Availability Attacks on Computing Systems through Alteration of Environmental Control: Smart Malware Approach

Keywhan Chung, Zbigniew T. Kalbarczyk, Ravishankar K. Iyer
kchung10,kalbarcz,rkiyer@illinois.edu
University of Illinois at Urbana-Champaign
Urbana, Illinois

ABSTRACT

In this paper, we demonstrate the feasibility of smart malware that advances state-of-the-art attacks by (i) *indirectly attacking* a computing infrastructure through a cyber-physical system (CPS) that manages the environment in which the computing enterprise operates, (ii) *disguising* its malicious actions as accidental failures, and (iii) *self-learning* attack strategies from cyber-physical system measurement data. We address all aspects of the malware, including the construction of the self-learning malware and the launch of a failure injection attack. We validate the attacks in a data-driven CPS simulation environment developed as part of this study.

CCS CONCEPTS

• Security and privacy → Distributed systems security; **• Computer systems organization → Embedded and cyber-physical systems**; **• Computing methodologies** → *Machine learning*.

KEYWORDS

Cyber Security, Cyber Physical Systems, CPS, Malware

ACM Reference Format:
Keywhan Chung, Zbigniew T. Kalbarczyk, Ravishankar K. Iyer. 2019. Availability Attacks on Computing Systems through Alteration of Environmental Control: Smart Malware Approach. In *10th ACM/IEEE International Conference on Cyber-Physical Systems (with CPS-IoT Week 2019) (ICCPS '19), April 16–18, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3302509.3311041

1 INTRODUCTION

The growing sophistication of cyber-physical systems (CPSes), while enabling new breakthroughs in different application domains, also brings new vulnerabilities that can be exploited by malicious actors. In a typical CPS, a computing infrastructure provides support to the physical infrastructure, and the resiliency of such systems gets significant attention from both researchers and practitioners. However, we find that a different scenario, in which a closed-loop CPS provides essential services to the computing infrastructure, has not yet been well investigated. More specifically, we consider a scenario in which an attacker attempts to shut down a computing infrastructure and cause a system-wide outage by compromising the CPS with self-learning malware.

A weakness in the cyber security of CPSes and computing infrastructure (CI) can introduce an unforeseen threat. By utilizing the often underestimated dependency of computing infrastructure on surrounding CPSes, an adversary who targets a high-value computing infrastructure (e.g., a data center or supercomputer) can deploy an *indirect attack* by exploiting the relatively weak security of CPSes that are critical to the operation of the computing infrastructure. In this scenario, the vulnerability of the CPS acts as a weak point that lowers the security barrier of the otherwise well-protected system. As demonstrated in [5], an indirect attack can further thwart detection by replaying known failure scenarios in the CPS such that their impact cascades to the target CI.

In this paper we discuss advances of cyber-attacks in the context of indirectness and automation (driven by self-learning abilities). More concisely, we present a type of smart (self-learning) malware that exploits the weaknesses of a CPS to induce a malicious attack that (i) masquerades (to monitoring entities of the CPS) as an accidental failure, and (ii) stealthily propagates to the computing infrastructure, causing a major failure that includes a system-wide outage. To be specific, in the context of a real computing facility (i.e., Blue Waters), we present a learning malware logic that mimics failures or abnormal events in the cooling facility (i.e., the National Petascale Computing Facility) that eventually lead to an outage of the HPC housed in the facility. Building on the hypothetical attack scenario in [5], we implement and evaluate variations of attack scenarios (leveraged by the degree of automation and learning). The most naive approach randomly pollutes the data segments of random packets, whereas the most advanced approach carefully crafts an attack strategy inferred from operational data.

As demonstrated in this paper, self-learning malware is no longer a remote possibility. Our experimental results show that our malware can effectively (i) classify the CPS measurements related to failure incidents of the computing infrastructure in an unsupervised manner, (ii) identify core CPS parameters related to the regulation of cooling capacity, and (iii) infer abnormal measurements in the parameters that triggered anomalies in the CPS whose impact propagated to the computing infrastructure. While the advances discussed in this paper might seem to benefit malicious entities, our approach (especially for inferring the attack strategy) can be used to preemptively eliminate the potential risks by detecting sensitive anomalies in real time; it can also be used to reinforce the control logic to properly handle such anomalies (see Section 5). Furthermore, the CPS simulator presented in this paper can be used to evaluate the resilience of the CPS against unforeseen abnormal conditions.

ICCPS '19, April 16–18, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6285-6/19/04. . . $15.00
https://doi.org/10.1145/3302509.3311041

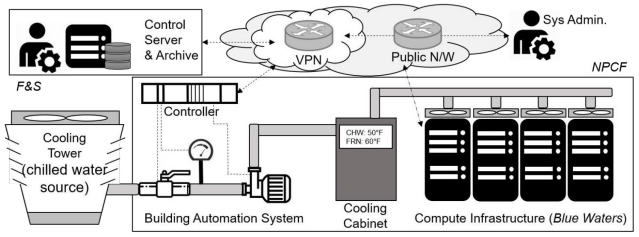


Figure 1: Chilled water system overview.

The contributions of this paper are the following:

- We show the possibility of an *indirect attack* on a computing infrastructure through intrusion in a cyber-physical system.
- We present *smart malware logic* that can learn attack strategies from CPS data and replay failure scenarios. The malware implemented 3 attack strategies that corrupt the cooling capacity.
- We present a *data-driven CPS simulator* and its usage in evaluating the resilience of a CPS against the malware. Comparison of simulation data with real data found a mean error of 3%.
- We discuss how our findings as a white-hat hacker can be used to secure a system from advanced threats.

2 SYSTEM OVERVIEW

As shown in Figure 1, we consider a typical configuration of a large-scale computing facility instantiated in the context of the Blue Waters supercomputer. In this configuration, the operational environment (e.g., room temperature, delivery of chilled water to cool the computing nodes) of the computing infrastructure is managed by a building automation system (and associated CPS) that optimizes the control over the actuators, given a set of measurements representing the parameters of the operational environment.

2.1 Computing infrastructure: Blue Waters

In our attack model, we set a computing infrastructure, or, to be more specific, its availability, as the final target. In this paper, we study the case of Blue Waters, a 13.3-petaflop supercomputer managed by the National Center for Supercomputing Applications (NCSA) at the University of Illinois. Blue Waters is composed of around 300 computing cabinets, each of which houses around 90 nodes (i.e., 3,000 16-core processors per cabinet, or 27,000 in total) [6]. To prevent damage to the physical parts from the massive heat generated during the operation of the supercomputer, a liquid-based cooling system has been deployed. For Blue Waters, 72 cooling cabinets are placed alongside the compute nodes, and each cabinet prepares and delivers Freon to the three adjacent compute nodes. The liquid form of Freon, travelling through the rack of compute nodes, then absorbs the heat from the compute nodes and returns to the cooling cabinet in the form of a gas.

2.2 Cyber-physical system: NPCF building automation system

The Blue Waters supercomputer is housed in a dedicated building, the National Petascale Computing Facility (NPCF). The 88,000-square-foot building uses a state-of-the-art building automation system (BAS) that is in charge of regulating the environmental parameters (i.e., pressure, flow, and source of the cooling system) of the building, including the server room. A detailed configuration of the system can be found in [5]. The building automation system (which consists of a control server, a set of programmable logic controllers-PLCs, sensors, and actuators) utilizes a set of measurements collected from the chilled water loop to regulate the chilled water delivered to the cooling cabinet, under three modes of operation: campus mode, mix mode, and economic (econ) mode. *Campus mode* is the used mode in most data centers that use chilled water bought from external providers. While the chilled water from such providers is well controlled (i.e., the temperature, flows, and pressure are kept within an agreed range), its usage results in an increase in the cost of operation. To reduce the cost of operation, NPCF has a set of dedicated cooling towers (which use cold temperatures to naturally chill the water) by means of which it can deploy an additional mode of operation: *economic* mode. By taking advantage of cold external temperatures throughout 2/3 of the year to prepare the chilled water, NPCF was able to significantly reduce its cost of operation, which compensated for the construction costs of the water towers after one year of operation. The *mix* mode is an intermediate mode that was introduced to enable smooth transition between economic and campus modes.

2.3 Data

The CPS operational data set is an archive of all measurements and control command values within the chilled water system of the building facility that were collected from September 2016 to May 2017. The data set contains 47 distinct parameters (see Table 1 in Appendix A) collected every 5 minutes. Sample parameters monitored and collected within the chilled water system include differential pressure, flow, and temperature of the campus input ("CAMP.CHW.DP, FLOW, TEMP"), control valve setting and measurements at the high loop ("CHW.HI.CV, TEMP, FLOW"). This data set is essential for analyzing the operation of the CPS and inferring critical information related to failure of the computing infrastructure.

The incident reports log incidents related to the computing infrastructure since the deployment of the system (i.e., December 2012). The incidents recorded in the reports include hardware part failures, cooling-system-related problems, and system-wide outages (SWOs) during which all 28,164 compute nodes were shut down.

2.4 Threat model

In our threat model, we assume:

- The cooling control system (CPS) is *less secure* than the target computing infrastructure and is relatively easier to exploit. (This motivates our indirect attack model.)
- The attacker (i) *can get (remote) access* (e.g., through phishing, stolen credentials, or insider attack) to the control network and the CPS control server, (ii) *can identify* the target parameter(s) (e.g., the flow and temperature of the chilled water supplied to the server room) based on the information available through the control server interface, and (iii) does not have access to the incident reports on the failures of the computing infrastructure.
- Our smart malware (i) *has access to* the data archive that stores the CPS measurements and the copies of the PLC programs kept in the control server as a backup of the actual logic that had been loaded onto the PLC, (ii) *can automatically derive attack strategies*

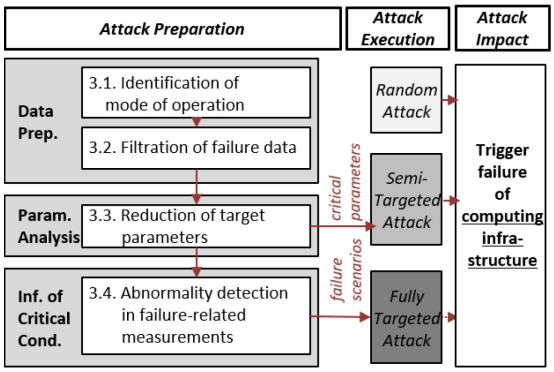


Figure 2: Approach and attack modes.

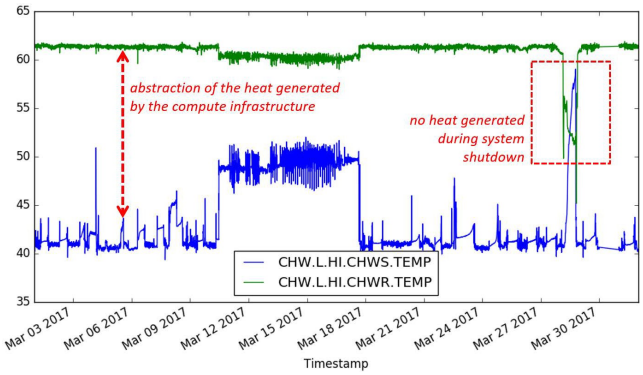


Figure 3: SWO observed through chilled water supply and return temperature measurements.

(as described in detail in Section 3), (iii) *can inject malicious control code* (that checks for the attack-triggering condition(s) and overwrites the critical parameters) into the controller program, and (iv) *can upload the program to the controller* by using the control server APIs. The technical details of the control server APIs and PLC control logic format can be obtained from publicly available documents on a given CPS (e.g., [22, 23]).

3 SELF-LEARNING MALWARE: AN UNSUPERVISED APPROACH

In this section, in terms of a set of steps (depicted in Figure 2), we describe our approach to developing self-learning malware that is capable of inducing malicious attacks against a CPS that controls the environment in which the computing infrastructure operates.

3.1 Step 1: Identification of mode of operation

Goal. As discussed in Section 2.2, a BAS relies on multiple input sources to deploy different modes of operation. The control logic and the measurements differ depending on the mode of operation, and hence, it is critical to accurately classify the data by the mode of operation to infer characteristics specific to each mode.

Approach. We apply (i.e., our malware runs) k-means clustering [11] to classify the data by the mode of operation. With the goal of interrupting the cooling capacity, we choose the two parameters that define the cooling capacity delivered to the server room: chilled water flow *(chw_flow)* and temperature difference *(tempDiff)* between the supply and return water.

Results. The clustering algorithm, with a k value of two, effectively captures the data clusters (campus and econ mode). When compared to the ground truth, we achieved a 97.96% true-positive rate (0.04% false-positive rate) for *campus mode*, a 99.76% true-positive rate for *econ mode*, and 98.62% overall accuracy.

3.2 Step 2: Filtration of failure data

Goal. To design attacks that masquerade as accidental failures, the primary step is to parse the CPS data that are related to system failures. However, with our assumption that the intrusion will be restricted to the level of the building control network, the identification of failures in the computing infrastructure becomes a nontrivial problem, as the information available to the attacker is limited. (I.e., the attack does not have access to failure logs collected from the computing infrastructure.) In order to overcome that limitation, we present an indirect approach to predicting the state of the computing infrastructure from the available CPS data.

Approach. A feasible approach is to infer the system status from the *returning chilled water temperature*, the only parameter within the measurements that reflects the status of the computing infrastructure of interest. In Figure 3, we plot the supply water temperature to the data center (*"CHW.R.HI.CHRS.TEMP"*) and the return temperature from the data center (*"CHW.R.HI.CHWR.TEMP"*) during a month of operation (March 2017). The distance between the two plots reflects the amount of heat generated by the computing infrastructure. What we focus on is the occasional nadirs (drops) among the relatively constant (compared to the supply temperature) return temperature. The nadirs indicate events in which the computing infrastructure did not generate heat, which happens only when the computing infrastructure is down. As a result, by detecting the nadir, the attacker can parse data related to maintenance of Blue Waters or its surrounding systems, and outages of Blue Waters.

Results. Analysis of CPS operational data over 9 months shows that the nadir detection approach reduces the data of interest from 72.5K entries (i.e., measurement points) to 2,938 entries (a 96% reduction). The 2,938 entries can be coalesced into 23 distinct events. (An event is represented as a set of measurement points collected within $\pm$ 3 hours of the extreme nadir.) By cross-validating those data with the incident reports (which are not available to external entities or attackers) that manually log the failure incidents of Blue Waters, we found that our prediction model captured 3 out of 5 system-wide outages that occurred during the period of study. The two SWOs that were not captured with our approach occurred when an outage did not last long enough for its impact to be visible at the chilled water system level. Reduction and filtration of the data in this step ensured that (i) the correlation analysis (Section 3.3) inferred the relationships among the parameters during normal operation (by excluding the data that were predicted to be related to computing infrastructure failure), and (ii) the abnormality analysis (Section 3.4) captured the anomalies in the failure data set by comparing it with the normal-operation data set.

3.3 Step 3: Reduction of target parameters

Goal. Within the control system under study, measurements of 47 parameters are monitored and shared among the controllers. While each and every parameter plays a role in the control logic, not all parameters are closely related to the goal of the attacker,

i.e., disrupting the delivery of chilled water to the computing infrastructure. While malware can be designed in a naive manner to pollute random parameters, its likelihood of success would be extremely low despite the high likelihood of exposure (or detection). Instead, we apply a statistical method to carefully choose the target parameters to be altered by our malware.

Approach. The attacker's goal is to reduce the cooling capacity of the chilled water delivered to the server room, and hence the attacker needs to monitor the final target parameters: the temperature and volumetric flow of the supplied chilled water. Given a series of measurements of the 47 parameters, the malware computes correlation between each pair of parameters, where a value closer to 1 (or −1) represents the significance of a linear relationship between the two parameters.

Results. For each mode of operation we choose the top ten parameters with the highest correlation. Example parameters include control valve setting, pump speed, and input water temperature/flow and pressure. In addition to reducing the number of target parameters, our correlation-based approach captures the relationships among the critical parameters to minimize contextual inconsistency during false-data injection. (I.e., when injecting false data into a critical parameter X, we also appropriately alter the data for parameters that report high correlation with the parameter X.) The reduced data set consists of 23 distinct events with 10 critical parameters each. In the next step, we filter out events with no abnormal values of critical parameters.

3.4 Step 4: Filtration of events with abnormal measurements

Goal. In step 2, we reduced the CPS measurement dataset to 23 events that are predicted to be related to a CI outage. In this step, we further reduce the number of events and **its** critical parameters by selecting those events for which the critical parameters show abnormal values. (We consider those events to be the root cause of the CPS-induced CI outages).

Approach. As part of defining an abnormal event for a parameter, we assume that the parameter values can be represented as a Gaussian distribution. Given a Gaussian distribution, measurements with z-scores greater than 3 are considered as outliers. For a stricter bound (to reduce false positives), we consider a measurement to be abnormal if its z-score exceeds 4. (In a perfect Gaussian distribution, 99.9936% of the measurements have z-scores smaller than 4.) To be specific, for each parameter in the 23 events, we check if there is an abnormal measurement that exceeds a threshold (whose $z = 4$). We remove the parameter from the list of critical parameters if no abnormal measurement/value is detected for this parameter. Also, if all the critical parameters are removed for an event, then we consider that event as "inappropriate for an attack strategy." After running the abnormality analysis over each critical parameter (from Section 3.3) for all events (from Section 3.2), we are left with a set of events to be implemented as part of an attack strategy.

Results. To demonstrate the effectiveness of this step, we applied our analysis to a known incident in the chilled water system that had impacted the operation of the computing infrastructure. Figure 4 shows the flow of the chilled water to the computing infrastructure (CI) during a maintenance operation on the campus loop. A closure in the tank of the campus chilled water (CHW) loop resulted

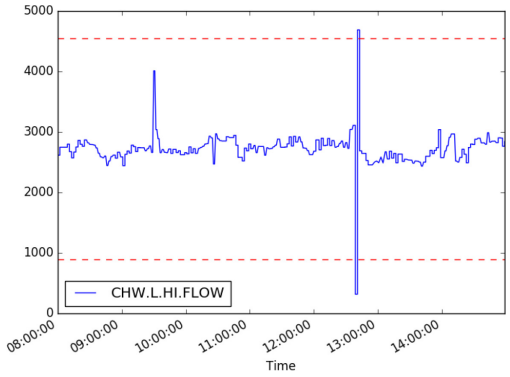


Figure 4: Loss of flow due to campus maintenance operation.

in an increase in CHW flow to the CI around 9:30; the CPS immediately regulated it by closing the valves. Later, after the maintenance operation was over around 13:00, the input flow returned to normal. However, with the current settings of the valve, the flow to the CI faced a dramatic drop that resulted in a lack of cooling capacity for a set of compute cabinets and caused an emergency power off (EPO) because of the increase in the internal temperature. The thresholds indicating the abnormal measurements of the parameter (chilled water flow) in Figure 4 ("CHW.L.HI.FLOW") are depicted as dashed lines. The figure shows that our threshold effectively identified the abnormal event, i.e., the dramatic drop in the water flow around 13:00 that was the root cause of the CI outage.

When we apply (or the malware applies) the abnormality detection method to each candidate parameter of the 23 events (identified in steps 1—3), we end up with 7 events that each include at least one abnormal measurement. In Section 3.5, we evaluate the events (in the context of causing a CI outage) and describe how the malware constructs attack strategies based on the selected events.

3.5 Variations of attack levels

The attack strategies that can be implemented by our malware are classified into three levels of sophistication (Figure 2): *random attacks*, *semi-targeted attacks*, and *fully targeted attacks*. Each level differs from the others in terms of the difficulty of preparing the attack and the likelihood of detection.

Random attack: This attack poisons the measurements of the BAS by overwriting the data segment of random parameters with random values. With consistent alteration of the measurements, there is a possibility that a malicious measurement will cause a BAS to make a faulty decision that leads to failure of the computing infrastructure. However, no sophistication is involved in this attack; hence, it has a low probability of success (from the attacker's perspective) and a high probability of detection (due to the lack of contextual consistency in the measurements). This attack is more likely to be effective if the attacker has some understanding of the physical system, but most likely there would be only one opportunity to carry out such an attack, since it would leave a highly visible, abnormal trace.

Semi-targeted attack: This attack strategically filters the parameters of interest. By reducing the target parameters, the attacker can increase the likelihood of success. Using the method discussed in Section 3.3, we identify 10 target parameters (out of 47 parameters available in the BAS) per mode of operation. When we validated

the 10 parameters with human expertise (which is not available to attackers), we found that 6 were critical to preparation of the chilled water for the campus mode and can potentially cause failure of computing infrastructure when corrupted. For the econ mode, we found 5 parameters to be critical. While the approach can increase the likelihood of a successful attack by reducing the target parameters from 47 to 5, the value to be injected into the reduced set of parameters remains unknown. A random injection into a selected set of parameters would leave the success of the attack at risk (though its chances are better than those of *random attacks)*.

Fully targeted attack: A smart attack, which we claim as the core contribution of this paper, aims at minimizing the probability of exposure while increasing the probability of success. Our attack model accomplishes that goal by emulating a failure incident. By applying the statistical methods discussed in sections 3.1–4 over the CPS data set, the malware infers seven attack strategies, where each strategy includes (i) *critical parameters* that can trigger a CI outage, (ii) *a sequence of values to overwrite* for each critical parameter, and (iii) *a critical condition* to initiate the attack (i.e., overwrite the values for each parameter with the sequence of values from the attack strategy). By cross-validating the timestamp of the strategies with the incident reports, we found that three of the seven strategies (4 for campus mode and 3 for economic mode) successfully captured incidents related to the CPS (see Table 2 in Appendix A). The remaining four strategies detected anomalies introduced by "the change of CPS operation mode" that was demonstrated to be "likely to cause a valve failure" (i.e., hardware failure in the control valve) in the cooling cabinet ([5]), or detected an anomaly in the return water temperature caused by the scheduled shutdown of the CI, which cannot cause a failure of the CI. Also, as occurrence of failures caused by "changes in operation mode" remains probabilistic, we left those strategies out of this paper.

4 RESULTS: ATTACK STRATEGIES AND THEIR DEPLOYMENT VIA THE SELF-LEARNING MALWARE

After the malware infers the strategies to be implemented as attacks, the malware identifies the controller (PLC) that uses the critical parameters as inputs or derives the values for the critical parameters as outputs. For the identified PLC, the malware modifies the PLC program (i) to check for attack- triggering conditions and (ii) tp overwrite the normal values of the critical parameters with the sequence of abnormal values defined by the attack strategy. The malware completes the preparation of the attack by uploading the modified program to the PLC. In the remainder of this section, we introduce the three attack strategies (from Section 3.5) and describe their implementation as smart malware.

4.1 Strategy 1: CHW abnormality due to a power interrupt (S1)

Incident. As shown in Figure 5a, an abnormality occurred in the temperature of the input chilled water ("CAMP.L.CHWS.TEMP", "CAMP.CHWS.TEMP") from the campus chilled water plant. Specifically, the input temperature increased by 10°F above the usual value (from 40°F to 50°F) and remained high for 30 minutes. By relating this observation to the corresponding incident report, we found that a power interrupt at the campus chilled water plant had impacted the computing facility. For this incident, the impact did not cascade into the computing infrastructure, because the incident occurred when computing infrastructure was not being heavily used (i.e., the computing infrastructure generated less heat and did not require significant cooling), but the consequences would have been different if the incident had happened during peak hours.

Attack. When this strategy is implemented and triggered as an attack, the malicious PLC logic on the controller overwrites the water temperature measurements as if the temperature had increased and forces the CPS to respond to the apparent abnormality. The CPS, in response to the "abnormality", increases the water flow to compensate for the (fake) loss of cooling capacity, which leads to excess cooling capacity to the CI cooling cabinets in the data center (i.e. a violation of the service-level agreement between the facility and the computing infrastructure). To avoid reaching the dew point, the CI cooling cabinets close the valve to slow down the flow. The attack becomes effective when the parameter values go back to normal. Once the (fake) temperature returns to normal, the CPS starts to close the valves, which reduces the flow of the chilled water to the CI. However, since the CI cooling cabinets do not react in a timely manner [5], the CI encounters a shortage in cooling capacity that increases the risk of an EPO.

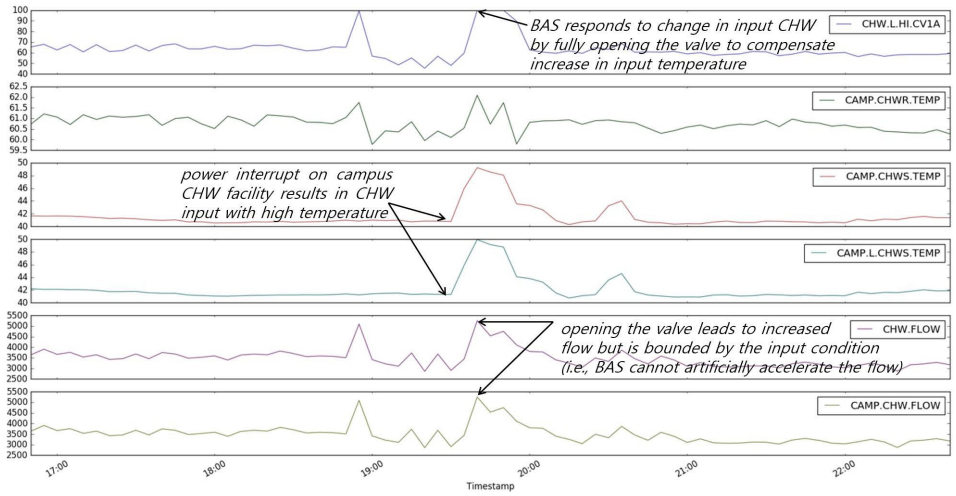
4.2 Strategy 2: Maintenance operation (S2)

Incident. Figure 5b shows a maintenance operation (replacement of a water valve) that was identified by the smart malware. Around 03:00, the CV1A valve gets closed even though the CPS was operating in campus mode. (The operation was allowed because the CI was shut down for planned maintenance.) The flow of the chilled water ("CHW.FLOW", "CAMP.CHW.FLOW") dramatically decreases, resulting in low cooling capacity. The entire event (including the process of replacing the control valve) lasted for 18 hours.

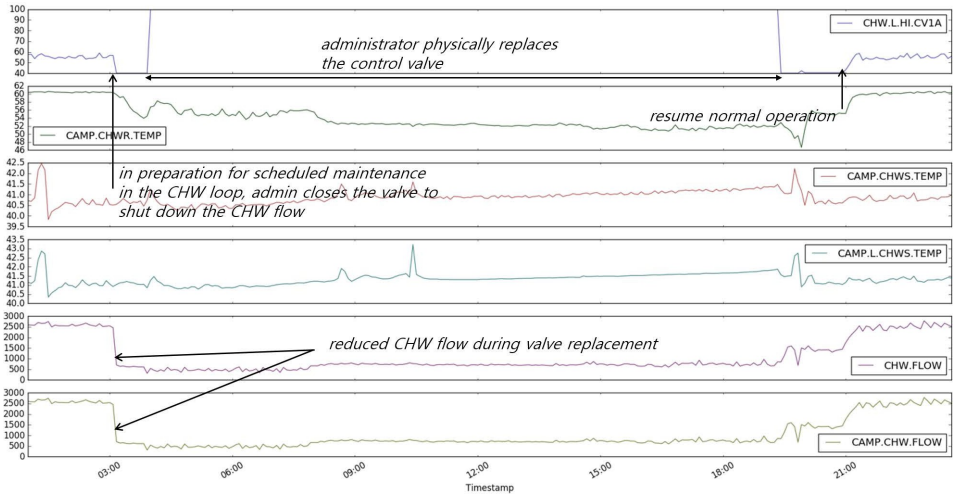
Attack. When this strategy is implemented and triggered as an attack, the malicious PLC logic on the controller overwrites the control valve opening ("CHW.L.HI.CV1A") with the strategic values (i.e., 40%). Without noticing the corruption in the command value, the actuator closes the valve for an hour, which results in reduced CHW flow (by 75%) and eventually a failure to deliver sufficient cooling capacity to the system (because of reduced chilled water flow); that failure would lead to an SWO of Blue Waters. As this attack emulates a maintenance operation, it can be relatively easy for the operators to notice it, and they may try to fix the problem by overwriting the suspicious commands from the PLC with manual commands from the control server. In such a case, the incident will not last for the entire period of the scenario (24 hours). However, as the impact of such an extreme loss in cooling capacity would quickly (i.e., in 2—3 minutes) cascade into the CI, the operator would not be able to prevent the SWO.

4.3 Strategy 3: Reduced cooling for an emergency outage (S3)

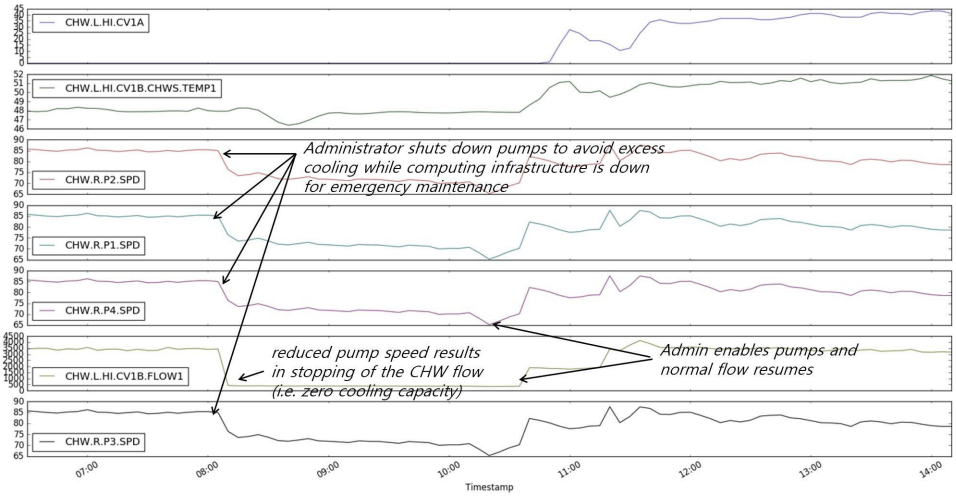
Incident. The last scenario captured by the malware is a reduction of chilled water flow in preparation for an emergency outage of the computing infrastructure. In economic mode, control valve 1B, which manages the water from the cooling tower, is fully open because of the relatively high input temperature, and the cooling



(a) Supply water temperature abnormality due to power interruption.



(b) CHW loop closure for maintenance operation.



(c) Reduced cooling capacity for emergency outage in computing infrastructure.

Figure 5: Attack scenarios derived by the smart malware. (A description of the parameters can be found in Appendix A.)

capacity is mainly managed by the pumps controlling the flow in the closed loop. As shown in Figure 5c, the speed of the four pumps (refer to "CHW.R.P1-4.SPD" in Figure 5c) is reduced, which results in a lower flow of chilled water, and eventually, low cooling capacity for the CI.

Attack. The reduction of cooling was intentional because the operator wanted to avoid reaching the dew point during the emergency outage. However, when the incident is replayed as an attack during normal CI operation, it can trigger an SWO of the CI. When this strategy is implemented and triggered as an attack, the malicious PLC logic on the controller overwrites the pump speed commands ("CHW.R.P1-4.SPD"). As in the real incident, the flow is reduced accordingly, which results in reduction of the cooling capacity available to the CI. As the cooling system fails to remove the heat in the CIs, the compute cabinets start to trigger EPOs to prevent critical damage in the hardware.

4.4 Malware implementation

Attack timeline. As demonstrated in [20], the majority of security attacks are detected by catching (i) the payload that the attacker executes or (ii) the trace left by the attacker in keeping a foothold in the system (e.g., a remote network connection). Under proper security monitoring, the likelihood of attack detection increases with the length of time the attacker stays in the system (which must be lengthy to enable investigation of the target and design of the attack) [26]. We consider the possibility that the attackers may avoid detection by deploying self-learning malware that can infer attack strategies (specific to the target system) from the data available in the system. In Figure 6, we show how an attack using our self-learning malware proceeds. The attacker starts by sending a phishing email to the CPS administrator (❶) to obtain the credentials of the administrator (❷). Using the credentials, the attacker gains access to the VPN-protected control network and the control server, and installs the smart malware (❸). Once started, the malware runs the learning algorithm (discussed in detail in Section 3) with the CPS measurement data archived in a database and derives potential attack strategies. The strategies are then injected into the controller (refer to Algorithm 1 in Appendix B) along with the attack control logic that determines when to trigger the attack (❹). In ❺, the malware updates the logic embedded in the controller with the malicious logic. Once embedded into the PLC (programmable logic controllers), the malicious code monitors the measurements and checks for a triggering condition (to maximize the probability of a successful attack,❻). When the condition is met, the output of the controller is overwritten with the strategic values (from ❹) to initiate a failure-triggering abnormal event (❼). The CPS responds to the abnormal measurements (i.e., tries to regulate the chilled water accordingly,❽), but, as in the failure incidents, the response is not satisfactory, and the impact (of loss of or excess cooling capacity) cascades to the compute cabinets (❾), triggering an EPO or an SWO.

Minimization of traces. The overhead of our approach is that it requires computing resources during data analysis. That requirement can be reduced via a number of tricks. First, instead of executing the malware as a single block executable, we divide the source code into snippets of logic modules (divided by functionality as described in Section 3), and have each module span the next module as an independent thread. After finishing its job, and executing the next module, it can terminate its execution, and the next module can remove the traces of the previous execution. Using that approach, detection of the malware's execution becomes harder as the execution proceeds. (Recall that the later steps require fewer computation resources because of the reduction of data.)

5 DISCUSSION

Generalization. We discussed our malware in the context of the CPS that manages the cooling system of Blue Waters. However, we believe that the same malware (with little or no modification) can be used in other computing facilities (HPCs, data centers, or the cloud) with dedicated cooling systems, as long as they use a CPS from the same vendor. Such a generalization is feasible because the attack strategies are not hard-coded into the malware, but dynamically derived from the environment that the malware observes. If the CPS were from a different vendor, the malware would need to be modified/adapted to account for different ways of programming and uploading a control logic to the controller(s). The rest of the procedures that constitute the presented self-learning malware (i.e., steps A through D as depicted in Figure 6) should remain the same.

The success of the self-learning malware, as presented in this paper, depends on the existence of CPS failures that lead to outages of the computing infrastructure. The events that would benefit the attacker (or malware) are rare (at NPCF, 2.73% of the 5,419 incidents from 2013 to 2016 were cooling-system-related [5]), and there is no guarantee that the malware can capture all such events.

Finally, although we focused on indirect (via CPS) attacks against the computing infrastructure, the ultimate success of an attack depends on the workload on the computing infrastructure, which is beyond the control of an attacker. For instance, in the case of a cooling system failure, only compute cabinets with high computational load (during the incident) can trigger an EPO.

Detection. Having demonstrated that malware with learning aspects is a potential risk, we must consider how to protect and secure our systems. System administrators, trying to protect a system from advanced attacks, can take advantage of the approach discussed in Section 3 in a supervised manner. In the following, we briefly discuss example feasible approaches. One is to *detect data obfuscation and abnormalities* using the steps in Section 3. The CPS under study deploys redundancy to enhance reliability, i.e., the transition to the backup CPS components happens only upon detection of a failure (e.g., a timeout on receiving heartbeat packets), and no validation process exists to detect malicious tampering with the measurements. As a solution for data tampering, we consider an intrusion detection system that analyzes data collected from the control network. For instance, the Bro IDS (intrusion detection system [19]) can be deployed in the control network. Bro can be used as a resiliency monitor to detect abnormal events (e.g., sudden changes, outliers, or measurements inconsistent with earlier observations). More specifically, unlike the malware, which has to conduct unsupervised learning because of limited information, the system administrators have full knowledge of (i) the CI and CPS, (ii) the data collected from those systems, and (iii) the ground truth on the CI status (which can be used to label the data).

Another feasible approach is to deploy *multi-layered monitoring*. We consider a monitoring strategy that collects security events from

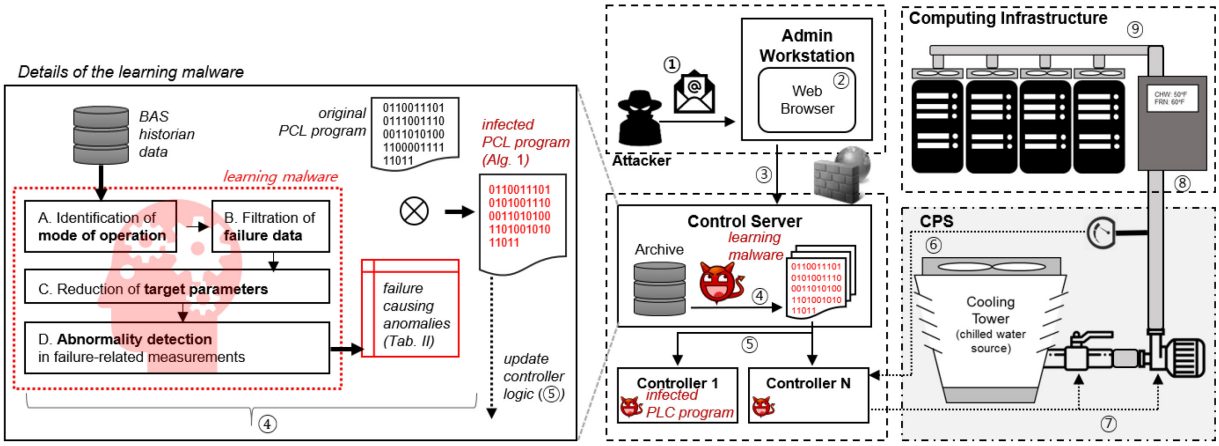


Figure 6: Overview of the fully targeted attack model (smart malware).

the CPS and the computing infrastructure. We can build probabilistic graphical models that effectively capture dependency among events and system states. For example, a factor graph based framework, for preemptive detection of security incidents, has been proposed in [3]. By correlating events that represent the behavior of the CI and CPS, we can enable early (i.e., before the CI is compromised) detection of security incidents.

6 VALIDATION

While the results described in the previous sections demonstrate that our malware logic can effectively parse attack scenarios (i.e., data from the time interval during which an anomaly in the CPS affected the operations of the computing infrastructure), its impact when injected is yet to be verified. To evaluate the effectiveness of the attack, we have implemented a software-driven simulation framework that replicates the operations of the BAS under study and (i) is driven by real CPS operational data, (ii) emulates the chilled water regulation process of the BAS, (iii) provides a cyber interface for the CPS used for communication and control of the components, and (iv) can test attack scenarios (as described in Section 4.4). After validating our framework's ability to emulate effectively the operation of the BAS, we evaluate the impact of the attack strategies generated by the smart malware by injecting the attacks in the simulated environment.

6.1 CPS simulation framework

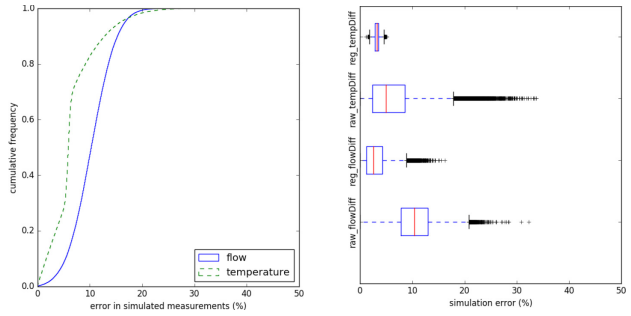
The most effective approach for validation would be to inject the attack scenarios into the CPS and its automation system. However, such experiments are necessarily restricted because (i) we cannot perform such an experiment without impacting users; (ii) the computing infrastructure is heavily used (24/7) for scientific research, so there is no down time for such experiments such that no users are impacted; and (iii) the final impact of the experiment is unknown (and could include hardware damage). Also, configuration of a physical test environment, which might be considered as an alternative, would not be easy, considering the scale and nature of the system (i.e., CPS of a building managing tons of water). Our simulation-based approach addresses such concerns by emulating the physical interaction in software. For extended usage of the simulator (i.e., for usage in CPSes other than our BAS under study), we have implemented the simulator in a layer-based architecture. As our simulator emulates the full functionality of the CPS (i.e., from the low-level interaction in the physical layer to communication between the controllers and control server in the cyber layer), we can efficiently evaluate the security and reliability of the CPS.

The simulation framework consists of a set of APIs, with the system divided into three layers: a physical layer (which models the interactions within the physical components), a programmable logic layer (which serves as the programmable logic controllers), and a cyber layer (which emulates the interaction between the controllers and the control server). We expect the simulator users to focus simply on the layer of interest, with the details in irrelevant layers remaining abstract.

Physical layer. Each of the physical components and their activities is defined as a Python function. In the current simulator (for the chilled water BAS under study), such components include the `pump`, `control valve`, `T-junction merge`, `T-junction distribution`, and `heat exchange`. Using Bernoulli's incompressible flow equation [12], we define the fluid-mechanical activities (i.e., changes in pressure, flow, and temperature) within each component. The inputs for each implemented component describe the physical characteristics of the chilled water fed into the component. By combining the inputs with the control values (from the controllers), the function can derive the physical characteristics of the output.

Programmable logic layer. The PLCs in the CPS are a critical component in the cyber layer of the CPS, as they provide the core intelligence that automates the control of the physical layer. For each controller in the system, we implemented the controller logic code (in our case from NPCF) in the Python language. Taking the same set of inputs, our Python code derives the output by using the logic described in the controller logic code. When using our framework for simulating a CPS other than our BAS, users must properly convert the controller logic code into the programming language of the framework. For such users, we provide a set of common instructions in the control logic code as Python functions.

The cyber layer provides interfaces for communication between the controllers and the server. By default, we provided the interface with the UDP protocol, but it can be extended to use other commonly used (or even custom) protocols (e.g., BACNet or any



(a) Cumulative frequency. (b) Boxplot.

Figure 7: Comparison of real and simulated data for flow and temperature of the chilled water going to the computing infrastructure.

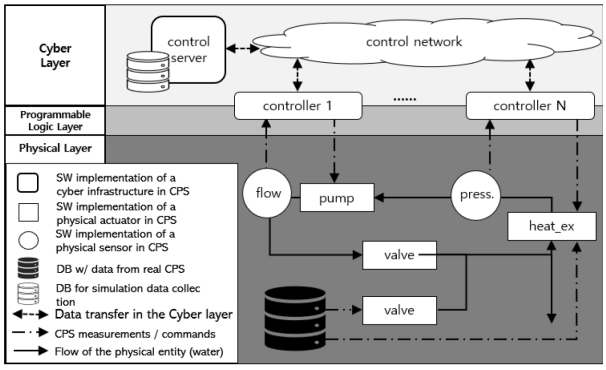


Figure 8: NPCF chilled water system simulator.

proprietary protocol). In addition, the API provides a feature for updating the controller logic. Since the controller logic is defined as a separate Python file, one can update it by executing a command in the control server. A summary of the interfaces available in the framework can be found in Table 3 in Appendix A.

6.2 Validation: simulation framework

To demonstrate the ability of the simulator to represent real operations of the system under study, we ran the simulator with input conditions (i.e., measurements of the chilled water input to the building) from the real system. Using the simulator APIs, we constructed a model of the CPS under study (Figure 8), and ran a simulation by feeding the system with real data that describe (i) the condition of the chilled water fed from the campus, and (ii) the condition of the chilled water returning from the server room (with the heat absorbed from the CI).

As shown in Figure 7, we found that the level of abstraction was sufficient for our purpose of replicating the physical system. In Figure 7a, the measurements from the simulator show a constant offset (by 403.07 gallons per minute for the flow and by 2.95°F for the temperature). In terms of error rate, a raw comparison (simply comparing the measurements from the simulator with those in the data) found error rates of 10.44% for the flow and 6.07% for the temperature. Those differences are due to the assumptions made in the design of the simulator (for instance, negligible major loss or loss due to friction). The excess energy (compared to the real operation) is represented as increased flow in the simulation. Furthermore, as the temperature of the chilled water is affected by the flow, the impact of the additional flow cascades into the temperature. We acknowledged that the offset was due to the abstractions adopted within the process of representing the real system with a simulator, so we removed the constant offsets from the simulation data. After we removed the offsets, the percentage difference between the simulation data and real data was reduced to 2.95% for the flow and 3.20% for the temperature (as shown in Figure 7b). Having confirmed the validity of the simulator, we could then evaluate the effectiveness of the strategies derived by the smart malware.

The simulation of the current framework is limited to the BAS studied in this paper, and the components of the framework are limited to those needed to simulate the BAS. For extended easy use of the simulator (e.g., for HVAC or water distribution), more components need to be added. In addition, in simulating physical aspects, the framework utilizes an abstract model that deploys a set of assumptions. Such an abstraction was reasonable for our purpose of verifying the smart malware strategies. However, for a more accurate simulation of the physical aspects, such assumptions should be removed, and the APIs should be updated accordingly.

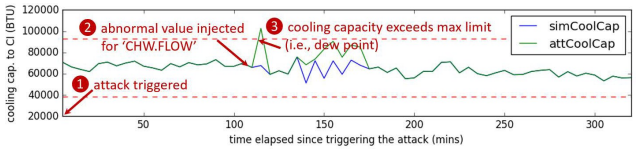
6.3 Validation: smart malware

Having verified the effectiveness of the CPS simulation framework, we now discuss the results of replaying the smart attack strategies (as described in Section 4) in the simulator through the following procedure implemented as part of the self-learning malware. (i) The malware analyzes the CPS operational data (archived in the database) and infer an attack strategy (e.g., altering measurements on the water temperature); (ii) it then embeds the selected attack strategy by injecting malicious code in the controller logic program of the target controller; (iii) it uploads the maliciously crafted controller logic program to the controller (PLC); and (iv) the malware then uses the maliciously altered PLC to monitor attack-triggering conditions, and, when the triggering conditions are satisfied, replaces the input/output values of the controller with the strategic values derived in step (i).

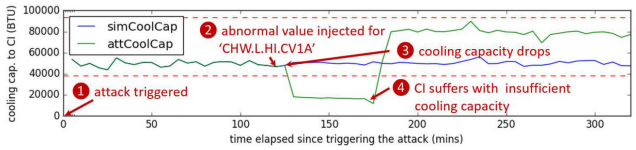
In order to validate the attack consequences, we set an upper and lower bound/limit on the cooling capacity. The cooling capacity is derived by multiplying the volumetric water flow by the temperature difference between the water going to and the water coming from the server room; the cooling capacity is measured in British thermal units (BTU). An attack is considered successful if the check on chilled water cooling capacity passes (i.e., the CPS perceives the supplied water cooling capacity to be within the predefined bounds) while the CPS measurements were being maliciously altered by our malware.

By injecting attacks in the simulated environment, we confirmed that the identified attack strategies can effectively introduce anomalies in the CPS such that the cooling capacity provided to the server room falls outside the normal range.

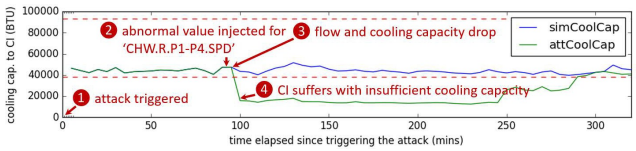
S1: CHW abnormality due to a power interrupt: 110 minutes (in simulation time) after the malicious PLC logic triggered the attack, the values for CHW flow measurements ("CHW.FLOW") were overwritten with abnormal values (derived by the smart malware), and the cooling capacity exceeded the upper limit of the maximum



(a) Simulated attack strategy 1 (S1)



(b) Simulated attack strategy 2 (S2)



(c) Simulated attack strategy 3 (S3)

Figure 9: Simulated impacts of attack strategies. (Dotted lines indicate the min/max normal cooling capacity.)

range. This anomaly occurred because the control system (given the fake data that reported an increase in chilled water input) increased the flow (to compensate for the increased temperature) by opening the control valve. However, as the real temperature was normal, the increased water flow resulted in excess cooling capacity, which is something the administrators try to avoid because of the risk that dew might form in the CI (Figure 9a).

S2: Maintenance operation (valve replacement): In this attack strategy, the malicious PLC logic started to provide the control valve opening command ("CHW.L.HI.CV1A") with abnormal values, 125 minutes after the attack was triggered. Initially, the controller (with its original logic) derived the correct control valve command (because the malicious PLC logic did not corrupt the input parameters). However, before sending out the command values, the malicious logic overwrote the correct values with the strategically crafted values, which caused the cooling capacity to drop below the lower limit. As shown in Figure 9b, while the simulation without injection kept the cooling capacity within the predefined boundaries, the attack effectively drove the cooling capacity outside the normal bounds.

S3: Reduced cooling for emergency outage: 90 minutes after the attack was triggered, the normal command values (i.e., "CHW.R.P1-P4.SPD") were overwritten with the maliciously crafted values, which leads to a loss of flow in the chilled water loop. As a result, the cooling system encountered a loss of cooling capacity such that it dropped below the lower limit 95 minutes after the attack was triggered. The cooling capacity was restored back to normal after 240 minutes (Figure 9c).

As demonstrated in our simulated environment, the attack strategies derived by the smart malware can effectively introduce an anomaly into the cooling system. Though the impact of the anomaly on the CI cannot be shown, we have successfully demonstrated that the attack can emulate incidents that led to SWOs in the past.

7 RELATED WORK

Energy-related attack models. In [15], the authors present an attack on the power system of a data center, which aggressively discharges a backup battery of a server rack and triggers a switch of power source to the discharged battery. The switch of the power source to the empty battery results in loss of service. Similar studies on different attacks were presented in [27] and [7]. Also, a number of previous efforts studied the impact of cooling failures on the availability of data centers. In [21], the author considered cooling systems as a vulnerable component of data centers and studied the impact of cooling losses due to power failure. Zhou et al., in [29], present a control logic that optimizes the operation of computer room air conditioning units. Papers [8, 9] investigate vulnerabilities introduced by aggressive heat policies in data centers and introduce a set of thermal attacks, and the authors of [2] demonstrate that DDoS attacks, by generating massive heat, can permanently damage server or switch hardware.

CPS Security. Multiple kinds of malware has been reported that take advantage of vulnerabilities in power systems [13, 14], such that the malware introduces a breakdown of critical components in the system or obfuscates the power grid, causing a blackout. The authors of [10] generalizes Stuxnet with a physical model embedded to derive the optimal attack vector that maximizes damage to the current target. Also, a number of studies have conducted research on the security of networked control systems [4, 24, 25, 28]. In [1], Alemzadeh et al. present malware that targets a teleoperated surgical robot, where the malware identifies packets that contain data for a certain state and injects faults that can cause catastrophic consequences. Furthermore, their attack model learns the optimal attack time that can maximize the impact of an attack. In [16], the authors present a method to identify network packets that update the control of a nuclear power plant's logic, and to overwrite the logic with failure-causing logic. In [18], an IDS that utilizes fused information from multiple sensors to detect energy theft is discussed. In terms of protection, Lin [17] deployed an open-source intrusion detection system over SCADA systems that expands the coverage of the IDS to control system-specific protocols. We expect such an approach to be effective against our attack model; different scripts can be added to detect policy violations and anomalies.

8 CONCLUSIONS

In this paper, we demonstrate the feasibility of security attacks that masquerade as failures and are delivered via self-learning malware that monitors the target system and launches the attack by injecting a strategic failure at a time and system location chosen to have a maximal impact. The target system we focus on is a CPS that manages and controls the cooling system for a large computing enterprise (i.e., the Blue Waters supercomputer). We address a range of issues, including the construction of the self-learning malware and the launch of a failure injection attack. Provided with real CPS data, our malware inferred 3 unique attack strategies which disrupt the cooling system while the attack masquerades as an accidental CPS failure. We validated the attacks in a data-driven CPS simulator developed as part of this study.

A TABLES

The tables can be found on the next page.

B ALGORITHM

Algorithm 1 Pseudocode of the malicious wrapper around the original controller logic

```
1: procedure DetermineOptimalStrategy
2:     if (α × par 1 + β×par 2 < thr 2)
         &(Timer == 0) then
3:         mode = econ
4: else
5:         mode = campus
6: op_strateϑy = strateϑy[mode]
7: procedure Original Controller Code
8: output _1 = orϑ_f unct 1(parX1, ..., parZ1)
9: ...
10:    output_N = orϑ_f unctN(parXN , ..., parZN )
11: procedure Output Overwrite
12:   if Timer < len(op_strateϑy[output _1]) then
13:       output _1 = op_strateϑy[output _1][Timer]
14:       ...
15:       output_N = op_strateϑy[output _N][Timer]
16:       Timer + = 1
17:   else
18:       Timer = 0
```

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos. 18-16673 and 15-45069. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Homa Alemzadeh, Daniel Chen, Xiao Li, Thenkurussi Kesavadas, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2016. Targeted Attacks on Teleoperated Surgical Robots: Dynamic Model-based Detection and Mitigation. In *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 395–406.

[2] Zahid Anwar and Asad Waqar Malik. 2014. Can a DDoS Attack Meltdown My Data Center? A Simulation Study and Defense Strategies. *IEEE Communications Letters* 18, 7 (July 2014), 1175–1178.

[3] Phuong Cao, Eric Badger, Zbigniew Kalbarczyk, Ravishankar Iyer, and Adam Slagell. 2015. Preemptive Intrusion Detection: Theoretical Framework and Real-world Measurements. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*. 1–12.

[4] Alvaro A Cárdenas, Saurabh Amin, and Shankar Sastry. 2008. Research Challenges for the Security of Control Systems. In *3rd USENIX Workshop on Hot Topics in Security*.

[5] Keywhan Chung, Valerio Formicola, Zbigniew T. Kalbarczyk, Ravishankar K. Iyer, Alexander Withers, and Adam J. Slagell. 2016. Attacking Supercomputers Through Targeted Alteration of Environmental Control: A Data Driven Case Study. In *Proceedings of the IEEE Conference on Communications and Network Security*. 406–410.

[6] Catello Di Martino, Zbigniew Kalbarczyk, Ravishankar K. Iyer, Fabio Baccanico, Joseph Fullop, and William Kramer. 2014. Lessons Learned from the Analysis of System Failures at Petascale: The Case of Blue Waters. In *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 610–621.

[7] Xing Gao, Dachuan Liu, Daiping Liu, and Haining Wang. 2016. On Energy Security of Smartphones. In *Proceedings of the ACM Conference on Data and Application Security and Privacy*. 148–150.

[8] Xing Gao, Zhang Xu, Haining Wang, Li Li, and Xiaorui Wang. 2017. Why Some Like It Hot Too: Thermal Attack on Data Centers. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems*. 23–24.

[9] Xing Gao, Zhang Xu, Haining Wang, Li Li, and Xiaorui Wang. 2018. Reduced Cooling Redundancy: A New Security Vulnerability in a Hot Data Center. In *Proceedings of the Network and Distributed System Security Symposium*.

[10] Luis Garcia, Ferdinand Brasser, Mehmet Hazar Cintuglu, Ahmad-Reza Sadeghi, Osama A. Mohammed, and Saman A. Zonouz. 2017. Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit. In *Proceedings of the Annual Network and Distributed System Security Symposium*.

[11] John A. Hartigan and Manchek A. Wong. 1979. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28, 1 (1979), 100–108.

[12] Wade W. Huebsch, Bruce R. Munson, Theodore H. Okiishi, and Donald F. Young. 2009. *Fundamentals of Fluid Mechanics*. John Wiley & Sons.

[13] Ralph Langner. 2011. Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Security & Privacy* 9, 3 (2011), 49–51.

[14] Robert M. Lee, Michael J. Assante, and Tim Conway. 2016. *Analysis of the Cyber Attack on the Ukrainian Power Grid*. Technical Report. Electricity Information Sharing and Analysis Center (E-ISAC).

[15] Chao Li, Zhenhua Wang, Xiaofeng Hou, Haopeng Chen, Xiaoyao Liang, and Minyi Guo. 2016. Power Attack Defense: Securing Battery-Backed Data Centers. In *Proceedings of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture*. 493–505.

[16] Bernard Lim, Daniel Chen, Yongkyu An, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2017. Attack Induced Common-Mode Failures on PLC-Based Safety System in a Nuclear Power Plant: Practical Experience Report. In *Proceedings of the IEEE Pacific Rim International Symposium on Dependable Computing*. 205–210.

[17] Hui Lin, Adam Slagell, Catello Di Martino, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. 2013. Adapting Bro into SCADA: Building a Specification-based Intrusion Detection System for the DNP3 Protocol. In *Proceedings of the Annual Cyber Security and Information Intelligence Research Workshop*. 5:1–5:4.

[18] S. McLaughlin, B. Holbert, A. Fawaz, R. Berthier, and S. Zonouz. 2013. A Multi-Sensor Energy Theft Detection Framework for Advanced Metering Infrastructures. *IEEE Journal on Selected Areas in Communications* 31, 7 (July 2013), 1319–1330.

[19] The Bro Project. 2012. The Bro Network Security Monitor. (2012). http://www.bro.org/

[20] Aashish Sharma, Zbigniew Kalbarczyk, James Barlow, and Ravishankar Iyer. 2011. Analysis of Security Data from a Large Computing Organization. In *Proceedings of the 41st Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 506–517.

[21] Shawn Shields. 2009. *Dynamic Thermal Response of the Data Center to Cooling Loss During Facility Power Failure*. Ph.D. Dissertation. Georgia Institute of Technology.

[22] Siemens. 2008. *SIMATIC S7 - OPENMODBUS / TCP Communication via the Integrated PN Interface of the PLC*. Siemens.

[23] Siemens Building Technologies. 2000. *APOGEE Powers Process Control Language (PPCL) User's Manual*. Siemens.

[24] André Teixeira, Daniel Pérez, Henrik Sandberg, and Karl Henrik Johansson. 2012. Attack Models and Scenarios for Networked Control Systems. In *Proceedings of the 1st International Conference on High Confidence Networked Systems (HiCoNS '12)*. 55–64.

[25] André Teixeira, Henrik Sandberg, and Karl H Johansson. 2010. Networked Control Systems under Cyber Attacks with Applications to Power Networks. In *Proceedings of the American Control Conference (ACC)*. IEEE, 3690–3696.

[26] U.S. Senate, Committee on Commerce, Science and Transportaion. 2014. *A 'Kill Chain' Analysis of the 2013 Target Data Breach*. Technical Report.

[27] Zhenyu Wu, Mengjun Xie, and Haining Wang. 2011. Energy Attack on Server Systems. In *Proceedings of the 5th USENIX Workshop on Offensive Technologies*.

[28] Heng Zhang, Peng Cheng, Ling Shi, and Jiming Chen. 2016. Optimal DoS attack scheduling in wireless networked control system. *IEEE Transactions on Control Systems Technology* 24, 3 (2016), 843–852.

[29] Rongliang Zhou, Zhikui Wang, Cullen E. Bash, Tahir Cade, and Alan McReynolds. 2012. Failure Resistant Data Center Cooling Control through Model-based Thermal Zone Mapping. In *Proceedings of the ASME Summer Heat Transfer Conference*. 751–757.

Table 1: Critical Parameters for the Attack Strategies

Mode of Operation	Parameter	Description
Campus	CHW.L.HI.CV1A	How far the control valve 1A is opened (%)
	CAMP.CHWR.TEMP	Temperature of chilled water returning to the campus CHW loop
	CAMP.CHWS.TEMP	Temperature of chilled water supplied from the campus
	CAMP.L.CHWS.TEMP	Temperature of chilled water supplied from the campus
	CHW.FLOW	Chilled water's volumetric flow
	CAMP.CHW.FLOW	Chilled water's volumetric flow from campus
Economic	CHW.L.HI.CV1A	How far the control valve 1A is opened (%)
	CHW.L.HI.CV1B.CHWS.TEMP1	Temperature of chilled water supplied to the server room
	CHW.L.HI.CV1B.FLOW1	Volumetric flow of chilled water through control valve 1B
	CHW.R.P1.SPD	Pump 1 speed (%)
	CHW.R.P2.SPD	Pump 2 speed (%)
	CHW.R.P3.SPD	Pump 3 speed (%)
	CHW.R.P4.SPD	Pump 4 speed (%)

Table 2: Attack strategies inferred by the fully targeted attack (smart malware)

Timestamp	Abnormal Parameter	Recorded Incident	Impact on CI
2016-10-25 09:25	control valve 1B flow	reduced cooling capacity for emergency outage (Figure 5c)	scheduled outage
2016-11-07 10:40	chw return temperature	(change in mode of operation)	
2016-12-05 08:10	chw return temperature	(reduction of heat absorbed from CI due to scheduled maintenance)	
2016-12-26 00:45	chw return temperature	(change in mode of operation)	
2016-12-26 07:35	chw return temperature	(change in mode of operation)	
2017-02-11 19:55	chw supply temperature	supply water anomaly due to power interrupt (Figure 5a)	SWO for 5 hrs
2017-03-28 09:15	pump 1 speed	chilled water loop maintenance (Figure 5b)	scheduled outage

Table 3: Summary of APIs provided in the simulation framework

Layer	API	Description
Cyber	`socket_init()/close()`	Initializes/closes the network sockets used for communication
	`ctrl_recvData()`	Controller receives packets from the network that contain commands (from the server) and data from different controllers
	`ctrl_sendData()`	Controller sends the data to be sent over the control network
	`server_recvData()`	Server receives data from the client and acknowledges receipt to the sender
	`server_sendData()`	Server prompts for operator command (e.g., update PLC), and sends the command to the controller over the network
Programmable Logic	`reportData()`	Controller reports the data to predefined destinations
	`recvPacket()`	Receives packets from the server or other controllers
	`controller_init()/close()`	Initializes the controller and its network socket and closes the socket
	`ctrl_logic()`	The control logic (for each controller) implemented in the Python language
	`PID()`	Function for PID logic (or the *LOOP* instruction in Siemens PPCL [23])
	`table()`	Function for the *TABLE* instruction in Siemens PPCL [23])
Physical	`HeatExchanger()`	Models the temperature during heat exchange
	`tJunction_distribution()`	Models the flow and pressure for a T-junction with one input and two outlets
	`tJunction_merge()`	Models the flow and pressure for a T-junction with two inputs and one outlet
	`controlValve()`	Models the flow by the control valve that takes commands from a controller
	`Pump()`	Models the flow and pressure by the pump operation that takes commands from the controller