Active Fuzzing for Testing and Securing Cyber-Physical Systems

Yuqi Chen
Singapore Management University
Singapore
yuqichen@smu.edu.sg

Bohan Xuan
Zhejiang University
China
xuanbohan@zju.edu.cn

Christopher M. Poskitt
Singapore Management University
Singapore
cposkitt@smu.edu.sg

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

Fan Zhang*
Zhejiang University
China
fanzhang@zju.edu.cn

ABSTRACT

Cyber-physical systems (CPSs) in critical infrastructure face a pervasive threat from attackers, motivating research into a variety of countermeasures for securing them. Assessing the effectiveness of these countermeasures is challenging, however, as realistic benchmarks of attacks are difficult to manually construct, blindly testing is ineffective due to the enormous search spaces and resource requirements, and intelligent fuzzing approaches require impractical amounts of data and network access. In this work, we propose *active fuzzing*, an automatic approach for finding test suites of packet-level CPS network attacks, targeting scenarios in which attackers can observe sensors and manipulate packets, but have no existing knowledge about the payload encodings. Our approach learns regression models for predicting sensor values that will result from sampled network packets, and uses these predictions to guide a search for payload manipulations (i.e. bit flips) most likely to drive the CPS into an unsafe state. Key to our solution is the use of *online active learning*, which iteratively updates the models by sampling payloads that are estimated to maximally improve them. We evaluate the efficacy of active fuzzing by implementing it for a water purification plant testbed, finding it can automatically discover a test suite of flow, pressure, and over/underflow attacks, all with substantially less time, data, and network access than the most comparable approach. Finally, we demonstrate that our prediction models can also be utilised as countermeasures themselves, implementing them as anomaly detectors and early warning systems.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Security and privacy** → *Intrusion detection systems*; • **Computing methodologies** → *Active learning settings*;

KEYWORDS

Cyber-physical systems; fuzzing; active learning; benchmark generation; testing defence mechanisms

ACM Reference Format:
Yuqi Chen, Bohan Xuan, Christopher M. Poskitt, Jun Sun, and Fan Zhang. 2020. Active Fuzzing for Testing and Securing Cyber-Physical Systems. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '20), July 18–22, 2020, Los Angeles, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3395363.3397376

1 INTRODUCTION

Cyber-physical systems (CPSs), characterised by their tight and complex integration of computational and physical processes, are often used in the automation of critical public infrastructure [78]. Given the potential impact of cyber-attacks on these systems [46, 51, 60], ensuring their security and protection has become a more important goal than ever before. The different temporal scales, modalities, and process interactions in CPSs, however, pose a significant challenge for solutions to overcome, and have led to a variety of research into different possible countermeasures, including ones based on anomaly detection [11, 15, 27, 45, 48, 52, 58, 61, 66, 69, 71], fingerprinting [12, 13, 44, 56], invariant-based monitoring [7, 8, 10, 18, 24, 25, 28, 39, 82], and trusted execution environments [74].

Assessing how effective these different countermeasures are at detecting and preventing attacks is another challenge in itself. A typical solution is to use established benchmarks of attacks [2, 41], which have the advantage of facilitating direct comparisons between approaches, e.g. as done so in [52, 58, 61]. Such benchmarks, unfortunately, are few and far between: constructing them manually requires a great deal of time and expertise in the targeted CPS (all while risking insider bias), and generalising them from one CPS to another is a non-starter given the distinct processes, behaviours, and complexities that different systems exhibit.

An alternative solution is to generate benchmarks using *automated testing and fuzzing*, with these techniques overcoming the complexity of CPSs by having access to machine learning (ML) models trained on their data (e.g. logs of sensor readings, actuator, states, or network traffic). Existing solutions of this kind, however, tend to make unrealistic assumptions about an attacker's capabilities, or require a large body of training data that might not be available. The fuzzer of [26], for example, can automatically identify actuator configurations that drive the physical states of CPSs to different extremes, but the technology assumes the attacker to

[*] Also with Zhejiang Lab, and Alibaba-Zhejiang University Joint Institute of Frontier Technologies.

ISSTA '20, July 18–22, 2020, Los Angeles, CA, USA

ACM ISBN 978-1-4503-8008-9/20/07…$15.00
https://doi.org/10.1145/3395363.3397376

have total control of the network and actuators, and is underpinned by a prediction model trained on complete sets of data logs from *several days* of operation. Blindly fuzzing without such a model, however, is ineffective at finding attacks: first, because the search spaces of typical CPSs are *enormous*; and second, because of the wasted time and resources required to be able to observe the effects on a system's physical processes.

In this paper, we present *active fuzzing*, an automatic approach for finding test suites of packet-level CPS network attacks, targeting scenarios in which training data is limited, and in which attackers can observe sensors and manipulate network packets but have no existing knowledge about the encodings of their payloads. Our approach constructs regression models for predicting future sensor readings from network packets, and uses these models to guide a search for payload manipulations that systematically drive the system into unsafe states. To overcome the search space and resource costs, our solution utilises *(online) active learning* [63], a form of supervised ML that iteratively re-trains a model on examples that are estimated to maximally improve it. We apply it to CPSs by flipping bits of existing payloads in a way that is guided by one of two frameworks: Expected Model Change Maximization [17], and a novel adaptation of it based on maximising behaviour change. We query the effects of sampled payloads by spoofing them in the network, updating the model based on the observed effect.

We evaluate our approach by implementing it for the Secure Water Treatment (SWaT) testbed [4], a scaled-down version of a real-world water purification plant, able to produce up to five gallons of drinking water per minute. SWaT is a complex multi-stage system involving chemical processes such as ultrafiltration, de-chlorination, and reverse osmosis. Communication in the testbed is organised into a layered network hierarchy, in which we target the ring networks at the 'lowest' level that exchange data using EtherNet/IP over UDP. Our implementation manipulates the binary string payloads of 16 different types of packets, which when considered together have up to 2^{2752} different combinations.

Despite the enormous search space, we find that active fuzzing is effective at discovering packet-level flow, pressure, and over/underflow attacks, achieving comparable coverage to an established benchmark [41] and an LSTM-based fuzzer [26] but with substantially less training time, data, and network access. Furthermore, by manipulating the bits of payloads directly, active fuzzing bypasses the logic checks enforced by the system's controllers. These attacks are more sophisticated than those of the LSTM-based fuzzer [26], which can only generate high-level actuator commands and is unable to manipulate such packets. Finally, we investigate the utility of the learnt models in a different role: defending a CPS directly. We use them to implement anomaly detection and early warning systems for SWaT, finding that when models are suitably expressive, they are effective at detecting both random and known attacks.

Summary of Contributions. We present active fuzzing, a black-box approach for automatically discovering packet-level network attacks on real-world CPSs. By iteratively constructing a model with active learning, we demonstrate how to overcome enormous search spaces and resource costs by sampling new examples that maximally improve the model, and propose a new algorithm that guides this process by seeking maximally different behaviour. We evaluate the efficacy of the approach by implementing it for a complex real-world critical infrastructure testbed, and show that it achieves comparable coverage to an established benchmark and LSTM-based fuzzer but with significantly less data, time, and network access. Finally, we show that the learnt models are also effective as anomaly detectors and early warning systems.

Figure 1: The Secure Water Treatment (SWaT) testbed

Organisation. In Section 2, we introduce the SWaT testbed, with a particular focus on its network and the structure of its packets. In Section 3, we present the components of our active fuzzing approach, and explain how to implement it both in general and for SWaT. In Section 4, we evaluate the efficacy of our approach at finding packet-level attacks, and investigate secondary applications of our models as anomaly detectors and early warning systems. In Section 5, we discuss some related work, then conclude in Section 6.

2 BACKGROUND

In the following, we present an overview of SWaT, a water treatment testbed that forms the critical infrastructure case study we evaluate active fuzzing on. We describe in more detail its network hierarchy and the structure of its packets, before stating the assumptions we make about the capabilities of attackers.

SWaT Testbed. The CPS forming the case study of this paper is Secure Water Treatment (SWaT) [4], a scaled-down version of a real-world water purification plant, able to produce up to five gallons of safe drinking water per minute. SWaT (Figure 1) is intended to be a *testbed* for advancing cyber-security research on critical infrastructure, with the potential for successful technologies to be transferred to the actual plants it is based on. The testbed has been the subject of multiple hackathons [9] involving researchers from both academia and industry, and over the years has established a benchmark of attacks to evaluate defence mechanisms against [41].

SWaT treats water across multiple distinct but co-operating stages, involving a variety of complex chemical processes, such as de-chlorination, reverse osmosis, and ultrafiltration. Each stage in the CPS is controlled by a dedicated Allen-Bradley ControlLogix programmable logic controller (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

```
###[ Ethernet ]###
  dst       = e4:90:69:a3:0c:f6
  src       = 00:1d:9c:c8:03:e7
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0xbc
     len       = 68
     id        = 18067
     flags     =
     frag      = 0
     ttl       = 64
     proto     = udp
     chksum    = 0xb1f3
     src       = 192.168.0.10
     dst       = 192.168.0.12
     \options   \
###[ UDP ]###
        sport     = 2222
        dport     = 2222
        len       = 48
        chksum    = 0xfca2
###[ ENIP_CPF ]###
           count     = 2
           \items     \
            |###[ CPF_AddressDataItem ]###
            |  type_id   = Sequenced Address Item
            |  length    = 8
            |###[ CPF_SequencedAddressItem ]###
            |     connection_id= 469820023
            |     sequence_number= 18743
            |###[ CPF_AddressDataItem ]###
            |  type_id   = Connected Transport Packet
            |  length    = 22
            |###[ Raw ]###
            |     load      = '~2\x01\x00\x00\x00\x00\x00
                               \r\x00\x00\x00\x00\x00\x00
                               \x00\x00\x00\x00\x00\x00
                               \x00'
```

Figure 2: A SWaT packet after dissection by Scapy

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 motorised 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. SCADA software and tools developed by Rockwell Automation are available to support some analyses.

The sensors in SWaT 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 (or 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.

SWaT implements a number of standard safety and security measures for water treatment plants, such as alarms (reported to the operator) for when these thresholds are crossed, and logic checks for commands that are exchanged between the PLCs. In addition, several attack defence mechanisms developed by researchers have been installed (see Section 5).

The network of the SWaT testbed is organised into a layered hierarchy compliant with the ISA99 standard [53], providing different levels of segmentation and traffic control. The 'upper' layers of the hierarchy, Levels 3 and 2, respectively handle operation management (e.g. the historian) and supervisory control (e.g. touch panel, engineering workstation). Level 1 is a star network connecting the PLCs, and implements the Common Industrial Protocol (CIP) over EtherNet/IP. Finally, the 'lowest' layer of the hierarchy is Level 0, which consists of ring networks (EtherNet/IP over UDP) that connect individual PLCs to their relevant sensors and actuators.

Tools such as Wireshark [6] and Scapy [3] can be used to dissect the header information of a Level 0 SWaT packet, as illustrated in Figure 2. Here, the source IP address (192.168.0.10) and target IP address (192.168.0.12) correspond respectively to PLC1 and its remote IO device. Actuator commands (e.g. "open valve MV101") are encoded in the binary string payloads of these packets. In Figure 2, the payload is 22 bytes long, but Level 0 packets can also have a payload length of 10 or 32 bytes. Randomly manipulating the payloads has limited use given the size of the search space (2^{2752} possibilities when considering the 16 types of packets we sample; see Section 3.1). Our solution uses active learning to overcome this enormous search space, establishing how different bits impact the physical state without requiring any knowledge of the encoding.

Attacker Model. In this work, we assume that attackers have knowledge of the network protocol (e.g. EtherNet/IP over UDP at Level 0 of SWaT), and thus are able to intercept (unencrypted) packets, dissect their header information, and manipulate their payloads. We assume that the packet payloads are binary strings, but *do not* assume any existing knowledge about their meaning or encoding schemes. We assume that attackers can always access the 'true' sensor readings while the system is operating, in order to be able to observe the effects of a packet manipulation, or to judge whether or not an attack was successful. These live sensor readings can be observed over several minutes at a time in order to perform some pre-training and active learning, but in contrast to other approaches (e.g. [26]), we do not require access to extensive sets of data for offline learning, and we do not require the ability to arbitrarily issue high-level actuator commands across the system—we do so *only* by manipulating payloads.

3 ACTIVE FUZZING

Our approach for automatically finding packet-level network attacks in CPSs consists of the following steps. First, data is *collected*: packets are sniffed from the network, their payloads are extracted, and (true) sensor readings are queried. Second, we *pre-train* initial regression models, that take concatenations of packet payloads and predict the future effects on given sensors. Third, we apply an *online active learning* framework, iteratively improving the current model by sampling payloads estimated to maximally improve it. Finally, we search for candidate attacks by flipping important bits in packet payloads, and using our learnt models to identify which of them will drive the system to a targeted unsafe state.

Algorithm 1 presents the high-level algorithm of these steps for active fuzzing. Note that the notation in Line 8 indicates concatenation of sequences. In particular, t_s copies of the vector p are appended to sequence P to add additional weight to the new example when the model is re-trained.

Algorithm 1: High-Level Overview of Active Fuzzing

```
Input: Sensor $s$, prediction time $t_s$, pre-training time $t_p$
Output: Prediction model $M_s$
Sniff packets and observe values of $s$ for $t_p$ minutes;
Construct a sequence $P$ of feature (bit-)vectors from packet
  payloads;
Construct a sequence $V$ such that each $V[i]$ contains the value of $s$
  observed $t_s$ seconds after $P[i]$ was sniffed;
(Pre-)train a regression model $M_s$ predicting $V$ from $P$;
repeat
    Sample a new feature vector $p$ using an active learning
      framework (Section 3.2);
    Wait for $t_s$ seconds then observe the value $v_s$ of $s$;
    $P := P^\frown \langle p \rangle^{t_s}$; [concatenation of $t_s$ copies]
    $V := V^\frown \langle v_s \rangle^{t_s}$;
until timeout;
Re-train $M_s$ to predict $V$ from $P$;
return model $M_s$;
```

In the following, we describe the steps of the algorithm in more detail, and present the details of one particular implementation for the SWaT water purification testbed.

3.1 Packet Sniffing and Pre-Training

Collecting Raw Data. Both the pre-training and active learning phases of our approach require access to two types of data from the CPS under test. First, they must be able to sniff the network packets and extract their binary string payloads. Second, they must be able to access the true readings of any sensors under consideration, as the idea is to be able to observe the effects on sensor readings of different payload manipulations.

For SWaT, our approach extracts packets from Level 0 of the network hierarchy, i.e. the packets exchanged between PLCs and remote IO devices. By targeting this lowest level of the network, we ensure that our manipulations are altering the actuator states directly. For our prototype, we physically connect some Raspberry Pis to the PLCs of SWaT and sniff the packets using a network connection bridge; in reality, an attacker might sniff packets by other means, e.g. exploiting the wireless connection when enabled. As Level 0 implements the EtherNet/IP protocol, we can use the tcpdump packet analyser to capture packets, and Scapy to further extract their contents.

For our prototype, we obtain the current sensor readings by querying SWaT's historian. We assume that the historian's data is true, i.e. that the system is not simultaneously under attack by another entity, and that it is operationally healthy. In reality, an attacker might access this information through an exploit in the historian, e.g. an EternalBlue exploit [1], or a backdoor connection (both of which were discovered in SWaT hackathons [9]).

Pre-Training Models. A goal of our approach is to minimise the amount of data required to train an accurate prediction model for sensor readings. We thus proceed in two phases: a pre-training phase, and an active learning phase. The pre-training phase uses network data to construct an *initial* prediction model, the idea being that it provides a reasonable enough starting point such that active learning will later converge. A key distinction between the two stages is how the attacker behaves: while pre-training, they sit silently to observe *normal* packets of the system; but while actively learning, they intervene by injecting (possibly) *abnormal* packets and then observe the effects. It is thus important to minimise the amount of the time spent in the latter phase to avoid detection.

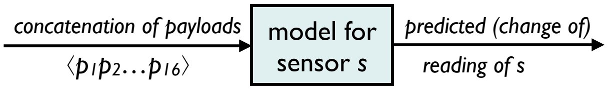
concatenation of payloads
$\langle p_1 p_2 \ldots p_{16} \rangle$
model for sensor s
predicted (change of)
reading of s

Figure 3: Input/output of a learnt model for sensor s

We require a series of regression models, one per sensor, that take as input the payloads of captured packets, and return as output a prediction of how the considered (true) sensor reading will evolve after a time period. To achieve this goal requires a number of system-specific decisions to be made, for example, the types of packets to train the model on, and a fixed time period that is appropriate to the processes involved (some will change the physical state more quickly than others). There are several types of regression models that are fit for the task. In this work, we focus on two: *linear models* and *gradient-boosting decision trees (GBDT)* [35]. A linear model is the simplest possible choice and thus serves as our baseline, whereas the GBDT is a well-known and popular example of a non-linear model. Both models can be integrated with existing active learning frameworks for regression, which was a key reason for their selection. Several more expressive models, such as neural networks, do not have any good online active learning strategies (to the best of our knowledge).

In SWaT, packets are collected from Level 0 (see Section 2) in the first four stages of the system. By observing the network traffic, we identified four different types of packets in each stage based on payload lengths and headers: packets that have payloads of length (1) 10 bytes; (2) 32 bytes; (3) 22 bytes, with a source IP address of 192.168.0.S0; and (4) 22 bytes, with a source IP address of 192.168.0.S2. Here, S is replaced with the given stage of the system (1, 2, 3, or 4). Across these four stages, there are thus 16 different types of packets in total. In constructing a feature vector for training, we make no assumptions about the meaning of these different packets, so select the first of *each* type of packet that is collected at a particular time point and concatenate their payloads together in a fixed order. This leads to feature vectors containing a series of 2752 bits.

Along with constructing a sufficient number of feature vectors (experimentally determined in Section 4), we also query the historian for sensor values after fixed time periods have passed. For flow and pressure sensors this time period is 5 seconds; for tank level sensors, it is 30 seconds, owing to the fact that they change state rather more slowly. With this data collected, we train linear and GBDT models for each individual sensor in turn, such that a sensor reading can be predicted for a given bit vector of payloads from the 16 types. An overview of the input/output of these models in given in Figure 3. Note that for flow and pressure sensors, the corresponding models predict their future *values*, whereas for tank level sensors, the corresponding models predict by how much they will change. This discrepancy is due to the fact that the effects of flow/pressure attacks stabilise at a final value very quickly.

3.2 Active Learning and Attack Discovery

Active Learning. After completing the pre-training phase, we should now have a model that is capable of making some reasonable predictions with respect to *normal* packets in the CPS network. However, the attacks we need for testing the CPS are not necessarily composed of normal packets. We need to train the model further on a broader set of examples, but cannot do it blindly owing to the expense of running the system and the enormity of the search space (2^{2752} potential combinations of feature vectors in SWaT).

Our solution is to train the model further using *(online) active learning* [63], a supervised ML approach that iteratively improves the current model. Theoretical studies have shown that active learning may exponentially reduce the amount of training data needed, e.g. [33, 37, 38]. The idea is to reduce the amount of additional data by sampling examples that are *estimated* to maximally change the current model in some way. In our case, we use one of two active learning frameworks to guide the construction of new feature vectors by flipping the bits of existing ones (this is more conservative than constructing payloads from scratch, but minimises the possibility of packet rejection). Once new feature vectors have been sampled, we can decompose them into their constituent packets, spoof them in the network, observe their effects on true sensor readings, then re-train the model accordingly.

While active learning for classification problems is well-studied, there are limited active learning frameworks for regression, and some of the ones that exist make assumptions unsuitable for our application (e.g. a Gaussian distribution [21]). However, the Expected Model Change Maximization (EMCM) approach of Cai et al. [17] avoids this assumption and is suitable for CPSs. Their framework is based on the idea of sampling new examples that are estimated to maximally change the model itself, i.e. the gradient in linear models, or a linear approximation of GBDTs based on 'super features' extracted from the trees (see [17] for details).

Inspired by EMCM, and motivated by the fact we can query live behaviour of the system, we also propose a variant of the framework called Expected Behaviour Change Maximisation (EBCM). Instead of sampling examples estimated to maximally change the model, EBCM attempts to identify examples that cause maximally different *behaviour* from what the system is currently exhibiting. For example, if a considered sensor reading is increasing, then EBCM may identify examples that cause it to decrease as much as possible instead. The intuition of the approach is that exploring different behaviour in a particular context is more informative. It also seeks to check that unfamiliar packets predicted to cause that behaviour *really do* cause that behaviour, updating the model otherwise.

Algorithm 2 summarises the steps of EBCM, in which a new feature vector is constructed by sampling additional packets, randomly flipping the bits of several copies, and choosing a vector that would have led to a maximally different reading than the original. Note that to ensure some variation, the feature vector is chosen from a set of several using Roulette Wheel Selection [43], which assigns to each candidate a probability of being selected based on its 'fitness', here defined as the absolute difference between what the sensor reading actually became (with respect to the original packets) and what the current model predicted for the candidate. If f_i is the fitness of one of n candidates, then its probability of being selected is $f_i / \sum_{j=1}^{n} f_j$. A random number is generated between 0 and the sum of the candidates' fitness scores. We then iterate through the candidates until the accumulated fitness is larger than that number, returning that final candidate as our chosen bit-vector.

```
Algorithm 2: Expected Behaviour Change Maximisation
    Input: Prediction model $M_s$, prediction time $t_s$, maximum number
           of bits to flip $n_m$
    Output: Feature (bit-)vector $p_f$
Sniff current packets and construct a feature vector $p_o$ based on
  their payloads;
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 selecting and
      flipping $n \leq n_m$ bits;
    $v_p := M_s(p)$;
    $P := P^\frown \langle p \rangle$;
    $D := D^\frown \langle |v_s - v_p| \rangle$;
until timeout;
Select a feature vector $p_f$ from $P$ using Roulette Wheel Selection
  with corresponding fitness values in $D$;
return feature (bit-)vector $p_f$;
```

For SWaT, we implemented both EMCM and EBCM, using the same construction of feature vectors (i.e. a concatenation of the payloads of 16 types of packets). Upon computing new feature vectors using these active learning frameworks, we then break the vectors down into their constituent packets, and spoof them in Level 0 of the network using Scapy. After spoofing, we wait either 5 or 30 seconds (when targeting flow/pressure or tank level sensors respectively) before querying the latest sensor value, then re-train the model based on the new packets and readings observed. This process is repeated until a suitable timeout condition (Section 4.2).

Attack Discovery and Fuzzing. In the final step of our approach, we use the learnt models (Figure 3) to discover attacks, i.e. packet manipulations that will drive targeted (true) sensor readings out of their safe operational ranges. In particular, after choosing a sensor to target, the corresponding model is used to evaluate a number of candidate packet manipulations and reveal the one that is (predicted) to realise the attack most effectively. The final part of our approach consists of *generating* those candidate packet manipulations for the model to evaluate.

Algorithm 3 presents the steps of our packet manipulation procedure for attack discovery. The idea of the algorithm is to identify the bits that are most *important* (i.e. have the most influence in the prediction), generate candidates by flipping fixed numbers of those bits, before broadening the search to other, less important bits too. As different candidates are generated, they are evaluated against a simple objective function that is maximised as the predicted sensor state becomes closer to an edge of its safe operational range. Suppose that v_s denotes a value of sensor s, and that L_s and H_s

Algorithm 3: Attack Discovery

```
Input: Prediction model $M_s$, number of bits to flip $n$, objective
       function $f$
Output: A bit-vector $p_{max}$
Sniff current packets and construct a feature vector $p_o$ based on
  their payloads;
Construct a sequence $\Phi$ of (0-based) indices of $p_o$, from the position
  with the highest feature importance (Section 3.2) to the lowest;
$k := n - 1$;
$Done := \emptyset$;
$f_{max} := 0$;
repeat
    $\Phi_k := \{i \mid i \in \Phi[0..k]\}$;
    $Combs := \{B \mid B \in 2^{\Phi_k} \wedge |B| = n \wedge B \notin Done\}$;
    for $c \in Combs$ do
        Construct $p$ from $p_o$ by flipping $p_o[i]$ for every $i \in c$;
        if $f(M_s(p)) > f_{max}$ then
            $f_{max} := f(M_s(p))$;
            $p_{max} := p$
        $Done := Done \cup c$;
    $k := k + 1$;
until $k == |\Phi|$ or timeout;
return bit-vector $p_{max}$;
```

respectively denote its lower and upper safety thresholds. Let:

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

A suitable objective function that is maximised by values approaching either of the thresholds would then be:

$$f(v_s) = \frac{1}{d_s/(H_s - L_s)}$$

We calculate *feature importance* in one of two ways, depending on the model used. For a linear model, the absolute value of the model's weight for that feature is taken as its importance. For a GBDT model, since it is a boosting ensemble model with a bunch of decision trees, we average the feature importance scores of these trees to obtain the feature importance of the overall model.

For SWaT, we implemented attack discovery for multiple flow, pressure, and tank level sensors, and used instances of the objective function above for each of them. The feature vectors returned by Algorithm 3 are broken into their constituent packets, then spoofed in the network using Scapy. If an attack successfully drives a targeted sensor out of its normal operational range (e.g. over/underflow), we record this, adding the particular packet manipulation used to a test suite of attacks, and document it accordingly (see Section 4 for an experimental evaluation). Recall that in SWaT, the models for tank levels sensors do not predict future values directly, but rather the magnitude by which they will change by: as a consequence, Algorithm 3 is adapted for these sensors by observing the current reading at the beginning, then using it to calculate the input for the objective function.

4 EVALUATION

We evaluate the effectiveness of active fuzzing for attack discovery and detection using the SWaT testbed (Section 2).

4.1 Research Questions

Our evaluation design is centred around the following key research questions (RQs):

RQ1 (Training Time): How much time is required to learn a high-accuracy model?

RQ2 (Attack Discovery): Which model and active learning setup is most effective for attack discovery?

RQ3 (Comparisons): How does active fuzzing compare against other CPS fuzzing approaches?

RQ4 (Attack Detection): Can the learnt models be used for anomaly detection or early warnings?

RQ1 is motivated by our assumption that attackers do not have access to large offline datasets for training, and may need to evade anomaly detection systems. How long would an attacker need to spend observing live sensor readings (pre-training) and spoofing packets (active learning) before obtaining a high-accuracy model? RQ2 aims to explore the different combinations of our regression models with and without active learning, in order to establish which is most effective for discovering packet-level CPS attacks, and to quantify any added benefit of active learning in conquering the huge search space. RQ3 is intended to check our work against a baseline, i.e. its effectiveness in comparison to random search and another guided CPS network fuzzer. Finally, RQ4 aims to explore whether our learnt models can have a secondary application as part of an anomaly detection or early warning system for attacks.

4.2 Experiments and Discussion

We present the design of our experiments for each of the RQs in turn, as well as some tables of results and the conclusions we draw from them. The programs we developed to implement these experiments on the SWaT testbed are all available online [5].

RQ1 (Training Time). Our first RQ aims to assess the amount of time an attacker would require to learn a high-accuracy model from live packets. To answer this question, we design experiments for the two phases of learning in turn.

First, we investigate how long the attacker must spend *pre-training* on normal live sensor readings (i.e. without any manipulation). Recall that our goal in this phase is not to obtain a highly accurate model, but rather to find a *reasonable* enough model as a starting point for active learning. To do this, we compute the *r2 scores* of linear and GBDT regression models for individual sensors after training for different lengths of time. An r2 score is the percentage of variation explained by the model, and reflects how well correlated the predictions of a sensor and their actual future values are. Prior to training, we collect 230 minutes of packet and sensor data, splitting 180 minutes of it into a training set and the remaining 50 minutes into a test set. For each sensor, we train linear and GBDT models using the full 180 minutes (our upper limit of the experiment), and compute their r2 scores using the test data. We repeat this process for 10 minutes of data, then 20 minutes, ... up to 150 minutes at various intervals until it is clear that model is converging. We judge that a model has converged when the importance scores of its features (see Section 3.2) have stabilised up to a small tolerance (0.5% for flow/pressure sensors; 5% for level sensors) as the model is re-trained on new samples. All steps are repeated ten times and medians are reported to reduce the effects of different starting states.

Second, given a model that has been pre-trained, we investigate how long the attacker must then spend *actively learning* before the model achieves a high accuracy. To do this, we pre-train linear and GBDT prediction models for each sensor for the minimum amount of time previously determined (in the first experiment). Then, for both variants of active learning (Section 3.2), we sample new sensor data from the system and retrain the models every 5 minutes. In this experiment, we record the amount of time it takes for a model to stabilise with a high r2 score, i.e. above 0.9, using the same 50 minutes of test data to compute this. We repeat these steps ten times and compute the medians.

Results. Table 1 presents the results of our first experiment. The columns correspond to the amount of training time (10 minutes through to 180), whereas the rows correspond to regression models for individual SWaT sensors, including Flow Indicator Transmitters (e.g. FIT101), a Differential Pressure Indicator Transmitter (DPIT301), and Level Indicator Sensors (e.g. LIT101). For the LITs, our models predict their values 30 seconds into the future (as tank levels rise very slowly), whereas for all other sensors our models make predictions for 5 seconds ahead. The values reported in the table are r2 scores: here, a score of 1 indicates that the model and test data are perfectly correlated, whereas a score of -1 would indicate that there is no correlation at all. When there is clear evidence of a model converging, we do not repeat the experiment for longer training periods (except 180 minutes, our upper limit).

All of our models eventually converge during pre-training, except the linear model for LIT401: the process involving this tank is too complicated to be represented as a linear model due to the multiple interactions and dependencies involving other stages of the testbed (the GBDT model does not suffer this problem). Note that while pre-training leads to relatively high r2 scores for a number of the models (e.g. the simpler processes involving flow), this does not necessarily imply that the models will be effective for attack discovery (as we investigate in RQ2). For the goal of determining a minimum amount of pre-training time, we fix it at 40 minutes, as all models (except Linear-LIT401) exhibit some positive correlation by then (≥ 0.3). Some scores are still low, but this will allow us to assess whether active learning is still effective when applied to cases that lack a good pre-trained model.

Table 2 presents the results of our second experiment. Here, the columns contain the sensors that each regression model is targeting, whereas the rows contain the type of model and active learning variant considered. The values reported are the number of minutes (accurate up to 5 minutes) of active learning that it takes before models achieve an r2 score above 0.9. Note that with active learning, none of the linear models for tank level sensors were able to exceed our r2 threshold (although they did converge for LIT101 and LIT301 with lower scores). All GBDT models were able to exceed the r2 threshold with active learning, indicating that the additional expressiveness is important for some processes of SWaT—likely because the actual processes *are* non-linear. The amount of time required varied from 10 up to 45 minutes. Taking the pre-training time into consideration:

> *Once pre-trained on 40 minutes of data observations, attackers can accurately predict SWaT's sensor readings after 10–45 minutes of active learning.*

This is a significantly reduced amount of time compared to SWaT's LSTM-based fuzzer [26], the model of which was trained for approximately two days on a rich dataset compiled from four days of constant operation.

RQ2 (Attack Discovery). Our second RQ aims to assess which combinations of models and active learning setups (including no active learning at all) are most effective for *finding attacks*, i.e. manipulations of packet payloads that would cause the true readings of a particular sensor to eventually cross one of its safety thresholds (e.g. risk of overflow, or risk of bursting pipe).

To do this, we experimentally calculate the *success rates* at finding attacks for all variants of models covering the flow, pressure, and tank level sensors. Furthermore, we do so while restricting the manipulation of the packets' payloads to different quantities of bit flips, from 1–5 and 10 such flips. For each model variant, we calculate the success rate by running our active fuzzer 1000 times with the given model, and recording as a percentage the number of times in which the resulting modified packet would cause the physical state to cross[1] a safety threshold. Note that it is important to flip existing payload bits, rather than craft packets directly, as the system's built-in validation procedures may reject them.

Results. Table 3 presents the results of our experiment for RQ2. Each sub-table reports on a restriction to a particular number of payload bit flips, ranging from 1–5 and then 10. The columns contain the sensors we are attempting to drive into unsafe states, whereas the rows contain the type of model and active learning variant considered (if any). The final row, Random (No Model), is discussed as part of RQ3. The values recorded are success rates (%s), where 100% indicates that all 1000 model-guided bit flips would succeed, and where 0% indicates that none of them would do. In the active learning models, pre-training was conducted for 40 minutes (as determined in RQ1). We also include a model that was pre-trained *only* for 90 minutes—roughly the time to do *both* pre-training and active learning—to ensure a fair comparison.

We can draw a number of conclusions from these results. First, linear models are not expressive enough in general for driving the bit flipping of payloads: their success rates for the LITs, for example, is mostly 0%, and even at 10 bit flips numbers for most sensors remain very low. GBDT quite consistently outperforms the linear models, often approaching 100% success rates. Like the linear models, GBDT struggled to attack the LITs for small numbers of bit flips (likely because multiple commands are needed to affect these sensors), but can attack them all once the restriction is lifted to ten bit flips.

> *The expressiveness of the underlying model is critically important: active learning alone is not enough to compensate for this.*

Another conclusion that can be drawn from the tables relates to the significantly higher success rates for variants using active learning, both for linear models and GBDT models. The combination of active learning with the expressiveness of GBDT, in particular, leads

[1] With the exception of the low threshold for flow sensors, which is 0.

Table 1: r2 scores (*higher is better*) of linear and GBDT sensor prediction models after different amounts of pre-training

Linear Models		10min	20min	30min	40min	50min	60min	70min	80min	100min	120min	150min	180min
Flow	FIT101	0.3399	0.5998	0.6698	0.7391	0.8214	0.8475	0.8896	0.8712	· · ·	· · ·	· · ·	0.8840
	FIT201	-0.7112	-0.0775	0.7394	0.8381	0.8962	0.8931	· · ·	· · ·	· · ·	· · ·	· · ·	0.7332
	FIT301	-0.481	0.5292	0.8227	0.8949	0.9121	0.9001	· · ·	· · ·	· · ·	· · ·	· · ·	0.8772
	FIT401	-1.2513	-0.6149	0.1123	0.3142	0.6634	0.7235	0.6695	0.6143	· · ·	· · ·	· · ·	0.6425
Pr.	DPIT301	-0.2511	0.6070	0.8648	0.9563	0.9651	0.9642	· · ·	· · ·	· · ·	· · ·	· · ·	0.9569
T. Level	LIT101	0.0624	0.1516	0.6024	0.6582	0.6824	0.6168	0.7172	0.772	0.7965	0.8197	0.8133	0.8254
	LIT301	-0.0806	0.0937	0.4248	0.4949	0.5963	0.6260	0.6583	0.4807	0.6209	· · ·	· · ·	0.5426
	LIT401	0.1543	0.0612	0.2942	0.0273	-0.2208	0.1119	-0.4902	0.007	0.1259	0.2412	0.0135	-0.6597

GBDT Models		10min	20min	30min	40min	50min	60min	70min	80min	100min	120min	150min	180min
Flow	FIT101	-0.2229	-0.0245	0.4313	0.7742	0.9112	0.9413	0.9755	0.9741	· · ·	· · ·	· · ·	0.9637
	FIT201	-0.7116	0.3545	0.9584	0.9633	0.9504	0.9642	· · ·	· · ·	· · ·	· · ·	· · ·	0.9421
	FIT301	-0.9453	0.2496	0.8051	0.9734	0.9731	0.9751	· · ·	· · ·	· · ·	· · ·	· · ·	0.9524
	FIT401	-0.2124	0.3120	0.7015	0.8125	0.8342	0.8861	0.8453	0.7921	· · ·	· · ·	· · ·	0.8025
Pr.	DPIT301	-0.5508	0.8218	0.9387	0.9757	0.9818	0.9831	0.9912	0.9901	0.9875	0.9706	0.9496	0.9085
T. Level	LIT101	-0.042	-0.1352	-0.153	0.3858	0.6459	0.7881	0.8536	0.8680	0.8961	0.9221	0.8721	0.9019
	LIT301	-0.0419	-0.2151	-0.0185	0.3863	0.7059	0.8208	0.6486	0.7938	0.5498	0.7363	· · ·	0.7291
	LIT401	-1.1415	0.1121	0.7123	0.8377	0.8503	0.8575	0.7731	0.7907	0.8764	0.8743	0.8741	0.8444

Table 2: Median time (mins; *lower is better*) for active learning (AL) configurations to achieve an r2 score above 0.9

	Flow				Pressure	Tank Level		
AL Config.	FIT101	FIT201	FIT301	FIT401	DPIT301	LIT101	LIT301	LIT401
Linear (EBCM)	25	15	30	15	30	—	—	—
Linear (EMCM)	20	20	45	15	40	—	—	—
GBDT (EBCM)	10	10	25	10	30	35	30	45
GBDT (EMCM)	10	10	25	10	20	40	40	45

to attacks being found in all cases for the 10 bit flip restriction. With active learning enabled, the difference is often significant (e.g. 0% vs. 100% for FIT401, 10 bit flips). The results suggest that active learning is key for finding the 'critical bits' in payloads, given its ability to sample and query new data. Models that have only been pre-trained just recognise trends observed in normal data, and do not necessarily know which bits involved in the patterns are the critical ones for enacting an attack.

Active learning is effective at identifying critical bits in payloads, and can lead to significantly higher success rates in attack discovery.

RQ3 (Comparisons). Our third RQ assesses how active fuzzing performs against two baselines. First, for every sensor (as targeted in RQ2), we randomly generate 1000 k-bit payload manipulations (where k is 1–5 or 10) and assess for them the attack success rates (a percentage, as calculated in RQ2). Second, we qualitatively compare our attacks against the ones identified by the LSTM-based fuzzer for SWaT [26] as well as an established benchmark of SWaT network attacks [41] that was manually crafted by experts.

Results. The results of the random flipping baseline are given in the final rows of Table 3. Clearly, this is not an effective strategy for finding attacks based on packet manipulation, as no success rate exceeds 0.5%. This is unsurprising due to the huge search space involved. Note also that for the more challenging over/underflow attacks, random bit flipping is unable to find any examples at all.

Regarding the LSTM-based fuzzer for SWaT [26], a side-by-side comparison is difficult to make as it does not manipulate packets but rather only issues high-level actuator commands (e.g."OPEN MV101"). Our approach is able to find attacks spanning the same range of sensed properties, but does so by manipulating the bits of packets directly (closer to the likely behaviour of a real attacker) and without the same level of network control (other than true sensor readings, which both approaches require). In this sense our attacks are more elaborate than those of the LSTM fuzzer. Our approach is also substantially faster: active fuzzing can train effective models in 50-85 minutes, whereas the underlying model used in [26] required approximately two whole days.

Our coverage of the SWaT benchmark [41] is comparable to that of [26], since both approaches find attacks spanning the same sensed properties. However, all of the attacks in [41] and [26] are implemented at Level 1 of the network. Active fuzzing instead generates packet-manipulating attacks at Level 0, which has the advantage of avoiding interactions with the PLC code, possibly making manipulations harder to detect (e.g. bypassing command validation checks). In this sense, the attacks that active fuzzing finds complement and enrich the benchmark.

Active fuzzing finds attacks covering the same sensors as comparable work, but with significantly less training time, and by manipulating packets directly.

RQ4 (Attack Detection). Our final RQ considers whether our learnt models can be used not only for attack discovery but also attack *prevention.* In particular, we investigate their use in two defence mechanisms: an anomaly detector and an early warning system. We then assess how effective they are detecting attacks.

To perform anomaly detection, we continuously perform the following process: we read the current values of sensors, and then use our learnt models to predict their values 5 seconds into the future (30 seconds for tank levels). After 5 (or 30) seconds have passed, the *actual* values v_a are compared with those that were predicted v_p, and an anomaly is reported if $|v_p - v_a|/v_m > 0.05$ (or $|v_p - v_a| > 5$ for tanks), where v_m is the largest possible observable value for the sensor. To evaluate the effectiveness of this detection scheme, we implement an experiment on actual historian data extracted from SWaT [41]. For each sensor in turn, we randomly generate 1000 spoofed sensor values by randomly adding or subtracting values (in

Table 3: Success rates (%s; *higher is better*) of different model configurations for finding packet manipulations (1-5 and 10 bit flips) that successfully drive SWaT's flow, pressure, and tank level readings to safety thresholds

		Flow				Pr.	Level		
	Models (1 Bit Flip)	FIT101	FIT201	FIT301	FIT401	DPIT301	LIT101	LIT301	LIT401
Linear	Pre-Train Only (40min)	0.5	4.4	1.3	1.8	0.9	0	0	0
	Pre-Train Only (90min)	0.8	3.8	1.4	0.2	2.5	0	0	0
	Active Learning (EBCM)	27	22.3	7.5	43.9	14.4	0	0	0
	Active Learning (EMCM)	29.4	19.2	7.9	36.6	9.1	0	0	0
GBDT	Pre-Train Only (40min)	0	59.3	0	0	0	0	0	0
	Pre-Train Only (90min)	0	57.7	30.3	0	0	0	0	0
	Active Learning (EBCM)	97.7	99.2	97.9	75.4	96.1	0	0	0
	Active Learning (EMCM)	97.6	99.4	97.6	13.5	95.3	0	0	0
—	Random (No Model)	0	0	0	0.2	0	0	0	0

		Flow				Pr.	Level		
	Models (2 Bit Flips)	FIT101	FIT201	FIT301	FIT401	DPIT301	LIT101	LIT301	LIT401
Linear	Pre-Train Only (40min)	0.7	8	1.9	3.1	1.7	0.2	0	0
	Pre-Train Only (90min)	2.7	9.4	4.1	0.6	6.6	0	0	0
	Act. Learning (EBCM)	46.1	39.1	15.1	77.6	30.1	2	0	0
	Act. Learning (EMCM)	47.4	31.8	16.5	72.4	17.9	0.6	0	0
GBDT	Pre-Train Only (40min)	0	98.7	0	0	0	0	0	0
	Pre-Train Only (90min)	0.3	96	59	0	0	0	0	0
	Act. Learning (EBCM)	100	99.9	100	100	99.9	76.4	0	0
	Act. Learning (EMCM)	100	100	99.7	100	99.9	87.1	0	0
—	Random (No Model)	0.3	0	0	0.2	0	0	0	0

		Flow				Pr.	Level		
	Models (3 Bit Flips)	FIT101	FIT201	FIT301	FIT401	DPIT301	LIT101	LIT301	LIT401
Linear	Pre-Train Only (40min)	1.2	10.6	3.5	3.5	3	0	0	0
	Pre-Train Only (90min)	4.2	12.5	5.5	0.5	8.2	0	0	0
	Act. Learning (EBCM)	58.6	50.6	25.7	91.7	37.2	4.2	0.1	0.1
	Act. Learning (EMCM)	60.4	45.1	21.4	88.7	24.2	2.7	0.3	0
GBDT	Pre-Train Only (40min)	0	100	0.1	0	0	0	0	0
	Pre-Train Only (90min)	1.1	100	80.6	0	0	0	0	0
	Act. Learning (EBCM)	100	100	100	100	100	97	8.1	23.7
	Act. Learning (EMCM)	100	100	99.7	100	100	99.2	3.7	32.3
—	Random (No Model)	0.1	0.2	0.1	0.1	0.3	0	0	0

		Flow				Pr.	Level		
	Models (4 Bit Flips)	FIT101	FIT201	FIT301	FIT401	DPIT301	LIT101	LIT301	LIT401
Linear	Pre-Train Only (40min)	0.9	13.8	5.4	4.9	4.3	0	0	0
	Pre-Train Only (90min)	6.4	15.1	7.5	0.9	12.1	0	0	0
	Act. Learning (EBCM)	71.6	65	27.9	97.7	49.1	11.4	0.1	0.2
	Act. Learning (EMCM)	75.1	26.8	31.1	95.7	29.2	5.4	0.3	0
GBDT	Pre-Train Only (40min)	0	100	0.7	0	0	0	0	0
	Pre-Train Only (90min)	1.2	100	96.1	0	0	0	0	0
	Act. Learning (EBCM)	100	100	100	100	100	100	20.4	55.7
	Act. Learning (EMCM)	100	100	99.7	100	100	100	17.7	67.3
—	Random (No Model)	0.1	0.1	0.1	0.2	0.2	0	0	0

		Flow				Pr.	Level		
	Models (5 Bit Flips)	FIT101	FIT201	FIT301	FIT401	DPIT301	LIT101	LIT301	LIT401
Linear	Pre-Train Only (40min)	1.7	19.3	6.2	5.7	4.3	0.4	0	0
	Pre-Train Only (90min)	6.6	18.4	8.7	1.1	13.5	0.1	0	0
	Act. Learning (EBCM)	78.3	71.6	35.4	99.5	56.6	16.3	0.5	0
	Act. Learning (EMCM)	81.5	62.6	36	97.9	39.3	9.3	0.5	0
GBDT	Pre-Train Only (40min)	0	100	3.1	0	0	0	0	0
	Pre-Train Only (90min)	3.1	100	99.8	0	0.1	0	0	0
	Act. Learning (EBCM)	100	100	100	100	100	100	29.7	76
	Act. Learning (EMCM)	100	100	99.7	100	100	100	34.9	82.2
—	Random (No Model)	0.2	0.4	0.1	0.1	0.2	0	0	0

		Flow				Pr.	Level		
	Models (10 Bit Flips)	FIT101	FIT201	FIT301	FIT401	DPIT301	LIT101	LIT301	LIT401
Linear	Pre-Train Only (40min)	4.7	34.9	9.6	12.5	10.7	0.8	0	0
	Pre-Train Only (90min)	13	38.3	19.2	2.3	29.1	1	0	0
	Act. Learning (EBCM)	96.4	91	61.8	100	81.5	35.7	6.1	2.1
	Act. Learning (EMCM)	96.7	89.1	59.5	100	63	29.1	5.7	0
GBDT	Pre-Train Only (40min)	0	100	31.5	0	0	0	0	0
	Pre-Train Only (90min)	12.1	100	99.8	0	2.3	0	0	0
	Act. Learning (EBCM)	100	100	100	100	100	100	72.2	99.3
	Act. Learning (EMCM)	100	100	99.7	100	100	100	77	99.7
—	Random (No Model)	0.2	0.4	0.4	0.1	0.5	0	0	0

Table 4: Success rates (%s; *higher is better*) of different anomaly detector models at detecting injected sensor values

		Flow				Pr.	Tank Level		
	Anomaly Detector	FIT101	FIT201	FIT301	FIT401	DPIT301	LIT101	LIT301	LIT401
Linear	Pre-Train Only (40min)	82.3	100	100	41.3*	100	8.2*	38.5*	40.1*
	Pre-Train Only (90min)	100	100	100	62.9*	100	67.8*	39.3*	13.2*
	Act. Learning (EBCM)	100	100	100	71.1*	100	59.8*	51.2*	64.1*
	Act. Learning (EMCM)	100	100	100	71.4*	100	57.5*	44.7*	50.7*
GBDT	Pre-Train Only (40min)	100	100	100	100	100	71.8*	74.6*	76*
	Pre-Train Only (90min)	100	100	100	100	100	95.3	92.3	74.1
	Act. Learning (EBCM)	100	100	100	100	100	91.8	95.5	84.1
	Act. Learning (EMCM)	100	100	100	100	100	92.5	95.8	83.4

the range 5-10 for LITs, or 0.05v_m through to 0.1v_m for the others) to sensor readings at different points of the data. We then use our learnt models to determine what would have been predicted from the data 5 or 30 seconds earlier, comparing the actual and predicted values as described. We record the success rates of our anomaly detectors at detecting these spoofed sensor readings.

Our early warning system is set up in a similar way, continuously predicting the future readings of sensors based on the current network traffic. The key difference is that rather than comparing actual values with previously predicted values, we instead issue warnings at the time of prediction if the future value of a sensor is outside of its well-defined normal operational range. To experimentally assess this, we manually subject the SWaT testbed to the Level 1 attacks identified in [26] (which itself covers more unsafe states that the SWaT benchmark [41]), targeting each sensor in turn. When each attack is underway, we use our learnt models to predict the future sensor readings. If a warning is issued at some point *before* a sensor is driven outside of its normal range, we record this as a success. We repeat this ten times for each sensor.

Results. Table 4 contains the results of our anomaly detection experiment. The columns indicate the sensors for which values in the data were manipulated, whereas the rows indicate the model and active learning variant used. The values are the success rates, i.e. the percentage of spoofed sensor values that were detected as anomalous. Asterisks $(*)$ indicate where false positive rates were above 5%, meaning the anomaly detectors were not practically useful. For the flow and pressure sensors, most variants of model

Table 5: Success rates (%s; *higher is better*) of different models at warning before sensors exit their safe ranges

Early Warning System		Tank Level		
		LIT101	LIT301	LIT401
Linear (all variants)		–	–	–
GBDT	Pre-Train Only (40min)	–	–	–
	Pre-Train Only (90min)	100	100	100
	Active Learning (EBCM)	100	100	100
	Active Learning (EMCM)	100	100	100

and active learning were able to successfully detect anomalies, the main exception being FIT401 for which the linear model performed poorly. The tank level sensors were more challenging to perform anomaly detection for, but the GBDT models have a clear edge over the linear ones. Active learning made little difference across the experiments, except to improve the accuracy of the original 40 minute pre-trained models.

Table 5 contains the results of our early warning detection experiment. The columns indicate the sensed properties that were targeted by attacks (e.g. drive LIT101 outside of its safe range), whereas the rows indicate the model and active learning variant used. The values are the success rates, i.e. the percentages of attacks that were warned about before succeeding. Cells containing dashes (–) indicate that more than 5% of the warnings were false positive, and thus too unreliable. The first thing to note is that the experiment only considered the tank level sensors: this is because the flow and pressure sensors can be forced into unsafe states very quickly, requiring more immediate measures than an early warning system. The tanks however take time to fill up or empty, and thus are a more meaningful target for this solution. Second, the model has a clear impact: GBDT models with either active learning or at least 90 minutes of pre-training are accurate enough to warn about 100% of the attacks, whereas the linear models are not expressive enough and suffer from false positives. Again, active learning improves the accuracy of 40 minute pre-trained models sufficiently, but otherwise is not critical: its key role is not in prevention but in *discovering* attacks, through its ability to identify the critical bits to manipulate.

> *Our models can be repurposed as anomaly detectors or early warning systems, but active learning is not as critical here as in attack discovery.*

4.3 Threats to Validity

While our work has been extensively evaluated on a real critical infrastructure testbed, threats to the validity of our conclusions of course remain. First, while SWaT is a fully operational water treatment testbed, it is not as large as the plants it is based on, meaning our results may not scale-up (this is difficult to assess, as access to such plants is subject to strict confidentially). Second, it may not generalise to CPSs in domains that have different operational characteristics, motivating future work to properly assess this. Finally, while our anomaly detector performed well, our sensor spoofing attacks were generated randomly, and may not be representative of a real attacker's behaviour (note however that the early warning system was assessed using previously documented attacks). Similarly, our early warning detection systems performed well at detecting known over/underflow attacks, but these attacks are of the kind that active fuzzing itself can generate: how the models perform against different kinds of attacks requires further investigation.

5 RELATED WORK

In this section, we highlight a selection of the literature that is related to the main themes of this paper: *learning from traffic* (including active learning), *defending CPSs*, and *testing/verifying CPSs*.

Learning from Network Traffic. The application of machine learning to network traffic is a vibrant area of research [68], but models are typically constructed to perform *classification* tasks. To highlight a few examples: Zhang et al. [84] combine supervised and unsupervised ML to learn models that can classify zero-day traffic; Nguyen and Armitage [67] learn from statistical features of sub-flows to classify between regular consumer traffic and traffic originating from online games; and Atkinson et al. [16] use a classifier to infer personal information by analysing encrypted traffic patterns caused by mobile app usage. All these examples are in contrast to active fuzzing, where *regression* models are learnt for predicting how a set of network packets will cause a (true) sensor reading of a CPS to change. We are not aware of other work building regression models in a similar context.

Similar to active fuzzing, there are some works that apply active learning, but again for the purpose of classification, rather than regression. Morgan [65], for example, uses active learning to reduce training time for streaming data classifiers, as do Zhao and Hoi [86] but for malicious URL classifiers.

Defending CPSs. Several different research directions on detecting and preventing CPS attacks have emerged in the last few years. Popular approaches include anomaly detection, where data logs (e.g. from historians) are analysed for suspicious events or patterns [11, 15, 27, 45, 48, 52, 58, 61, 66, 69, 71]; digital fingerprinting, where sensors are checked for spoofing by monitoring time and frequency domain features from sensor and process noise [12, 13, 44, 56]; and invariant-based checks, where conditions over processes and components are constantly monitored [7, 8, 10, 18, 24, 25, 28, 39]. These techniques are meant to complement and go beyond the built-in validation procedures installed in CPSs, which typically focus on simpler and more localised properties of the system.

The strengths and weaknesses of different countermeasures has been the focus of various studies. Urbina et al. [77] evaluated 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 defences), and suggest ways of mitigating this. Cárdenas et al. [19] 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. centralised vs. distributed.

As a testbed dedicated for cyber-security research, many different countermeasures have been developed for SWaT itself. These include anomaly detectors, typically trained on the publicly released dataset [2, 41] using unsupervised learning techniques, e.g. [42, 52, 58]. A supervised learning approach is pursued by [24, 25], who inject faults into the PLC code of (a high-fidelity simulator) in order to obtain abnormal data for training. Ahmed et al. [12, 13] implemented fingerprinting systems based on sensor and process noise for detecting spoofing. Adepu and Mathur [7, 8, 10] systematically and manually derived physics-based invariants and other conditions to be monitored during the operation of SWaT. Feng et al. [32] 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.

Testing and Verifying CPSs. Several authors have sought to improve the defences of CPSs by constructing or synthesising attacks that demonstrate flaws to be fixed. Liu et al. [62] and Huang et al. [50], for example, synthesise attacks for power grids that can bypass bad measurement detection systems and other conventional monitors. Dash et al. [30] target robotic vehicles, which are typically protected using control-based monitors, and demonstrate three types of stealthy attacks that evade detection. Uluagac et al. [76] presented attacks on sensory channels (e.g. light, infrared, acoustic, and seismic), and used them to inform the design of an intrusion detection system for sensory channel threats. Active fuzzing shares this goal of identifying attacks in order to improve CPS defences.

Fuzzing is a popular technique for automatically testing the defences of systems, by providing them with invalid, unexpected, or random input and monitoring how they respond. Our active fuzzing approach does exactly this, guiding the construction of input (network packets) using prediction models, and then observing sensor readings to understand how the system responds. The closest fuzzing work to ours is [26], which uses an LSTM-based model to generate actuator configurations, but requires vast amounts of data and system access to function effectively. Fuzzing has also been applied for testing CPS models, e.g. CyFuzz [29] and DeepFuzzSL [72], which target models developed in Simulink. Outside of the CPS domain, several fuzzing tools are available for software: American fuzzy lop [83], for example, uses genetic algorithms to increase the code coverage of tests; Cha et al. [22] use white-box symbolic analysis on execution traces to maximise the number of bugs they find; and grammar-based fuzzers (e.g. [40, 49]) use formal grammars to generate complex structured input, such as HTML/-JavaScript for testing web browsers. Fuzzing can also be applied to network protocols in order to test their intrusion detection systems (e.g. [79]). Our work, in contrast, assumes that an attacker has *already compromised* the network (as per Section 2).

There are techniques beyond fuzzing available for analysing CPS models in Simulink. A number of authors have proposed automated approaches for falsifying such models, i.e. for finding counterexamples of formal properties. To achieve this, Yamagata et al. [14, 81] use deep reinforcement learning, and Silvetti et al. [73] use active learning. Chen et al. [23] also use active learning, but for mining formal requirements from CPS models. Note that unlike these approaches, active fuzzing is applied directly at the network packet level of a real and complex CPS, and therefore does not make any of the abstractions that modelling languages necessitate.

A number of approaches exist that allow for CPSs to be *formally* verified or analysed. These typically require a formal specification or model, which, if available in the first place, may abstract away important complexities of full-fledged CPS processes. Kang et al. [55], for example, construct a discretised first-order model of SWaT's first three stages in Alloy, and analyse it with respect to some safety properties. This work, however, uses high-level abstractions of the physical process, only partially models the system, and would not generalise to the packet-level analyses that active fuzzing performs. Sugumar and Mathur [75] analyse CPSs using timed automata models, simulating their behaviour under single-stage single-point attacks. Castellanos et al. [20], McLaughlin et al. [64], and Zhang et al. [85] perform formal analyses based on models extracted from the PLC programs, whereas Etigowni et al. [31] analyse information flow using symbolic execution. If a CPS can be modelled as a hybrid system, then a number of formal techniques may be applied, including model checking [34, 80], SMT solving [36], reachability analysis [54], non-standard analysis [47], process calculi [59], concolic testing [57], and theorem proving [70]. Defining a formal model that accurately characterises enough of the CPS, however, is the *hardest* part, especially for techniques such as active fuzzing that operate directly at the level of packet payloads.

6 CONCLUSION

We proposed *active fuzzing*, a black-box approach for automatically building test suites of packet-level CPS network attacks, overcoming the enormous search spaces and resource costs of such systems. Our approach learnt regression models for predicting future sensor values from the binary string payloads of network packets, and used these models to identify payload manipulations that would achieve specific attack goals (i.e. pushing true sensor values outside of their safe operational ranges). Key to achieving this was our use of online active learning, which reduced the amount of training data needed by sampling examples that were estimated to maximally improve the model. We adapted the EMCM [17] active learning framework to CPSs, and proposed a new version of it that guided the process by maximising behaviour change.

We presented algorithms for implementing active fuzzing, but also demonstrated its efficacy by implementing it for the SWaT testbed, a multi-stage water purification plant involving complex physical and chemical processes. Our approach was able to achieve comparable coverage to an established benchmark and LSTM-based fuzzer, but with significantly less data, training time, and resource usage. Furthermore, this coverage was achieved by more sophisticated attacks than those of the LSTM-based fuzzer, which can only generate high-level actuator commands and is unable to manipulate packets directly. Finally, we showed that the models constructed in active learning were not only useful for attack *discovery*, but also for attack *detection*, by implementing them as anomaly detectors and early warning systems for SWaT. We subjected the plant to a series of random sensor-modification attacks as well as existing actuator-manipulation attacks, finding that our most expressive learnt models were effective at detecting them.

ACKNOWLEDGMENTS

We are grateful to the three anonymous ISSTA referees for their very constructive feedback. This research / project is supported by the National Research Foundation, Singapore, under its National Satellite of Excellence Programme "Design Science and Technology for Secure Critical Infrastructure" (Award Number: NSoE_DeST-SCI2019-0008). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore. It is also supported in part by a Major Scientific Research Project of Zhejiang Lab (2018FD0ZX01), Alibaba-Zhejiang University Joint Institute of Frontier Technologies, Zhejiang Key R&D Plan (2019C03133), the Fundamental Research Funds for the Central Universities (2020QNA5021), and by an SUTD-ZJU IDEA Grant for Visiting Professor (SUTD-ZJU VP 201901).

REFERENCES

[1] 2016. CVE-2017-0144. Available from MITRE, CVE-ID CVE-2017-0144.. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0144

[2] 2020. iTrust Labs: Datasets. https://itrust.sutd.edu.sg/itrust-labs_datasets/. Accessed: May 2020.

[3] 2020. Scapy. https://scapy.net/. Accessed: May 2020.

[4] 2020. Secure Water Treatment (SWaT). https://itrust.sutd.edu.sg/itrust-labs-home/itrust-labs_swat/. Accessed: May 2020.

[5] 2020. Supplementary material. https://github.com/yuqiChen94/Active_fuzzer.

[6] 2020. Wireshark. https://www.wireshark.org/. Accessed: May 2020.

[7] Sridhar Adepu and Aditya Mathur. 2016. 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, 449–460.

[8] Sridhar Adepu and Aditya Mathur. 2016. 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) (IFIP AICT)*, Vol. 471. Springer, 91–104.

[9] Sridhar Adepu and Aditya Mathur. 2018. Assessing the Effectiveness of Attack Detection at a Hackfest on Industrial Control Systems. *IEEE Transactions on Sustainable Computing* (2018).

[10] Sridhar Adepu and Aditya Mathur. 2018. Distributed Attack Detection in a Water Treatment Plant: Method and Case Study. *IEEE Transactions on Dependable and Secure Computing* (2018).

[11] Ekta Aggarwal, Mehdi Karimibiuki, Karthik Pattabiraman, and André Ivanov. 2018. CORGIDS: A Correlation-based Generic Intrusion Detection System. In *Proc. Workshop on Cyber-Physical Systems Security and PrivaCy (CPS-SPC 2018)*. ACM, 24–35.

[12] Chuadhry Mujeeb Ahmed, Martín Ochoa, Jianying Zhou, Aditya P. Mathur, Rizwan Qadeer, Carlos Murguia, and Justin Ruths. 2018. *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, 483–497.

[13] Chuadhry Mujeeb Ahmed, Jianying Zhou, and Aditya P. Mathur. 2018. 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, 566–581.

[14] Takumi Akazaki, Shuang Liu, Yoriyuki Yamagata, Yihai Duan, and Jianye Hao. 2018. Falsification of Cyber-Physical Systems Using Deep Reinforcement Learning. In *Proc. International Symposium on Formal Methods (FM 2018) (LNCS)*, Vol. 10951. Springer, 456–465.

[15] Wissam Aoudi, Mikel Iturbe, and Magnus Almgren. 2018. 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, 817–831.

[16] John S. Atkinson, John E. Mitchell, Miguel Rio, and George Matich. 2018. Your WiFi is leaking: What do your mobile apps gossip about you? *Future Generation Comp. Syst.* 80 (2018), 546–557.

[17] Wenbin Cai, Ya Zhang, and Jun Zhou. 2013. Maximizing Expected Model Change for Active Learning in Regression. In *Proc. IEEE 13th International Conference on Data Mining (ICDM 2013)*. IEEE Computer Society, 51–60.

[18] Alvaro A. Cárdenas, Saurabh Amin, Zong-Syun Lin, Yu-Lun Huang, Chi-Yen Huang, and Shankar Sastry. 2011. Attacks against process control systems: risk assessment, detection, and response. In *Proc. ACM Symposium on Information, Computer and Communications Security (AsiaCCS 2011)*. ACM, 355–366.

[19] Alvaro A. Cárdenas, Robin Berthier, Rakesh B. Bobba, Jun Ho Huh, Jorjeta G. Jetcheva, David Grochocki, and William H. Sanders. 2014. A Framework for Evaluating Intrusion Detection Architectures in Advanced Metering Infrastructures. *IEEE Transactions on Smart Grid* 5, 2 (2014), 906–915.

[20] John H. Castellanos, Martín Ochoa, and Jianying Zhou. 2018. Finding Dependencies between Cyber-Physical Domains for Security Testing of Industrial Control Systems. In *Proc. Annual Computer Security Applications Conference (ACSAC 2018)*. ACM, 582–594.

[21] Rui M. Castro, Rebecca Willett, and Robert D. Nowak. 2005. Faster Rates in Regression via Active Learning. In *Proc. Annual Conference on Neural Information Processing Systems (NIPS 2005)*. 179–186.

[22] Sang Kil Cha, Maverick Woo, and David Brumley. 2015. Program-Adaptive Mutational Fuzzing. In *Proc. IEEE Symposium on Security and Privacy (S&P 2015)*. IEEE Computer Society, 725–741.

[23] Gang Chen, Zachary Sabato, and Zhaodan Kong. 2016. Active learning based requirement mining for cyber-physical systems. In *Proc. IEEE Conference on Decision and Control (CDC 2016)*. IEEE, 4586–4593.

[24] Yuqi Chen, Christopher M. Poskitt, and Jun Sun. 2016. Towards Learning and Verifying Invariants of Cyber-Physical Systems by Code Mutation. In *Proc. International Symposium on Formal Methods (FM 2016) (LNCS)*, Vol. 9995. Springer, 155–163.

[25] Yuqi Chen, Christopher M. Poskitt, and Jun Sun. 2018. 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, 648–660.

[26] Yuqi Chen, Christopher M. Poskitt, Jun Sun, Sridhar Adepu, and Fan Zhang. 2019. Learning-Guided Network Fuzzing for Testing Cyber-Physical System Defences. In *Proc. IEEE/ACM International Conference on Automated Software Engineering (ASE 2019)*. IEEE Computer Society, 962–973.

[27] Long Cheng, Ke Tian, and Danfeng (Daphne) Yao. 2017. Orpheus: Enforcing Cyber-Physical Execution Semantics to Defend Against Data-Oriented Attacks. In *Proc. Annual Computer Security Applications Conference (ACSAC 2017)*. ACM, 315–326.

[28] Hongjun Choi, Wen-Chuan Lee, Yousra Aafer, Fan Fei, Zhan Tu, Xiangyu Zhang, Dongyan Xu, and Xinyan Xinyan. 2018. Detecting Attacks Against Robotic Vehicles: A Control Invariant Approach. In *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS 2018)*. ACM, 801–816.

[29] Shafiul Azam Chowdhury, Taylor T. Johnson, and Christoph Csallner. 2017. 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) (LNCS)*, Vol. 10107. Springer, 46–60.

[30] Pritam Dash, Mehdi Karimibiuki, and Karthik Pattabiraman. 2019. Out of control: stealthy attacks against robotic vehicles protected by control-based techniques. In *Proc. Annual Computer Security Applications Conference (ACSAC 2019)*. ACM, 660–672.

[31] Sriharsha Etigowni, Dave (Jing) Tian, Grant Hernandez, Saman A. Zonouz, and Kevin R. B. Butler. 2016. CPAC: securing critical infrastructure with cyber-physical access control. In *Proc. Annual Conference on Computer Security Applications (ACSAC 2016)*. ACM, 139–152.

[32] Cheng Feng, Venkata Reddy Palleti, Aditya Mathur, and Deeph Chana. 2019. 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.

[33] Shai Fine, Ran Gilad-Bachrach, and Eli Shamir. 2002. Query by committee, linear separation and random walks. *Theoretical Computer Science* 284, 1 (2002), 25–51.

[34] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable Verification of Hybrid Systems. In *Proc. International Conference on Computer Aided Verification (CAV 2011) (LNCS)*, Vol. 6806. Springer, 379–395.

[35] Jerome H. Friedman. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics* 29 (2001), 1189–1232.

[36] Sicun Gao, Soonho Kong, and Edmund M. Clarke. 2013. dReal: An SMT Solver for Nonlinear Theories over the Reals. In *Proc. International Conference on Automated Deduction (CADE 2013) (LNCS)*, Vol. 7898. Springer, 208–214.

[37] Ran Gilad-Bachrach, Amir Navot, and Naftali Tishby. 2003. *Kernel Query By Committee (KQBC)*. Technical Report. Leibniz Center, The Hebrew University.

[38] Ran Gilad-Bachrach, Amir Navot, and Naftali Tishby. 2005. Query by Committee Made Real. In *Proc. Annual Conference on Neural Information Processing Systems (NIPS 2005)*. 443–450.

[39] Jairo Giraldo, David I. Urbina, Alvaro Cardenas, Junia Valente, Mustafa Amir Faisal, Justin Ruths, Nils Ole Tippenhauer, Henrik Sandberg, and Richard Candell. 2018. A Survey of Physics-Based Attack Detection in Cyber-Physical Systems. *Comput. Surveys* 51, 4 (2018), 76:1–76:36.

[40] Patrice Godefroid, Hila Peleg, and Rishabh Singh. 2017. Learn&Fuzz: machine learning for input fuzzing. In *Proc. IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*. IEEE Computer Society, 50–59.

[41] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. 2016. A Dataset to Support Research in the Design of Secure Water Treatment Systems. In *Proc. International Conference on Critical Information Infrastructures Security (CRITIS 2016)*.

[42] Jonathan Goh, Sridhar Adepu, Marcus Tan, and Zi Shan Lee. 2017. Anomaly detection in cyber physical systems using recurrent neural networks. In *Proc. International Symposium on High Assurance Systems Engineering (HASE 2017)*. IEEE, 140–145.

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

[44] Qinchen Gu, David Formby, Shouling Ji, Hasan Cam, and Raheem A. Beyah. 2018. Fingerprinting for Cyber-Physical System Security: Device Physics Matters Too. *IEEE Security & Privacy* 16, 5 (2018), 49–59.

[45] Yoshiyuki Harada, Yoriyuki Yamagata, Osamu Mizuno, and Eun-Hye Choi. 2017. Log-Based Anomaly Detection of CPS Using a Statistical Method. In *Proc. International Workshop on Empirical Software Engineering in Practice (IWESEP 2017)*. IEEE, 1–6.

[46] Amin Hassanzadeh, Amin Rasekh, Stefano Galelli, Mohsen Aghashahi, Riccardo Taormina, Avi Ostfeld, and M. Katherine Banks. 2019. A Review of Cybersecurity Incidents in the Water Sector. *Journal of Environmental Engineering* (09 2019).

[47] Ichiro Hasuo and Kohei Suenaga. 2012. Exercises in Nonstandard Static Analysis of Hybrid Systems. In *Proc. International Conference on Computer Aided Verification (CAV 2012) (LNCS)*, Vol. 7358. Springer, 462–478.

[48] Zecheng He, Aswin Raghavan, Guangyuan Hu, Sek M. Chai, and Ruby B. Lee. 2019. Power-Grid Controller Anomaly Detection with Enhanced Temporal Deep Learning. In *Proc. IEEE International Conference On Trust, Security And Privacy In Computing And Communications (TrustCom 2019)*. IEEE, 160–167.

[49] Christian Holler, Kim Herzig, and Andreas Zeller. 2012. Fuzzing with Code Fragments. In *Proc. USENIX Security Symposium (USENIX 2012)*. USENIX Association, 445–458.

[50] Zhenqi Huang, Sriharsha Etigowni, Luis Garcia, Sayan Mitra, and Saman A. Zonouz. 2018. 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, 151–162.

[51] ICS-CERT Alert. 2016. Cyber-Attack Against Ukrainian Critical Infrastructure. https://ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01. document number: IR-ALERT-H-16-056-01.

[52] Jun Inoue, Yoriyuki Yamagata, Yuqi Chen, Christopher M. Poskitt, and Jun Sun. 2017. 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, 1058–1065.

[53] ISA. 2020. ISA99, Industrial Automation and Control Systems Security. https://www.isa.org/isa99/. Accessed: May 2020.

[54] Taylor T. Johnson, Stanley Bak, Marco Caccamo, and Lui Sha. 2016. Real-Time Reachability for Verified Simplex Design. *ACM Transactions on Embedded Computing Systems* 15, 2 (2016), 26:1–26:27.

[55] Eunsuk Kang, Sridhar Adepu, Daniel Jackson, and Aditya P. Mathur. 2016. Model-based security analysis of a water treatment system. In *Proc. International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS 2016)*. ACM, 22–28.

[56] Marcel Kneib and Christopher Huth. 2018. 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, 787–800.

[57] Pingfan Kong, Yi Li, Xiaohong Chen, Jun Sun, Meng Sun, and Jingyi Wang. 2016. Towards Concolic Testing for Hybrid Systems. In *Proc. International Symposium on Formal Methods (FM 2016) (LNCS)*, Vol. 9995. Springer, 460–478.

[58] Moshe Kravchik and Asaf Shabtai. 2018. 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, 72–83.

[59] Ruggero Lanotte, Massimo Merro, Riccardo Muradore, and Luca Viganò. 2017. A Formal Approach to Cyber-Physical Attacks. In *Proc. IEEE Computer Security Foundations Symposium (CSF 2017)*. IEEE Computer Society, 436–450.

[60] John Leyden. 2016. Water treatment plant hacked, chemical mix changed for tap supplies. *The Register* (2016). https://www.theregister.co.uk/2016/03/24/water_utility_hacked/ Accessed: May 2020.

[61] Qin Lin, Sridhar Adepu, Sicco Verwer, and Aditya Mathur. 2018. 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, 525–536.

[62] Yao Liu, Peng Ning, and Michael K. Reiter. 2011. False data injection attacks against state estimation in electric power grids. *ACM Transactions on Information and System Security* 14, 1 (2011), 13:1–13:33.

[63] Edwin Lughofer. 2017. On-line active learning: A new paradigm to improve practical useability of data stream modeling methods. *Information Sciences* 415 (2017), 356–376.

[64] Stephen E. McLaughlin, Saman A. Zonouz, Devin J. Pohly, and Patrick D. McDaniel. 2014. A Trusted Safety Verifier for Process Controller Code. In *Proc. Annual Network and Distributed System Security Symposium (NDSS 2014)*. The Internet Society.

[65] Jillian Morgan. 2015. *Streaming Network Traffic Analysis Using Active Learning*. Ph.D. Dissertation. Dalhousie University.

[66] Vedanth Narayanan and Rakesh B. Bobba. 2018. Learning Based Anomaly Detection for Industrial Arm Applications. In *Proc. Workshop on Cyber-Physical Systems Security and PrivaCy (CPS-SPC 2018)*. ACM, 13–23.

[67] Thuy T. T. Nguyen and Grenville J. Armitage. 2006. Training on multiple sub-flows to optimise the use of Machine Learning classifiers in real-world IP networks. In *Proc. Annual IEEE Conference on Local Computer Networks (LCN 2006)*. IEEE Computer Society, 369–376.

[68] Thuy T. T. Nguyen and Grenville J. Armitage. 2008. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials* 10, 1-4 (2008), 56–76.

[69] Fabio Pasqualetti, Florian Dorfler, and Francesco Bullo. 2011. 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, 2195–2201.

[70] Jan-David Quesel, Stefan Mitsch, Sarah M. Loos, Nikos Arechiga, and André Platzer. 2016. How to model and prove hybrid systems with KeYmaera: a tutorial on safety. *International Journal on Software Tools for Technology Transfer* 18, 1 (2016), 67–91.

[71] Peter Schneider and Konstantin Böttinger. 2018. High-Performance Unsupervised Anomaly Detection for Cyber-Physical System Networks. In *Proc. Workshop on Cyber-Physical Systems Security and PrivaCy (CPS-SPC 2018)*. ACM, 1–12.

[72] Sohil Lal Shrestha, Shafiul Azam Chowdhury, and Christoph Csallner. 2020. DeepFuzzSL: Generating models with deep learning to find bugs in the Simulink toolchain. In *Proc. Workshop on Testing for Deep Learning and Deep Learning for Testing (DeepTest 2020)*. ACM. To appear.

[73] Simone Silvetti, Alberto Policriti, and Luca Bortolussi. 2017. An Active Learning Approach to the Falsification of Black Box Cyber-Physical Systems. In *Proc. International Conference on integrated Formal Methods (iFM 2017) (LNCS)*, Vol. 10510. Springer, 3–17.

[74] Chad Spensky, Aravind Machiry, Marcel Busch, Kevin Leach, Rick Housley, Christopher Kruegel, and Giovanni Vigna. 2020. TRUST.IO: Protecting Physical Interfaces on Cyber-physical Systems. In *Proc. IEEE Conference on Communications and Network Security (CNS 2020)*. IEEE. To appear.

[75] Gayathri Sugumar and Aditya Mathur. 2017. 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, 138–145.

[76] A. Selcuk Uluagac, Venkatachalam Subramanian, and Raheem A. Beyah. 2014. Sensory channel threats to Cyber Physical Systems: A wake-up call. In *Proc. IEEE Conference on Communications and Network Security (CNS 2014)*. IEEE, 301–309.

[77] David I. Urbina, Jairo Alonso Giraldo, Alvaro A. Cárdenas, Nils Ole Tippenhauer, Junia Valente, Mustafa Amir Faisal, Justin Ruths, Richard Candell, and Henrik Sandberg. 2016. Limiting the Impact of Stealthy Attacks on Industrial Control Systems. In *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS 2016)*. ACM, 1092–1105.

[78] US National Science Foundation. 2018. Cyber-Physical Systems (CPS). https://www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf18538&org=NSF. document number: nsf18538.

[79] Giovanni Vigna, William K. Robertson, and Davide Balzarotti. 2004. Testing network-based intrusion detection signatures using mutant exploits. In *Proc. ACM Conference on Computer and Communications Security (CCS 2004)*. ACM, 21–30.

[80] Jingyi Wang, Jun Sun, Yifan Jia, Shengchao Qin, and Zhiwu Xu. 2018. Towards 'Verifying' a Water Treatment System. In *Proc. International Symposium on Formal Methods (FM 2018) (LNCS)*, Vol. 10951. Springer, 73–92.

[81] Yoriyuki Yamagata, Shuang Liu, Takumi Akazaki, Yihai Duan, and Jianye Hao. 2020. Falsification of Cyber-Physical Systems Using Deep Reinforcement Learning. *IEEE Transactions on Software Engineering* (2020). Early access.

[82] Cheah Huei Yoong, Venkata Reddy Palleti, Arlindo Silva, and Christopher M. Poskitt. 2020. Towards Systematically Deriving Defence Mechanisms from Functional Requirements of Cyber-Physical Systems. In *Proc. ACM Cyber-Physical System Security Workshop (CPSS 2020)*. ACM. To appear.

[83] Michał Zalewski. 2017. American fuzzy lop. http://lcamtuf.coredump.cx/afl/. Accessed: May 2020.

[84] Jun Zhang, Xiao Chen, Yang Xiang, Wanlei Zhou, and Jie Wu. 2015. Robust Network Traffic Classification. *IEEE/ACM Transactions on Networking* 23, 4 (2015), 1257–1270.

[85] Mu Zhang, Chien-Ying Chen, Bin-Chou Kao, Yassine Qamsane, Yuru Shao, Yikai Lin, Elaine Shi, Sibin Mohan, Kira Barton, James R. Moyne, and Z. Morley Mao. 2019. Towards Automated Safety Vetting of PLC Code in Real-World Plants. In *Proc. IEEE Symposium on Security and Privacy (S&P 2019)*. IEEE, 522–538.

[86] Peilin Zhao and Steven C. H. Hoi. 2013. Cost-sensitive online active learning with application to malicious URL detection. In *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2013)*. ACM, 919–927.