SAND2021-15349C

SCAPHY: Detecting Modern ICS Attacks by Correlating Behaviors in SCADA and PHYsical

***Abstract*—Modern Industrial Control Systems (ICS) attacks evade existing tools because they use knowledge of ICS elements and victim's environment to blend their activities with benign Supervisory Control and Data Acquisition (SCADA) operation, causing physical world damages. We present SCAPHY to detect ICS attacks in SCADA hosts by leveraging the unique *execution phases* of SCADA to identify the limited set of legitimate behaviors to control the physical world, which differentiates from attacker's activities. For example, it is typical for SCADA to setup new ICS device objects during *initialization*, but anomalous during *process-control*. To extract unique behaviors of each SCADA execution phase, SCAPHY first leverages ICS open platform communications (OPC) conventions to generate a novel physical process dependency and impact graph (PDIG) to identify disruptive physical states. SCAPHY then uses PDIG to inform a *physical process-aware* dynamic analysis, whereby code paths of *process-control* operations are induced to execute API call behaviors unique to legitimate *process-control* phase. Through this, SCAPHY selectively monitors attacker's physical world-targeted activities that violates legitimate *process-control* behaviors. We evaluated SCAPHY at a U.S. national lab state-of-the-art ICS environment. We operated 24 diverse ICS scenarios, across 5 ICS industries, and diverse attacks on realistic ICS settings including an adapted Texas Pan Handle power grid. We detected 95% of all attacks, including ICS malware. We analyze SCAPHY's resiliency to futuristic ICS attacks where attacker knows our approach.**

I. INTRODUCTION

Unlike Information Technology (IT) cyber attacks, Industrial Control System (ICS) attacks cause physical world damage to life-dependent industrial processes such as electricity and water supply [1, 2]. ICS physical processes are controlled by Supervisory Control and Data Acquisition (SCADA) hosts, which run special software and hardware to control the physical world [3–6]. Modern ICS attacks [1, 2, 7] are launched from SCADA, where attackers utilize legitimate ICS resources to blend their malicious activities with benign SCADA operations, causing *targeted* process disruptions. Unfortunately, ICS attacks continue to rise, with many attacks in 2021 alone having real economic and safety impacts [1, 8].

To detect ICS attacks, statistical analysis of ICS traffic [5, 9–19] are effective against *noisy* and abnormal traffic such as denial of service and illegal protocol fields. However, modern ICS attacks evade them by not only using legitimate protocols, but knowledge of ICS parameters to cause specific (not noisy) process disruptions [2–4]. In addition, physical models monitor sensor data to know when *observed* physical states deviate from *expected* [20–23]. These techniques use historical sensor data to fit a *linear* model. However, in real-world ICS settings, such models may not be available or easily derived [23, 24]. Further, physical models trigger false alarms when plugged into production due to noise and configuration changes, such that benign physical states are outside the model [4, 21, 23]. In general, existing ICS tools are evaded by modern ICS attacks and prone to false alarms due to analyzing traffic or sensor data in isolation, and therefore they cannot tie their analysis to the attack-execution context in SCADA. Detecting ICS attack in SCADA hosts is hard because attackers use the same behaviors (or API calls) as benign SCADA programs. For example, the Industroyer malware, which shutdown Ukraine power grid [2, 3], performed malicious actions that are part of normal SCADA activity such as creating new ICS device handles. Hence, existing host agents that monitors *non-SCADA* or unknown APIs will not detect it. Similarly, Florida water poisoning attack [1] used benign Human Machine Interface (HMI) to dump toxic chemicals into the city's water supply.

We found that while these attack behaviors are normal SCADA activities, they are suspicious when performed at certain *execution phases* in SCADA lifecycle. Instead of treating SCADA as one monolithic execution, we consider its behaviors in unique execution phases, which are *initialization* and *process-control*. We observe that for Industroyer attack to work, the attacker had to execute API calls that are *atypical* of legitimate process-control, but needed to launch the attack. For example, to hijack SCADA ICS device handles, Industroyer executed many Registry Setup APIs, which are typical of *initialization* but performed during process-control phase. To attack physical processes, attackers must *inject* atypical executions or circumvent normal process-control behavior. Hence if we can identify the limited set of legitimate process-control behaviors, we can selectively monitor and detect attacker's physical world-targeted activities that violate them.

Further, because SCADA responds to physical process changes, we can induce SCADA to execute process-control behaviors by stimulating the change. However, this requires a physical model of ICS processes and their possible states. Interestingly, we found that we can leverage widely deployed open platform communications (OPC) conventions to identify process states via connected ICS elements. We can then *toggle* each element state to induce SCADA to exhibit its process-control behaviors, enabling us to identify them. Further, we can derive the *impact* of these state changes to identify *disruptive* process states. This can allow us to detect disruptive physical world effects caused by (state-changing) control signals.

We present SCAPHY, a new hybrid technique to detect ICS attacks by correlating SCADA behaviors with physical world effects to identify the limited set of API calls to legitimately control the physical world, which differentiates from attacker's activities. SCAPHY leverages the distinct execution phases of SCADA to identify behaviors unique to each process-control phase, which attacker must *inject* into or *circumvent* to attack the physical world. To identify the *channels* through which

SCADA accesses the physical, we introduce a new reference model, *SCADA Software Stack* (S^3), to characterize the internal software and hardware layers of SCADA activities. SCAPHY uses S3 layers to selectively monitor steps along attacker's activities towards attacking the physical. Through S^3, SCAPHY can detect attacks that *circumvent* proper S^3 layers (e.g., SCADA rootkits) but sends disruptive signals to the physical.

SCAPHY uses a physical model to identify disruptive control signals sent to the physical world. SCAPHY generates this model by leveraging OPC conventions (*tags* and *alarms*) to extract and map ICS elements to their processes based on a novel process dependency and impact graph (PDIG) model. PDIG enables SCAPHY to assign each element state an *impact coefficient* ($I_C(s)$) based on how they impact (decrease or increase) process outcomes. Further, SCAPHY leverages PDIG to help induce and extract legitimate process-control behaviors. To do this, SCAPHY performs a *physical process-aware* dynamic analysis, whereby a SCADA engine [25] is induced to execute process-control code paths by iteratively switching ICS element states connected to the process. During this, SCAPHY records executed API calls to establish a set of PHYSical world Impact Call Specialization (PHYSICS) constraints to identify *legitimate* process-control behavior.

SCAPHY detects modern ICS attacks that are missed by existing tools. By correlating activities in SCADA and physical, SCAPHY provides better alerts for ICS operators and enables them to make threat remediations at both SCADA and physical plant. Via PHYSICS constraints, SCAPHY limits the operations an attacker can execute to disrupt a physical process and can detect attacker's *injected* executions in each process-control phase. SCAPHY's physical model detects when control signals cause a physical process to have *inconsistent state* or driven outside its *set point* ranges. SCAPHY is device agnostic because it uses generic OPC tags not based on any device or controller, making it work for any OPC-based ICS deployment. We evaluated SCAPHY at a U.S. national lab state-of-the-art ICS testbed. We operated 24 diverse ICS scenarios, spanning 5 industries, and launched 40 attacks on realistic ICS settings including an open-source Texas Pan Handle power grid [26]. SCAPHY detected 95% of all attacks, including ICS malware attacks. We make the following contributions:

1) We propose a hybrid technique to detect ICS attacks by correlating SCADA activities with physical world effects.
2) We present an ICS physical model via OPC conventions to both identify disruptive control signals and extract legitimate process-control phase behaviors in SCADA.
3) We introduce a new reference model, *SCADA Software Stack* S^3, to characterize internal software and hardware layers of SCADA operation. Through S^3, host agents can monitor specific S^3 layers to detect SCADA host attacks.
4) Using diverse ICS scenarios/attacks, SCAPHY achieved 95% accuracy and 3.5% false positives (FP), compared to 47.5% accuracy and 25% FP of existing work [5, 20].
5) Due to lack of research resources to support diverse ICS security research [27–30], we make available over 200GB of new ICS experiment scenarios and their corresponding attacks in both SCADA and physical aspects, and diverse physical processes developed for the FactoryIO ICS Engine [25] and hardware-in-the-loop device support.

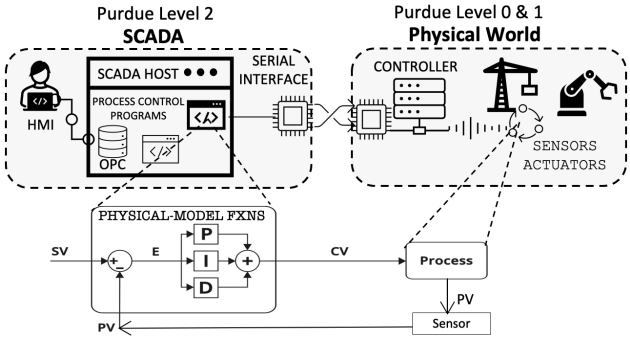


Fig. 1: Showing SCADA Process-Control Operation: A physical-model function compute a control variable *CV* to effect change on the physical world

II. Background and Motivation

We present a background on ICS and use recent attacks to motivate our problem; 2021 Florida water poisoning attack [1], and 2016 Industroyer power grid attack [3]. In Section IV, we detail SCAPHY's approach using real world ICS settings.

A. Real-World Motivating Examples

Florida Water Poisoning Attack. An attacker raised *dosing rate* of Sodium Hydroxide (NaOH) in FL water treatment plant to toxic levels, endangering citizens. NaOH is used to balance water PH but is toxic in high amounts. After gaining access to SCADA, the attacker started an HMI program to issue attack signals to disrupt the *level control* and *dosing* processes, increasing NaOH from normal 100 ppm to 11,100ppm [1, 32].

Industroyer Power Grid Attack. Industroyer shutdown a Ukranian power station by sending malicious signals from a SCADA host to a Siemens SPIROTEC device that runs circuit breaker Remote Terminal Units (RTUs). To perform the attack, Industroyer hijacked host serial COM ports, stole the breaker's *tag* from OPC to *address* its payload, and opened the breakers. This caused power imbalance that shutdown the station [2, 3].

B. ICS/SCADA Operations

Figure 1 describes ICS operation based on Purdue model [6, 33]. SCADA host run at Level 2 to control physical processes which runs at Level 0-1 via actuators, sensors, and controllers. SCADA programs constantly monitor running processes and when change is needed, they execute physical-model functions such as *proportional integral derivative (PID)* to compute a control variable (*CV*), which is sent to actuators to effect the change. For example, the *level control* process in the FL water plant controls chemical level in a tank via a *PID* function that control how much fluid enters and leaves the tank [30]. This is known as *process-control* and is the main function of SCADA.

Unfortunately, ICS attacks are launched from SCADA due to direct access to the physical world as in Industroyer and FL attacks. An attacker can gain access to SCADA hosts using IT means such as phishing. Then he can leverage host ICS resources such as OPC and HMIs to send malicious signals to actuators. OPC services are used in ICS for interoperability by serving data about ICS elements in the plant.

Existing work to detect ICS attacks focus mainly on analyzing ICS traffic such as control signals and sensor data.

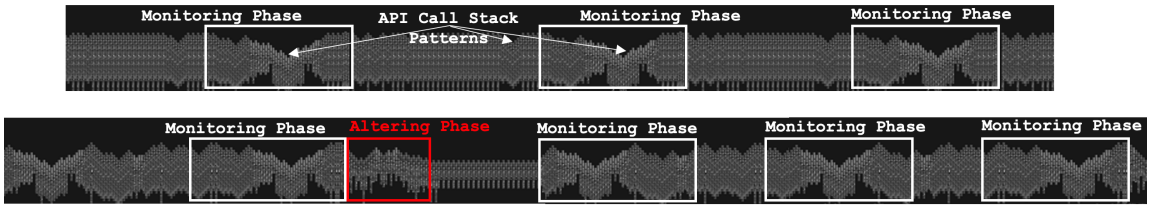


Fig. 2: Cycles of Process-Monitoring and Process-Altering phases based on Siemens S7 WinSPS [31] control of Water Treatment Plant operation

However, they are evaded by modern attacks that use legitimate ICS resources in SCADA hosts to produce signals that blend with benign traffic. Due to physical world constraints in analyzing SCADA operations, existing work do not significantly consider SCADA host executions, which can identify attacker's activities that originates the malicious signals. PLC-based techniques [34, 35] tries to prevent malicious ladder logic code from loading or executing on the PLC. However, SCADA control signals (if it passes via a PLC) are mostly forwarded to actuators, and not analyzed as in PLC ladder logic. Table I is a taxonomy of recent works in ICS attack detection, categorized by their detection data points. The Evaluation row shows their limitations in detecting modern ICS attack in realistic settings, such as only using one ICS scene to evaluate their work (water plants or power grid). We found that this is due to limited research resources to support diverse ICS security research [27–30]. In general, existing techniques are evaded by modern ICS attacks and prone to false alarms due to analyzing traffic or sensor data in isolation, and therefore, cannot tie their analysis to attack-execution context in SCADA.

Traffic Analysis. Network flow-based approaches [9, 11, 12, 36] analyze abnormal ICS channels and function codes. However, they are evaded by modern attacks such as Industroyer and FL water plant attacks, which uses legitimate SCADA channels and function codes. Traffic timing analysis [5, 10, 37] are effective for catching anomalous round trip time delays and inter-arrival times. However, they are only effective against attack behaviors that are *chatty* [10, 19] such as attacker scans, but not modern attacks which are targeted.

Physical Models. use sensor data to know when observed behavior deviates from expected [21, 23]. Expected behavior can be learned by fitting historical sensor data into a linear dynamic state space or auto regressive model [20–22]. However, in real-world ICS settings, such models may not be available or easily derived [23, 24]. Further, physical models trigger false alarms when in production due to noise and config changes, such that benign states are outside the model [21, 23]. We found that existing physical models cannot reason across multiple processes, which leads to false alarms when a high sensor deviation is benign in one process but anomalous in another.

C. SCADA Host Attacks and Security Challenges

Due to physical ramifications of SCADA executions, existing host analysis techniques cannot be directly applied to secure SCADA hosts [2, 42]. SCADA platforms contain many proprietary and domain-specific programs such as ICS drivers and physical-model libraries, due to complex safety constraints in physical tasks. Due to legacy and third-party components, code-signing cannot be strictly enforced to whitelist benign programs, enabling attackers to modify benign programs. For

Techniques	Yang [38]	Ihab [39]	Ponoma. [5]	Ghaeini [20]	Dina [22]	Aoudi [23]	Niang [40]	Mulder [34]	Formby [35]	Lee [41]	SCAPHY
ICS Traffic Analysis											
protocol fields/func code	•		•							•	•
time analysis	•	•	•								
Physical Behavior											
deviation (LDS/AR)				•	•						
deviation (SSA)						•					
disruptive impact											•
PLC Control Logic											
logic execution								•	•		
logic verification							•				
SCADA Host Behavior											
detects DLL inject										•	•
PHYSICS constraints											•
Evaluation											
tested in-the-wild attacks							•				•
tested diverse ICS apps											•

TABLE I: Taxonomy of Recent Related works in ICS Attack detection

example, Industroyer executed custom DLLs, and Stuxnet [7] injected into Siemens programs. In contrast to IT apps, SCADA programs are hard to analyze due to physical domain requirements. As such, analysis using state-of-the-art concolic tools [43–45] will be intractable due to hardware-constrained code paths and dynamic environment need [43, 45].

Our Insight. We found that the nature of physical world tasks, which happen in sequence of repeated steps, requires SCADA to exhibit two distinct execution phases: *initialization* and *process-control*. SCADA uses process-control to make changes in the physical via *process-monitoring* and *process-altering* sub-phases. We observe that for SCADA host attacks to work, the attacker has to execute operations that are atypical of legitimate process-control behavior but needed to launch the attack. For example, Industroyer made OPC calls in process-monitoring phase to access its target's device tags. However, OPC calls are typical for *initialization*. In addition, Industroyer created new ICS service handles while already in the process-altering phase (to hijack access to the physical world), which is a process-monitoring behavior, and suspicious as an attack.

SCADA Execution Phases. SCADA starts with *initialization* which is performed once and involves setting up the environment such as reading config files and loading ICS drivers and services. After *initialization*, process-monitoring starts. It involves updating physical process states in memory and checking for events. Data from here is presented to operator HMIs and recorded in Historians. When process change is needed, process-monitoring transitions into process-altering to perform the change, and then returns until change is required again. Process-altering involves invoking physical-model functions that takes a *setpoint* variable *SV* and sends a domain-specific *CV* to the process. Figure 2 shows the unique pattern

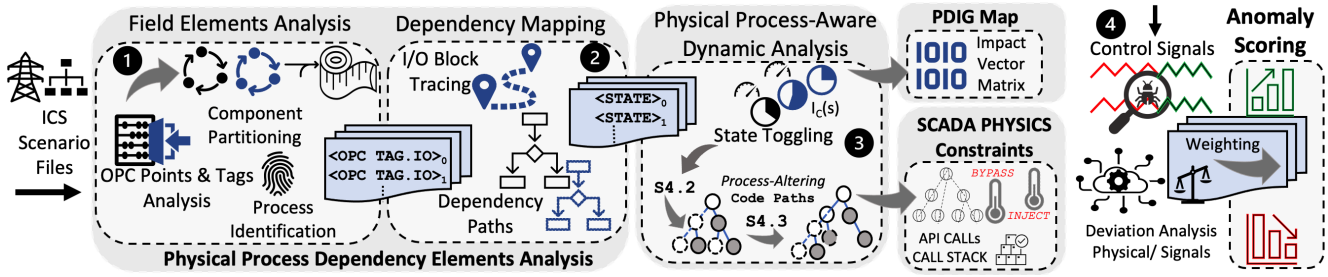


Fig. 3: SCAPHY Architecture: ICS scenario file is parsed. Extracted ICS elements are analyzed to identify processes and dependency elements. Each dependency element state is toggled to measure impact on process output, during which SCADA process-control PHYSICS constraints are identified for each process.

of API call stack behavior of process monitoring and process-altering phases of SCADA process-control of a water treatment plant based on Siemens S7 WinSPS SCADA platform [31].

To identify behaviors unique to each process-control phase, SCAPHY first leverages OPC conventions to build a physical model of ICS processes. It then uses the model to inform a *physical process-aware* dynamic analysis to induce and extract the limited set of legitimate API calls unique to each process-control phase behavior, which differentiates from attacker's activities. Through the API calls, SCAPHY establishes a PHYSical world Impact Call Specialization (PHYSICS) constraints, which attackers must *circumvent* or *inject* (*atypical* APIs) to attack the physical world. In addition, SCAPHY develops a new reference model, *SCADA Software Stack* (S^3) to characterizes the internal software and hardware *layers* to access the physical. For example, calling *ReadFile* indicates *process-monitoring* and *WriteFile* indicates *process-altering*, both of which accesses an ICS device object handle (e.g, COM Ports), the 3rd layer in S^3, to access physical devices. Through S^3, SCAPHY can detect attacks that *circumvent* legitimate S^3 layers (e.g., SCADA rootkits) but sends disruptive signals to the physical. Further, SCAPHY's physical model detects disruptive effects of control signals such as when a physical process has an *inconsistent state* or in *outside setpoint* ranges.

III. Threat Model and Assumptions

We assume a threat model similar to ICS detection approaches for SCADA-originated attacks [3–5, 19, 42, 46]. We assume the Purdue model [6], whereby SCADA hosts send control signals to actuators. We assume the attacker has compromised the SCADA host with root privileges, can utilize legitimate ICS protocols, or install malware to attack the physical world. We make the following practical assumption about our system: The attacker cannot tamper the network interface where SCAPHY listens for ICS signals. We note that ICS control signals are typically due to real-time safety needs and processing power. We do not consider indirect attacks (such as side channels) that do not pass through the interface that SCAPHY monitors, such as attacks not originating from SCADA. We note that over 90% of *in-the-wild* ICS attacks are SCADA-originated [3, 4, 7, 46]. Malware that originates and runs *entirely* on PLC such as in [47] are rarely seen in the wild, and are mostly academic or notional. This is due to attacker's practical cost of developing reusable malware code for embedded architectures which device platforms are mostly based on [4, 48]. We note that preventing Man-in-the-middle PLC sensor tampering has been explored and prevented by existing work [49–53] and in secure deployments by channelling sensor data via data diode gateways [54], and hence is outside the scope of this work.

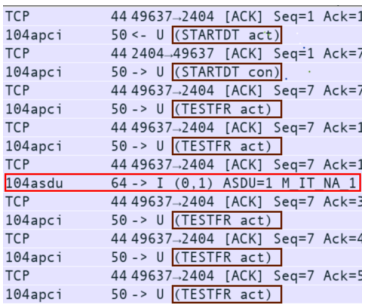
IV. SCAPHY Approach

Input and Output. SCAPHY's takes as input, an ICS scenario (OPC data & function block logic) and outputs (i) a physical model and (ii) PHYSICS constraints. The physical model maps each ICS element to a process and assigns each element state, an impact vector based on how the state impacts process outcomes. SCAPHY uses this physical model to detect signals that are disruptive to process outcomes. PHYSICS constraints are a set of API calls of legitimate *process-control* behaviors, which differentiates from attacker's activities, such that he must *inject into* or *circumvent* them to attack processes, allowing SCAPHY to detect it as a violation of PHYSICS constraints.

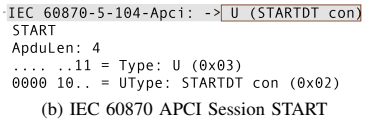
Deployment. SCAPHY can be deployed as a kernel driver or in a hypervisor hosting SCADA virtual machines (VM). For our work, we deployed SCAPHY in Dom 0 of XEN hypervisor. We chose this approach because many ICS settings use virtual solutions to provide compute redundancy in SCADA [55–58]. In the hypervisor, SCAPHY analyzes only *state-changing* signals leaving the physical interface. These signals perform *WRITE* operation on a device based on the protocol *function codes*. ICS protocols such as Modbus, IEC, and DNP3 have function code that specifies *WRITES* and *READS* operations. For example, DNP3 $0x02$ and Modbus $0x05$ denotes *WRITE* signal [10, 59]. Although many traffic analyzers exist to identify WRITE function codes in ICS protocols [11, 12, 60], we prototyped SCAPHY for Modbus and IEC traffic. Further, we leverage LibVMI to monitor API calls in SCADA VMs [61, 62].

A. End-to-End Operation

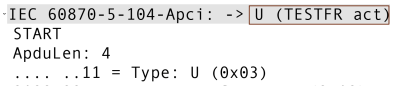
SCAPHY works in four phases as shown in Figure 3. ❶ SCAPHY parses ICS scenario files to extract, analyze, and partition ICS elements into terminal and non-terminal sets based on OPC tags. Terminal elements identifies processes. ❷ SCAPHY then traces each element's connections to map dependent element sequences to their process. SCAPHY then loads the scenario in a SCADA engine [25, 31] and performs a *physical process-aware* dynamic analysis ❸, whereby the engine is induced to execute code paths of *process-control* operations by iteratively switching each element state. During this, SCAPHY records the API calls executed during process-altering and process-monitoring phases separately, to establish a PHYSical world Impact Call Specialization (PHYSICS) constraints, which identifies the limited set of API calls of



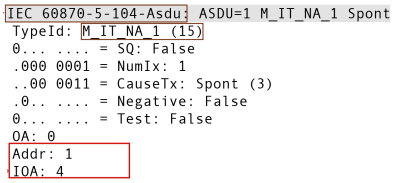
(a) Industroyer's IEC 60870 Attack Traffic Signals



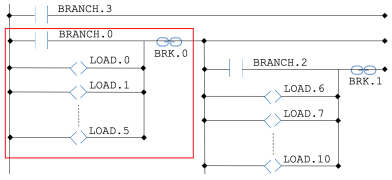
(b) IEC 60870 APCI Session START



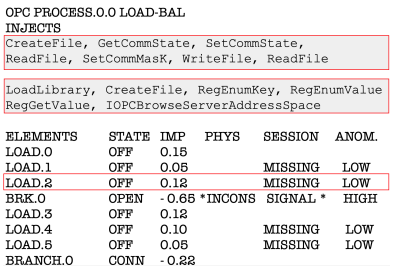
(c) IEC 60870 APCI TEST



(d) Industroyer Payload Addressed to Circuit Breaker IOA 4 OPC tag



(e) Power Load Balancing Part Ladder Logic



(f) SCAPHY Output on detecting Industroyer attack

Fig. 4: Delienation of Industroyer's power grid attack signals based on the IEC 60870-5-104 Communication Protocol

legitimate *process-control* behaviors. Further, the change in process output caused by each state switching is averaged over several scan cycles to derive an impact vector for each state relative to others. States without significant impact to the process are pruned. For states with an oscillating impact (i.e., process output may increase or decrease), SCAPHY derives a *set point range*, which defines a minimum and maximum impact vector for the process. SCAPHY raises an alert when executed APIs in *process-control* phase deviates from the PHYSICS constraints. SCAPHY ❹ analyzes control signal to detect process *inconsistent state* and *outside setpoint* outcomes, as well as *missing*, *extraneous*, and *out-of-order* signals. If any of these is detected, SCAPHY computes an anomaly score.

B. Detecting Industroyer Attack Behavior with SCAPHY

Industroyer shutdown Ukranian power station by maliciously opening circuit breakers connected to load lines [3]. This attack disrupted the load balancing of power demand and supply, a known weakness in power systems, leading to outtages [63]. To replicate the attack, we adapted an open-source Texas Pan Handle power grid setup in a U.S. National Lab. Our lab setup is detailed in Section VI. We executed Industroyer in its "intended" environment; a SCADA host with COM ports and OPC, connected to an IEC 608070 device with simulated circuit breaker RTUs. SCAPHY raised detection alarm in under 9 seconds for a PHYSICS *injection* violation and a process *inconsistent state* anomaly.

Industroyer PHYSICS Violation. Industroyer executed several *LoadLibrary* calls, although a normal SCADA API, was performed after Industroyer entered the process-altering phase indicated by a prior *CreateFile(WRITE)* call. We found that *LoadLibray* is normal for process-monitoring and *initialization* but not process-altering, per the PHYSICS constraints established from the power scenario. We found that Industoryer used *LoadLibrary* to load *OPCClientDemo.dll*, to gain OPC capability. Thereafter, Industroyer transitioned to process-monitoring indicated by a *ReadFile* call. It then invoked *IOPCBrowseServerAddressSpace* OPC call to extract circuit breaker Information Object Addresses (IOAs) to send its payload as shown in Figure 4d. OPC calls are typical of *initialization*, but not process-monitoring. In addition, Industroyer created new ICS device handles while already in the process-altering phase (to highjack COM Ports) which is malicious in process-altering.

Industroyer's Physical Anomalies. The Industroyer attack generated eight IEC 608070 signals as shown in Figure 4, comprising of two IEC 60870 Application Protocol Control Information (ACPI) *START* frames to begin its communication, two ACPI *TEST* frames to check controller status, and one Application Service Data Unit (ASDU) payload sent to the circuit breaker IOA. Industroyer also issued three last *TEST* signals to verify controller's post attack status as shown in Figure 4c. Based on the physical model mapping of the element IOA in the payload, SCAPHY identified the target process as *load balancing* (LB). LB's dependency elements are load lines *LOADS.0-5*, breaker *BRK.0*, and *Branch.0* as shown in Figure 4e. SCAPHY output (Figure 4f) show that Industroyer issued a *WRITE* signal to *BRK.0* (i.e., indicated by SIGNAL*), with no signals for the other elements (missing) in LB's control session. However, *BRK.0*'s new open state has an opposing impact vector to load lines's OFF state per SCAPHY physical model, allowing SCAPHY to detect an *inconsistent state* anomaly (detailed in Subsection V-C). Using *BRK.0*'s impact vector of 65%, SCAPHY derived a high anomaly score.

V. SYSTEM DESIGN

The goal of SCAPHY is to (*I*) identify the limited set of legitimate API calls of SCADA process-control execution phases that differentiates from attacker activities, and (*II*) build a physical model of ICS processes to identity disruptive physical effects and control signals. To achieve (*I*), we leverage the physical model in (*II*) to inform a *physical process-aware* dynamic analysis. To achieve (*II*), we leverage OPC conventions to generate a process dependency and impact graph (PDIG).

A. Automated Physical Process Comprehension

OPC Points and Tags Analysis. SCAPHY parses OPC *points and tags* to discover ICS element's physical identifiers such as their IOA [42]. OPC naming conventions allows SCAPHY to extract element's possible states, which allows SCAPHY to automatically *switch* element states during impact modelling.

For example, the full OPC tag in the Industroyer example *BRK.0.BOOL=1* denotes an element *BRK*, an ID of 0, and a boolean state of 1. By the *BOOL* tag, SCAPHY knows to switch its state between 1 and 0, as opposed to to an integer.

Element Partitioning and Process Identification. A physical process comprises of elements that work together to achieve a process outcome, which is determined by the state of a *terminal* element. For example, the tank in the *level control* process of the FL water plant is the terminal element because the process output depends on the water level held by the tank, which is measured by a level *meter* sensor. SCAPHY partitions ICS elements into two sets; terminal and non-terminal sets (E_{Term}, E_{NTerm}). SCAPHY identifies unique processes based on E_{Term}. To partition elements, SCAPHY analyzes OPC Alarms data and extracts *process identifiers*, which are unique tuples of a sensor and terminal element which uniquely identifies processes. SCAPHY represents processes in the form of $P_j = (SID, E_{Term_j})$; where SID is a process sensor ID and E_{Term_j} is a *terminal* element for process P_j. SID monitors E_{Term_j} state, which corresponds to the outcome of P. For example, SCAPHY represents the level control process as *LevelControl=(meter,tank)*. Other elements the plant such as pumps and valves are in E_{NTerm}. Based on identifying all processes, SCAPHY partitions the set of elements E such that:

$$(E = E_{Term} \cup E_{NTerm}) \tag{1}$$

$$\{\forall i, j \in (E_{Term}, E_{NTerm}); i \neq j; i \cap j = \emptyset\} \tag{2}$$

$$|E_{Term}| := |P| \tag{3}$$

$|E_{Term}|$ equals the number of processes, $|P|$

Process Dependency Mapping. After identifying processes, SCAPHY analyzes element's connections (also called 2-wire diagrams) to map non-terminal elements to their connected processes. This connection describes the physical and sequential logic of elements, which captures the information flow between them. Connection schemas are automatically generated by SCADA programs during design of ICS scenarios. From each E_{Term} node, SCAPHY traces its connection *paths* until all elements connected to E_{Term} are identified. Each process now has a list of paths, *PATHS*, which contain their dependency paths, *DepPath*. *DepPath* is a set of E_{NTerm} nodes arranged in sequential order from the E_{Term} node identifying process p, and given as follows:

$$DepPath(p) := \{E_{NTerm_0}, ..., E_{NTerm_n}\} \tag{4}$$

$$PATHS(p) := \{DepPath_i(p), ..., DepPath_n(p)\} \tag{5}$$

SCAPHY then aggregates all elements in all *DepPath* of a process into a dependency element set $DEP(P)$ such that:

$$\{\forall i \in PATHS(p) : DEP(p) := \bigcup_{j=0}^{n} i(p)\} \tag{6}$$

where $DEP(P)$, the dependency element set of P, is the union of all elements in all the dependency paths of a process path-list $PATHS$. SCAPHY keeps tracks of the internal ordering among these elements as developed from Equation 4. SCAPHY uses this order information to detect out-of-order signals.

Functional Inter-Process Relationships. SCAPHY identifies functional relationship between processes who share common elements among them. If a shared element is in E_{Term} of P_1 and in E_{NTerm} in P_2, then P_1 depends on P_2. SCAPHY models such element as an *inter-process transfer points* (PTP), and P_1 and P_2 as *PTP* sink and *PTP* source respectively. PTP instances is common among physical elements of the Boolean type such as valves, switches, and relays. SCAPHY leverages PTP instances to detect attacks spanning multiple processes, such as in the Florida water poisoning attack (shown in Subsection VI-B). PTP events occur when a control signal causes a PTP element to change state, which causes the PTP *sink* to assume the value of the PTP *source*'s E_{Term}. Through this, SCAPHY detects disruptive impact on the PTP *sink* process stemming from the PTP *source*. Further, when processes share an element that is non-terminal to both processes, SCAPHY models this as multiplexed (MP) elements. When a MP state change causes an anomaly, SCAPHY aggregates the anomaly score based on all affected processes.

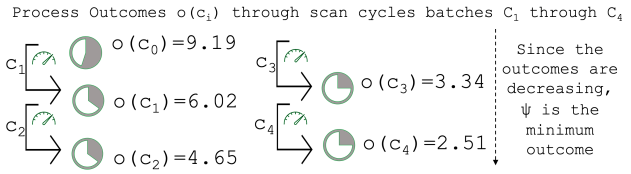
B. Modelling Process Dependency and Impact

SCAPHY models how each dependency element state of a process impacts the process using a novel *process dependency and impact graph* (PDIG) model. SCAPHY leverages PDIG to automatically know which elements maps to what process and which elements states decrease or increase the process output.

Sketching PDIG Gragh. PDIG is a set of vertices which represents elements, and edges which connects element nodes based on their relationships. Undirected *process edges* connect terminal elements to each non-terminal element in the *same* $DEP(p)$ of the process. Undirected *element edges* connect non-terminal elements that don't have any ordering constraints. Directed *element edges* connect two non-terminal elements that have ordering constraints among them in the direction of the ordering. Each element node has an attribute that captures its possible states and impact vector. In the PDIG, SCAPHY automatically identifies and annotates PTP instances when there is an undirected element edge from a terminal element (of the PTP *source*) to a non-terminal element (i.e., the PTP element). Further, multiplexed elements are identified when an element has more than one process edge.

Deriving Impact to Model Physical Effects. SCAPHY assigns each element state an impact vector $I_C(s)$ based on how they impact a process. $I_C(s)$ is used in anomaly scoring of disruptive impact based on the affected element state. SCAPHY also uses $I_C(s)$ to prune out non-impactful states, which reduces the number of elements to consider during attack detection. To derive $I_C(s)$, SCAPHY leverages a SCADA ICS engine to drive the physical process under analysis. In our work, we leveraged in the Siemens S7 WinSPS program [31] and FactoryIO ICS Engine [25] to load and drive physical processes. SCAPHY then iteratively switches each state in the process and analyzes the moving average of process outputs via reading the process sensor element. SCAPHY stops evaluating the change in output when successive output changes become negligible (less than 1%) or reaches a *steady state*. We normalize $I_C(s)$ with respect to the scan cycles analyzed, which bounds its value between 0.0 and 1.0. This succinctly describes the impact of each state relative to other states.

Formulation of $I_C(s)$ We define a process *outcome transition* set τ_n, which is a set of ordered outcomes o for a process



(a) Process operation scan cycles

Let the Calibrated range of P be 0 - 20.50cm, and P's outcome after cycle C_i be $O(c_i)$, Impact Coefficient $I_C(c_i)$ is as follows:

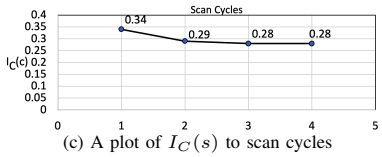
$$I_C(c_1) = \frac{\frac{(6.02-9.19)}{(9.19-0)}}{1} = -0.34$$

$$I_C(c_2) = \frac{\frac{(6.02-9.19)}{(9.19-0)} + \frac{(4.65-6.02)}{(6.02-0)}}{2} = -0.29$$

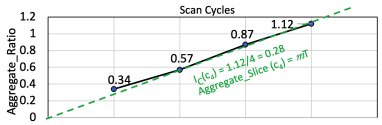
$$I_C(c_3) = \frac{\frac{(6.02-9.19)}{(9.19-0)} + \frac{(4.65-6.02)}{(6.02-0)} + \frac{(3.34-4.65)}{(4.65-0)}}{3} = -0.28$$

$$I_C(c_4) = \frac{\frac{(6.02-9.19)}{(9.19-0)} + \frac{(4.65-6.02)}{(6.02-0)} + \frac{(3.34-4.65)}{(4.65-0)} + \frac{(2.51-3.34)}{(3.34-0)}}{4} = -0.28$$

(b) Computation of Impact Coefficient



(c) A plot of $I_C(s)$ to scan cycles



(d) Plot of Aggregate Slices to scan cycles

Fig. 5: SCAPHY Impact Coefficient $I_C(s)$ Derivation for Each Element State

from scan cycle c_j to c_n:

$$\tau_n := \{o(c_j), o(c_{j+1}), o(c_{j+2}), ..., o(c_n)\} \tag{7}$$

where c_j is the first scan cycle following SCAPHY switching of an elemetary state and c_n is the last or current scan cycle observed where $0 \leq j \leq n, n \in \mathbb{Z}$. SCAPHY keeps track of the highest or lowest recorded outcome ψ (i.e., the boundary outcome). For any scan cycle c_j, we define the $I_C(s)$ of the element state under analysis as follows:

$$I_C(s) = \frac{\sum_{i=j}^{n} \frac{o(c_i)-o(c_{i-1})}{ab(\psi-o(c_{i-1}))}}{|\tau_n|} \tag{8}$$

where $o(c_i) - o(c_{i-1})$ is change in process outcome, $ab(\psi - o(c_i))$ is the absolute difference between the highest or lowest outcome and the preceding value at the scan cycle $i-1$. Further, $|\tau_n|$ is cardinality of the scan cycles from c_j to c_n.

We see that $\frac{o(c_i)-o(c_{i-1})}{ab(\psi-o(c_{i-1}))}$ is the ratio of the current process change (i.e., $o(c_i) - o(c_{i-1})$) to maximum change ($\psi - o(c_{i-1})$). If we aggregate this ratio for each scan cycle, we can compute $I_C(s)$ instantaneously at any scan cycles we chose without having to always compute $I_C(s)$ through all previous scan cycles. Using the aggregate ratio to compute the instantaneous derivation of $I_C(s)$ is given as follows:

$$I_C(s) = \frac{\delta}{\delta T} Aggregate\_Slice(o(c_n)) \tag{9}$$

where, for all scan cycles T, $Aggregate\_Slice(o(c_n))$ is the

Algorithm 1 DeriveImpactCoefficient($I_C(s)$)

Input: ElementState s, Process p
Output: $I_C(s)$: ▷ Initialization

$Cycle, Cycle_{MAX} \leftarrow GetCycleBatchAndMax$
$\psi \leftarrow GetOutcomeBoundCalib$
$O_{PREV} \leftarrow GetProcessInitOutcome(p)$
$SteadyState \leftarrow GetSteadyState(p)$
$Aggregate\_Slice \leftarrow 0$
while $Cycle_{MAX} > 0$ **do**
  $SDK\_RunSim(s, p)$
  $O_{c_i} \leftarrow GetProcessOutcome(p)$
  $O_{DIFF} \leftarrow O_{c_i} - O_{PREV}$
  $Aggregate\_Slice \leftarrow (Aggregate\_Slice + O_{DIFF})$
  $Aggregate\_Slice \leftarrow Aggregate\_Slice/ABS(O_{PREV}) - \psi)$
  $I_C(s) \leftarrow Aggregate\_Slice/Cycle$ ▷ check if steady state is reached, if so return $I_C(s)$
  if $O_{DIFF} < SteadyState$ **then**
    Return $I_C(s)$
  else
    $Cycle++$
    $Cycle_{MAX}--$
    $\psi \leftarrow UpdateOutcomeBound(\psi, O_{c_i})$ ▷ update ψ if neccesary
Return $I_C(s)$

sum of current process change to maximum change from c_j to c_n, and defined as:

$$\{\forall c_i \in T : Aggregate\_Slice(o(c_n)) := \sum_{i=j}^{n} \frac{o(c_i) - o(c_{i-1})}{\psi - o(c_{i-1})}\} \tag{10}$$

Figure 5b illustrates a derivation of $I_C(s)$ through scan cycles c_1 to c_4. At each scan cycle transition, the generated process outcomes, 9.19 through 2.15 were inputted into the $I_C(s)$ formula to compute the $I_C(s)$ scores. Notice that at scan cycle c_4, the difference in the subsequent $I_C(s)$ (i.e., from c_3) was negligible (i.e., 0). SCAPHY uses this observation as a heuristic to detect steady states. Otherwise, SCAPHY sets a maximum bound to stop the simulation. Figure 5d shows $I_C(s)$ for the end scan cycle c_n ($n = 4$) using the iterative form and the instantaneous $I_C(s)$ form (green dotted line). The $Aggregate\_Slice(o(c_n))$ function is a straight line ($Aggregate\_Slice(c_n) = mT$) drawn from origin to the point $c_n \in T$, where m is the slope. Taking the derivative of $Aggregate\_Slice(c_n) = mT$ gives the instantaneous $I_C(s)$.

Algorithm 1 shows SCAPHY algorithmic approach to derive to $I_C(s)$. $Aggregate\_Slice$ is the aggregate ratio of measured impact to maximum possible impact, and $Cycles$ is the total cycle batches in terms of scan cycles.

C. Characterizing Physical and Signal Anomalies

Inconsistent State. SCAPHY detects when a process is in *inconsistent state* when two dependency element states have *opposing* impact vectors. Recall that state's $I_C(s)$ in the PDIG drives process output in *one* direction towards its goal (i.e, not opposing direction). For example, Industroyer attacked *LB* process aims for a *LB* factor of 1.0 between power supply and demand. Although *LB* is affected by several factors, the states of load lines and circuit breakers play a role. If load decreases or disconnect (e.g., due to low demand), process-control reacts by sending control signals to open circuit breakers to bring back the balance. So, *load disconnecting* is balanced by *breaker opening* and drives the process towards its goal. However, a control session involving *opening* circuit breakers and *connecting* load lines is inconsistent and disruptive to

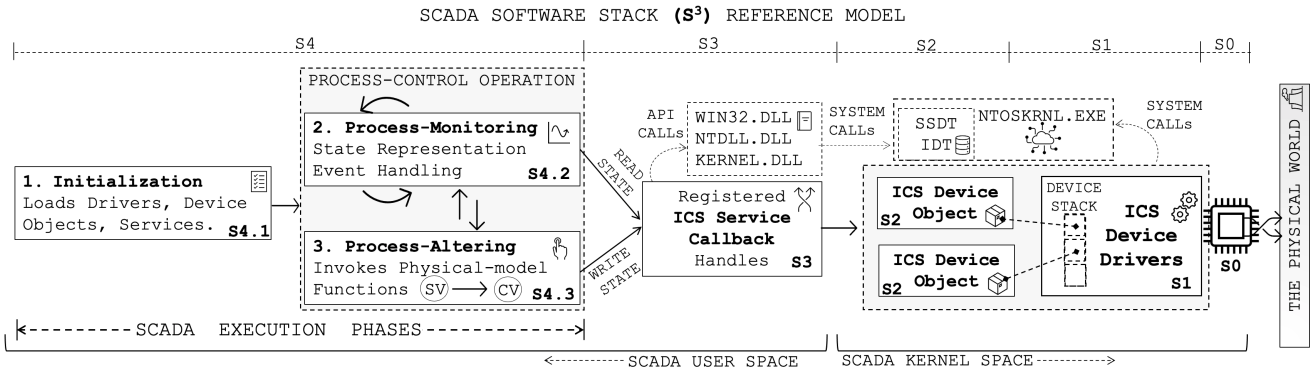


Fig. 6: SCADA Software Stack (S^3) and SCADA Execution Phases: Showing SCADA Host Interactions with S^3 Layers to access the Physical World

load balance and should never occur in any legitimate setting. SCAPHY's $I_C(s)$ model captures this element state relationships and detects such *inconsistent state* physical anomalies. Through this, SCAPHY detected the Industroyer attack based on physical effects of the attack signal (Section IV).

Outside Setpoint. SCAPHY detects when control signals drive a process to exceed what it is *operationally* caliberated for based on learning the highest and lowest bounds of the process output recovered during the $I_C(s)$ derivation.

Signal Anomalies. SCAPHY detects (i) *missing signals* when a process' control session traffic has incomplete signals based on the number of its process' dependency elements. We observe that targeted attacks send isolated or incomplete signals to disrupt a specific element. (ii) *Extraneous signals* arise when a process' control session traffic contains signals for elements not associated with the process. (iii) *Out-of-Order signals* occur when analyzed sequence of signals in a process' control session are not in the expected ordered flow based on PDIG's element ordering. Although not all process requires control signals be ordered, some device logic demand ordering constraints on the incoming signals to properly operate a process.

Scoring Anomalies. SCAPHY computes anomaly scores based on affected element's $I_C(s)$. Let m be number of affected elements, and s_i is the state transitioned by the suspicious control signals. Let n be the number of the process dependency elements and $I_{C_{MAX}}(j)$ be the maximum $I_C(s)$ for an element. Anomaly score is given by:

$$\{\forall j \in DEP(p) : Anomaly\_Score = \frac{\sum_{i=0}^{m} I_C(s_i)}{\sum_{j=0}^{n} I_{C_{MAX}(j)}}\} \quad (11)$$

Using $I_{C_{MAX}}$ in the denominator normalizes and bounds the scores between 0 and 1 which succinctly captures the deviation measure relative to all process elements. Based on this normalization, we calibrated a detection threshold boundary for low, medium, high anomalies as 0.0-0.25, 0.26-0.60, and 0.61-1.0 respectfully. We found that this produced the best accuracy categorization for all scenarios we tested. However, as with all anomaly systems, operators can fine tune this threshold boundaries based on their need.

D. Analyzing Physical World-Targeted Executions in SCADA

Given an ICS scenario (OPC data and Function block logic), SCAPHY aims to generate the limited set of API calls unique to legitimate SCADA process-control operations, referred to as PHYSical world Impact Call Specialization (PHYSICS) constraints. To do this, SCAPHY leverages its physical model to inform a *physical process-aware* dynamic analysis, whereby a SCADA engine is induced to execute code paths of process-monitoring and altering behaviors. However, this requires first knowing each phase identifier and boundary

Leveraging SCADA Software Stack (S^3) to Characterize Process-Control Behaviors. Through in-depth analysis of process-control in diverse ICS settings, we introduce a new reference model, *SCADA Software Stack* (S^3). S^3 layers does not replace Purdue ICS Levels [33]. Rather, it breakdown Purdue Level 2 (i.e., control systems) into 5 layers to characterize the internal software and hardware layers involved in SCADA process-control, as shown in Figure 6. We hope that via S^3, Antivirus companies can develop SCADA-specific host agents to monitor accesses to specific S^3 layers to detect attacks. We describe the S^3 layers shown in Figure 6 using Windows operating system (OS) due to its dominance in ICS.

SCADA programs ($S4$) do not access ICS devices directly but do so using *device objects* ($S2$), which are software handles that enable the OS to mediate access to physical I/O ($S0$). To support diverse ICS devices, Windows provides a Driver Model (WDM) to allow device vendors to run ICS drivers ($S1$) in the kernel. In WDM, *driver objects* of an ICS driver represent instances of ICS devices the driver supports. For example, Windows supports 16550 UART devices via Windows Serial Driver, which allows SCADA programs to declare device objects, called COM ports, to communicate with devices.

ICS Callback Functions. To access devices, SCADA programs invoke ICS callback functions ($S3$) registered during ICS driver load. This callback function invokes *CreateFile* which returns the device object handle as shown in Figure 6. Parameters *lpFileName* specifies device object name (e.g., COM1); *dwDesiredAcces* specifies READ or WRITE access mode; *dwShareMode* enables SCADA programs to deny other programs (such as malware) access to physical devices. Unfortunately, attackers (Havex, Industroyer, Oldrea) subvert this access control by killing benign SCADA processes to release their ICS handles. For example, Industroyer killed *D2MultiCommService.exe* and hijacked all serial COM ports to the Siemens SPIROTEC device. Havex [46] scanned COM Ports to discover connected devices. After obtaining device handles, SCADA programs (and attackers) can invoke ReadFile to read device states or *WriteFile* to send signals to them. Based on this analysis of S^3 layers, SCAPHY can monitor

ICS scenarios	Industry	Physical World Dependency & Impact Model: physical processes	ID	*OPC* E_{Term}	*PDIG* nodes	I_C avg/max	PHYSICS Constraints: behavior calls	behavior stack	verify TP	verify FP	time (min)	ICS program files: OPC points and tags size	element	wires
power grid	Power Plan	load balancing	1.1	c-breaker	6	.71/.76	6	7	6	0	4.3	9K	19	35
		pwr distribution	1.2	load lines	4	.59/.66	4	6	4	0	3.5	9K	19	35
water treatment	Water Plan.	level control	2.1	holding tank	4	.56/.86	10	12	10	0	6.2	11.5K	13	23
		dosing	2.2	dose valve	2	.66/.86	4	11	4	0	5	11.5K	4	9
auto warehouse	Manufactur	pallet alignment	3.2	Axes X,Z	6	.47/.62	8	13	8	0	5.9	9K	10	19
		throughput	3.2	entry conveyor	2	.7/.84	4	11	3	1	5.5	9K	6	11
assembler	Manufactur	product quality	4.1	clamp lid/base	2	.67/.77	4	7	4	0	7	9.5K	8	19
		load balancing	4.2	conveyor2	5	.8/.84	6	14	6	0	4.6	9.5K	11	19
palletizer	Shipping	load alignment	5.1	push clamp	3	.47/.71	7	13	7	0	7.2	7.8K	7	13
		prod protection	5.2	entry conveyor	6	.32/.69	4	17	4	0	5	7.8K	9	13
hvac system	Gas	heat setpoint	6.1	room space	3	.46/.79	8	12	7	1	6.1	6K	8	15
		heat flow	6.2	vent	3	.6/.82	6	19	6	0	5	6K	7	14
converge station	Shipping	path throughput	7.1	load/unload	2	.47/.6	7	13	7	0	5	6.2K	9	14
		alt throughput	7.2	transfer	3	.67/.88	6	12	5	1	4.8	6.2K	9	17
production line	Manufactur	alignment	8.1	control arm	2	.8/.8	6	15	5	1	2.6	8K	11	23
		throughput	8.2	conveyors	4	.6/.72	6	16	6	0	3.3	8K	11	14
sort station	Shipping	sort accuracy	9.1	unloader	2	.54/.85	4	13	4	0	4.7	9K	6	13
		throughput	9.2	conveyor	7	.5/.9	6	17	6	0	4.4	9K	16	14
separator	Shipping	accuracy	10.1	pusher1-2	6	.53/.72	5	12	4	1	5.9	4.9K	17	19
		throughput	10.2	conveyors	7	.69/.81	4	8	4	0	4.8	4.9K	15	10
elevator	Manufactur	prod safety	11.1	conveyor1-3	5	.77/.83	6	13	6	0	9.2	11K	13	14
		throughput	11.2	entry conveyor	5	.33/.68	4	19	4	0	5	11K	12	24
queue processor	Manufactur	spacing	12.1	buffer conveyor	6	.63/.71	6	13	6	0	5.7	9K	17	9
		throughput	12.2	entry conveyor	2	.67/.8	5	11	4	1	4.7	9K	14	11

TABLE II: ICS Scenarios: SCAPHY's physical world modelling results and generated SCADA PHYSICS constraints with Diverse ICS Industry Applications

CreateFile, [*WriteFile* | *ReadFile*] to triage attacker's process-altering and process-monitoring activities, respectively.

Identifying Process-Control Phase Windows. Existing execution phase analysis for web servers [64] rely on developer-supplied *boundaries* to identify phase transitions. This approach is manual and will not work if boundaries are not available such as in proprietary settings like ICS. However, SCAPHY leverages S^3 layers to analyze the cyclical nature of SCADA process-control to identify its transitions from process-monitoring to process-altering. We know that accessing device objects ($S2$) using *CreateFile* in the "WRITE" mode is a process-altering behavior, and in "READ" mode is process-monitoring, but we need to know when they start and ends (i.e., phase window) to accurately specialize the extracted API calls. Based on the cyclical API call stack behavior of these phases shown in Figure 2, we know that process-altering *follows* process-monitoring. As shown, after process-altering are a few memory freeing operations which we found was to free up memory buffers used in physical-model computations.

We analyze process-monitoring to know its phase window. Process-Monitoring comprise of two main sub phases; *process state representation* (reading the process state), and *event handling* (analyzing events to see if change is needed). If no change is needed, it repeats as shown in Figure 2. SCAPHY analyzes the changing *IP* register and call stack depths of the loop structure to identify the end of event handling the first time it returns to the *IP* it started. When event handling ends but *IP* returns to a different location and call stack depth, SCAPHY identifies this as the start of process-altering execution. Finally, SCAPHY performs a final check by confirming the expected S^3 phase identifiers in each phase, which are [*CreateFile*, *ReadFile* || *WriteFile*], *CloseHandle* API call sequences for process-monitoring and process-altering respectfully.

Physical Process-Aware Dynamic Analysis. SCAPHY leverages an ICS emulation engine (FactoryIO [25]) and a SCADA engine (Siemens S7 WinSPS [31]) to perform a *physical-process-aware* dynamic analysis of process-control behavior to generate PHYSICS constraints. FactoryIO (which also provides a SCADA engine) provides an environment to setup and drive physical processes. Its emulation engine supports interfacing with real hardware-in-the-loop PLCs using real ICS protocols such as Modbus, and allows loading of generic ICS scenario function blocks. On starting each process, we monitor the SCADA host API executions to know when each process-control phase start based on the *phase windows*. For each process, we *induce* the SCADA execution to *re-compute* the process control variable (CV) (i.e., to send to the physical) by iteratively switching each element state in the process. This drives SCADA execution down the process-monitoring and process-altering code paths to effect change on the process, enabling SCAPHY to record the API calls of each phase.

Process-Aware PHYSICS Constraints. SCAPHY's physical world model enables generated PHYSICS constraints to be process-aware, i.e., generated per physical process. Although many code paths exist in SCADA executions, our PHYSICS constraints cover *only* process-control paths. SCAPHY covers these paths by inducing the legitimate SCADA process-control logic to react to each element state change, ensuring that all "state-changing" relevant logic paths are dynamically covered. Because element states are derived from the deployed environment via OPC, SCAPHY's PHYSICS constraints succinctly represent the limited legitimate APIs to control the physical.

PHYSICS Violations by Injection. To perform attacks, attackers execute atypical APIs outside of benign process-control phase. SCAPHY detects APIs not in established PHYSICS constraints as *injection violations*. As such, SCAPHY can detect when malicious code injected into benign SCADA programs run (as was done by Stuxnet [7]) as wells as redirected API calls via *hooking* the Import Address Table. Stuxnet-type attacks will evade existing tools that *whitelists* benign SCADA programs. However, because SCAPHY focuses on executed APIs, not the *executor*, injected APIs will be detected.

PHYSICS Violations by Bypass. Rootkits can bypass the S^3 layers and directly send malicious signals to physical I/O using

	MITRE ICS Attack ID	Attack Description	In-the-wild ICS Reference	MITRE ICS TTP	Physical Process IDs (in Table II))	PHYSICS bypass	PHYSICS inject	Physical Anom. incons	Physical Anom. setpoint	Signal Anom. miss	Signal Anom. extran	Signal Anom. ooo	Metrics TP	Metrics FP
Process Altering	T872	wipe host/registers	Killdisk	Evasion	1.1, 1.2, 4.1	✓		✓			✓		3	0
	T836	modify parameter	Stuxnet	Impair proc contrl	4.1, 4.2, 8.1, 8.2		✓		✓		✓		3	0
	T831	contrl manipulation	Stuxnet	Impact control	2.1, 2.2, 6.2		✓			✓		✓	3	1
	T889	kernel driver attack	Blaster	Modify Program fxn	5.1, 5.2, 10.2	✓			✓				2	0
	T855	unauthorised cmd msg	Industroyer	Impair proc contrl	3.1, 3.2, 11.1		✓	✓			✓		3	1
	M1.2	corrupt registers	Triton	Impair proc ctl	7.1, 10.2, 11.2		✓	✓			✓	✓	4	0
Non-Process Altering	T874	library hooking	Triton	Execution	5.1, 5.2, 10.1, 10.2		✓						1	0
	T801	monitor proc state	Industroyer	Collection	6.1,6.2, 8.1		✓						1	0
	T861	points/tags identifica.	Backdoor.Oldrea	Collection (OPC)	6.1, 6.2, 8.2		✓						1	0
	T816	device shutdown	Industroyer	Inhibit Resp fxn	7.1, 9.1, 9.2, 7.2		✓						1	0
	T888	network Enumeration	Havex(as is)	Discovery fxn	4.1, 4.2, 8.1, 8.2		✓						1	0
	T805	block serial COM	Industroyer(as is)	Inhibit Resp fxn	1.1, 1.2, 9.1		✓						1	0

TABLE III: Deployed Attacks and Detection Metrics. We leverage the MITRE ICS Attack Framework [65] to categorize the attack TTPs

a direct driver call *0x6b DeviceIOControl*. However, because SCAPHY sees all *WRITE* traffic on the physical interface ($S0$), it detects the attack as *Bypass* violation because no $S2$ activity was seen, allowing SCAPHY to know that a kernel-space entity bypassed proper process-altering S^3 channels to send signals.

VI. EVALUATION

We evaluate SCAPHY's ability to (i) accurately detect a variety of ICS attacks across diverse ICS scenarios, and (ii) outperform existing tools in detection accuracy. We launched 40 ICS attacks on 24 diverse ICS scenarios across 5 ICS industries to show SCAPHY versatility, as shown in Table II. SCAPHY detected 95% of all attacks, including real ICS malware (Havex[1] and Industroyer[2]), with only 3.5% false positives. Due to lack of research resources to support diverse ICS security research [27–30], we make available over 200GB of new ICS experiments and ICS attacks against both SCADA and physical targets, developed for FactoryIO ICS Engine [25].

Experimental Setup. We leveraged a U.S. national lab state-of-the-art ICS testbed, which supports fast deployment of ICS topologies, OPC, HMIs, and Windows 7 SCADA VMs, prepared with ICS resources to control physical processes such as UART interfaces and Windows Serial Driver. For SCADA control, we used 3 platforms: S7 WinSPS [31], MyScada [66], and FactoryIO SDK. This makes our testbed suited to evaluate SCAPHY against ICS attacks in realistic settings. To support diverse processes, we leveraged Simulink [67], PowerWorld [68], and FactoryIO [25] Engine to emulate physical process components in Remote Terminal Units (RTUs).

ICS Attacks Performed. We performed diverse modern attacks from 4 categories: attacks that (**I**) maliciously alter element states in running processes, (**II**) blocks SCADA access to the physical, (**III**) collects attack-relevant data from SCADA, and (**IV**) exploit bugs in ICS devices. We leveraged Mitre ICS Attack Framework and ICSSploit [69] to develop realistic attacks, indicated in "In-the-wild" column of Table III and Table IV, each tailored against their pertinent ICS target.

Physical World Model and PHYSICS Constraints. Table II presents our analyzed ICS scenarios details and SCAPHY's derived physical world models and PHYSICS constraints. To verify accuracy of generated PHYSICS constraints (API calls), we produced forensic execution traces of our SCADA program using the Time-Travel debugging feature of Windows Debugger (WinDBG), which we manually stepped to see the APIs of each process-control path. Table II column 8-11 shows the number of unique *process-altering* APIs called on average per path. As shown, we found that only few unique APIs were seen (most times in loops) depicting that *process-altering* is very specialized per physical domain, but the high stack depth shows that *process-altering* is executed deep in SCADA logic. We found that the false positives (FP) were due to rare element states not parsed correctly from OPC. SCAPHY's physical model allowed for an efficient impact analysis of dependency elements, which allowed SCAPHY to prune non-impactful elements from the original pool extracted from OPC. We observe that many ICS elements such as sensors and repeaters do not have an operational impact on process output, hence were pruned off during $I_C(s)$ derivation. As such, we saw over 50% reduction (on avg) from extracted OPC elements to PDIG nodes. This outperforms naively analyzing all states regardless of impact as done in [20] (compared in Subsection VI-D).

Further, the reduced element pool contributed to an $I_C(s)$ (impact) average above 70% of their process output, showing that SCAPHY modelled the relevant or *impactful* elements whose malicious state change are more disruptive to the process. SCAPHY physical-process-aware dynamic analysis takes about 6 mins, which is reasonable per deployed scenarios sizes (last 3 columns). Our ICS scenarios are adapted from real-world models developed by domain experts. For example, our power grid scenario was adapted from an open-source Texas Pan Handle power grid [26], simulated in *PowerWorld* RTUs. We use the power grid example to explain Table II: SCAPHY accurately identified the process terminal element (E_{Term}) as circuit breaker and load lines. As shown, SCAPHY's impact analysis pruned the OPC element pool of 19 nodes to 10 PDIG nodes, which allows SCAPHY to be efficient. The average impact of all PDG nodes was over 50% with max at 0.76 and 0.66, meaning that attack involving them are most disruptive.

A. ICS Attack Detection

Table III shows details of attack categories **I**, **II**, and **III**, and SCAPHY's detection metrics. Attack category I are shown in the first row-group, "Process Altering". Category II and III are shown in the second-row group, Non-Process Altering. SCAPHY detected PHYSICS bypass and inject attacks for both categories. However, SCAPHY detected physical and signal anomalies for only category I, because category II and III do not send *WRITE* control signals. That is, they do not alter running processes but either block SCADA access to the physical or collect data about physical devices from SCADA. For example, in $T805$, Industroyer issued *CreateFile* on available

[1] 7f249736efc0c31c44e96fb72c1efcc028857ac7
[2] 2cb8230281b86fa944d3043ae906016c8b5984d9

ICSSPLOIT Attack ID	Attack Description	Real-world Device Targets	In-the-Wild CVEs/ICSAs	Exploit Type	*SCADA Software Stack* (S^3) Activity							*Metrics*		
					$S4.1$	$S4.2$	$S4.3$	$S3$	$S2$	$S1$	$S0$	TP	FP	FN
SPLOIT1.1	stop controller/CPU	Siemens Simatic-1200	ICSA-11-186-01	Unprotected Port	-	-	✓	✓	✓	✓	✓	5	0	2
SPLOIT1.2	remote code execution	QNX SDP 660	CVE-2006-062	Buffer Overflow	-	✓	✓	✓	✓	✓	✓	4	0	1
SPLOIT1.3	remote device halt	Schneider Quantum	ICSA-13-077-01	I/O corruption	-	✓	✓	✓	✓	✓	✓	5	0	1
SPLOIT1.4	crash RTOS service	QNX INETDd	CVE-2013-2687	Buffer Overflow	-	✓	✓	✓	✓	✓	✓	5	0	1
SPLOIT1.5	RPC device crash	WindRiver VXWorks	CVE-2015-7599	Integer Overflow	-	✓	✓	✓	✓	✓	✓	4	0	1
SPLOIT1.6	denial of service	Siemens S7-300/400	CVE-2016-9158	Input Validation	-	-	✓	✓	✓	✓	✓	4	0	2

TABLE IV: Deployed Attacks and Detection Metrics. We leverage Attack Modules ICSSPloit Attack Modules [69] to categorize the attack TTPs

COM ports to block our SCADA program from accessing the physical ("as is" mean we executed the In-the-wild malware). Similarly, Havex in $T888$ enumerated all *COM* device objects stored in Registry keys *HKLMSYSTEMCurrentControlSetServices* (registry value *SERVICE_KERNEL_DRIVER* are device driver services) to identify connected devices by issuing loops of *OueryKey*, *OpenKey*, *QueryValueKey* API calls. These API calls were atypical of process-monitoring, allowing SCAPHY to detect their PHYSICS inject violations.

SCAPHY detected two PHYSICS bypass violations in attack category I, $T872$, and $T889$, which are kernel level attacks that bypassed SCAPHY monitored S^3 layers. Specifically, $T889$ and $T872$ used *DeviceIOControl* direct driver call to send signals out the serial I/O. SCAPHY detected the attack because they sent out *WRITE* signals without accessing S^3 layers, allowing SCAPHY to know that a kernel-space entity bypassed proper *process-altering* and S^3 channel to attack the physical. Further, $T872$ used the same call to send control code to the storage device driver to delete whole drives (/DosDevices/C:). The Last 3 columns show detection metrics. Ground truth was derived from open-source attack data [65, 70]. We found that FPs were due to missed APIs during PHYSICS analysis due to rare element states not parsed from OPC.

Evaluating SCADA Software Stack (S^3) Activity. For attack category **IV**, we demonstrate S^3's ability to *pinpoint* steps along SCADA access to the physical, which shows that S^3 layers are practical host *monitoring taps* for ICS attack detection. To do this, SCAPHY monitored access to each S^3 layer based on the appropriate API call for each layer. E.g., the *ReadFile* accesses $S2$, the ICS device object. We leveraged ICSSPLOIT [69] to compile real-world exploits that were developed as proof-of-concepts exploit code against real bug in ICS devices. We launched these attacks against their simulated ICS targets and then check which S^3 component was triggered in the SCADA host. Table IV details the results. SCAPHY detected all five layers of S^3 in all attacks as shown. However, in layer $S4$, SCAPHY did not detect *initialization* behavior ($S4.1$). This is because the exploits where self-contained and did not issue any API to setup environment. However, SCAPHY detected their *process-altering* behavior ($S4.3$) when the *WriteFile* API was called to send signal to the physical. We hope that via S^3, Antivirus companies can develop SCADA-specific host agents to monitor and correlate accesses to specific S^3 layers to detect attacks.

B. Case Study: 2021 Florida Water Plant Poisoning Attack

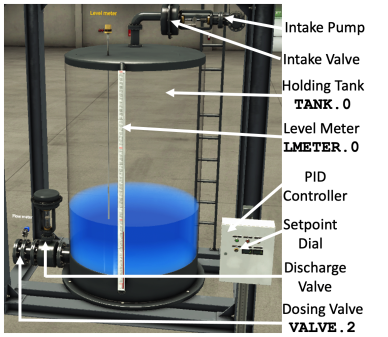
We replicated the FL incident with realistic water treatment scenario using FactoryIO Engine[25] and open-source data [27, 28]. The attacker targeted the chemical dosing operation by manipulating a *proportional* P parameter to raise toxic levels of NaOH in the water outside the set point (*SV*) [1]. Chemical dosing involves two processes: *Level control* and *Dosing*. The HMI is shown in Figure 7a. Level control aims to fill a holding tank, *TANK.O* with chemical based on *SV*, after which Dosing will open a valve, *VALVE.2* to let chemical into the water supply [27, 28]. Level control is controlled by a domain-specific physical model function, *Proportional Integral Derivative* (*PID*). Because *SV* cannot be reached in one shot, PID operation involves several "intake" and "discharge" cycles, (shown in Figure 7d) whereby an intake pump fills chemical into *TANK.O*, and a discharge valve remove accesses. P controls how aggressive the intake and discharge cycles are driven. E.g, a high P pumps an *initial* excessive volume into *TANK.O*. Figure 7c shows how different P values affect how *SV* is reached. *SV* is set via a hardware dial on the PID controller, so attacker cannot modify it using cyber means.

Attack and Detection. The attacker issued 2 control signals to raise P (dumps excessive chemical into *TANK.O*) and open *VALVE.2*. We launched the attack using a Modbus payload, but in a less suspicious manner, that is, make attack code self-contained, without triggering any PHYSICS violation. At the SCADA side, the modbus operation triggered all S^3 layers to send out the signal, which is not by itself malicious. We focus on the physical aspects to detect the attack and correlate with the *process-altering* activity observed at the SCADA host. SCAPHY's detected a high *outside set point* physical anomaly and a low *Extraneous* signal anomaly. Figure 7b shows the detection data SCAPHY outputs to ICS operators

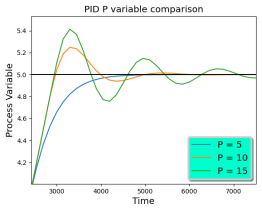
Explanation SCAPHY detection is based on the functional process relationship captured in the physical model between Level Control and Dosing. Recall Subsection V-A, *VALVE.2* is a PTP element between Level control (PTP source) and Dosing (PTP sink). When *VALVE.2*'s state changes from *CLOSE* to *OPEN*, the Dosing process output *assumes TANK.O*'s value (i.e., the E_{Term} of the PTP Source) which is measured by the meter sensor *LMETER.0*. This Dosing process outcome (*LMETER.0*'s value) was outside the set point for Dosing derived during $I_C(s)$ derivation, which allow SCAPHY to detect it. Finally, a high anomaly score is calculated based on *VALVE.2*'s $I_C(s)$ of 0.87, allowing SCAPHY to detect an *outside setpoint* physical anomaly. Further, SCAPHY detects an *extraneous* signal anomaly because the signal to open *VALVE.2* was "extraneous" in Level control's process control session.

C. Case Study: SCADA Rootkit that knows SCAPHY *approach*

We discuss SCAPHY's ability to detect an evasive rootkit that knows SCAPHY approach. Recall in Subsection VI-A, Rootkits $T872$ and $T889$ used a direct driver call *DeviceIOControl* to send out control signals. SCAPHY detected the attack because no $S2$ activity was seen, allowing SCAPHY to know that a kernel-space entity bypassed proper *process-altering* and S^3 channels to send signals. We analyze a rootkit that leverage kernel-space behaviors to perform just enough legitimate



(a) Chemical Dosing Operation HMI

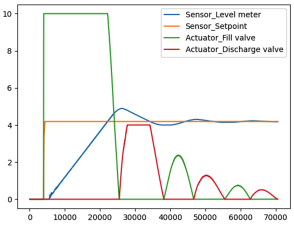


(c) Effects of different P's

```
OPC PROCESS.0.0 LEVEL CTR
OPC PROCESS.0.1 DOSING

ELEMENTS   STATE    IMP   PHY  SESSION   ANOMA.
PUMP.0     ON       0.15       MISSING   LOW
VALVE.0    OPEN     0.15       MISSING   LOW
VALVE.1    CLOSE    -0.2       MISSING   LOW
PVAR.0     VOLT 10  0.63       SIGNAL *  LOW
LMETER.0   VOLT 31   -          -
VALVE.2    OPEN     0.87 *OUT SETP EXTRAN* HIGH
```

(b) SCAPHY Output



(d) Intake/discharge cycles

process-altering behavior (i.e., access device objects in $S2$ without violating any PHYSICS constraints), without using *DeviceIOControl*. The rootkit leverages *ZwCreateFile* native API which handles parameters differently in kernel-space than the userspace *CreateFile* counterpart [71]. Unlike in userspace, calling *ZwCreateFile* with an ICS driver name returns the driver base address, which can be traversed to locate device objects in driver data structure. Using this object, the rootkit can send signals to the device using native ZwWriteFile.

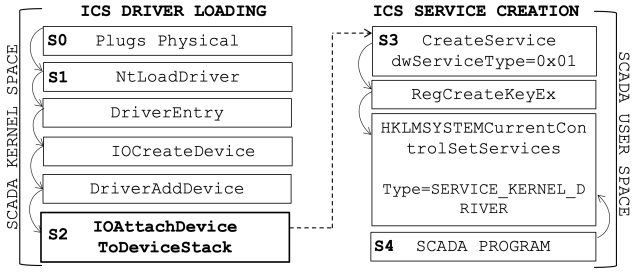


Fig. 8: S^3 Initialization at Driver Load: To detect malicious Rootkit *DriverAddDevice* $S2$ operation to add itself on a Device Stack in Kernel Land

Attack and Detection. We used Metasploit to develop such rootkit and installed it as an ICS driver. On execution, *ZwCreateFile* and ZwWriteFile call signified $S2$ *process-altering* activities, which sent out a signal, and no PHYSICS violations were observed, which evades SCAPHY. To detect this attack, SCAPHY can (**i**) ensure that the ICS device objects (used to access ICS devices) are obtained via proper device identifiers (e.g., COM1) registered in *HKLMSYSTEMCurrentControlSetServices* Registry during driver loading: (**ii**) add a feature to track malicious driver loads in the kernel.

Explanation Option **i** simply adds more constraints on the attacker and prevents him from bypassing legitimate SCADA channels ($S3$) from kernel space. However, advanced adversaries can theoretically find other OS evasions to satisfy these constraints. In option **ii**, we note that for the attack to work, the rootkit has to first *attach* itself on the *device stack* of the target device. Figure 8 shows the steps ICS drivers use to attach themselves to a device stack during driver load. Device stack is a kernel data structure that tells the kernel subsystem which drivers can access I/O of a physical device. Hence SCAPHY can track when new drivers are added to a device stack, allowing it to catch when *additional* drivers are introduced.

We note that rootkits cannot modify the kernel subsystem (*NTOSKRNL.EXE*) per *Windows Kernel Patch Protection* [72], a feature that prevents third-party components (e.g., drivers) from "patching" or modifying part of the kernel subsystem, such as the service descriptor table (SSDT) and device stack. This prevents rootkits from accesses I/O to a device that it does not exist on its stack. Figure 8 shows the API sequence that describes steps from driver loading to registering ICS callback functions. Note that argument *dwServiceType*=$0x01$ in *CreateService* registers an ICS driver service. Interestingly, these APIs follows and initializes our proposed S^3 layers. On seeing these APIs *after* an ICS driver has already been registered, SCAPHY detects this rootkit.

D. Comparing SCAPHY with Existing Techniques

We compared SCAPHY to existing ICS techniques that use physical models [20] and traffic classifier [5]. [20] analyzes sensor data's cumulative sum of residuals, and raises alarm if the difference between sensor and expected behavior is higher than a threshold. [5] analyzes spatial-temporal properties of ICS signals such as packet arrival times and size, using a REBTree classifier. [5] raises alarm when traffic features are *outside* a running average. To do this comparison, we use sensor and traffic data from the experiments presented in Table II, which we parsed into *@.ARFF* format. We leveraged open-source tools to setup these techniques. For [5], we leveraged WEKA [73] to generates a REPTree classifier that builds a decision tree using information gain and variance in the extracted traffic fields. For [5], we leveraged Scikit-Learn to generate a linear regressive model to fit the sensors values of the physical elements in the normal running mode.

To launch attacks, we follow the format in Table III, which produced 40 attacks and 146 normal instances. Table V shows the results. Existing tools did not detect any SCADA attacks due not being able to reason about SCADA execution behavior. However, this is where SCAPHY detected most attacks (90%, and 95% overall). [20] detected 19 attacks (47.5%), with a high FP of 37 (25%). Its FP is due to flagging high sensor deviations that are part of benign behaviors. One of those instances is the Florida water attack where a high *proportional* P variable causes a high but *temporary* sensor value. Although it deviates significantly from expected set point, it is benign in the Level Control process, and only anomalous in the Dosing process if dosing valve is open. Unlike SCAPHY's physical model, existing linear models [20–23] based on sensor deviations do not capture these inter-process relationships (such as between the Dosing and Level Control process), hence are prone to false alarms due to high but temporary benign deviations. SCAPHY's physical model TP is due to approximating analog states. That is for an analog voltage level of 0 - 10v, SCAPHY switches just three levels 0v, 5v, 10v, which saves space but is less precise. [5] detected 11 attacks (27.5%), but with low FP (12.3%) because most modern attack is similar to benign. SCAPHY signal anomaly had more FPs because it flagged many benign missing signals, showing that missing signals are not good indicator for attacks, which we can mitigate by raising its detection threshold.

Techniques	Approach	Attacks/ Normal	Attack Detection Metrics								
			SCADA			Physical			CTRL Signals		
			TP	FP	FN	TP	FP	FN	TP	FP	FN
Sensor [20] Analysis	Linear Regressive M.	40/146	-	-	-	19	37	21	-	-	-
Traffic [5] Analysis	Decision Tree Classifier	40/146	-	-	-	-	-	-	11	18	29
Hybrid SCAPHY	SCADA Corr. with Physical	40/146	36	5	4	21	14	19	18	21	2

TABLE V: Comparison with Existing Techniques

VII. DISCUSSIONS

Robustness of PHYSICS Constraints. PHYSICS constraints should be generated per physical process and SCADA program. Although, PHYSICS constraints are domain-specific (e.g., power scenarios), we found that *process-altering* PHYSICS constraints generated from different SCADA program do not differ much. In our work, we tested 3 SCADA platforms and they produced very similar API calls. We observe that this is because the physical-model functions (e.g., *PID*) adhere to strict invariants of the physical model. As such, when SCADA platforms update features, the core physical-model logic to drive physical processes is rarely updated.

LibVMI. is the state-of-the-art VM introspection tool [61, 62] and allows SCAPHY to monitor API calls executed in SCADA VMs per system process. Through this, SCAPHY can analyze each processes' physical world-targeted executions.

ICS Attack Difficulty. Unlike in IT, developing ICS attacks is hard due to finding the right SCADA environment and physical device target. As such, we spent months developing many modern attack scenarios, and tested more attacks than existing work, including a few ICS malware to run on ICS targets. Further, in attack category *IV* we executed real ICS attacks in the SCADA hosts that exploits historical ICS device CVEs.

VIII. LIMITATIONS AND FUTURE WORK

Based on our evaluations, SCAPHY greatly limits attacker's abilities to cause process disruptions. However, in Subsection VI-C, we see that an advanced attacker can find ways (such as using rop chains [74]) to present "good" data to SCAPHY. Although we showed that detecting malicious driver rootkits can mitigate the problem, rootkits can theoretically circumvent our assumed Windows kernel protections. As future work, we will investigate robust techniques to block SCADA rootkits from loading. In addition, we will investigate integrating call stack data as additional security layer when legitimate API calls are executed. This approach is reasonable given that the *process-control* behaviors have well-defined call stack behavior as was shown in Figure 2.

IX. RELATED WORKS

What differentiates ICS from IT systems is that physical tasks follow immutable laws of physics [20, 21], which can be learned to build prediction models [75]. Several works [20, 76] leveraged this idea to fit observed sensor data in a Linear Dynamic State Space [20] or Auto Regressive Models [22] to predict when observed behavior deviates from expected. However, in real-world ICS settings, such models may not be available or easily derived [23, 24]. Further, physical models trigger false alarms when in production due to noise and config changes, such that benign states are outside the model [23]. We found that these physical models cannot reason across multiple processes, which leads to false alarms when a high sensor deviation is benign in one process but anomalous in another.

Statistical analysis of ICS traffic [5, 9–19] are effective against *noisy* and abnormal traffic such as attacker's probings, and illegal protocol fields. However, modern ICS attacks evade them by not only using legitimate protocols but employ knowledge of ICS parameters to cause specific (i.e., not noisy) process disruptions [2–4]. Similarly, host agents [41] that monitor *system calls* executions in SCADA hosts cannot accurately detect modern attacks because they make use of same system calls as benign SCADA programs. Network flow-based detection approaches [9, 11, 12, 36] detects attacks based on abnormal traffic function codes and channels, such as demonstrated in [77, 78]. However, they are evaded by modern attacks such as Industroyer and FL water plant attacks, which uses legitimate HMI channels and function codes. Further, timing analysis [5, 10, 37] are effective for catching anomalous round trip time delays and inter-arrival times. However, they are only effective against attack behaviors that are *chatty* [10, 19] such as attacker scans, but not targeted attacks.

Sequence/Pattern-based techniques [11, 59, 79] are promising to detect anomalous communication patterns such as *isolated* signals. However, without knowing the physical behavior the signal impacts on the physical world, or if a flagged signal is actually malicious (since anomalous may or may not disrupt a process), it is prone to false positives since benign events such as noise may have same pattern. Further, pattern-based modeling of traffic baseline suffers from slight configuration changes (e.g., addition of new devices), which causes false positives due to model's high sensitivity [11, 79]. In SCAPHY, each signal is weighed based on their physical impact, which is essential to not only triaging isolated signals, but sequences of signals that may impact processes at different impact levels. State-based approaches [79] detects *critical states* in ICS but requires manual rules, which do not scale. In addition, they track and store all state transitions, which grows exponentially when ICS parameters increase. In contrast, SCAPHY is stateless and only tracks current element states. Further, by only focusing on a far-reduced pool of *impactful* element (i.e., prunes non-impactful elements), SCAPHY increases its scalability.

X. CONCLUSION

We present SCAPHY to detect ICS attacks in SCADA by leveraging the unique *execution phases* of SCADA to identify the limited set of legitimate behaviors to control the physical world, which differentiates from attacker's activities. To extract unique behaviors of each SCADA execution phase, SCAPHY first leverages OPC conventions to generate a novel physical process dependency and impact graph (PDIG) to detect disruptive physical states. SCAPHY then uses PDIG to inform a *physical process-aware* dynamic analysis, whereby code paths of *process-control* operations are induced to execute API call behaviors unique to legitimate *process-control* phase. Through this, SCAPHY detects attacker's activities that violates legitimate *process-control* behaviors. We evaluated SCAPHY with diverse ICS scenarios and attacks. SCAPHY detected 95% of all attacks, and outperformed existing tools.

REFERENCES

[1] *Oldsmar Water Treatment Facility Cyber Attack*. https://www.dragos.com/blog/industry-news/recommendations-following-the-oldsmar-water-treatment-facility-cyber-attack/.

[2] Defense Use Case. "Analysis of the cyber attack on the Ukrainian power grid". In: *Electricity Information Sharing and Analysis Center (E-ISAC)* (2016).

[3] *WIN32/INDUSTROYER: A new threat for industrial control systems*. https://www.welivesecurity.com/wp-content/uploads/2017/06/Win32_Industroyer.pdf.

[4] Ruimin Sun et al. "SoK: Attacks on Industrial Control Logic and Formal Verification-Based Defenses". In: *arXiv preprint arXiv:2006.04806* (2020).

[5] Stanislav Ponomarev. *Intrusion Detection System of industrial control networks using network telemetry*. Louisiana Tech University, 2015.

[6] *What is the Purdue Model for ICS Security?* https://www.zscaler.com/resources/security-terms-glossary/what-is-purdue-model-ics-security.

[7] *W32.Stuxnet Dossier*. https://www.wired.com/images_blogs/threatlevel/2011/02/Symantec-Stuxnet-Update-Feb-2011.pdf.

[8] *Ransomware Attack leads to shutdown of Major U.S. Pipeline System*. https://www.washingtonpost.com/business/2021/05/08/cyber-attack-colonial-pipeline/.

[9] Jeong-Han Yun et al. "Burst-based anomaly detection on the DNP3 protocol". In: *International Journal of Control and Automation* 6.2 (2013), pp. 313–324.

[10] Celine Irvene et al. "If I Knew Then What I Know Now: On Reevaluating DNP3 Security using Power Substation Traffic". In: *Proceedings of the Fifth Annual Industrial Control System Security (ICSS) Workshop*. 2019, pp. 48–59.

[11] Niv Goldenberg and Avishai Wool. "Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems". In: *international journal of critical infrastructure protection* 6.2 (2013), pp. 63–75.

[12] James Halvorsen and Julian L Rrushi. "Target Discovery Differentials for 0-Knowledge Detection of ICS Malware". In: *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing*. IEEE. 2017, pp. 542–549.

[13] Basem Al-Madani, Ahmad Shawahna, and Mohammad Qureshi. "Anomaly detection for industrial control networks using machine learning with the help from the inter-arrival curves". In: *arXiv preprint arXiv:1911.05692* (2019).

[14] Nils Svendsen and Stephen Wolthusen. "Modeling and detecting anomalies in SCADA systems". In: *International Conference on Critical Infrastructure Protection*. Springer. 2008, pp. 101–113.

[15] Leandros A Maglaras and Jianmin Jiang. "Intrusion detection in SCADA systems using machine learning techniques". In: *2014 Science and Information Conference*. IEEE. 2014, pp. 626–631.

[16] Barnaby Stewart et al. "A novel intrusion detection mechanism for scada systems which automatically adapts to network topology changes". In: *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems* 4.10 (2017).

[17] Leandros A Maglaras, Jianmin Jiang, and Tiago Cruz. "Integrated OCSVM mechanism for intrusion detection in SCADA systems". In: *Electronics Letters* 50.25 (2014), pp. 1935–1936.

[18] Tiago Cruz et al. "Improving cyber-security awareness on industrial control systems: the CockpitCI approach". In: *Journal of Information Warfare* 13.4 (2014), pp. 27–41.

[19] David Formby et al. "Who's in Control of Your Control System? Device Fingerprinting for Cyber-Physical Systems." In: *NDSS*. 2016.

[20] Hamid Reza Ghaeini et al. "State-aware anomaly detection for industrial control systems". In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. 2018, pp. 1620–1628.

[21] David I Urbina et al. "Limiting the impact of stealthy attacks on industrial control systems". In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, pp. 1092–1105.

[22] Dina Hadziosmanovic et al. "Through the eye of the PLC: towards semantic security monitoring for industrial control systems". In: *Proc. ACSAC*. Vol. 14. Citeseer. 2014, p. 22.

[23] Wissam Aoudi, Mikel Iturbe, and Magnus Almgren. "Truth will out: Departure-based process-level detection of stealthy attacks on control systems". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 817–831.

[24] Istvan Kiss, Bela Genge, and Piroska Haller. "A clustering-based approach to detect cyber attacks in process control systems". In: *2015 IEEE 13th international conference on industrial informatics (INDIN)*. IEEE. 2015, pp. 142–148.

[25] *Next Gen PLC Training*. https://factoryio.com.

[26] *ACTIVSg2000: 2000-bus synthetic grid on footprint of Texas*. https://electricgrids.engr.tamu.edu/electric-grid-test-cases/activsg2000/.

[27] Jonathan Goh et al. "A dataset to support research in the design of secure water treatment systems". In: *International conference on critical information infrastructures security*. Springer. 2016, pp. 88–99.

[28] *Secure Water Treatment (SWaT)*. https://itrust.sutd.edu.sg/itrust-labs-home/itrust-labs_swat/.

[29] Thomas Morris and Wei Gao. "Industrial control system traffic data sets for intrusion detection research". In: *International Conference on Critical Infrastructure Protection*. Springer. 2014, pp. 65–78.

[30] Thomas Morris et al. "A control system testbed to validate critical infrastructure protection concepts". In: *International Journal of Critical Infrastructure Protection* 4.2 (2011), pp. 88–103.

[31] *WinSPS-S7 Programming AND simulation tool for Siemens S7-300-PLCs*. https://www.mhj-tools.com/?page=winsps-s7.

[32] *Dangerous Stuff: Hackers Tried to Poison Water Supply of Florida Town*. https://www.nytimes.com/2021/02/08/us/oldsmar-florida-water-supply-hack.html.

[33] Rockwell Automation. *Converged Plantwide Ethernet (CPwE) Design and Implementation Guide*. 2011.

[34] John Mulder et al. "WeaselBoard: zero-day exploit detection for programmable logic controllers". In: *Sandia report SAND2013-8274, Sandia national laboratories* (2013).

[35] David Formby and Raheem Beyah. "Temporal execution behavior for host anomaly detection in programmable logic controllers". In: *IEEE Transactions on Information Forensics and Security* 15 (2019), pp. 1455–1469.

[36] Chetna Singh, Ashwin Nivangune, and Mrinal Patwardhan. "Function code based vulnerability analysis of DNP3". In: *2016 IEEE International Conference on Advanced Networks and Telecommunications Systems*. IEEE. 2016, pp. 1–6.

[37] Ihab Darwish and Tarek Saadawi. "Attack Detection and Mitigation Techniques in Industrial Control System-Smart Grid DNP3". In: *2018 1st International Conference on Data Intelligence and Security (ICDIS)*. IEEE. 2018, pp. 131–134.

[38] Huan Yang, Liang Cheng, and Mooi Choo Chuah. "Deep-learning-based network intrusion detection for SCADA systems". In: *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2019, pp. 1–7.

[39] Ihab Darwish and Tarek Saadawi. "Attack Detection and Mitigation Techniques in Industrial Control System -Smart Grid DNP3". In: *2018 1st International Conference on Data Intelligence and Security (ICDIS)*. 2018, pp. 131–134. DOI: 10.1109/ICDIS.2018.00028.

[40] Mohamed Niang et al. "Formal Verification for Validation of PSEEL's PLC Program." In: *ICINCO (1)*. 2017, pp. 567–574.

[41] Jae-Myeong Lee and Sugwon Hong. "Keeping host sanity for security of the SCADA systems". In: *IEEE Access* 8 (2020), pp. 62954–62968.

[42] Benjamin Green, Marina Krotofil, and Ali Abbasi. "On the significance of process comprehension for conducting targeted ICS attacks". In: *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*. 2017, pp. 57–67.

[43] Omar Alrawi et al. "Forecasting Malware Capabilities From Cyber Attack Memory Images". In: *30th USENIX Security Symposium*. 2021.

[44] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. "S2E: A platform for in-vivo multi-path analysis of software systems". In: *Acm Sigplan Notices* 46.3 (2011), pp. 265–278.

[45] Insu Yun et al. "{QSYM}: A practical concolic execution engine tailored for hybrid fuzzing". In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018, pp. 745–761.

[46] Julian Rrushi et al. "A quantitative evaluation of the target selection of havex ics malware plugin". In: *Industrial Control System Security (ICSS) Workshop*. 2015.

[47] Luis Garcia et al. "Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit." In: *NDSS*. 2017.

[48] *GuardLogix Controller Systems*. https://literature.rockwellautomation.com/idc/groups/literature/documents/rm/1756-rm093_-en-p.pdf.

[49] Anhtuan Le, Utz Roedig, and Awais Rashid. "LASARUS: Lightweight Attack Surface Reduction for Legacy Industrial Control Systems". In: June 2017, pp. 36–52. ISBN: 978-3-319-62104-3. DOI: 10.1007/978-3-319-62105-0_3.

[50] Mohsen Salehi and Siavash Bayat-Sarmadi. *PLCDefender: Improving Remote Attestation Techniques for PLCs Using Physical Model*. 2021. DOI: 10.1109/JIOT.2020.3040237.

[51] Zeyu Yang et al. "PLC-Sleuth: Detecting and Localizing {PLC} Intrusions Using Control Invariants". In: *23rd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2020)*. 2020, pp. 333–348.

[52] Chuadhry Mujeeb Ahmed et al. "Noiseprint: Attack detection using sensor and process noise fingerprint in cyber physical systems". In: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. 2018, pp. 483–497.

[53] Long Cheng, Ke Tian, and Danfeng Yao. "Orpheus: Enforcing cyber-physical execution semantics to defend against data-oriented attacks". In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. 2017, pp. 315–326.

[54] *Prevent intrusion and maintain network integrity with Data Diodes*. https://advenica.com/en/cds/data-diodes.

[55] Antoine Lemay, José Fernandez, and Scott Knight. "An isolated virtual cluster for SCADA network security research". In: *1st International Symposium for ICS & SCADA Cyber Security Research 2013 (ICS-CSR 2013) 1*. 2013, pp. 88–96.

[56] Qais Qassim et al. "A survey of SCADA testbed implementation approaches". In: *Indian Journal of Science and Technology* 10.26 (2017), pp. 1–8.

[57] Daniel T Sullivan and Edward J Colbert. *Demonstration of SCADA Virtualization Capability in the US Army Research Laboratory (ARL)/Sustaining Base Network Assurance Branch (SBNAB) SCADA Hardware Testbed*. Tech. rep. RAYTHEON TECHNICAL SERVICES CO LLC DULLES VA, 2015.

[58] Avik Dayal et al. "VSCADA: A reconfigurable virtual SCADA test-bed for simulating power utility control center operations". In: *2015 IEEE Power & Energy Society General Meeting*. IEEE. 2015, pp. 1–5.

[59] Igor Nai Fovino et al. "Modbus/DNP3 state-based intrusion detection system". In: *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. IEEE. 2010, pp. 729–736.

[60] *VTSCADA*. https://www.vtscada.com/help/Content/Welcome.htm?tocpath=Welcome.

[61] Haiquan Xiong et al. "Libvmi: a library for bridging the semantic gap between guest OS and VMM". In: *2012 IEEE 12th International Conference on Computer and Information Technology*. IEEE. 2012, pp. 549–556.

[62] Tamas K Lengyel et al. "Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system". In: *Proceedings of the 30th Annual Computer Security Applications Conference*. 2014, pp. 386–395.

[63] Saleh Soltan, Prateek Mittal, and H Vincent Poor. "BlackIoT: IoT botnet of high wattage devices can disrupt the power grid". In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018, pp. 15–32.

[64] Seyedhamed Ghavamnia et al. "Temporal system call specialization for attack surface reduction". In: *29th Security Symposium ({USENIX} Security 20)*. 2020, pp. 1749–1766.

[65] *ATTACK for Industrial Control Systems*. https://collaborate.mitre.org/attackics/index.php/Main_Page.

[66] *MYSCADA SCADA Automation and HMI Solutions*. https://www.myscada.org/en/.

[67] *Simulation and Model Based Design*. https://www.mathworks.com/products/simulink.html.

[68] *The Visual Approach to Electric Power System*. https://www.powerworld.com.

[69] *ICSSPLOIT*. https://github.com/dark-lbp/isf/tree/master/icssploit.

[70] *Metasploit Modules for SCADA-related Vulnerabilities*. https://scadahacker.com/resources/msf-scada.html.

[71] *Using Nt and Zw Versions of the Native System Services Routines*. https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/using-nt-and-zw-versions-of-the-native-system-services-routines.

[72] Mark Ermolov and Artem Shishkin. *Microsoft Windows 8.1 Kernel Patch Protection Analysis*.

[73] Mark Hall et al. "The WEKA data mining software: an update". In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18.

[74] Andrea Bittau et al. "Hacking blind". In: *2014 IEEE Symposium on Security and Privacy*. IEEE. 2014, pp. 227–242.

[75] Lennart Ljung. "System identification". In: *Signal analysis and prediction*. Springer, 1998, pp. 163–173.

[76] Yong Wang et al. "Srid: State relation based intrusion detection for false data injection attacks in scada". In: *European symposium on research in computer security*. Springer. 2014, pp. 401–418.

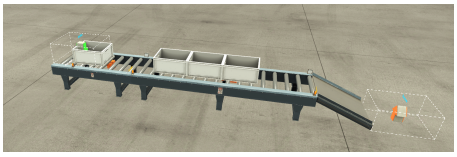
[77] Samuel East et al. "A Taxonomy of Attacks on the DNP3 Protocol". In: *International Conference on Critical Infrastructure Protection*. Springer. 2009, pp. 67–81.

[78] Nicholas Rodofile, Kenneth Radke, and Ernest Foo. "Real-time and interactive attacks on DNP3 critical infrastructure using Scapy". In: *Proceedings of the 13th Australasian Information Security Conference (AISC 2015)[Conferences in Research and Practice in Information Technology (CRPIT), Volume161]*. Australian Computer Society Inc. 2015, pp. 67–70.

[79] Alfonso Valdes and Steven Cheung. "Communication pattern anomaly detection in process control systems". In: *2009 IEEE Conference on Technologies for Homeland Security*. IEEE. 2009, pp. 22–29.

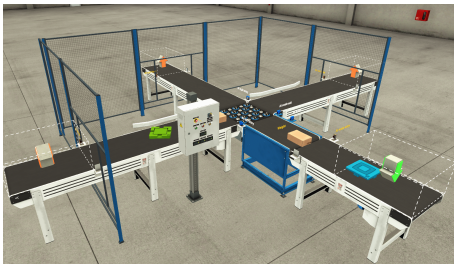
XI. Additional Technical Material

(a) HMI for Elevator (Advanced) Scene

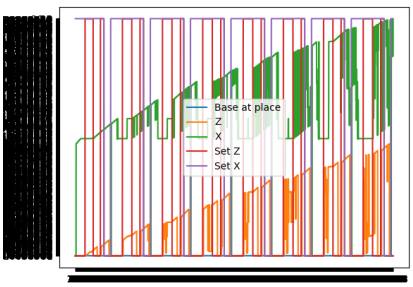


(b) HMI for Queue Processor Scene

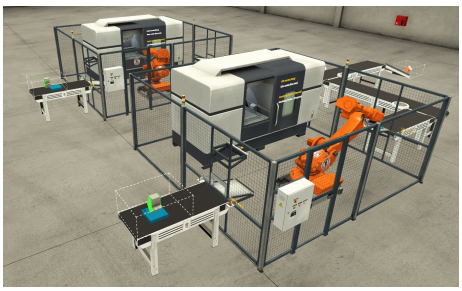
(c) HMI for Converge Station Scene



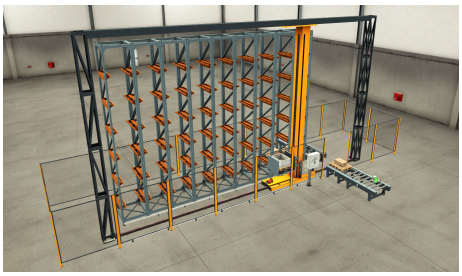
(d) HMI for Sorting Station Scene



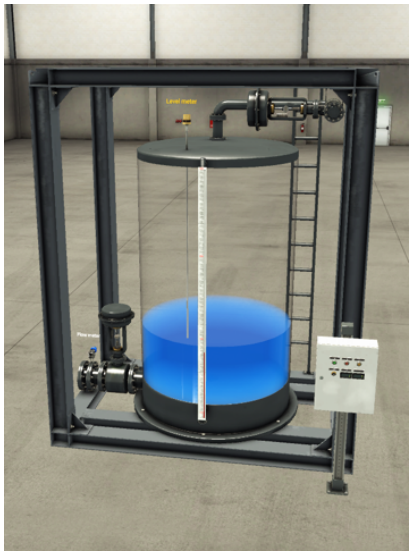
(e) Graphical HMI for AssemblerArm Positioning



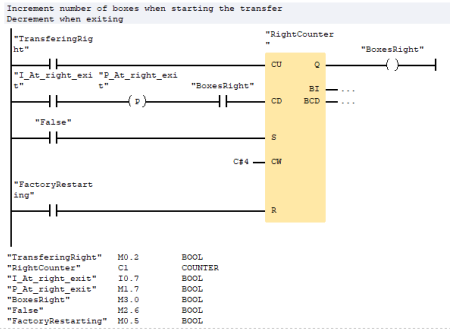
(f) HMI for Production Line Scene



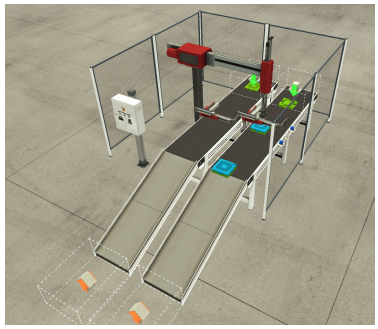
(g) HMI for Automated Warehouse Scene



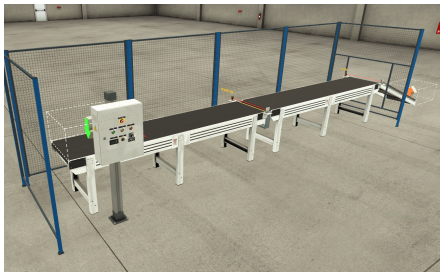
(h) HMI for Chemical Dosing Scene



(i) 2-wire or Logic Schema diagram: Ladder logic for the Sorting Station

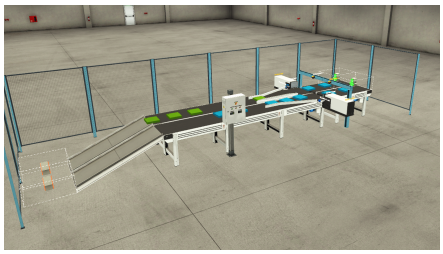


(j) HMI for Assembler Scene



(k) HMI for Buffer Station Scene

(l) HMI for Palletizer Scene



(m) HMI for Separating Station Scene

...	Symbol	Address	Type
1	LeftCounter	C 1	COUNTER
2	MiddleCounter	C 2	COUNTER
3	RightCounter	C 3	COUNTER
4	At_Exit	I 0.0	BOOL
5	I_Factory_Running	I 0.6	BOOL
6	At_Entry	I 0.7	BOOL
7	FactoryRestarting	M 0.0	BOOL
8	Transit	M 0.1	BOOL
9	Entry_Conveyor	Q 0.0	BOOL
10	ConveyorBelt_1	Q 0.1	BOOL
11	ConveyorExtended_1	Q 0.2	BOOL
12	ConveyorBelt_2	Q 0.3	BOOL
13	ConveyorExtended_2	Q 0.4	BOOL
14	ConveyorBelt_3	Q 0.5	BOOL
15	ConveyorExtended_3	Q 0.6	BOOL

(n) Sorting Station | PLC Memory Map

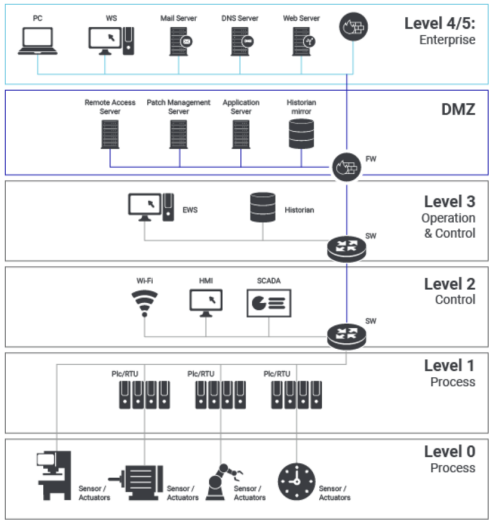


Fig. 10: The Purdue ICS Model

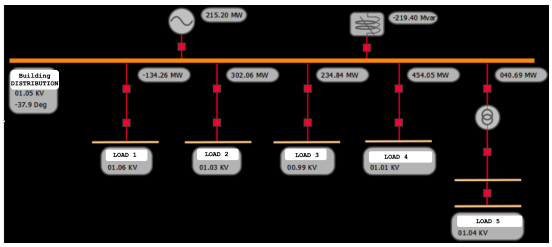
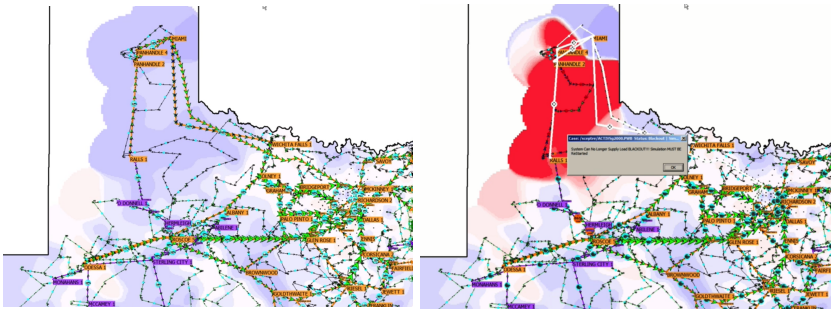


Fig. 11: Power ICS Scenario: Showing Load Lines Elements



(a) Normal (b) Attacked

Fig. 12: Adapted Texas Pan Handle Power Grid: Showing Visualization of Normal and Disrupted Load Lines

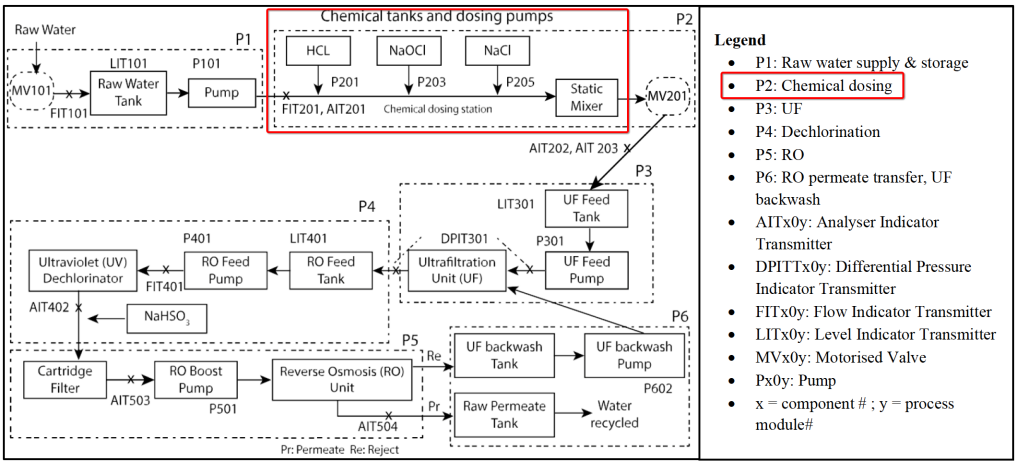


Fig. 13: Complete water treatment plant based on [27, 28]: Showing the chemical dosing operation that was attacked in the Florida water poisoning attack