiring new training data, we apply the concept of active learning that selects training data which would improve the model most and accordingly search the vectors that tends to maximally change the actuator configurations (i.e., the output of the model). Retraining the model with such vectors could maximally improve our model. By this way, we can save time and effort in acquiring new training data. In the fuzzing part, we fuzz the sensor readings over the network to fool the system to automatically generate attack sequences that drive the system into a targeted unsafe state. This fuzzing is guided by the learnt model: potential manipulations of the sensors are searched for (*e.g.,* with a genetic algorithm [35] or simulated annealing algorithm [36]), and then the model predicts which of them would drive the CPS closest to the unsafe state. Combining *learning* and *fuzzing*, our approach intelligently and automatically construct attacks the would drive the system into different categories of unsafe physical states and hence can be utilized as 'test suites'.

Our design for active sensor fuzzing was driven by four key requirements. First, that it should be *general*, in the sense that it is not devised for specific CPS, but is able to apply in a variety of sensors and actuators. Second, that the approach should be *comprehensive*, in that the suites of attacks it constructs should systematically cause different categories of unsafe states for CPS, rather than just a select few. Third, that it should be *efficient*, with each attack achieving its goal quickly (within an acceptable and reasonable time interval), posing additional challenge for countermeasures. Finally, that it should be *practical*, 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.

To evaluate our approach against these requirements, we implemented it for a real-world CPS testbed: the Secure Water Treatment (SWaT) testbed [37]. SWaT is a fully operational water treatment plant consisting of 42 sensors and actuators, able to produce five gallons of drinking water per minute. The design of the testbed was based on real-world industrial purification plants and distribution networks, and thus reflect many of their complexities. The evaluation shows that active sensor fuzzing could automatically identify suites of attacks that drove SWaT into 15 different unsafe states involving water flow, pressure and tank levels. Furthermore, it covered nine unsafe states beyond those in an established expert-crafted benchmark [29] and two unsafe states beyond those constructed by another intelligent fuzzing framework [38]. Finally, we study the difference between the models trained with and without active learning. We show how active learning improves the model performance and helps construct attack suites.

In summary, we make the following contributions.

- We propose active sensor fuzzing, which makes use of ML models and metaheuristic methods for automatically constructing test suites (or benchmarks) of network attacks for different CPSs. We utilize active learning to improve the accuracy of our models and the performance of the attack suites.
- We implement active sensor fuzzing for a real-world CPS testbed, identifying attacks that drive it into 15 different unsafe states, including nine that were not present in an expert-crafted benchmark and two that were not covered by an existing fuzzing framework.
- We successfully launch targeted attacks and multi-point attacks, which to the best of our knowledge, is never achieved automatically before.
- We analyze the impact of active learning on those trained models and try to explain why active learning can result in better performance in constructing attack suites.

This paper is an extension of our previous publication [38] by complementing the original framework with a new option of the victim components in the system: sensor readings. The attacks that involved manipulating sensor readings were not evaluated in the previous work, which means a huge class of real-world attacks were not possible to find. Meanwhile, sensor-manipulating attacks are important to include in test suites. When we override the sensor readings, the system itself will automatically assign the corresponding actuator configurations to get back to its 'normal' states. Such procedures are more covert, especially when the system is deployed with some intrusion detection systems (IDS). So the benchmarks based on sensor readings are more likely to achieve the attack goal without triggering any defense mechanisms. Apart from the new attack targets, We learn some extra models to better illustrate the relation between actuators and sensors and introduce a new search algorithm to guide the fuzzing process. We also introduce the *active learning* technique into the process of learning to generate a more accurate model. Furthermore, we complement the original attack benchmarks by implementing two sophisticated attacks: targeted and multi-point attack.

II. Background and Motivational Example

Here, we clarify our assumptions of CPSs and fuzzing, before introducing our real-world CPS case study, SWaT. We discuss an example of active sensor fuzzing on SWaT.

CPSs and Fuzzing. We define CPSs as systems in which algorithmic control and physical processes are tightly integrated. Concretely, we assume that they consist of computational elements (the 'cyber' part) such as programmable logic controllers (PLCs), distributed over a network, and interacting with their processes via sensors and actuators (the 'physical'

Fig. 1. The Secure Water Treatment (SWaT) testbed

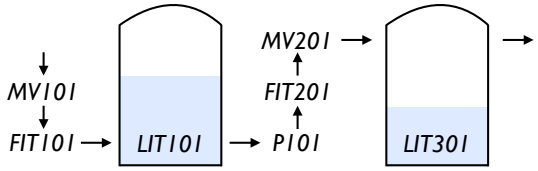


Fig. 2. Some interconnected components of SWaT's first three stages

part). 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. We assume that the sensors read continuous data (*e.g.,* temperature, pressure, flow) and that the states of the actuators are discrete.

These characteristics together make CPSs very difficult to reason about: while individual control components (*e.g.,* PLC programs) may be simple in isolation, reasoning about the behavior 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.

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 [39]. Many of the most well-known tools perform fuzzing on programs, *e.g.,* [40], [41], but in the context of CPSs, we consider fuzzing at the *network* level.

SWaT Testbed. One CPS that satisfies the aforementioned assumptions, and thus will form the case studies of this paper, is Secure Water Treatment (SWaT) [37], testbed built for cyber-security research. (Due to the impact of COVID-19 pandemic, we do not evaluate our approach on WADI testbed like our previous submission [38] did.) SWaT (Figure 1) is a fully operational water treatment plant with the capability of producing five gallons of safe drinking water per minute. The testbed is scaled-down versions of actual water treatment plant in a city, and exhibit many of their complexities.

SWaT treats water across six distinct but co-operating stages, involving a variety of complex chemical processes. Each stage is controlled by a dedicated Allen-Bradley ControlLogix PLC, which communicates with the sensors and actuators relevant to that stage over a ring network, and with other PLCs over a star network. Each PLC cycles through its program, computing the appropriate commands to send to actuators based on the latest sensor readings received as input. The system consists of 42 sensors and actuators in total, with sensors monitoring physical properties such as tank levels, flow, pressure, and pH values, and actuators including motorized valves (for opening an inflow pipe) and pumps (for emptying a tank). A historian regularly records the sensor readings and actuator commands during SWaT's operation, facilitating data logs for offline analyses and machine learning.

The sensors in the testbed are associated with manufacturer-defined ranges of *safe* values, which in normal operation, they are expected to remain within. If a sensor reports a (true) reading outside of this range, we say the physical state of the CPS has become *unsafe*. If a level indicator transmitter, for example, reports that the tank in stage one has become more than a certain percentage full (or empty), then the physical state has become unsafe due to the risk of an overflow (resp. underflow). Unsafe pressure states indicate the risk of a pipe bursting, and unsafe levels of water flow indicate the risk of possible cascading effects in other parts of the system.

A number of countermeasures have been developed to prevent SWaT from entering unsafe states. Ghaeini and Tippenhauer [42], for example, monitor the network traffic with a hierarchical intrusion detection system, and Ahmed et al. [16], [17] detect attacks by fingerprinting sensor and process noise. Other approaches learn models from physical data logs, and use them to evaluate whether or not the current state represents normal behavior or not: most (*e.g.,* [7], [12], [43]) use unsupervised learning to construct these models, although Chen et al. [23], [25] use supervised learning by automatically seeding faults in the control programs (of a high-fidelity simulator). Feng et al. [44] also generate invariants, but use an approach based on learning and data mining that can capture noise in sensor measurements more easily than manual approaches.

Active Sensor Fuzzing Example. To illustrate how our approach works in practice, we informally describe how it is able to automatically find an attack for overflowing a tank in the first stage of SWaT. Figure 2 depicts the relationship between some interconnected components across the first three stages. It includes some sensors: Level Indicator Transmitters (LITs) for reporting the levels of different tanks; and Flow Indicator Transmitters (FITs) for reporting the flow of water in some pipes. It also includes some actuators: Motorized Valves (MVs), which if open, allow water to pass through; and a Pump (P101), which if on, pumps water out of the preceding tank. The physical flow of water throughout this subpart of the system is controlled by some inter-communicating PLCs. If the value of LIT301 becomes too low, for example, the PLC controlling valve MV201 will open it. Furthermore, the PLC controlling pump P101 will switch it on to pump additional water through, causing the value of LIT301 to rise.

Before launching our fuzzer, two choices must be made: first, what is the *attack goal* (characterized as a fitness function); and second, which search algorithm should be used to identify actuator configurations that achieve it? As our goal is to overflow the tank monitored by LIT101, we must define

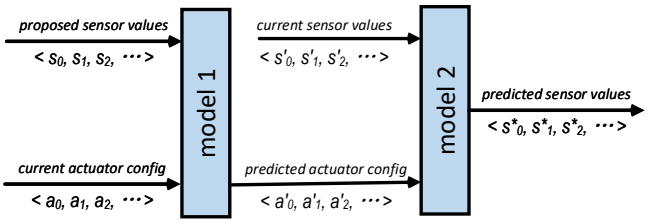


Fig. 3. Inputs/outputs of the learnt models

a fitness function on the sensor readings that is maximized as we get closer to overflowing the tank. A simple function achieving this takes as input a vector of predicted sensor states $\langle \text{LIT101}, \text{LIT301}, \ldots, \text{FIT101}, \text{FIT201}, \ldots \rangle$ and returns simply the value of LIT101. Initially, we randomly search through the space of sensor readings since we do not assume any prior knowledge.

Upon launching the fuzzer, several different values of the system's sensors are (randomly) generated and sent to override the current sensor readings. For each set of 'fake' values, two models are used to predict the future sensor values. Note that for assessing the effects of the attacks, separate variables are used to record 'true' sensor readings rather than the 'fake' sensor values we generate. Specifically, one model is used to *predict* the future actuator configurations that the PLC will issue according to the spoofed sensor readings, and another model, as same as the model trained in our previous publication [38], would *predict* the corresponding changes of sensor readings according to the actuator configurations. Note that the change of sensor readings can also be predicted with one model, which takes the 'fake' sensor values, the 'true' sensor values and the current actuator configurations as the input. The intuition of training two models is that we can observe how different states of sensor readings affect the evolution of actuators and furthermore have a clearer understanding about the interaction among the different components of the CPS.

Our fitness function is maximized by future states with the highest LIT101 readings, which in Figure 2, would result from a configuration of actuators where MV101 is open, P101 is off, P102 is off, and P601 is on (as water will be flowing into the tank, and the pumps will not be removing it). Note that P102 and P601 are not pictured: the former is a backup pump (redundancy for P101); the latter is a pump for driving in water from stage 6. Such actuator configurations can be achieved when the readings of FIT101 and LIT101 are low, the readings of FIT201 and LIT301 are high, so that the PLC will open MV101 and P601 to supply LIT101 and close P101, P102 and MV201 to prevent the overflow of LIT301. With suitable sensor readings identified: `FIT101 0.00`, `LIT101 400.00`, `FIT201 3.00`, and `LIT301 1100.00`, the relevant commands will be issued by PLCs: `OPEN MV101`, `OFF P101`, `OFF P102`, and `ON P601`. Eventually, LIT101 will reach its upper unsafe range (risk of overflow), *i.e.,* the attack goal.

III. How Sensor Fuzzing Works

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 two models of the CPS that can predict the effects of sensor readings and actuator configurations based on the physical state respectively, as summarized in Figure 3.

Let x[n] denote a plant state at time n. Let s[n] = S(x[n]) be the equation denoting the relation between the plant state x[n]. Let a[n] = A(x[n]) denote the actuator configuration of x[n]. Let x[n+1] = f(x[n], a[n]) denote the dynamical evolution of the plant state at time n+1 in terms of its state and actuator input at time n. The first model tries to learn the composition of S and f, it takes as input the current configuration of all actuators and proposed readings of the sensors, returning as output a prediction of the actuator configurations that would result from adopting those readings for a fixed time interval. The second model tries to learn the composition of f and A, it takes all of the true sensor readings and the actuator configurations predicted by the first model as input and outputs the prediction of the true sensor values after a fixed time interval. Notice that we can also train a single model which takes the configuration of actuators and proposed sensor readings and directly predicts the future sensor values. However, with training two models , we are able to further analyze the inner relation among the different devices (*e.g.,* sensors and actuators) of the CPS that can later be used to inform which combination of sensor readings is likely to drive the system closer to a targeted unsafe state.

Unlike actuator configurations, which are composed of discrete states, the sensor reading is a continuous value. Considering the precision of the target CPS testbed, the search space of sensor readings is enormous; hence blindly manipulating the sensor reading has limited effectiveness to generate the meaningful attacks suites. Therefore, in addition to the normal training procedure, we also consider utilizing *online active learning* to overcome the huge search space. Active learning is a form of supervised ML technology, which is proposed to overcome the high cost of labeling by iteratively selecting unlabeled data from the data pool and re-training the model. It is proven [31] that active learning is useful to reduce the training overhead and enhance the model performance. In our approach, we adapt the Expected Behavior Change Maximization (EBCM) framework from [31]. The intuition is that exploring different behavior in a particular context is more informative. To learn aforementioned models, we extract a time series of sensor and actuator data from the system logs and apply a suitable machine learning algorithm.

The second step of our approach searches for sensor readings to fuzz the sensor with that will drive the CPS into an unsafe physical state. The sub-steps of this part are summarized in Figure 4. To find the right combination of sensor readings, our approach applies search algorithms over the space of sensor readings, returning the configuration that is predicted by the cascaded 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,

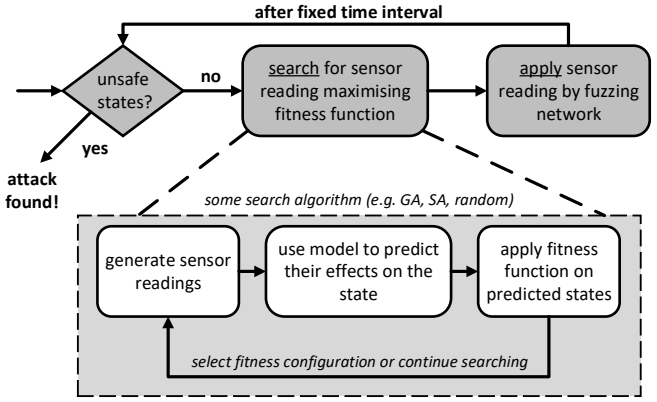


Fig. 4. Overview of our ML-guided sensor fuzzing (details of any particular search algorithm are omitted here)

but also metaheuristic (*e.g.,* genetic algorithms and simulated annealing) given that the search space of sensor readings can grow quite large (Since each sensor reading is stored in single-precision floating-point format in PLC). We use *fitness functions* to evaluate predicted sensor states with respect to the attack goal.

Assumptions. Given that active sensor fuzzing, ultimately, is intended to generate attacks for evaluating CPS defense solutions, it is important to detail our assumptions about systems and attackers, thus characterizing the kinds of attacks that will be found (and just as importantly, those that will not be).

In the learning phase of our approach, we assume the integrity of the data that the machine learning models are trained on (*i.e.,* an attacker has not manipulated the learning of the model). In the fuzzing phase, we search for network attacks that could be executed by attackers that are able to compromise the communication links between PLCs and sensors. In particular, we assume that attackers can monitor all genuine sensor readings and actuator commands, and can manipulate the readings of arbitrary sensors at will. Notice that in order to make our attacks more stealthy, all the modified sensor readings are within the range when the system runs without tampering. Furthermore, we assume that the genuine readings originating from any sensors can be intercepted and blocked from ever reaching the PLCs. Note that while in general we assume a rather powerful attacker, it is possible to use our technique with *more restricted capabilities* too (*e.g.,* particular sensors or stages only), in order to test defense mechanisms in a more realistic context.

A. Step One: Learning a Prediction Model

Data Collection. Our method requires a dataset from which the relationship between actuators, sensor readings and the evolution of the physical states can be learnt. A suitable raw format for such data is a *time series* of sensor readings and actuator configurations, recorded at regular intervals during the regular operation of the CPS. The required size of the time series depends upon how many 'modes' of behavior the system exhibits, and how quickly the effects of actuator commands are propagated through the physical state. In general, logs from several days of operation may be sufficient to span enough of the system's normal operational behavior.

Since our method is aimed at engineers or researchers who are seeking to assess the defenses of a CPS (*i.e.,* not an external attacker), we assume that such data is readily available, *e.g.,* from the system's historian. But in principle, it could also be collected by monitoring the network traffic.

For SWaT, time series datasets are already available online [29], and were obtained by running the testbeds non-stop for seven days without any interruption from attacks or faults. The datasets include the states of all sensors and actuators as recorded by the historians every 1s.

Training a Model. Our method builds two models cascaded together as Figure 3 depicts. In this paper, we consider three different ML algorithms that are popular and well-suited for the particular form of training data—a time series of actuator states and (continuous) sensor values. First, we consider a Long Short-Term Memory (LSTM) network [45], a deep learning model with an architecture that supports the learning of longer-term dependencies in the data for both models. Second, we consider a Support Vector Classification (SVC) [46] for the first model to predict the states of the actuators. Most of the actuators can only be in one of two states (*i.e.,* 0 for off, 1 for on), and thus we can consider it as a binary classification problem, where each class represents one physical states of the actuator. For the second model, whose outputs are sensor readings, we choose Support Vector Regression (SVR) model [47], an extension of support vector machines for handling continuous values.

To apply these ML algorithms to a dataset, the sensor and actuator values must be extracted from the raw logs into a fixed vector format. As we want to learn the relation between actuator states and the evolution of sensor values, a possible form of the vector is $\langle\langle s_0, s_1, \ldots, a_0, a_1, \ldots\rangle, \langle s'_0, s'_1, \ldots, a'_0, a'_1, \ldots\rangle\rangle$, where the s_i and a_i are the readings/states of sensors and actuators at time point t, and the s'_i and a'_i are the readings/states of the same sensors and actuators but at time point $t+i$ for some fixed time interval i. Depending on the particular algorithm, the sensor values may also need to be normalized. After extracting as many vectors as possible from the raw data, as is typical in learning, the algorithm should only be trained on a fixed portion of them (*e.g.,* 80%), leaving the remainder as test data for assessing the accuracy of the learnt model's predictions.

We trained one set of LSTM and SVC/SVR models using standard Python library implementations: the Keras neural networks API [48] and scikit-learn [49] respectively. We extracted vectors for training from days 2–5 of the SWaT dataset, with vectors from the remaining days reserved for testing the learnt models. Our two LSTM models (model 1 and model 2) used a traditional architecture consisting of an LSTM layer followed by a dense, fully-connected layer, and took approximately 17 hours of training on a server with four TITAN Xp GPUs. The training of SVC and SVR models took approximately 3 hours and 7 hours, repectively. The accuracy

TABLE I
THE ACCURACY (%) OF MODEL 1 AND MODEL 2 APPLYING THREE ML TECHNOLOGIES

	LSTM	SVC	SVR
Model 1	98.8%	98.2%	—
Model 1-AL	99.2%	99.0%	—
Model 2	96.3%	—	95.1%

Algorithm 1: Expected Behavior Change Maximization

Input: Prediction model M_s, prediction time t_s
Output: Feature vector p_f

Sniff current sensor readings and actuator configurations and construct a feature vector p_o based on their values;
Wait for t_s seconds then observe the value v_s of s;
Let $P := \langle\rangle$; [empty sequence]
Let $D := \langle\rangle$;
repeat
  Construct a new vector p from p_o by randomly manipulating sensor values of p_o;
  $v_p := M_s(p)$;
  $P := P^\frown\langle p\rangle$; [Include the new vector into P]
  $D := D^\frown\langle|v_s - v_p|\rangle$; [Include the value difference into D]
until *timeout*;
Select a feature vector p_f from P using *Roulette Wheel Selection* with corresponding values in D;
return feature vector p_f;

of each model is given in Table I. Note that upon testing of model 2, which outputs continuous values, the accuracy is measured with a tolerance between the actual and predicted values of 5%.

We remark that in our SWaT implementations, in addition to learning models able to predict *all* the sensor values and actuator configurations $\langle s'_0, s'_1, s'_2, \ldots, a'_0, a'_1, a'_2, \ldots\rangle$, we also learnt a series of simpler models that predict *only* a single sensor reading or actuator status. These models are useful in later experiments (Section IV) when the fuzzer is attempting to drive one sensor in particular to an unsafe state, since they can provide the necessary predictions more efficiently, and are also faster to learn (5-10 minutes on an off-the-shelf laptop). Furthermore, for SVC/SVR, it allowed us to vary the time interval in predictions for different sensors. While most were set at 1s (as for LSTM), we found that we needed a larger interval of 7s in SVC for the valves and the pumps and 100s in SVR for the water tank level sensors, due to the fact that the actuators take a certain time to open/close and tank levels change more slowly. (This is only an issue for SVC/SVR, since the architecture of LSTM is specially designed to handle long lags between events.)

As for the active learning part, we adopt the EBCM algorithm in [31] which samples examples estimated to cause maximally different *behavior* from the current state of the system. We only apply active learning on model 1 since applying it on model 2 would take a lot longer time (the change of water levels is much slower than the change of the states of actuators). It is unreasonable to assume that the attacker can access the CPS and manipulate its data for such amount of time without triggering any alarms. Algorithm 1 summarizes the steps of EBCM. We first construct a new feature vector by sampling the *true* sensor readings and actuator configurations. Then the sensor readings are randomly manipulated and a vector that would have led to a maximally different actuator states than the original is chosen as the output. We use Roulette Wheel Selection [35] to choose the feature vector from a set of several to ensure some variation. Each candidate is assigned a probability of being selected based on its 'fitness'. Herein we encode the actuator states with a vector consists with 0/1, and calculate the *difference* between two sets of actuator states with hamming distance. The 'fitness' of each candidate is defined as the *difference* between what the actuator configurations actually are and what the current model predict for the candidate.

In our active learning setting, the initial models are trained using the data only from day 2, day 3 and half of day 4. Then the active learning process will be repeated for 10 rounds. In each round, a set of sensor readings will be generated according to the EBCM algorithm and the prediction of the model. The genuine sensor readings will be replaced by the falsified ones and the actuators will respond to the new readings. We wait for 10 seconds to ensure that all the actuators complete the on/off operations and collect the new sensor readings and actuator states. Such procedure will be repeated for 10 times in each round and the model will be updated according to the newly collected data when a round is finished. The whole active learning procedure will take approximately 20 minutes (10 rounds*100 seconds/rounds, added up with the time to generate sensor readings), which is a acceptable time compared with the training period. The accuracies of the pre-trained models are 90.4% and 84.2% for LSTM and SVC, respectively. The performances with active learning are also given in Table I. We can conclude that with fewer data samples, models with better performance are achieved while using active learning.

B. Step Two: Fuzzing to Find Attacks

Fitness Functions. We use *fitness functions* to quantify how close some (predicted) sensor readings are to an unsafe state we wish to reach. Intuitively, a fitness function takes a vector of sensor values as input, and returns a number that is larger the 'closer' the input is to an unsafe state. The goal of a search algorithm is then to find sensor readings that are predicted (by the LSTM/SVM models) to bring about genuine sensor states that *maximize* the fitness function.

Fitness functions are manually defined for the CPS under consideration, and should characterize *precisely* what an unsafe state is. There is a great deal of flexibility in how they are defined: it is possible to define them in terms of the unsafe ranges of individual sensors, of groups of related sensors, or of different combinations therein. In this paper, we perform sensor fuzzing using fitness functions for the

individual sensors in turn, to ensure a suitable variety of attacks, and to avoid the problem of a single sensor always dominating (*e.g.,* because it is easier to attack).

Defining a fitness function for a single sensor s is straightforward. If its unsafe range consists of all the values above a certain threshold, then a suitable fitness function would be the sensor reading s itself, since maximizing it brings it closer to (or beyond) that threshold. If its unsafe range consists of values below a certain threshold, then a suitable fitness function would be the *negation* of the sensor reading, $-s$.

A general fitness function can be defined for any group of the system's sensors. Let v_s denote the current value of sensor s, L_s denote its lower safety threshold, H_s denote its upper safety threshold, and $r_s = H_s - L_s$ denote its range of safe values. Let

$$d_s = \begin{cases} \min\left(|v_s - L_s|, |v_s - H_s|\right) & L_s \leq v_s \leq H_s \\ 0 & \text{otherwise} \end{cases}$$

i.e., the absolute distance of the sensor reading from the nearest safety threshold. Then, the fitness function for a set S of the CPS's sensors is $\sum_{s \in S} \frac{1}{d_s / r_s}$. This function tends to infinity as individual sensor readings get closer to either their lower or upper safety thresholds. In other words, the fitness function returns an increasingly large number as the system moves in the direction of an unsafe state.

For SWaT, we defined fitness functions on a per-sensor basis, and treated the values above and below these thresholds as two *separate* cases. We did this for two reasons: first, to ensure that sensor fuzzing can find attacks that violate each sensor in both ways (where applicable); and second, as it increases the diversity of attacks in cases it is much easier to attack a sensor in one direction. We remark that a preliminary investigation found more general fitness functions (*e.g.,* favoring *any* kind of attack) not to be useful for SWaT, as the attacks found by fuzzing were dominated by sensors that were easier to drive to unsafe ranges, such as the water flow indicators in the first stage.

Searching and Fuzzing. Note that as described in our active fuzzing example (see Section 2), in the context of sensor fuzzing, the sensor value injection will influence the state of actuators firstly. Afterward, these actuators will drive the 'true' sensor reading to the unsafe state it's going to reach, *i.e.,*, lower or upper safety thresholds. In addition to the single-point attack presented in our previous paper [38], we also perform targeted and multi-point attacks (see Section 4.2 RQ3). In order to generate the attack benchmarks, we consider three different algorithms for searching across the space of sensor readings. First, we consider a simple random search, in which several combinations of sensor readings are randomly generated. The network is then fuzzed to apply the readings that is predicted (by the LSTM/SVM model) to maximize the fitness function.

Second, we consider a genetic algorithm (GA) [35], a metaheuristic search inspired by the "survival of the fittest" principle from the theory of natural selection. The high-level steps of our particular GA for finding sensor reading configurations are summarized in Algorithm 2. In [38],the actuator configurations are chosen as the fuzzing targets, which can be encoded as bit vectors. However, in this paper we fuzz sensor readings, which are usually stored as floating-point numbers. Although we can map these floats into discrete space and encode them afterwards, it may not be effective (due to the complication of encoding floating number operations using bit vector operators). Therefore, we adapt our GA algorithm to deal with floating-point data. The implementation of the GA algorithm can be described as follows: First, a population of sensor reading configurations is randomly generated. Note that all the generated sensor readings are within the range of the minimum to the maximum readings from the data logs we collect (See Section III-A) for each individual sensor. The inspiration of this setting is that all the fuzzed sensors readings *have appeared* during the normal operations of the testbed. So that the attack may bypass some simple defense mechanisms that only based on the sensor readings. Next, we calculate the fitness of each candidate in the population by: (1) applying the cascaded model; then (2) applying the fitness function to the resulting predicted state.

Algorithm 2: GA for Sensor Readings

Input: Vector of actuator configurations A, prediction cascaded model M, fitness function f, population size n, no. of parents k, mutation probability p_m
Output: Vector of sensor readings

```
Randomly generate population P of n sensor readings;
Compute fitness of each candidate c ∈ P with f(M(c, A));
repeat
    Select k parents from P using Roulette Wheel Selection;
    Generate new candidates from parents using crossover;
    Generate new candidates from parents using
      floating-point mutation with probability p_m;
    Compute fitness of new candidates c with f(M(c, A));
    Replace P with the n fittest of the new and old
      candidates;
until timeout;
return candidate c ∈ P that maximizes f(M(c, A));
```

At this point we enter the main loop, in which the fittest candidates are selected for generating "offspring" from. We select the candidates using roulette wheel selection [35], which assigns to each candidate a probability of being selected based on its fitness. If f_i is the fitness of one of the n candidates, then its probability of being selected is $f_i / \sum_{j=1}^{n} f_j$. Next, we sample candidates based on the probabilities using the following implementation. First, a random number is generated between 0 and the sum of the candidates' fitness scores. Then, we iterate through the population, until the accumulated fitness score is larger than that number. At this point we stop, selecting the last candidate as a "parent". We repeat this until we have selected k parents (possibly including duplicates).

From the parents, we generate new candidates (offspring) by applying crossover and mutation. For the former, we have

$$\begin{aligned} x_A^{'} &= \alpha x_B + (1-\alpha) x_A \\ x_B^{'} &= \alpha x_A + (1-\alpha) x_B \end{aligned} \tag{1}$$

where α is a random number between 0 and 1.

Algorithm 3: SA for Sensor Readings

Input: Vector of actuator configurations A, prediction cascaded model M, fitness function f, initial temperature T_{in}, minimum temperature T_{min}, no. of initial solutions n, no. of inner iterations k, temperature constant t
Output: Vector of sensor readings

```
Randomly generate set S of n sensor reading configurations;
Compute fitness of each candidate c ∈ S with f(M(c, A));
Select c ∈ S that maximizes f(M(c, A));
Set initial temperature T as T = T_in;
repeat
    repeat
        Generate a random noise Δc;
        Generate a new candidate c_new by adding Δc to c;
        if f(M(c_new, A)) > f(M(c, A)) then
            c = c_new;
        else
            Generate a random number r between 0 an 1;
            if r < exp((f(M(c_new, A)) − f(M(c, A)))/T) then
                c = c_new;
            else
                continue;
    until iteration k is reached;
    Update T with new temperature T/(1 + t);
until timeout or T < T_min;
return the final candidate c;
```

For the latter, we have

$$x_A^{'} = \begin{cases} x_A + k \cdot (x_{\max} - x_A) \cdot r, & \text{rand }()\%2 = 0 \\ x_A - k \cdot (x_A - x_{\min}) \cdot r, & \text{rand }()\%2 = 1 \end{cases} \quad (2)$$

where k is 0.2, r is a random number between 0 and 1, $x_{\max}$ and $x_{\min}$ are the maximum value and the minimum value of the corresponding sensor in the historian data logs, respectively.

The fitness of all the new candidates is calculated, then the n fittest candidates from the old and the new candidates are carried forward as the new population, with the other, less fit candidates all eliminated. This iteration continues until a fixed timeout is reached. The fittest sensor reading configuration is then returned, and the relevant genuine sensor readings are overwritten by the falsified ones.

Our third searching algorithm is simulated annealing (SA) [36], a metaheuristic method models a physical process where a solid is heated and then slowly cools down to decrease defects to solve optimization problems. The high-level steps of our particular SA for finding sensor readings are summarized in Algorithm 3. The details of how we implement SA algorithm can be described as follows: First, we randomly generate a set of sensor reading configurations (like GA, all the sensor readings are in the normal range). Next, we calculate the fitness of each individual configuration by : (1) applying the cascaded model; then (2) applying the fitness function to the resulting predicted state. We single out the configuration with the maximum fitness and set it as our first *solution.*

Then we enter the main loop. We generate a new *solution* by randomly adding up a tiny noise Δc to the previous *solution*. The fitness f_{new} of the new *solution* will be calculated. If the f_{new} is larger than the fitness of the previous *solution* f_{old}, the previous *solution* will be replaced by the new one. Else, we generate a random number r between 0 and 1. If r is smaller than $\exp(f_{new} - f_{old})/T)$, the previous *solution* will also be replaced by the new one. Else, the *solution* remains unchanged. Such procedure will be repeated for k times. Then the temperature T will be updated with a lower temperature $T/(1 + t)$. After this, we enter the next loop. The main loop continues until a fixed timeout is reached or T is lower than the minimum temperature T_{min}.

In our GA implementation, we set the population size as $n = 100$, number of parents as $k = 100$, the crossover probability as 0.5, and the mutation probability as 0.1. These parameters were chosen to ensure that the algorithm will converge before the 10s timeout. Furthermore, we cap the maximum number of iterations at 100 (if reached before the timeout). In our SA implementation, we set the initial temperature as $T_{in} = 1$, minimum temperature as $T_{min} = 7 \cdot 10^{-3}$, the number of initial solutions n as 100, the number of inner iterations k as 10, the temperature constant t as 1. These parameters were also chosen to ensure the time constraint. For the random search, the maximum number of the generated combinations is set to 1000. For the GA and SA searches, we set a timeout of 10s, after which the best sensor reading found so far would be the one that is applied. While this 10s timeout is clearly longer than the 1s prediction time interval of our models (except for the 100s time interval for tank levels in SVM), the predictions are still meaningful because the physical states of the testbeds evolve so slowly in the meantime. For CPSs that change faster, a longer prediction time interval may need to be considered for the search to remain practically useful.

Once our random, GA or SA search algorithm identifies the fittest sensor readings, we use the Python package pycomm [50] to fuzz the sensor readings over the network. As long as the system continues to approach the targeted unsafe state, the sensor readings are held; otherwise, the search is repeated to identify a more fruitful configuration.

IV. Evaluation

We evaluate the effectiveness of active sensor fuzzing on the SWaT testbeds (Section II).

A. Research Questions

Our evaluation addresses six research questions based on our original design requirements for active sensor fuzzing (Section I):

- **RQ1 (Efficiency):** How quickly is active sensor fuzzing able to find an attack?
- **RQ2 (Comprehensiveness):** How many unsafe states can the attacks of active sensor fuzzing cover?
- **RQ3 (Complexity):** Is active sensor fuzzing able to generate sophisticated attacks?
- **RQ4 (Setup):**Which combination of model and search algorithm is most effective?
- **RQ5 (Comparisons):** How do the attacks compare against those of other approaches or those in benchmarks?

RQ6 (Improvement): How active sensor fuzzing complements our original fuzzing framework ?

B. Experiments and Discussion

RQs 1–2 consider whether active sensor 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 investigates whether active sensor fuzzing is able to generate sophisticated attacks, *i.e.,* targeted and multi-point attacks. RQ 4 considers how different setups of active sensor fuzzing (*i.e.,* different models or search algorithms) impact its ability to find attacks. RQ 5 compares the effectiveness of active sensor fuzzing against two other approaches: the baseline of randomly manipulating sensor readings *without* reference to a model of the system, and an established, manually constructed benchmark of attacks [29]. Finally, RQ6 clarifies how the new features induced in active sensor fuzzing complement our previous work, which uses ML-guided fuzzing to mutate actuator states [38], and discusses how they promote the predictive models and the procedure of finding attacks in the context of cyber physical systems.

We design four experiments for the SWaT testbed to evaluate our research questions. The programs we built to perform these experiments and all supplementary material are available online to download [51].

Experiment #1: RQs 1, 2&4. In our first experiment, we systematically target the different possible unsafe sensor ranges in SWaT, using our tools in six different model/search setups: LSTM-GA, LSTM-SA, LSTM-Random, SVR-GA, SVR-SA and SVR-Random (Each model setup contains models trained with and without active learning, see Section 3). Our goal is to collate the data and obtain an overall picture of how the different ML-guided setups perform, how quickly attacks are found, and how many different unsafe states are covered.

For each of the setups and fitness functions (usually two per sensor, targeting the lower and upper safety thresholds separately), we ran the experiment as follows. First, we "reset" the testbed by operating it normally until all sensors entered their safe ranges. Upon reaching this state, we launched the fuzzer with the given setup and fitness function, and let it run without interference for up to 20 minutes (long enough to ensure that over/underflow attacks are able to complete). If the unsafe state targeted by the fitness function was reached within that time, we disabled the fuzzer and recorded the time point at which the transition to unsafe state occurred. (If other sensors happen to enter their unsafe states along the way, we record this too, but do not disable the fuzzer for them.) Once the fuzzer was disabled, the testbed was allowed to reset and return to a safe state. We repeated these steps 5 times for *every* combination of fuzzer setup and fitness function, and recorded the median so as to remove biases resulting from differences in starting states. Note that in this experiment, we did not specifically target overflow or underflow attacks. So our tool would automatically choose the *nearest* unsafe state as the goal.

Results. The results of this experiment are given in the first twelve rows of Table II. The rows denote the different fuzzing setups (model trained with active learning is indicated by their subscripts *AL*), whereas the columns denote the different possible unsafe states for each sensor. These include: Flow Indicator Transmitters (FITs), which measure water flow in pipes and can become too High or Low; a Differential Pressure Sensor (DPIT), which can become too High or Low; and Level Indicator Transmitters (LITs), which measure the water levels of tanks, and can indicate a risk of Overflow or Underflow. We do not include the Analyzer Indicator Transmitters (AITs), which measure properties such as pH, as the testbeds currently take raw water from the mains (*i.e.,* already close to pH 7); as a result, the readings barely vary during the plant's operation. The numbers indicate the amount of time taken, in seconds, for a particular fuzzing setup to reach an unsafe state (RQ1); they are the medians obtained from 5 repetitions per combination of fuzzing setup and fitness function. Numbers indicated with an asterisk ($*$) indicate that one or more repetitions that were unable to reach the unsafe state within 20 minutes. Furthermore, $1200+$ indicates that despite approaching the given unsafe state, none of the repetitions were able to cross the threshold. The dashes (—) indicate which sensors never approached an unsafe state. From Table II, we can conclude that active sensor fuzzing is able to find attacks that drive SWaT into 15 different unsafe states involving water flow, pressure, and tank levels (RQ2).

For the unsafe states of FIT or DPIT, the median time for active sensor fuzzing to successfully launch the attack across different setups (models and search algorithms) is essentially the same. However, when the search algorithm is random, some repetitions fails to reach the unsafe states (the number indicated with an asterisk ($*$)). While almost every repetition equipped with certain searching strategy manages to cross the safety boundary. The only exception is one repetition of which the setup is LSTM-SA (the corresponding number is indicated with an asterisk as shown in Table II). In this repetition, the search algorithm of SA failed to generate an appropriate solution before the maximum iteration was reached.

When the attack targets are LITs, the ML-GA and ML-SA setups noticeably outperform their Random variants at driving every SWaT's water tank level into an unsafe state for both overflow and underflow attacks (Table II). For LIT sensors under a relatively simple circumstance (*i.e.,* LIT101), the Random setups are still able to cross the safety boundary within the given time but take more time than those using metaheuristic algorithms. For the sensors under the complex circumstances (*i.e.,* LIT301 and LIT401, which are relevant to three or four of the six stages), the Random setups fail to reach the threshold (indicated with $1200+$) when targeting the underflow states. Meanwhile, ML-GA and ML-SA setups are able to generate at least one successful attack within the given time, which proves the superiority of applying metaheuristic search in the setup of active sensor fuzzing (RQ4). We thus have the following conclusion:

TABLE II
RESULTS: MEDIAN TIME TAKEN (S) TO DRIVE SWAT'S FLOW, DIFFERENTIAL PRESSURE, AND LEVEL INDICATOR TRANSMITTERS INTO UNSAFE STATES

		Flow (High)			Flow (Low)					Pr. (L)	Tanks (Overflow)			Tanks (Underflow)		
		FIT101	FIT201	FIT601	FIT101	FIT201	FIT301	FIT401	FIT501	DPIT301	LIT101	LIT301	LIT401	LIT101	LIT301	LIT401
Our work	SVM-GA	14	9	18	14	8	18	16	5	14	353	519	497	550	576	751*
	SVM$_{AL}$-GA	12	11	18	13	7	16	15	5	14	340	431	484	538	534	744*
	SVM-SA	13	15	18	14	8	19	16	6	13	407	477*	503	571*	585	802*
	SVM$_{AL}$-SA	15	12	17	14	8	17	16	5	11	391	472	452	526	571	793*
	SVM-Random	15	13*	21*	15	9*	20*	15	5	14*	520	577	681	655	—	—
	SVM$_{AL}$-Random	16	14	19*	15	7	17*	15	5	12	512*	669*	682*	607*	—	1200+
	LSTM-GA	14	21	18	14	8	19	15	5	13	396	443	483	527	548	739*
	LSTM$_{AL}$-GA	10	17	16	15	8	17	14	4	13	366	405	478	509	502	737*
	LSTM-SA	17*	11	17	13	7	20	16	5	12	370	525	507	533*	570*	772*
	LSTM$_{AL}$-SA	15	12	17	13	7	16	14	5	10	381	539	499	514	533	750*
	LSTM-Random	12*	19*	19*	15	9	18	16	6	11*	662*	730	614*	679*	—	1200+
	LSTM$_{AL}$-Random	14	17	17*	14*	8	18	15	5	13*	614*	719*	599*	623*	1200+	—
Other	Random (No Model)	16*	17*	22*	14*	9*	—	—	6*	14*	—	—	—	—	—	1200+
	Benchmark [29]	14	18	—	14	18	—	—	—	—	454*	771*	1200+	—	—	1200+
	Smart Fuzzing [38]	13	10	16	13	7	16	14	5	11	333	439	681	511	1200+	1200+

*If attacks can only occur under strict conditions, an accurate prediction model and a sophisticated search algorithm (*e.g., *GA or SA) are required for generating attacks.*

Experiment #2: RQs 3&4. To further investigate the ability of our tool to find attacks, our second experiment studies whether active sensor fuzzing is able to generate more sophisticated and restricted benchmarks. In our experiment #1, we attack each individual sensor separately and do not aim any specific unsafe states. While in some real-world attack scenarios, the attackers are meant to drive the system out of some certain threshold (*e.g.,* cause the water tank to overflow rather than underflow), or drive multiple components of the system into dangerous states at the same time (*e.g.,* drive multiple water tanks to overflow or underflow simultaneously). Therefore, we design two sub-experiments to evaluate whether our tool still work given the aforementioned restriction.

Considering the complexity induced by different combinations of sensors, we single out three water tank level indicator transmitters (LIT101, LIT301 and LIT401), which work in three different stages of SWaT, as our victim sensors. The reason is that the overflow and underflow of water tanks usually cause more damage in real-world CPSs, hence the attack benchmarks of these three sensors are more representative.

The first sub-experiment aims to launch targeted attacks. We ran the experiment as follows. For each sensor, we separately targeted overflow and underflow states. The setting of target states could be achieved by modifying d_s of the fitness function:

$$d_s = \begin{cases} |v_s - L_s| & L_s \leqslant v_s, target = underflow \\ |v_s - H_s| & v_s \leqslant H_s, target = overflow \\ 0 & otherwise \end{cases} \quad (3)$$

We chose the LSTM models retrained with active learning as our ML-guiding tool. We launched the fuzzer and let it run as described in Experiment #1. For every combination of fuzzer setup and fitness function, we repeated the steps 10 times and recorded how many times the water level successfully reached the specified states.

Results. The results of this sub-experiment are given in the first three rows of Table III. We record the success rate for each sensor to reach each specified unsafe states. The intuition is that from the success rate we can infer the difficulty to construct a certain attack suite for our tool. From the table, we find that when our tool is applied with GA or SA searching algorithm, it could drive three water tanks into any unsafe states we assign with a high success rate. The repetitions under Random setup were still able to across the safety boundary, but with a relatively low success rate. Especially when the attack can only occur under strict conditions (*i.e.,* the underflow attack of LIT301 and LIT401). Such findings further proves the necessity of utilizing a more sophisticated search algorithm if the attacker want to launch more sophisticated attacks. We thus have the following conclusions:

*When equipped with the sophisticated search algorithm (*e.g., *GA or SA), our tool is able to launch targeted attack with high success rate.*

TABLE III
RESULTS: SUCCESS RATE (%) OF LAUNCHING ATTACKS TOWARDS TARGETED UNSAFE STATES (OVERFLOW OR UNDERFLOW)

		Overflow			Underflow		
		LIT101	LIT301	LIT401	LIT101	LIT301	LIT401
Our work	LSTM$_{AL}$-GA	100	100	90	100	90	80
	LSTM$_{AL}$-SA	100	100	100	100	90	70
	LSTM$_{AL}$-Random	60	50	50	50	10	10
Other	Random (No Model)	10	0	0	10	0	0

TABLE IV
RESULTS: SUCCESS RATE (%) OF LAUNCHING MULTI-POINT ATTACKS WITH DIFFERENT COMBINATIONS OF UNSAFE STATES

		LLL	LLH	LHL	LHH	HLL	HLH	HHL	HHH
Our work	SVM-GA	—	84	88	64	—	80	80	88
	SVM$_{AL}$-GA	4	88	88	92	—	84	88	92
	SVM-SA	—	72	80	64	—	76	76	96
	SVM$_{AL}$-SA	—	84	80	84	4	76	84	96
	SVM-Random	—	8	4	4	—	4	4	12
	SVM$_{AL}$-Random	4	8	4	12	—	8	4	20
Other	Random (No Model)	—	4	—	—	—	—	4	8

The second sub-experiment aims to investigate the ability of our tool to attack multiple components of the CPS at the same time. We ran the experiment as follows. Three LITs were chosen as our multi-point attack targets. Each sensor had two unsafe states (underflow and overflow), so there were 8 combinations of unsafe states in total. For each combination, we manually set up the fitness function $\sum_{s \in S} \frac{1}{d_s / r_s}$ where S included LIT101, LIT301 and LIT401. We chose the SVM models as our ML-guiding tool. Then We launched the fuzzer and let it run as described in Experiment #1. Notice that we only stopped the fuzzer when all the unsafe states of different sensors were reached or timeout. For every combination of fuzzer setup and fitness function, we repeated the steps 25 times and recorded the number of times that the attack successfully drove SWAT into multi-point unsafe states.

Results. The results of this experiment are given in the first six rows of Table IV. We record the success rate of launching the specified multi-point attacks. For each sensor, we use L to denote that the water level reaches the lower threshold (*i.e.,* underflow) and H to denote that the water level reaches the upper threshold (*i.e.,* overflow). We use three letters to represent three unsafe states for each sensor (*e.g.,* LLH represents that LIT101 is underflow, LIT301 is underflow and LIT401 is overflow). From Table IV, we can conclude that our tool is able to find multi-point attacks for six combinations of unsafe states out of eight. Focus on the success rate, we find that the ML-GA and ML-SA setups outperform their Random variants in most cases. Furthermore, when the model is trained with active learning, the fuzzer is able to generate attacks with a higher success rate. Note that when the unsafe state targets are set to LLL or HLL, almost every repetition fails to cross the safety boundary. We find such failures result from the control logic of SWaT. When the underflow of LIT401 is chosen as the target states, the most efficient way to achieve the attack goal is to set the tank level of LIT401 to a large number. The CPS would close P301 and open P401 and the water in LIT401 will be pumped out without being pumped in. However, when P301 is close, there will be no way for the water in LIT301 to be pumped out, hence the underflow of LIT301 would be hard to reach. Therefore, under the setup of LLL and HLL, it is unrealistic to generate a successful attack in most cases. The only three successful repetitions occurred when the tank levels of LIT301 were low enough that they could reach the lower boundary within a short period of time. Such a contradiction does not appear in LHH or HHH attacks, because the CPS does not forbid P301 to open when LIT301 is low. Therefore, the attack benchmark can successfully drive the water level of LIT401 to reach the upperbound. We thus have the following conclusion:

> *Active learning and metaheuristic search algorithms improve the success rate when launching multi-point attacks.*

Experiment #3: RQ 5. Our third experiment seeks to compare the results of our active sensor fuzzing setups against those of other approaches. We make a comparison against two different baselines. First, we compare against an automatic approach in which all of the "smartness" is removed, and the manipulation to the sensor values are simply generated and applied *randomly*, without reference to any prediction model for the testbed (in contrast to LSTM-Random, SVM-Random). Second, and in contrast, we compare against the attacks of an established, expert-crafted benchmark [29], which were systematically derived from an attack model. For the different attacks derived from these two sources, this experiment is roughly similar to the first: launch them, and record which sensors are driven into unsafe ranges (and at which time points). The idea of the experiment is to establish where the effectiveness of active sensor fuzzing can be positioned between two extremes: a simplistic, uninformed search on the one hand; and an expert-crafted, comprehensive benchmark on the other.

We ran the experiment as follows. For the random baseline, we wrote a program to randomly generate 10 distinct sets of sensor readings. For each set in turn, we began by 'resetting' SWaT to a normal state, using the same procedure as the previous experiment. Once in such a state, we then fuzzed the network to manipulate the sensor readings with the given random configuration for 20 minutes. If during the run any real sensor readings were driven into an unsafe range, we recorded the sensors (and ranges) in question, and the time points at which they *first* became unsafe. After the run, the system was allowed to reset, before repeating the process for the other randomly generated configurations.

For the benchmark [29], we manually extracted all attack sequences that were intended drive the system into unsafe physical states. For each of these six attacks in turn, we fuzzed the network manually to recreate the attack sequence, overriding the sensor and actuator states as prescribed by the benchmark. If during the run any (*actual*) sensor readings were driven into an unsafe range, we recorded the sensors and time points at which they *first* became unsafe. This was repeated

10 times for each attack sequence.

Results. The results of this experiment are given in the bottom rows of Tables II. The numbers indicate the time at which sensor entered an unsafe state for the first time. For "Random (No Model)", they are the medians across runs for 10 randomly generated sensor readings; for "Benchmark", they are the medians across 10 repetitions for each of attacks that target some unsafe state. By comparing these numbers with active sensor fuzzing **(RQ5)**, we show that our approach drives *more* of the sensors to unsafe states and does so *faster*.

The comparison with Random (No Model) makes clear that most of the attacks are not possible to find by "accident", or just by simply fuzzing at random without any artificial or human intelligence. At the other end, the comparison with the SWaT benchmark illustrates that active sensor fuzzing was able to drive the system to nine additional types of unsafe states that were not covered by the benchmark. Also, its comparable attacks were slower than those of active sensor fuzzing. Most of our attacks achieves the shortest time while attacking the LITs (Especially under the setup of LSTM$_{AL}$-GA). The comparison suggests that our approach could complement the benchmark and improve its comprehensiveness and efficiency.

> *Active sensor fuzzing complemented an established benchmark by covering nine additional unsafe states.*

Apart from the comparison of the ability to generate basic attacks (*i.e.,* single-point and attacks without targeting any specified states) of other approaches. We also seek to find out whether those approaches can perform more sophisticated attacks as our tool does. The manually crafted benchmark [29] does not give any implementations on how to launch those sophisticated attacks, so we compare against the Random approach.

We ran the experiment as follows.We generated 10 distinct sets of sensor readings for the experiment of targeted attacks and 20 sets for the experiment of multi-point attacks. Then the network was fuzzed to manipulate the sensor readings. The fuzzer only stopped when the assigned unsafe states were reached or the 20 minutes timeout. We then recorded the number of sensor reading sets that successfully drove the system across the safety boundary.

Results. The results of the random baseline is given in the last row of Table III and Table IV. For the targeted attack, only one repetition drives LIT101 to overflow and one repetition drives LIT101 to underflow while others fail to attack. For the multi-point attack, the repetition do not success under most cases. Only by little chance could the randomly generated attack suite succeed in driving all three tanks to unsafe states simultaneously. Compared with the random approach combined with any prediction models (*e.g.,* LSTM-Random, SVM-Random), it is clear that the "smartness" induced by those prediction model make the attack suites more feasible. We thus have the following conclusions:

Experiment #4: RQ 6. In our fourth experiment, we reveal and analyze how active learning benefit the fuzzing framework compared with our previous work [38]. In this work, we apply new prediction models in the step of learning. Therefore, we design this experiment to figure out the influence of these changes.

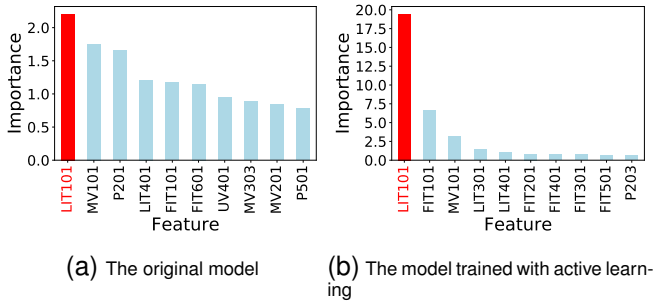
First, we study whether active sensor fuzzing can complement the attack benchmarks generated by the original fuzzing framework. We singled out the attack with the best performance (attacks with shortest time to drive the system into an unsafe state) of each individual sensor from our original fuzzing tool [38]. The results are given in the last row of Table II, indicated by "Smart Fuzzing". In our original fuzzing framework, there are 13 kinds of attack benchmarks generated for SWaT in total. In this work, with applying active learning, our framework successfully generates attack suites that can drive the system reach two more unsafe states within the given time (*i.e.,* the underflow thresholds of LIT301 and LIT401). Thus, we have the following conclusion:

> *Active sensor fuzzing complemented the original approach and discovered two more attacks within the given time.*

Next, we explain why active learning could promote the procedures of attack discovery. From the aforementioned experiments (see Table I, II, III and IV), we can conclude that active learning enhances the prediction models to construct sophisticated attack suites in less time. For a better understanding of the effect of active learning, we use *feature importance* [52], a technique that calculates a score for all the input features for a prediction model based on how they influence the result, to interpret the original model and the model trained with active learning. The intuition is that under most cases, the components of CPS evolve according to their related sensor values (*e.g.,* a pump opens when the tank level is high and closes when the tank level is low). Thus, by calculating the feature importance before and after active learning, we can find out which sensor plays an essential role while the prediction model generates its output, and how active learning influence these feature importance.

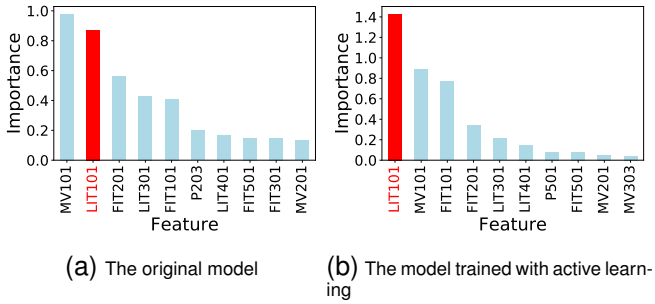
We calculated *feature importance* based on the model we used. For the linear SVC model, we took the absolute value of the model's weight for the feature as its importance. For the LSTM model, we referred to [53] and used the prediction loss after shuffling a certain feature as its importance. We picked MV101, a motorized valve in stage 1, as an example to explain the impact of active learning.

Results. The calculated feature importance of the SVC model and the LSTM model that predict the evolution of MV101 are given in Figure 5 and Figure 6, respectively. In Figure 5, compared to the original model, the feature importance of LIT101 (marked in red color) is significantly larger than these of other sensors. In Figure 6, the feature importance of LIT101 is ranked first for the active learning model, while ranked second for the original model. After analyzing the logic of the Swat testbed, we found that the behavior of MV101 is determined by the status of LIT101, *i.e.,* MV101 closes when LIT101 is high and opens when LIT101 is low. The larger the feature importance is, the higher probability the corresponding sensor is selected to be manipulated in the fuzzing phase. Therefore, active learning is substantial for



(a) The original model (b) The model trained with active learning

Fig. 5. The feature importance of the SVC model that predicts the evolution of MV101 which is trained (a) normally and (b) with active learning techniques



(a) The original model (b) The model trained with active learning

Fig. 6. The feature importance of the LSTM model that predicts the evolution of MV101 which is trained (a) normally and (b) with active learning techniques

finding the important targeted sensor and improves the success rate to launch an attack.

We thus have the following conclusions:

> *Active learning is effective at finding the critical sensors for manipulation.*

C. Discussion on attack detection mechanism

Some hybrid systems have built-in attack detection mechanisms, such as invariant monitoring [54]. These mechanisms statistically analyze whether all system states obey a set of predefined rules. For example, if the valve MV101 is open, the reasonable range of water flow indicator FIT101 is 0.4-2.0. If the actual reading falls outside of this range, the detection mechanism will trigger an alarm. Other mechanisms conduct statistical tests based on the likelihood of an observation given an internal state of the system. If these mechanisms are well-designed and the attacker has no means of compromising or falsifying the data, our approach is unlikely to generate any possible attack vectors. However, if the rules or statistical tests are insufficient to cover all vulnerabilities, then active fuzzing could be able to detect some vulnerabilities.

D. Threats to Validity

Finally, we remark on some threats to the validity of our evaluation and approach. First, our approach was only performed on one testbed. Second, our approach was implemented for a CPS testbed: while it is a real, fully operational plants based on the designs of industrial ones, it is still smaller, and our results may therefore not scale-up (this is difficult to test due to the confidentiality surrounding plants in cities). Finally, the initial states of the testbeds were not controlled, other than to be within their normal ranges, meaning that our performance results may vary lightly.

V. Related Work

In this section, we highlight a selection of the literature that is particularly relevant to some of the main themes of this paper: constructing attacks, fuzzing, and assessing robustness. We remark that related works on *attack defense mechanisms* are discussed earlier in the paper in Section I, with some mechanisms specific to SWaT discussed in Section II.

A number of papers have considered the systematic *construction* or *synthesis* of attacks for CPSs in order to test (or demonstrate flaws in) some specific attack detection mechanisms. Liu et al. [55], for example, target electric power grids, which are typically monitored based on state estimation, and demonstrate a new class of data injection attacks that can introduce arbitrary errors into certain state variables and evade detection. Huang et al. [56] also target power grids, presenting an algorithm that synthesises attacks that are able to evade detection by conventional monitors. The algorithm, based on ideas from hybrid systems verification and sensitivity analysis, covers both discrete and continuous aspects of the system. Urbina et al. [57] evaluate several attack detection mechanisms in a comprehensive review, concluding that many of them are not limiting the impact of stealthy attacks (*i.e.,* from attackers who have knowledge about the system's defenses), and suggest ways of mitigating this. Sugumar and Mathur [58] address the problem of assessing attack detection mechanisms based on process invariants: their tool simulates their behavior when subjected to single stage single point attacks, but must first be provided with some formal timed automata models. Cárdenas et al. [59] propose a general framework for assessing attack detection mechanisms, but in contrast to the previous works, focus on the business cases between different solutions. For example, they consider the cost-benefit trade-offs and attack threats associated with different methods, *e.g.,* centralized vs. distributed.

Fuzzing has been a popular research topic in security and software engineering for many years, but the goals of previous works tend to differ from ours, which is a general approach/tool for discovering CPS network attacks. Fuzzing has previously been applied to CPS models, but for the goal of *testing* them, rather than finding attacks. CyFuzz [60] and DeepFuzzSL [61] are two such tools, which offer support for testing Simulink models of CPSs. American fuzzy lop [41] targets programs, and uses GAs to increase the code coverage of tests and find more bugs. Cha et al. [40] also target software, using white-box symbolic analysis on execution traces to maximize the bugs it finds in programs. Grammar-based fuzzers (*e.g.,* [62], [63]) generate complex structured input, such as HTML-JavaScript for testing web browsers, with formal grammars. A number of works (*e.g.,* [64]) have targeted the fuzzing of network protocols in order to test their

intrusion detection systems. In contrast, our work starts from the assumption that an attacker has *already compromised* the network, and uses ML-guided fuzzing to find the different ways that such an attacker might drive the system to an unsafe state. The attacks that it uncovers then form test suites for attack detection mechanisms. The work of Chen et al. [31], which also introduces active learning to guide the fuzzing process, is close to our fuzzing framework. But in contrast to our work, the framework of [31] constructs regression models and fuzzes based on the network payloads of CPSs.

There are more *formal* approaches that could be used to analyze a CPS and construct a benchmark of different attacks. However, these typically require a *formal specification*, which, if available in the first place, may be too simple to capture all the complexities in the physical processes of full-fledged CPSs. Kang et al. [65], for example, construct a discretised first-order model of SWaT's first three stages in Alloy, and analyze it with respect to some safety properties. However, the work uses very simple abstractions of the physical processes, and only partially models the system (we consider *all* of it without the need for any formal model). Castellanos et al. [66] automatically extract models from PLC programs that highlight the interactions among different internal entities of a CPS, and propose reachability algorithms for analyzing the dependencies between control programs and physical processes. McLaughlin et al. [67] describe a trusted computing base for verifying safety-critical code on PLCs, with safety violations reported to operators when found. Etigowni et al. [68] define a CPS control solution for securing power grids, focusing on information flow analyses based on (potentially verifiable) policy logic and symbolic execution. Beyond these examples, if a CPS can be modeled as a hybrid system, there are several formal methods that can be applied to it, including model checking [69], [70], SMT solving [71], non-standard analysis [72], process calculi [73], reachability analysis [74], concolic testing [75], and theorem proving [76]. Defining a formal model that accurately characterizes enough of the physical process and its interactions with the PLCs is, however, the *hardest* part. Active sensor fuzzing in contrast can achieve results on real-world CPSs without the need for specifying a model at all: it learns one implicitly and automatically from the data logs.

VI. Conclusion

In this work, we proposed active sensor fuzzing, a black-box tool to automatically generate CPS attack suites based on manipulating sensor values. We used active learning to overcome the enormous search space and the huge cost to label data in the context of cyber physical system. We trained a series of cascaded models from which the relevance between the actuators and the sensor readings of the system could be learnt. Online active learning was used to enhance the performance of our models. We adapted the EBCM [31] active learning framework, which was designed to maximize the change of behavior patterns of CPSs. We implemented two metaheuristic approaches to guide the fuzzing process: genetic algorithm (GA) and simulated annealing (SA).

We than evaluated the efficiency and efficacy of our tool on a multi-stage water purification testbed: SWaT, which consists of multiple computation, communication and physical processes and related components. The experiments showed that our tool was able to find attacks that could drive the system into 15 different unsafe states, 9 more than an expert-crafted benchmark [43]. Furthermore, we implemented two sophisticated attacks: targeted and multi-point attack. We showed that active sensor fuzzing was able to launch targeted attacks with high success rate while guided by some sophisticated search algorithm and a well-trained model. As for the multi-point attacks, the benchmark generated by our tool could also succeed in most cases, as long as the attack wasn't in contradiction to the control logic of the system. Finally, we introduced how active sensor fuzzing extended our original fuzzing framework [38]. We complement the attack benchmarks with 2 more kinds of attack suites. Furthermore, we analyzed how active learning could help refine the distribution of feature importance of the prediction model. When the model was updated, the procedure of finding attacks in the context of CPS was also improved.

Acknowledgements

This work was supported in part by National Natural Science Foundation of China (62227805, 62072398), by National Key R&D Program of China (2020AAA0107700), by SUTD-ZJU IDEA Grant for visiting professors (SUTD-ZJUVP201901), by Alibaba-Zhejiang University Joint Institute of Frontier Technologies, by National Key Laboratory of Science and Technology on Information System Security (6142111210301), by State Key Laboratory of Mathematical Engineering and Advanced Computing, and by Key Laboratory of Cyberspace Situation Awareness of Henan Province (HNTS2022001).

References

[1] US National Science Foundation, "Cyber-physical systems (CPS)," https://www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf18538&org=NSF, 2018, document number: nsf18538.

[2] R. Rajkumar, I. Lee, L. Sha, and J. A. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proc. Design Automation Conference (DAC 2010)*. ACM, 2010, pp. 731–736.

[3] J. Leyden, "Water treatment plant hacked, chemical mix changed for tap supplies," *The Register*, 2016, acc.: September 2019. [Online]. Available: https://www.theregister.co.uk/2016/03/24/water_utility_hacked/

[4] ICS-CERT Alert, "Cyber-attack against Ukrainian critical infrastructure," https://ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01, 2016, document number: IR-ALERT-H-16-056-01.

[5] L. Cheng, K. Tian, and D. D. Yao, "Orpheus: Enforcing cyber-physical execution semantics to defend against data-oriented attacks," in *Proc. Annual Computer Security Applications Conference (ACSAC 2017)*. ACM, 2017, pp. 315–326.

[6] Y. Harada, Y. Yamagata, O. Mizuno, and E. Choi, "Log-based anomaly detection of CPS using a statistical method," in *Proc. International Workshop on Empirical Software Engineering in Practice (IWESEP 2017)*. IEEE, 2017, pp. 1–6.

[7] J. Inoue, Y. Yamagata, Y. Chen, C. M. Poskitt, and J. Sun, "Anomaly detection for a water treatment system using unsupervised machine learning," in *Proc. IEEE International Conference on Data Mining Workshops (ICDMW 2017): Data Mining for Cyberphysical and Industrial Systems (DMCIS 2017)*. IEEE, 2017, pp. 1058–1065.

[8] F. Pasqualetti, F. Dorfler, and F. Bullo, "Cyber-physical attacks in power networks: Models, fundamental limitations and monitor design," in *Proc. IEEE Conference on Decision and Control and European Control Conference (CDC-ECC 2011)*. IEEE, 2011, pp. 2195–2201.

[9] E. Aggarwal, M. Karimibiuki, K. Pattabiraman, and A. Ivanov, "CORGIDS: A correlation-based generic intrusion detection system," in *Proc. Workshop on Cyber-Physical Systems Security and PrivaCy (CPS-SPC 2018)*. ACM, 2018, pp. 24–35.

[10] W. Aoudi, M. Iturbe, and M. Almgren, "Truth will out: Departure-based process-level detection of stealthy attacks on control systems," in *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS 2018)*. ACM, 2018, pp. 817–831.

[11] Z. He, A. Raghavan, S. M. Chai, and R. B. Lee, "Detecting zero-day controller hijacking attacks on the power-grid with enhanced deep learning," *CoRR*, vol. abs/1806.06496, 2018.

[12] M. Kravchik and A. Shabtai, "Detecting cyber attacks in industrial control systems using convolutional neural networks," in *Proc. Workshop on Cyber-Physical Systems Security and PrivaCy (CPS-SPC 2018)*. ACM, 2018, pp. 72–83.

[13] Q. Lin, S. Adepu, S. Verwer, and A. Mathur, "TABOR: A graphical model-based approach for anomaly detection in industrial control systems," in *Proc. Asia Conference on Computer and Communications Security (AsiaCCS 2018)*. ACM, 2018, pp. 525–536.

[14] V. Narayanan and R. B. Bobba, "Learning based anomaly detection for industrial arm applications," in *Proc. Workshop on Cyber-Physical Systems Security and PrivaCy (CPS-SPC 2018)*. ACM, 2018, pp. 13–23.

[15] P. Schneider and K. Böttinger, "High-performance unsupervised anomaly detection for cyber-physical system networks," in *Proc. Workshop on Cyber-Physical Systems Security and PrivaCy (CPS-SPC 2018)*. ACM, 2018, pp. 1–12.

[16] C. M. Ahmed, M. Ochoa, J. Zhou, A. P. Mathur, R. Qadeer, C. Murguia, and J. Ruths, "*NoisePrint*: Attack detection using sensor and process noise fingerprint in cyber physical systems," in *Proc. Asia Conference on Computer and Communications Security (AsiaCCS 2018)*. ACM, 2018, pp. 483–497.

[17] C. M. Ahmed, J. Zhou, and A. P. Mathur, "Noise matters: Using sensor and process noise fingerprint to detect stealthy cyber attacks and authenticate sensors in CPS," in *Proc. Annual Computer Security Applications Conference (ACSAC 2018)*. ACM, 2018, pp. 566–581.

[18] Q. Gu, D. Formby, S. Ji, H. Cam, and R. A. Beyah, "Fingerprinting for cyber-physical system security: Device physics matters too," *IEEE Security & Privacy*, vol. 16, no. 5, pp. 49–59, 2018.

[19] M. Kneib and C. Huth, "Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks," in *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS 2018)*. ACM, 2018, pp. 787–800.

[20] A. A. Cárdenas, S. Amin, Z. Lin, Y. Huang, C. Huang, and S. Sastry, "Attacks against process control systems: risk assessment, detection, and response," in *Proc. ACM Symposium on Information, Computer and Communications Security (AsiaCCS 2011)*. ACM, 2011, pp. 355–366.

[21] S. Adepu and A. Mathur, "Using process invariants to detect cyber attacks on a water treatment system," in *Proc. International Conference on ICT Systems Security and Privacy Protection (SEC 2016)*, ser. IFIP AICT, vol. 471. Springer, 2016, pp. 91–104.

[22] ——, "Distributed detection of single-stage multipoint cyber attacks in a water treatment plant," in *Proc. ACM Asia Conference on Computer and Communications Security (AsiaCCS 2016)*. ACM, 2016, pp. 449–460.

[23] Y. Chen, C. M. Poskitt, and J. Sun, "Towards learning and verifying invariants of cyber-physical systems by code mutation," in *Proc. International Symposium on Formal Methods (FM 2016)*, ser. LNCS, vol. 9995. Springer, 2016, pp. 155–163.

[24] S. Adepu and A. Mathur, "Distributed attack detection in a water treatment plant: Method and case study," *IEEE Transactions on Dependable and Secure Computing*, 2018.

[25] Y. Chen, C. M. Poskitt, and J. Sun, "Learning from mutants: Using code mutation to learn and monitor invariants of a cyber-physical system," in *Proc. IEEE Symposium on Security and Privacy (S&P 2018)*. IEEE Computer Society, 2018, pp. 648–660.

[26] H. Choi, W. Lee, Y. Aafer, F. Fei, Z. Tu, X. Zhang, D. Xu, and X. Xinyan, "Detecting attacks against robotic vehicles: A control invariant approach," in *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS 2018)*. ACM, 2018, pp. 801–816.

[27] J. Giraldo, D. I. Urbina, A. Cardenas, J. Valente, M. A. Faisal, J. Ruths, N. O. Tippenhauer, H. Sandberg, and R. Candell, "A survey of physics-based attack detection in cyber-physical systems," *ACM Computing Surveys*, vol. 51, no. 4, pp. 76:1–76:36, 2018.

[28] "iTrust Labs: Datasets," https://itrust.sutd.edu.sg/itrust-labs_datasets/, 2019, accessed: September 2019.

[29] J. Goh, S. Adepu, K. N. Junejo, and A. Mathur, "A dataset to support research in the design of secure water treatment systems," in *Proc. International Conference on Critical Information Infrastructures Security (CRITIS 2016)*, 2016.

[30] S. Adepu and A. Mathur, "Assessing the effectiveness of attack detection at a hackfest on industrial control systems," *IEEE Transactions on Sustainable Computing*, 2018.

[31] Y. Chen, B. Xuan, C. M. Poskitt, J. Sun, and F. Zhang, "Active fuzzing for testing and securing cyber-physical systems," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 14–26.

[32] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *Tools and Algorithms for the Construction and Analysis of Systems: 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26–April 3, 2011. Proceedings 17*. Springer, 2011, pp. 254–257.

[33] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems." in *CAV*, vol. 10. Springer, 2010, pp. 167–170.

[34] E. Lughofer, "On-line active learning: A new paradigm to improve practical useability of data stream modeling methods," *Information Sciences*, vol. 415, pp. 356–376, 2017.

[35] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[36] P. J. Van Laarhoven and E. H. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.

[37] "Secure Water Treatment (SWaT)," https://itrust.sutd.edu.sg/itrust-labs-home/itrust-labs_swat/, 2019, accessed: September 2019.

[38] Y. Chen, C. M. Poskitt, J. Sun, S. Adepu, and F. Zhang, "Learning-guided network fuzzing for testing cyber-physical system defences," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 962–973.

[39] A. Takanen, J. D. Demott, and C. Miller, *Fuzzing for Software Security Testing and Quality Assurance*, 2nd ed. Artech House, 2018.

[40] S. K. Cha, M. Woo, and D. Brumley, "Program-adaptive mutational fuzzing," in *Proc. IEEE Symposium on Security and Privacy (S&P 2015)*. IEEE Computer Society, 2015, pp. 725–741.

[41] M. Zalewski, "American fuzzy lop," http://lcamtuf.coredump.cx/afl/, 2017, accessed: September 2019.

[42] H. R. Ghaeini and N. O. Tippenhauer, "HAMIDS: hierarchical monitoring intrusion detection system for industrial control systems," in *Proc. Workshop on Cyber-Physical Systems Security and Privacy (CPS-SPC 2016)*. ACM, 2016, pp. 103–111.

[43] J. Goh, S. Adepu, M. Tan, and Z. S. Lee, "Anomaly detection in cyber physical systems using recurrent neural networks," in *Proc. International Symposium on High Assurance Systems Engineering (HASE 2017)*. IEEE, 2017, pp. 140–145.

[44] C. Feng, V. R. Palleti, A. Mathur, and D. Chana, "A systematic framework to generate invariants for anomaly detection in industrial control systems," in *Proc. Annual Network and Distributed System Security Symposium (NDSS 2019)*. The Internet Society, 2019.

[45] F. A. Gers, J. Schmidhuber, and F. A. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[46] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[47] H. Drucker, C. J. C. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support vector regression machines," in *Proc. Advances in Neural Information Processing Systems (NIPS 1996)*. MIT Press, 1996, pp. 155–161.

[48] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[49] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *Proc. ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[50] A. Ruscito, "pycomm," https://github.com/ruscito/pycomm, 2019, accessed: September 2019.

[51] "Supplementary material," https://sav-smu.github.io/supplementary-material/ase2019.html, 2019.

[52] M. Wojtas and K. Chen, "Feature importance ranking for deep learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5105–5114, 2020.

[53] A. Altmann, L. Toloşi, O. Sander, and T. Lengauer, "Permutation importance: a corrected feature importance measure," *Bioinformatics*, vol. 26, no. 10, pp. 1340–1347, 2010.

[54] S. Adepu and A. Mathur, "From design to invariants: Detecting attacks on cyber physical systems," in *IEEE International Conference on Software Quality, Reliability and Security (QRS 2017) Companion*. IEEE, 2017, pp. 533–540.

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

[56] Z. Huang, S. Etigowni, L. Garcia, S. Mitra, and S. A. Zonouz, "Algorithmic attack synthesis using hybrid dynamics of power grid critical infrastructures," in *Proc. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2018)*. IEEE Computer Society, 2018, pp. 151–162.

[57] D. I. Urbina, J. A. Giraldo, A. A. Cárdenas, N. O. Tippenhauer, J. Valente, M. A. Faisal, J. Ruths, R. Candell, and H. Sandberg, "Limiting the impact of stealthy attacks on industrial control systems," in *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS 2016)*. ACM, 2016, pp. 1092–1105.

[58] G. Sugumar and A. Mathur, "Testing the effectiveness of attack detection mechanisms in industrial control systems," in *Proc. IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C 2017)*. IEEE, 2017, pp. 138–145.

[59] A. A. Cárdenas, R. Berthier, R. B. Bobba, J. H. Huh, J. G. Jetcheva, D. Grochocki, and W. H. Sanders, "A framework for evaluating intrusion detection architectures in advanced metering infrastructures," *IEEE Transactions on Smart Grid*, vol. 5, no. 2, pp. 906–915, 2014.

[60] S. A. Chowdhury, T. T. Johnson, and C. Csallner, "CyFuzz: A differential testing framework for cyber-physical systems development environments," in *Proc. Workshop on Design, Modeling and Evaluation of Cyber Physical Systems (CyPhy 2016)*, ser. LNCS, vol. 10107. Springer, 2017, pp. 46–60.

[61] S. L. Shrestha, S. A. Chowdhury, and C. Csallner, "Deepfuzzsl: Generating models with deep learning to find bugs in the simulink toolchain," in *2nd Workshop on Testing for Deep Learning and Deep Learning for Testing (DeepTest)*, 2020.

[62] C. Holler, K. Herzig, and A. Zeller, "Fuzzing with code fragments," in *21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 445–458.

[63] P. Godefroid, H. Peleg, and R. Singh, "Learn&fuzz: Machine learning for input fuzzing," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 50–59.

[64] G. Vigna, W. K. Robertson, and D. Balzarotti, "Testing network-based intrusion detection signatures using mutant exploits," in *Proc. ACM Conference on Computer and Communications Security (CCS 2004)*. ACM, 2004, pp. 21–30.

[65] E. Kang, S. Adepu, D. Jackson, and A. P. Mathur, "Model-based security analysis of a water treatment system," in *Proc. International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS 2016)*. ACM, 2016, pp. 22–28.

[66] J. H. Castellanos, M. Ochoa, and J. Zhou, "Finding dependencies between cyber-physical domains for security testing of industrial control systems," in *Proc. Annual Computer Security Applications Conference (ACSAC 2018)*. ACM, 2018, pp. 582–594.

[67] S. E. McLaughlin, S. A. Zonouz, D. J. Pohly, and P. D. McDaniel, "A trusted safety verifier for process controller code," in *Proc. Annual Network and Distributed System Security Symposium (NDSS 2014)*. The Internet Society, 2014.

[68] S. Etigowni, D. J. Tian, G. Hernandez, S. A. Zonouz, and K. R. B. Butler, "CPAC: securing critical infrastructure with cyber-physical access control," in *Proc. Annual Conference on Computer Security Applications (ACSAC 2016)*. ACM, 2016, pp. 139–152.

[69] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *Proc. International Conference on Computer Aided Verification (CAV 2011)*, ser. LNCS, vol. 6806. Springer, 2011, pp. 379–395.

[70] J. Wang, J. Sun, Y. Jia, S. Qin, and Z. Xu, "Towards 'verifying' a water treatment system," in *Proc. International Symposium on Formal Methods (FM 2018)*, ser. LNCS, vol. 10951. Springer, 2018, pp. 73–92.

[71] S. Gao, S. Kong, and E. M. Clarke, "dReal: An SMT solver for nonlinear theories over the reals," in *Proc. International Conference on Automated Deduction (CADE 2013)*, ser. LNCS, vol. 7898. Springer, 2013, pp. 208–214.

[72] I. Hasuo and K. Suenaga, "Exercises in nonstandard static analysis of hybrid systems," in *Proc. International Conference on Computer Aided Verification (CAV 2012)*, ser. LNCS, vol. 7358. Springer, 2012, pp. 462–478.

[73] R. Lanotte, M. Merro, R. Muradore, and L. Viganò, "A formal approach to cyber-physical attacks," in *Proc. IEEE Computer Security Foundations Symposium (CSF 2017)*. IEEE Computer Society, 2017, pp. 436–450.

[74] T. T. Johnson, S. Bak, M. Caccamo, and L. Sha, "Real-time reachability for verified simplex design," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 2, pp. 1–27, 2016.

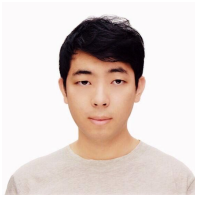
[75] P. Kong, Y. Li, X. Chen, J. Sun, M. Sun, and J. Wang, "Towards concolic testing for hybrid systems," in *Proc. International Symposium on Formal Methods (FM 2016)*, ser. LNCS, vol. 9995. Springer, 2016, pp. 460–478.

[76] J. Quesel, S. Mitsch, S. M. Loos, N. Arechiga, and A. Platzer, "How to model and prove hybrid systems with KeYmaera: a tutorial on safety," *International Journal on Software Tools for Technology Transfer*, vol. 18, no. 1, pp. 67–91, 2016.

Fan Zhang is a professor of the College of Computer Science and Technology, Zhejiang University, China. His main research interests include hardware security, system security, web security, cryptography and security of emerging system platforms. He is also a visiting professor of Singapore University of Technology and Design. He has served as the technical program committee members of many security conferences, including DAC, AsiaCCS, FDTC, ASHES, SPACE, AsianHOST and more. As a co-author, he received seven best/distinguished paper awards from conferences including COSADE, AsianHOST and Chinacrypt. He is the program chair for the International Workshop on Security Proofs for Embedded Systems (PROOFS) 2020. He also has a lot of journal publications including IEEE TIFS, TPDS, IACR CHES and more. He received his Ph.D. degree in Computer Science and Engineering from University of Connecticut.

Qianmei Wu is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Zhejiang University. She received the bachelor's degree from the College of Computer Science and Technology, Jilin University, in 2020. Her research interests include cyber-physical system security, hardware security, cryptography.

Bohan Xuan is currently a post-graduate student pursuing his master's degree in Zhejiang University, China. He received his bachelor's degree in Zhejiang University. His research interests include the security of neural network and the defense and attestation of cyber-physical systems.



Yuqi Chen is an Assistant Professor at the School of Information Science and Technology at ShanghaiTech University. He received his B.Sc. in computer science from the South China University of Technology in 2015 and his Ph.D. from the Singapore University of Technology and Design in 2019. Before joining ShanghaiTech, Yuqi was a Research Scientist in the System Analysis and Verification group at Singapore Management University. Yuqi's research interests lie at the intersection of software engineering and security. He employs a range of techniques, including testing, reverse engineering, program analysis, and formal methods, to develop practical solutions for securing critical cyber-physical systems.

Wei Lin is currently a PhD student in SUTD, Before joining SUTD, he earned his master's degree and bachelor's degree in National University of Singapore and Nanyang Technological University, respectively. His current research interests are network security and IoT Security.

Christopher M. Poskitt is an Assistant Professor of Computer Science (Education) at Singapore Management University (SMU), where he is a member of the Software Analysis and Verification Group. Prior to SMU, he held research and teaching positions at ETH Zürich, Switzerland, and the Singapore University of Technology and Design. His research broadly addresses the problem of engineering correct and secure software/systems, towards which he has co-developed techniques for testing/defending cyber-physical systems, tools for analysing execution models of concurrency APIs, and logics for reasoning about the correctness of graph-rewriting programs.

Jun Sun is currently a full professor at School of Computing and Information Systems, Singapore Management University. He received Bachelor and PhD degrees in computing science from National University of Singapore (NUS) in 2002 and 2006. In 2007, he received the prestigious LEE KUAN YEW postdoctoral fellowship. He has been a faculty member since 2010 and was a visiting scholar at MIT from 2011-2012. Jun's research interests include software engineering, cyber-security and formal methods. He is the co-founder of the PAT model checker.

Binbin Chen (M'11) received the B.Sc. degree in computer science from Peking University and the Ph.D. degree in computer science from the National University of Singapore. Since July 2019, he has been an Associate Professor in the Information Systems Technology and Design (ISTD) pillar, Singapore University of Technology and Design (SUTD). He currently also holds a joint appointment as Principal Research Scientist at Advanced Digital Sciences Center, which is a University of Illinois research center located in Singapore. His current research interests include wireless networks, cyber-physical systems, and cyber security for critical infrastructures.