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

Moses Ike†‡, Kandy Phan†, Keaton Sadoski†, Romuald Valme‡, Wenke Lee†
†Georgia Institute of Technology, ‡Sandia National Laboratories
Email: {mike, kphan, dksados, rvalme}@sandia.gov, mosesjike@gatech.edu, wenke@cc.gatech.edu

Abstract—Modern Industrial Control Systems (ICS) attacks evade existing tools by using knowledge of ICS processes 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 by leveraging the unique execution phases of SCADA to identify the limited set of legitimate behaviors to control the physical world in different phases, which differentiates from attacker's activities. For example, it is typical for SCADA to setup ICS device objects during initialization, but anomalous during process-control. To extract unique behaviors of SCADA execution phases, SCAPHY first leverages open ICS 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 SCADA process-control execution is induced to reveal API call behaviors unique to legitimate process-control phases. Using this established behavior, SCAPHY selectively monitors attacker's physical world-targeted activities that violates legitimate process-control behaviors. We evaluated SCAPHY at a U.S. national lab ICS testbed environment. Using diverse ICS deployment scenarios and attacks across 4 ICS industries, SCAPHY achieved 95% accuracy & 3.5% false positives (FP), compared to 47.5% accuracy and 25% FP of existing work. We analyze SCAPHY's resilience to futuristic attacks where attacker knows our approach.

I. INTRODUCTION

Unlike Information Technology (IT) attacks, Industrial Control System (ICS) attacks cause physical damages to life-dependent physical processes such as power and water supply. ICS processes are controlled by Supervisory Control and Data Acquisition (SCADA) hosts, which run special programs to control the physical world [1–3]. Modern ICS attacks [4–7] are launched from SCADA, where attackers utilize legitimate ICS resources to blend their activities with benign SCADA operations and send malicious signal to disrupt processes.

To detect ICS attacks, statistical analysis of ICS traffic [3, 8–15] are effective against noisy behaviors (e.g., network scans and malformed protocols), but are evaded by modern attacks which use legitimate protocols and knowledge of ICS parameters to cause targeted (not noisy) disruptions [1, 2, 5]. In addition, physical models monitor sensors to know when observed physical states deviate from expected, by fitting historical sensor data into linear models [16–18]. However, [19, 20] show that in practice, such models may require experts to build, and a detailed process model may be unavailable. Further, physical models trigger false alarms when deployed in production due to noise and configuration changes, such that benign states appear outside the model [17, 19]. In general, existing ICS tools are evaded by modern attacks and prone to false alarms due to analyzing traffic/sensor data in isolation, and therefore they cannot tie their analysis to attack-execution context in SCADA. Detecting ICS attacks in SCADA is hard because attackers use the same API call behavior as benign SCADA programs. For example, Industroyer malware, which shutdown Ukraine power grid [1, 5], performed malicious actions that are part of normal SCADA activity such as accessing ICS device objects. Similarly, 2021 Florida water poisoning attack [4] used normal Human Machine Interface (HMI) commands to dump toxic chemicals into the water supply. Hence, existing host agents that looks for "non-SCADA" APIs will not detect them.

We found that while these attack behaviors are normal SCADA activities, they are anomalous when performed in atypical execution phases in SCADA. Therefore, in this work, instead of treating SCADA as one monolithic execution, we specialize 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 process-control but needed for the attack. For example, to hijack ICS device handles, Industroyer executed Registry Setup APIs in process-control, which is typical for initialization, hence anomalous. After infecting SCADA, attackers must "setup/connect" their tools to attack the physical world. These attack behaviors do not align with SCADA's phase-based behavior and leads to atypical APIs in wrong phases. Hence if we identify the limited set of legitimate process-control behaviors, we can selectively monitor and detect attacker's physical-targeted activities that violate them.

Further, because SCADA responds to physical world changes, we can induce SCADA to reveal process-control behaviors by stimulating relevant changes. However, this requires a physical model of ICS processes and their elements’ states. Interestingly, we found that we can leverage ICS open platform communications (OPC) conventions [21–23] to characterize processes via ICS element configurations. 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 execution phase-specific behaviors with physical world impact. SCAPHY identifies the limited set of API calls unique to each SCADA execution phase, which differentiates from attacker's activities in these phases, allowing SCAPHY to detect them. To characterize the "required" internal host channels SCADA uses to control the physical world, we introduce a new reference model, SCADA
Software Stack ($S^3$), which SCAPHY leverages to selectively monitor steps along attacker’s physical world-bound activities in each execution phases. Through $S^3$, SCAPHY can detect attacks that circumvent proper $S^3$ layers (e.g., SCADA rootkits) but sends disruptive control signals to the physical world.

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 [24] 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 behaviors.

SCAPHY detects modern ICS attacks that are missed by existing tools. By correlating behaviors in SCADA and physical, SCAPHY provides contextual alerts for ICS operators to respond to attacks at both SCADA and physical plant. Via PHYSICS constraints, SCAPHY limits the operations an attacker can execute to disrupt a physical process by detecting attacker’s anomalous API behavior in atypical execution phases. SCAPHY’s physical model detects when control signals cause a physical process to have inconsistent state or driven outside its setpoint ranges. SCAPHY’s use of open OPC convention makes it device agnostic and not based on any device/controller, making it work for any OPC-supported ICS deployment. We evaluated SCAPHY at a U.S. national lab state-of-the-art ICS testbed. We launched 40 attacks on 24 diverse ICS scenarios across 4 industries, including an open-source Texas Pan Handle power grid [25]. SCAPHY detected 95% of all attacks with only 3.5% false positives, including real world ICS malware attacks. We make the following contributions:

1. We propose a hybrid technique to detect ICS attacks by correlating SCADA behaviors with physical world effects.
2. We present an ICS physical model via OPC conventions to both identify disruptive physical states and extract legitimate process-control phase behaviors in SCADA.
3. We introduce a new reference model in ICS, SCADA Software Stack ($S^3$), to characterize internal software & hardware layers of SCADA operation. Through $S^3$, host agents can selectively monitor $S^3$ layers to detect 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 [3, 16].
5. Due to limited resources and datasets for diverse ICS security research [26–29], we make available diverse ICS experiment scenarios and datasets derived from them, both at the physical sensor layer and from the SCADA components. These experiments can be run in the FactoryIoT ICS Engine [24] and Siemens Step7-based development suite, WinSIPS [30].

---

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 [4, 31].

Industroyer Power Grid Attack. Industroyer shutdown a Ukrainian 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 [1, 5].

B. ICS/SCADA Operations Background

Fig 1 describes ICS operation based on Purdue model [32]. SCADA hosts at Purdue Level 2 control physical processes, which run at Level 0 and 1 via ICS elements (comprising of actuators, sensors, and parameters in programmable logic controllers or PLC). SCADA constantly monitor running processes and when change is needed, they execute physical-domain logic such as proportional integral derivative (PID) to compute a control variable (CV) to modify element’s states and effect the change (Fig 1). For example, the level control process in the FL water plant controls chemical level in a tank via a PID logic that control how much fluid enters and leaves the tank [29]. 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. Attacker gain access to SCADA via IT means such as phishing. Then he can leverage ICS resources such as OPC, HMIs, and SCADA channels to "address" and send malicious signals to ICS elements.

OPC is a functional part of SCADA and used for interoperability to exchange data about plant information in standardized-mentioned SCADA using OPC [21–23]. OPC is used in widely deployed ICS platforms such as Siemens Tia Portal & Siceider OASYS. For
example, Industroyer malware accessed the Ukrainian power grid’s OPC server to extract the circuit breaker’s OPC tag to address its attack payload. In our example, this OPC tag is BRK.0.BOOL, which specifies element BRK id=0 with BOOL parameter — a heuristic about its possible states: 0 or 1 [33].

**Limitations of Existing Work.** Existing work to detect ICS attacks focus on analyzing traffic and/or sensor data in isolation, without SCADA execution context, which limits their ability to tie their analysis to SCADA for better accuracy and less false alarms. PLC techniques [34, 35] to detect malicious ladder logic on PLCs do not analyze SCADA control signals as in ladder logic but forwards signals to device. Lee [36] only monitors for host DLL injection, which may not happen in ICS attacks. Table I is a taxonomy of leading recent works in ICS attack detection, categorized by their technique and data point. We highlight their limitations such as not having ICS diversity in their evaluation and testing in-the-wild (adapted) attacks. We found that this is due to limited resources to emulate end-to-end and diverse ICS security research [26–29].

**ICS Traffic** approaches [8, 10, 11, 37] analyze abnormal traffic flows and protocols. However, modern attacks such as Industroyer and FL water plant attacks use legitimate ICS protocols to emit traffic flows that blends with benign traffic. Traffic timing analysis [3, 9, 38] are effective for analyzing round trip time delays and inter-arrival times but are only effective against attack behaviors that are chatty [9] such as network scans, but not modern attacks which are targeted.

**Physical Models** detects deviation from expected physical behavior by training sensor data using linear and auto regressive models [16–18]. However, [19, 20] that show in practice, such models may require experts to build, and a detailed process model may be unavailable. Further they trigger false alarms in production due to noise and configuration changes such that benign states are outside the model. Reinforcement and Deep Learning [39–42] which uses game-theory to learn normal and attack behaviors, requires a high-interaction environment, known attacks, and expert reward function, which may limit its use in diverse ICS practice. Process-aware approaches [43–47] focus on specific physical functions, which reduces ambiguity in detection, but requires experts to specify process-specific violations (e.g., [43, 45] uses BRO rules for safety thresholds).

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 [5, 33]. Due to complex safety constraints in physical tasks, SCADA runs many proprietary and physical domain-specific programs such as ICS drivers and hardware tools. As such, unlike IT programs, using existing concolic analysis tools [48–50] will be intractable due to hardware-constrained code paths and environment need [48, 50]. Due to legacy and third-party components, code-signing cannot be strictly enforced to whitelist programs, enabling attackers to modify benign programs. For example, Industroyer executed custom API’s, and Stuxnet [6] injected into Siemens programs.

Our Insight. We found that the nature of physical attacks, which occur in cycles of repeated steps, requires SCADA to exhibit two distinct execution phases: initialization and process-control. Process-control comprise of process-monitoring and process-altering sub-phases. After infecting SCADA, attackers must “setup” and “connect” their tools to attack the physical world. These attack activities do not align with SCADA phase-specific operation, hence results in API calls executed in inappropriate SCADA phases. For example, to access target device tags, Industroyer made OPC calls in the process-monitoring phase, but OPC calls are used in initialization. To hijack SCADA physical channels (from benign program), Industroyer and Havex malware [51] created ICS device objects while in the process-altering phase, but that was a process-monitoring behavior, hence indicative of an attack. Therefore, SCAPHY deems API calls as “anomalous” when executed in atypical SCADA execution phase.

**SCADA Execution Phases.** SCADA starts with initialization which is performed once to setup environment such as loading ICS drivers. After initialization, process-monitoring starts. It involves updating process states in memory and operator HMIs. When physical change is needed, process-monitoring transitions to process-altering to perform the change and return until change is needed again. Process-altering invokes physical-domain logic on a setpoint variable SV to output a CV which is sent to the process (Fig 1). Fig 2 shows the unique

| Technique | ICS Traffic | PLC Logic | Physical Models | Has SCADA Context |
|-----------|-------------|-----------|----------------|-------------------|
| Traffic Analysis | Protocol fields | Time analysis | | |
| PLC Logic | Logic execution | Logic verification | | |
| Physical State | State Deviation | Physical Impact | | |
| RL Models | Online POMDP | Reward weights | Multi-agent game | |
| Process-Aware | Predetermined rules | Power-flow specs | Power prediction | |
| SCADA Context | DLL inject-based | PHYSICS constr. | Evaluation | |
| In-the-wild attacks | ICS diversity | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 4 |

**TABLE I:** Taxonomy of Recent Related works in ICS Attack detection
execution patterns of process monitoring and process-altering phases of a water treatment operation based on Siemens S7 WinSPS SCADA platform [30]. This pattern is based on API call stack behavior, which identifies these SCADA phases.

To identify behaviors unique to each process-control phase, SCAPHY first leverages domain-knowledge in OPC to build a physical model of ICS processes, which enables it to detect physical anomalies of process state changes such as when a process has inconsistent state. SCAPHY 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 behaviors. Through the API calls, SCAPHY establishes a PHYSical world Impact Call Specialization (PHYSICS) constraints, which attackers must violate (i.e., execute atypical APIs) to attack the physical world. Further, SCAPHY develops a new reference model, SCADA Software Stack (S3) to characterize the "required" internal SCADA channels to access the physical based on the execution phases. For example, calling ReadFile indicates process-monitoring and WriteFile indicates process-altering. Both APIs access ICS device objects (e.g., COM Ports) in 3rd layer of S3, to send signals to physical devices. Through S3, SCAPHY can detect attacks that circumvent or bypasses required S3 layers (e.g., SCADA rootkits) but sends disruptive signals to devices.

Resilience Against Evasion in SCADA. To attack the physical world, an attacker must communicate over SCADA's physical interface, through (only) which the physical can be accessed. SCAPHY monitors all network communication in the SCADA host via a hypervisor, which runs at a higher privilege than the SCADA system. SCAPHY enforces that all access to the physical world go through the expected physical world-bound API calls by monitoring accesses to the S3 layers at runtime from the hypervisor using virtual machine introspection (VMI). If an attacker bypasses these expected APIs and sends signals to the physical world, SCAPHY will see that the emitted signal does not have the expected provenance (i.e., no matching API call occurred), and consequently detect the attacker’s influence on the system.

III. THREAT MODEL AND ASSUMPTIONS

We assume a threat model similar to existing work for SCADA-originated ICS attacks [3, 9, 15, 16], whereby attacker has infected SCADA and can use available tools to send attack signals to devices. We developed SCAPHY for the Windows-based SCADA systems, which has unmatched dominance in ICS. We make the following practical assumptions: We do not consider attacks that do not originate from SCADA such as side-channel [55, 56]. Majority of in-the-wild ICS attacks are SCADA-originated [57]. Notional malware that originates runs only on PLC such as in [58] are rarely seen in the wild due to attacker’s cost of developing reusable malware for non-traditional CPUs [2, 59]. We note that PLC Man-In-The-Middle (MITM) has been addressed by existing work [60–62] and in practice via non-PLC diode gateways [63], and hence is outside the scope of this work. SCAPHY assumes the hypervisor as Trusted Computing Base (TCB) that cannot be compromised. In the hypervisor, SCAPHY sees all physical-bound API calls on SCADA virtual machine (VM). Further, SCAPHY relies on the Windows Kernel Patch Protection (KPP) to detect when rootkits inject into the ICS Device Stack (2nd layer of S3) to MITM legitimate SCADA programs. KPP ensures that third-party kernel drivers (e.g., rootkits) cannot modify the kernel subsystem, which mediates hardware-access (i.e., rootkits cannot hide hardware-bound API calls). Relying on KPP is practical because it is widely used in Windows.

IV. SCAPHY APPROACH

Input and Output. SCAPHY’s takes as input, an ICS scenario’s OPC element data and function block diagram (FBD) and outputs (i) a physical model and (ii) PHYSICS constraints. The physical model assigns an impact score to each element state in a process, based on how the state impacts the process output. SCAPHY uses impact scores to detect signals that are disruptive to a process. PHYSICS constraints are a set of API calls of legitimate process-control phase 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 runs in hypervisors of SCADA VMs. Specifically, we deployed SCAPHY in Dom 0 of Xen hypervisor. This approach is practical because ICS plants are increasingly adopting virtualization to provide redundancy in SCADA such as in Enel power plants [64] and recent surveys [65, 66]. For example, Nozomi Networks, an ICS leader, provides several security solutions (e.g., Vantage and Guardian) that relies on virtualized end points [67–69]. In Xen, SCAPHY leverages LibVMI to trace API calls in SCADA VMs. At the network interface, SCAPHY monitor for control signals that changes element states. ICS protocols such as Modbus and DNP3 specifies function code for control signals (e.g., DNP3 0x02/Modbus 0x05). Many traffic analyzers exist to identify function codes in ICS protocols [10, 11, 70].

A. End-to-End Operation

SCAPHY works in four phases as shown in Fig 3. 1) SCAPHY parses ICS scenario files to analyze and partition ICS elements.
into terminal and non-terminal sets based on heuristics from OPC. Terminal elements identifies processes. SCAPHY then traces each element’s connections to map dependent element to their process. SCAPHY then loads the scenario in an ICS engine [24, 30] and performs a physical process-aware dynamic analysis, whereby the engine is induced to execute code paths of process-control operations by iteratively switching element states. During this, SCAPHY records the API calls executed during process-altering and process-monitoring phases, to establish PHYSical Impact Call Specialization (PHYSICS) constraints. Further, the change in process output caused by each state switching is averaged over several scan cycles to derive an impact score for each state relative to others. For states with oscillating impact (i.e., process output may increase or decrease), SCAPHY derives a setpoint range, which defines a minimum and maximum impact score for the process. SCAPHY raises alarms when executed APIs in each process-control phase are not in the PHYSICS constraints, when control signals causes inconsistent state or outside setpoint anomalies, and when missing, extraneous, and out-of-order signals are seen. SCAPHY then computes an anomaly score.

B. Detecting Industroyer Attack Behavior with SCAPHY

Industroyer shutdown Ukrainian power grid by opening circuit breakers connected to load lines [1]. This attack disrupted load balancing of power demand & supply, a weakness in power systems, leading to outages [71]. To replicate the attack, we adapted an open-source Texas Pan Handle grid [25] setup in PowerWorld [72] at a U.S. National Lab. Our lab setup is detailed in Section VII. We ran Industroyer in its “intended” environment; a SCADA host with COM ports and OPC, connected to IEC 608070 device with simulated circuit breaker RTUs. SCAPHY raised detection alarm in under 9 seconds for a PHYSICS injection violation and an inconsistent state anomaly.

Industroyer PHYSICS Violation. Industroyer made several LoadLibrary calls, although a normal SCADA API, was performed after the process-altering phase indicated by a prior CreateFile(Write) call. We found that LoadLibrary is normal for process-monitoring and initialization but not process-altering, per the PHYSICS constraints established from the power scenario. We found that Industroyer used LoadLibrary to load OPCClientDemo.dll to gain OPC capability. Thereafter, Industroyer transitioned to process-monitoring indicated by a ReadFile call. It then invoked IPOCBrowseServerAddressSpace OPC call to extract circuit breaker Information Object Address (IOA tag) to send its payload as shown in Fig 4c. OPC calls are typical of initialization, but not process-monitoring. In addition, Industroyer created new OPC devices 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 8 IEC 60870 signals as shown in Fig 4a; two IEC 60870 Application Protocol Control Information (ACPI) START frames to begin communication, two ACPI TEST frames to check controller status, and one Application Service Data Unit (ASDU) payload sent to the circuit breaker RTU. Industroyer also issued 3 last TEST signals to verify controller’s post attack status as shown in Fig 4d. 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 Fig 4d. SCAPHY output (Fig 4f) show that Industroyer issued a control signal to BRK.0 (i.e., indicated by SIGNAL*), with none for other elements (i.e., missing). However, BRK.0’s new open state has an opposing impact vector to load lines’ OFF state per SCAPHY physical model, allowing SCAPHY to detect an inconsistent state anomaly (detailed in Section 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 identify disruptive physical impact of control signals. To achieve (I), we leverage the process dependency mappings in (II) to inform a physical process-aware dynamic analysis. To achieve (II), we leverage ICS OPC domain knowledge and impact scores derived during (I) to build a process dependency and impact graph (PDIG).

A. Automated Physical Process Comprehension

OPC Tag Analysis. SCAPHY parse OPC tags of ICS scenarios to discover ICS elements and their parameters. OPC naming conventions allows SCAPHY to extract element’s possible states, which allows SCAPHY to automatically switch element’s states during impact modelling. For example, the extracted OPC tag in Industroyer example BRK.0.BOOL=1 specifies element BRK, ID of 0, and boolean parameter, which tells SCAPHY to switch its state 1→0, as opposed to an integer.

ICS Element Partitioning and Process Identification. A process comprises of elements that work together to achieve a physical goal specified by the plant. This goal is determined by the state of a terminal element (a heuristic SCAPHY uses to identify unique processes). For example, the Tank in the Level Control (LC) process of the FL attack example is
the terminal element because the LC goal depends on the Tank’s level. This level is monitored by a level Meter sensor. When Tank’s level reaches the plant’s specified SV for LC, the process concludes. OPC captures this information via OPC Alarms&Event, a data structure that specifies monitored process parameters [73, 74]. To identify terminal elements, SCAPHY analyzes Alarms&Event and extracts process parameter tuples, which corresponds to a sensor and terminal element pair, e.g., the Meter and Tank. SCAPHY then partitions ICS elements into two sets; terminal and non-terminal sets \((E_{Term}, E_{NTerm})\). SCAPHY represents a process in the form of \(P_j = (S_j, E_{Term})\); where \(S_j\) is the process sensor that monitors the terminal element \(E_{Term}\), which corresponds to process \(P_j\)’s output goal. For example, SCAPHY represents LC process as \(LC=\{\text{Meter, Tank}\}\). Other elements in 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:

\[
(E = E_{Term} \cup E_{NTerm})
\]

\[
\{\forall i, j \in (E_{Term}, E_{NTerm}); i \neq j; i \cap j = \emptyset\}
\]

\[
|E_{Term}| = |P|
\]

**Process Dependency Mapping.** After identifying processes, SCAPHY traces element’s connections to identify dependency elements of a process. To do this, SCAPHY converts the ICS scenario’s function block diagram (FBD) into Statement Lists (STL), a textual representation of FBD logic. STL are network-like statements which connects elements via logic arithmetic. An example STL is shown in (Section A). FBD are automatically generated by SCADA programs. From each \(E_{Term}\) node in STL, SCAPHY traces its connection paths until all connected elements are identified. Each process (identified by its terminal element) 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:

\[
DepPath(p) := \{E_{NTerm0}, \ldots, E_{NTermn}\}
\]

\[
PATHS(p) := \{DepPath0(p), \ldots, DepPathn(p)\}
\]

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

\[
\{\forall i \in PATHS(p) : DEP(p) := \bigcup_{j=0}^{n} i(p)\}
\]

\(DEP(P)\) is the union of all elements in the dependency paths of \(P\)’s pathlist. SCAPHY keeps tracks of the internal ordering among these elements as developed from Equation 4. SCAPHY uses this ordering constraints to detect out-of-order signals.

**Functional Inter-Process Relationships** SCAPHY identifies functional relationship between processes with common elements among them. If an 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 \(inter-process transfer points (PTP)\), and \(P_1\) and \(P_2\) as \(PTP sink\) and \(PTP source\) respectively. PTP instances is common among Boolean elements such as valves and switches. SCAPHY leverages PTP to detect attacks spanning multiple processes, such as in the FL water attack (detailed in Section VII-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.

**B. Modelling Process Dependency and Impact**

SCAPHY models how each dependency element state of a process impacts (decrease or increase) the process output using a novel process dependency and impact graph (PDIG) model.

**Sketching PDIG Graph.** PDIG is a set of nodes which represents elements, and edges which connects element nodes based on their relationship. Example PDIG is shown in Section A for the LC process. Undirected \(process edges\) connect \(E_{Term}\) elements to each \(E_{NTerm}\) element in the same \(DEP(P)\). Undirected element edges connect \(E_{NTerm}\) elements without any ordering constraint. Directed element edges connect two \(E_{NTerm}\) elements with ordering constraints among them in the direction of the ordering. Each \(E\) node is annotated with its possible states and impact score (derived later). In the PDIG, SCAPHY 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).

**Deriving Impact to Model Physical Effects.** SCAPHY assigns each element state \(s\) an impact score \(I_c(s)\) based on how they impact a process relative to other states. When a control signal changes an element state, \(I_c(s)\) is used to compute the anomaly score to quantify impact. SCAPHY also uses \(I_c(s)\) to prune out non-impactful states, which reduces the number of elements to monitor. Note that SCAPHY aims to derive “relative” impact score of an element state, not actual score, or score for state combinations. As such, SCAPHY does not consider other elements’ state when deriving \(I_c(s)\), which avoids combinational state explosion [10, 75, 76]. To make the scoring robust/fair, all score derivation starts from the same initial process configuration and is driven until the process reaches steady state. To derive \(I_c(s)\), SCAPHY leverages an ICS environment; Siemens S7 WinSPS [30] and FactoryIO [24] to load and drive the ICS scenarios’ processes. SCAPHY then iteratively switches each state in the process and analyze the moving average of process outputs via reading the process sensor element. SCAPHY stops evaluating the change in output when successive changes become negligible (less than 1%). We normalize \(I_c(s)\) with respect to scan cycles run, 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 \(\tau_n\), which is a set of ordered outcomes \(o\) for a process from scan cycle \(c_j\) to \(c_n\):

\[
\tau_n = \{o(c_j), o(c_{j+1}), o(c_{j+2}), \ldots, o(c_n)\}
\]

where \(c_j\) is the first scan cycle following SCAPHY switching of an elementary state and \(c_n\) is the last or current scan cycle observed where \(0 \leq j \leq n, n \in Z\). SCAPHY keeps track of the highest or lowest recorded outcome \(\psi\) (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:

\[
I_c(s) = \frac{\sum_{i=0}^{n} o(c_j) - o(c_{j-1})}{|\tau_n|}
\]

where \(o(c_j) - o(c_{j-1})\) is change in process outcome, \(ab(\psi -
(a) Process operation scan cycles

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

\[
I_C(C_i) = \frac{1}{\sum_{i=1}^{n} \left| \frac{o(C_i) - o(C_{i-1})}{\psi(o(C_i) - o(C_{i-1}))} \right|}
\]

(b) Computation of Impact Coefficient

\[
I_C(s) = \frac{\delta}{\delta T} \text{Aggregate}_\text{Slice}(o(C_n))
\]

where, for all scan cycles $o(C_i)$ 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 $\frac{o(C_i) - o(C_{i-1})}{\psi(o(C_i) - o(C_{i-1}))}$ 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:

\[
I_C(s) = \frac{\delta}{\delta T} \text{Aggregate}_\text{Slice}(o(C_n))
\]

Fig 5b illustrates a derivation of $I_C(s)$ through scan cycles $C_1$ to $C_n$. 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. Fig 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 $\text{Aggregate}_\text{Slice}(o(C_n))$ function is a straight line (Aggregate_Slice($c_m$) = $mT$) drawn from origin to the point $C_m$ in $T$, where $m$ is the slope. Taking the derivative of $\text{Aggregate}_\text{Slice}(c_m) = mT$ gives the instantaneous $I_C(s)$. Algorithm 1 shows SCAPHY algorithmic approach to derive to $I_C(s)$.

Algorithm 1 DeriveImpactCoefficient($I_C(s)$)

Input: Element State $s$, Process $p$
Output: $I_C(s)$:

```plaintext
Cycle, Cycle_MAX ← GetCycleBatchAndMax
ψ ← GetOutcomeBoundCalib
Op_REV ← GetProcessInitOutcome(p)
SteadyState ← GetSteadyState(p)
Aggregate_Slice ← 0
while Cycle_MAX > 0 do
    SDK_RunSim(s, p)
    $O_j$ ← GetProcessOutcome(p)
    $O_{DIFF}$ ← $O_j$ - $Op_REV$
    Aggregate_Slice ← (Aggregate_Slice + $O_{DIFF}$)
    Aggregate_Slice ← Aggregate_Slice/ABS(Op_REV - ψ)
    if $O_{DIFF}$ < SteadyState then
        Return $I_C(s)$
    else
        Cycle +
        $C_{MAX}$ = −
        ψ ← UpdateOutcomeBound(ψ, $O_j$)
    end if
Return $I_C(s)$
```

C. Characterizing Physical and Signal Anomalies

Inconsistent State. SCAPHY detects when a process has inconsistent state if two dependency element states have opposite impact vectors. To explain, recall that state's $I_C(s)$ in the PDIG drives process output in one direction towards its goal (i.e., not opposing goals). For example, Indroyster attacked LB process aims for a LB factor of 1.0 between power supply and demand. Although LB is affected by other factors, load lines and circuit breaker state play a role. If load decreases/disconnect (e.g., low demand), process-control responds by opening circuit breakers to bring back the balance. Hence, load disconnecting is consistent with breaker opening and drives the process towards its goal. However, opening circuit breakers and connecting load lines are inconsistent and disruptive to LB 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 Indroyster attack based on physical impact of the attack (Section IV).

Outside Setpoint. SCAPHY detects when control signals drive a process to exceed what it is operationally calibrated 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 traffic has incomplete signals based on number of its dependency elements. This is useful for targeted attack signals (ii) Extraneous signals is when control traffic contains signals for elements not associated with the process. (iii) Out-of-Order signals occur signals sequences in control traffic are not in the expected ordered flow based on PDIG.

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

\[
\text{Anomaly Score} = \frac{\sum_{j=1}^{n} I_C(s_j)}{\sum_{j=1}^{n} I_{MAX}(j)}
\]
0.25, 0.26-0.60, and 0.61-1.0 respectfully. These gave the best accuracy for all scenarios tested. However, operators can fine tune these detection thresholds as needed.

D. Analyzing Physical World-Targeted Executions in SCADA

Given an ICS scenario, SCAPHY aims to generate the limited set of API calls unique to legitimate SCADA process-control operations, referred to as PHYSical world Impact Call Special-ization (PHYSICS) constraints. To do this, SCAPHY leverages its physical model to inform a physical process-aware dynamic analysis, whereby a SCADA engine is executed to induce 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 does not replace Purdue Levels [32]. Rather, it breakdown Purdue Level 2 (i.e., control systems) into 5 layers to characterize the internal host layers involved in SCADA process-control, shown in Fig 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.

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. Callback functions invoke CreateFile which returns the device object handle as shown in Fig 6. Parameters lpFileName specifies device object name (e.g., COM1); dwDesiredAccess specifies Read/Write access mode; dwShareMode enables SCADA programs to deny other programs (e.g., malware) access to devices. Unfortunately, attackers (e.g., Havex, Industroyer, Oldrea) subvert this access control by killing SCADA processes to release their handles. For example, Industroyer killed D2MultiCommService.exe to hijack all COM ports to Siemens SPIROTEC device. Havex [51] scanned COM ports to discover connected devices. After obtaining device handles, SCADA programs (and attackers) invoke ReadFile to read device states or WriteFile to send signals to them. Based on this behavior of S^3 layers, SCAPHY can monitor CreateFile, [WriteFile | ReadFile] on device objects as identifiers of process-altering and process-monitoring, respectively.

Identifying Process-Control Phase Windows. Existing work for web servers [77] rely on developer-supplied boundaries to identify phase transitions. This manual approach 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. Recall that accessing device objects (S2) using CreateFile in "Write" mode identifies process-altering, and in "Read" mode identifies process-monitoring, but we need to know when they begin and end (i.e., phase window) to specialize the extracted API calls. Based on their cyclical API call stack behavior such as shown in Fig 2, we found that process-altering follows process-monitoring. We also found that SCADA performs memory freeing operations thereafter to free up memory buffers used in physical-domain computations and then returns to monitoring.

We analyze process-monitoring to know its phase window. Process-Monitoring comprise of two sub-phases; process state representation (reads element states), and event handling, which checks if change is needed, otherwise it repeats as shown in Fig 2. SCAPHY analyzes the changing EIP register and call stack depth of the repeating loop to know the end of event handling the first time it returns to the EIP it started. When event handling ends but EIP returns to a different location and call stack depth, SCAPHY identifies this as the start of process-altering. Finally, SCAPHY performs a check to confirm the expected S^3 identifiers in each phase, which are [CreateFile, ReadFile | WriteFile], CloseHandle call sequences for process-monitoring & process-altering respectfully.

Physical Process-Aware Dynamic Analysis. SCAPHY leverages an ICS emulation engine (FactoryIO [24]) and a SCADA platform (Siemens S7 WinSPS [30]) to perform a physical-process-aware dynamic analysis of process-control behavior to generate PHYSICS constraints. FactoryIO (which also provides a SCADA tool) provides an environment to setup and drive physical processes. Its emulation engine supports hardware-in-the-loop PLCs using real protocols such as Modbus and allows loading of generic ICS scenario FBDs. On starting each process, we monitor the SCADA 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-

---

Fig. 6: SCADA Software Stack (S^3) and SCADA Execution Phases: Showing SCADA Host Interactions with S^3 Layers to access the Physical World
Process-aware PHYSICS Constraints. SCAPHY’s physical model makes generated PHYSICS constraints process-aware, i.e., captures process functions. Although many code paths exist in SCADA, our PHYSICS constraints cover only process-control paths by inducing the benign SCADA process-control logic to react to each state change, ensuring that relevant “state-changing” logic paths are dynamically covered. Because element states are derived from the plant’s deployed OPC, SCAPHY’s PHYSICS constraints succinctly represent the limited legitimate APIs to control the physical.

**PHYSICS Violations by Injection.** SCAPHY detects anomalous APIs not in PHYSICS constraints as injection violations. As such, SCAPHY can detect attack code injected into SCADA programs (as done by Stuxnet [6]) and redirected API calls via Import Address Table hooking. Stuxnet-type attacks will evade existing tools that whitelist benign SCADA programs. However, because SCAPHY focuses on executing APIs, not the *executor*, injected APIs will be detected.

**PHYSICS Violations by Bypass.** Rootkits can bypass $S^3$ layers and directly send malicious signals to physical I/O using kernel calls such as `0x6b DeviceIOControl`. However, because SCAPHY sees all WRITE traffic on the physical interface (SO), 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. IMPLEMENTATION**

We leveraged domain knowledge in OPC convention [21–23] to automatically extract and analyze ICS process information from input ICS scenarios files (OPC element data and FBD).

**Accessing ICS Scenario Files in plants.** We developed an OPC client [78] to perform ReadRequest on OPC server (port 48031) using OPC UA protocol, which specifies a `nodesToRead` field [79] to return elements and parameter in JSON. Further, we read OPC Alarms&Event data from the `UsrServer_event` structure [73, 74] also in JSON. Lastly, we exported FBD’s STL using SCADA software’s provided API.

**Process Identification and Element Tracing.** We parse the OPC Alarm&Event data using python to extract the monitored (i.e., terminal) element and the sensor element. The terminal element is a fitting heuristic to identify processes because it represents a process goal. Next, to identify the OPC tags in the STL and trace element’s connection, we first parsed the STL and matched its tags to tags from OPC. Then, we traced the STL statements from the terminal element tag to other (non-terminal) elements. Next, our python script “partitions” terminal and non-terminal elements in two sets.

**Tracing SCADA APIs with LibVMI.** Our testbed SCADA runs in a Windows Dom U VM in Xen Hypervisor, installed in a bare-metal intel machine, with a Linux Dom 0 VM, where SCAPHY runs. In Dom 0, SCAPHY’s monitoring tool is implemented in C++ (77 lines) which calls LibVMI’s `alt+2m` module to initialize VM introspection on our SCADA VM via `create_view` method. To trace SCADA execution, our C++ code invokes LibVMI’s `SETUP_INTERRUPT_EVENT` to “trap” and forward executed APIs to a SCAPHY’s analysis tool (232 lines of python) via a Linux Pipe. We run SCAPHY as follows: `scaphy_monitor 1 scaphy_analysis.py`

**VII. EVALUATION**

We evaluate SCAPHY’s ability to (i) 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 4 ICS industries to show SCAPHY versatility, as shown in Table II. SCAPHY detected 95% of all attacks, including real ICS malware (Havex2 and Industroyer3), with only 3.5% false positives. Due to lack of resources to support diverse ICS security research [26–29], we make available over 200GB of new ICS experiments and...
attacks in both SCADA and physical aspects, developed for FactoryIO Engine [24] and Siemens S7 platform [30].

**Experimental Setup.** We leveraged a U.S. national lab testbed, which supports fast deployment of ICS topologies, OPC, HMI, and SCADA VMs, prepared with ICS tools to control physical processes such as UART interfaces and Windows Serial Driver. We used 3 SCADA platforms: S7 WinSPPS, MyScada [81], and FactoryIO SDK. This makes our testbed suited to evaluate SCPHY against ICS attacks in realistic settings. To support diverse processes, we leveraged Simulink [82], PowerWorld [72], and FactoryIO Engine to emulate physical processes in Remote Terminal Units (RTUs).

**ICS Attacks Performed.** We performed diverse modern ICS 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 [83] 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 shows our ICS scenarios details and SCPHY’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 a 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 FPs were due to rare element states not parsed correctly from OPC. SCPHY’s physical model allowed it to prune non-impactful elements from the original pool extracted from OPC, which is efficient. E.g., many ICS elements such as repeaters do not have any 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 naive analyzing all states regardless of impact as done in [16].

Further, the reduced element pool contributed to an $I_C(s)$ (impact) average above 70% of their process output, showing that SCPHY modelled the relevant or impactful elements whose malicious state change are more disruptive to the process. SCPHY physical-process-aware dynamic analysis takes about 8 mins, which is reasonable per deployed scenarios sizes (last 3 columns). Our ICS scenarios are adapted from real-world models developed. For example, our power grid scenario was adapted from an open-source Texas Pan Handle power grid [25], simulated in PowerWorld RTUs. We use the power grid example to explain Table II; SCPHY accurately identified the process terminal element ($E_{Term}$) as shown. SCPHY’s impact analysis pruned the OPC element pool of 19 nodes to 10 PDIG nodes, which is 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 attack category I, II, & III, and SCPHY’s results. Attack category I are shown in the first row-group, “Process Altering”. Category II & III are shown in the second-row group, Non-Process Altering. SCPHY detected PHYSICS bypass and inject attacks for both categories. However, SCPHY detected physical and signal anomalies for only category I, because category II & III do not send control signals but block SCADA access to the physical or collect data about devices. For example, in T805, Industrouter issued CreateFile on all COM ports to block our SCADA program from accessing the physical (“as is” mean we executed In-the-wild malware). Similarly, Havex in T888 enumerated all COM device objects stored in Registry keys HKLM\SYSTEM\CurrentControlSet\Services (registry value SERVICE\KERNEL\DRIVER are device driver services) to identify connected devices by issuing loops of QueryKey, OpenKey, QueryValueKey API calls. These API calls were atypical of process-monitoring, allowing SCPHY to detect their PHYSICS inject violations.

SCPHY detected two PHYSICS bypass violations in category I, T872 & T889, which are kernel attacks that bypassed SCPHY monitored $S^3$ layers. Specifically, T889 & T872 used DeviceIoControl direct driver call to send signals out the serial I/O. SCPHY detected the attack because they sent out signals without accessing $S^3$ layers, allowing SCPHY to know that a kernel entity bypassed proper process-altering $S^3$ 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 results. Ground truth was derived from open-source attack data [80, 84]. 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, SCPHY tracked access to each $S^3$ layer based on the API call identifiers for each layer. E.g., ReadFile accesses

---

**TABLE III:** Deployed Attacks and Detection Metrics. We leverage the MITRE ICS Attack Framework [80] to categorize the attack TTPs

| MITRE ICS Attack ID | MITRE ICS Description | Physical Process ID (in Table II) | PHYSICS bypass | PHYSICS inject | Physical Anom. I/T | Signal Anom. I/T | Metrics detect | TP FP Miss Time(s) |
|---------------------|------------------------|----------------------------------|----------------|---------------|------------------|-----------------|----------------|------------------|
| T872                | write host/registers   | Evilkid                          | Yes            | Yes           | Yes              | Yes             | Yes             | 7 0 9.8          |
| T836                | kernel driver attack   | Triton                          | Yes            | Yes           | No               | No              | No              | 4 1 9.4          |
| T889                | device shutdown        | Industrouter                     | No             | No            | No               | No              | No              | 1 0 4.1          |
| T875                | device shutdown        | Industrouter                     | No             | No            | No               | No              | No              | 1 0 4.1          |
| M1.2                | network Enumeration    | Havex circa                      | Yes            | Yes           | Yes              | Yes             | Yes             | 4 1 9.4          |

**IV Experimental Setup.** We leveraged a U.S. national lab testbed, which supports fast deployment of ICS topologies, OPC, HMI, and SCADA VMs, prepared with ICS tools to control physical processes such as UART interfaces and Windows Serial Driver. We used 3 SCADA platforms: S7 WinSPPS, MyScada [81], and FactoryIO SDK. This makes our testbed suited to evaluate SCPHY against ICS attacks in realistic settings. To support diverse processes, we leveraged Simulink [82], PowerWorld [72], and FactoryIO Engine to emulate physical processes in Remote Terminal Units (RTUs).

**ICS Attacks Performed.** We performed diverse modern ICS 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 [83] 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 shows our ICS scenarios details and SCPHY’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 a 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 FPs were due to rare element states not parsed correctly from OPC. SCPHY’s physical model allowed it to prune non-impactful elements from the original pool extracted from OPC, which is efficient. E.g., many ICS elements such as repeaters do not have any 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 naive analyzing all states regardless of impact as done in [16].

Further, the reduced element pool contributed to an $I_C(s)$ (impact) average above 70% of their process output, showing that SCPHY modelled the relevant or impactful elements whose malicious state change are more disruptive to the process. SCPHY physical-process-aware dynamic analysis takes about 8 mins, which is reasonable per deployed scenarios sizes (last 3 columns). Our ICS scenarios are adapted from real-world models developed. For example, our power grid scenario was adapted from an open-source Texas Pan Handle power grid [25], simulated in PowerWorld RTUs. We use the power grid example to explain Table II; SCPHY accurately identified the process terminal element ($E_{Term}$) as shown. SCPHY’s impact analysis pruned the OPC element pool of 19 nodes to 10 PDIG nodes, which is 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 attack category I, II, & III, and SCPHY’s results. Attack category I are shown in the first row-group, “Process Altering”. Category II & III are shown in the second-row group, Non-Process Altering. SCPHY detected PHYSICS bypass and inject attacks for both categories. However, SCPHY detected physical and signal anomalies for only category I, because category II & III do not send control signals but block SCADA access to the physical or collect data about devices. For example, in T805, Industrouter issued CreateFile on all COM ports to block our SCADA program from accessing the physical (“as is” mean we executed In-the-wild malware). Similarly, Havex in T888 enumerated all COM device objects stored in Registry keys HKLM\SYSTEM\CurrentControlSet\Services (registry value SERVICE\KERNEL\DRIVER are device driver services) to identify connected devices by issuing loops of QueryKey, OpenKey, QueryValueKey API calls. These API calls were atypical of process-monitoring, allowing SCPHY to detect their PHYSICS inject violations.

SCPHY detected two PHYSICS bypass violations in category I, T872 & T889, which are kernel attacks that bypassed SCPHY monitored $S^3$ layers. Specifically, T889 & T872 used DeviceIoControl direct driver call to send signals out the serial I/O. SCPHY detected the attack because they sent out signals without accessing $S^3$ layers, allowing SCPHY to know that a kernel entity bypassed proper process-altering $S^3$ 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 results. Ground truth was derived from open-source attack data [80, 84]. 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, SCPHY tracked access to each $S^3$ layer based on the API call identifiers for each layer. E.g., ReadFile accesses...
S2, the ICS device object. We leveraged ICSSPLOIT [83] to compile real-world exploits that were developed as proof-of-concepts exploit code against real bugs in ICS devices. We launched these attacks against their simulated ICS targets and then check which S3 component was accessed in the SCADA host. Table IV details the results. SCAPHy detected all five layers of S3 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 S3, Antivirus companies can develop SCADA-specific host agents to correlate accesses to specific S3 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 and open-source data [26, 27]. The attacker targeted the chemical dosing operation by manipulating a proportional P parameter to raise toxic levels of NaOH in the water outside the setpoint (SV) [4]. Chemical dosing involves two processes: Level control (LC) and Dosing. The HMI is shown in Fig 7a. LC 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 [26, 27]. LC is controlled by a physical domain logic, Proportional Integral Derivative (PID). Because SV cannot be reached in one shot, PID operation involves several "intake" and "discharge" cycles, (shown in Fig 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. Fig 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 via cyber.

**Attack and Detection.** The attacker issued 2 control signals to raise P (dumps excess chemical into TANK.O) and open VALVE.2. We launched the attack using a Modbus payload, but made the attack code self-contained, without triggering any PHYSICS violation. At the SCADA side, the attack triggered all S3 layers to send out signal, which was not malicious by itself. We now focus on the physical aspects. SCAPHy’s detected a high outside set point physical anomaly and a low Extraneous signal anomaly. Fig 7b shows SCAPHy’s outputs.

**Explanation** SCAPHy detection is based on the functional process relationship captured in the physical model between LC and Dosing. Recall Section V-A, VALVE.2 is a PTP element between LC (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)

| Attack ID | Device Targets | Type | SV/T | TP | FP | FN |
|-----------|----------------|------|------|----|----|----|
| 4         | Siemens Simatic S7-300/400 | Unprotected Port | 4 0 1 | 4 | 0 | 1 |
| 5         | Schneider Quantum | I/O corruption | 5 0 1 | 4 | 0 | 1 |
| 4         | QNX INETDD | Buffer Overflow | 4 0 2 | 4 | 0 | 1 |
| 5         | WindRiver VXWorks | Integer Overflow | 5 0 2 | 4 | 0 | 1 |
| 4         | Siemens Simatic S7-300/400 | Input Validation | 4 0 2 | 4 | 0 | 1 |

**Table IV: Deployed Attacks and Detection Metrics.** We leverage Attack Modules ICSSPloit Attack Modules [83] to categorize the attack TTPs.
ICS device object (S2 layer). Unlike naive rootkits T872 & T889, this rootkit exploits ZwCreateFile native API which handles parameters differently in kernel than the userspace CreateFile [88]. Calling ZwCreateFile with the driver name returns the driver base object address (a kernel object) which is traversed to locate ICS device objects in driver data structure. With this object, the rootkit (i) inserts itself on the ICS Device Stack of the physical interface, from which it can send signals, and (ii) invokes ZwWriteFile to complete fake S3 behavior.

**Detection.** Executing the rootkit emitted S2 process-altering APIs with no PHYSICS violations based on ZwCreateFile & ZwWriteFile calls. This evades SCAPHY. However, SCAPHY can detect this rootkit by simply monitoring for ICS driver attachment to the Device Stack, a protected kernel object. **Explanation** For the attack to work, the rootkit had to attach itself on the ICS Device Stack of the physical interface. This is done using IoAttachDeviceToDeviceStack during driver loading as shown in Fig 8, which shows the steps ICS drivers (and rootkit) must follow to attach themselves to a Device Stack. Device Stack is a kernel structure that tells the kernel 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 or unknown drivers are introduced.

We note that rootkits cannot modify the kernel subsystem (NTOSKRNL.EXE) per Windows Kernel Patch Protection (KPP) [89], which prevents third-party code (e.g., drivers) from "patching" the kernel subsystem, which includes Device Stack. KPP prevents unauthorized access of device I/O. Hence ICS drivers (and rootkits) must follow the API steps in Fig 8, which SCAPHY can monitor with high-fidelity. To test this, we added this API behavior to SCAPHY model and ran the rootkit. On observing a repeat of IoAttachDeviceToDeviceStack (i.e., after the benign ICS driver has already been attached), SCAPHY detects this malicious Device Stack tampering. Making this addition to SCAPHY was seamless because the API sequence in Fig 8 follows SCAPHY’s proposed $S^3$ layers, which shows the ubiquity of the $S^3$ reference model. For example, the argument dwServiceType=0x01 in CreateService registers an ICS driver service, which returns a handle used by ICS callback functions ($S^3$) to access device objects ($S^2$).

**Hypervisor-Level DKOM.** This second attack requires the rootkit to escape the VM and launch DKOM in Xen hypervisor (Dom 0 Linux) where SCAPHY runs. Although this is outside our threat model, as the hypervisor is our TCB (Section III), we wanted to show a concrete way to evade SCAPHY. Recall in Section VI that SCAPHY’s LibVMI-based monitoring tool uses a Linux Pipe to feed API traces to SCAPHY analysis tool. This DKOM works by redirecting this Pipe’s memory buffer in kernel to another pipe the rootkit controls. Specifically, our rootkit modified the Page Pointer, PAGE *, the first argument of pipe_buffer, a struc object of pipe_inode_info kernel object, which manages Pipe operations in Linux kernel. Since Linux Pipes are mounted in file-system "pipe://" it was easy to find SCAPHY’s pipe via ls. Then, the rootkit modified SCAPHY’s Pipe’s PAGE * in kernel to a malicious Pipe buffer address created with dcom_write 1 dcom_read, which allowed attacker to control what the SCAPHY analysis sees, without SCAPHY knowing. In our attack, we killed the LibVMI side of SCAPHY after the pipe manipulation for the attack to work.

**D. Comparing SCAPHY with Existing Techniques**

We compared SCAPHY to existing ICS techniques that use physical models [16] and traffic classifier [3]. [16] is based on [17] and analyzes sensor data’s cumulative sum of residuals, and raises alarm if the difference between sensor and expected behavior is higher than a threshold. [3] analyzes spatial-temporal properties of ICS signals such as packet arrival times and size, using a REBTree classifier. [3] raises alarm when traffic features are outside a running average. To do this comparison, we use sensor and traffic data from the experiments in Table II, which we parsed into ARFF format (sample shown in Section A), and make available via this work. We leveraged open-source tools to setup these techniques. For [16], we leveraged Scikit-Learn to generate a linear regressive model to fit the sensors values of the physical elements in the normal running mode. For [3], we leveraged WEKA [90] to generates a REPTree classifier that builds a decision tree using information gain and variance in the extracted traffic fields.

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 to no SCADA context. However, this is where SCAPHY detected most attacks (90%, and 95% overall). We note that diverse ICS experiments (such as ours) may affect the performance of tools designed for specific ICS domains. [16] 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 instance is the FL water attack where a high $P$ variable causes a high but temporary sensor deviation. Although it deviates greatly from the setpoint, it is benign in Level Control (LC), but anomalous in the Dosing process if dosing valve is open. Unlike SCAPHY’s physical model, existing linear models [16–19] do not capture inter-process operations (e.g., between Dosing and LC), hence are prone to false alarms due to high but temporary benign deviations. SCAPHY’s physical model lower TP is due to approximating analog states such as switching 0-10v with only 3 levels 0v,5v,10v, which saves space but is less precise. [3] detected 11 attacks (27.5%), but with low FP (12.3%) because most modern attack is similar to benign.

**TABLE V: Comparison with Existing Techniques**

| Techniques | Approach | Sensor | Traffic | Hybrid | Attack Detection Metrics |
|------------|----------|--------|---------|--------|--------------------------|
|             |          | Normal | SCADA   | Physical | CTRL Signals |
| Sensor [16] | Linear   | 40/146 | -       | 19/37  | 21/11  |
| Analysis   |          |        |         |         |             |
| Traffic [3] | Regression M. | -       | -       | 29/46  | -       |
| Analysis   |          |        |         |         |             |
| Hybrid     | SCADA Corr. | 40/146 | 46/54  | 21/14  | 19/21  |

Note that diverse ICS experiments (such as ours) may affect the performance of tools designed for specific ICS domains. [16] 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 instance is the FL water attack where a high $P$ variable causes a high but temporary sensor deviation. Although it deviates greatly from the setpoint, it is benign in Level Control (LC), but anomalous in the Dosing process if dosing valve is open. Unlike SCAPHY’s physical model, existing linear models [16–19] do not capture inter-process operations (e.g., between Dosing and LC), hence are prone to false alarms due to high but temporary benign deviations. SCAPHY’s physical model lower TP is due to approximating analog states such as switching 0-10v with only 3 levels 0v,5v,10v, which saves space but is less precise. [3] detected 11 attacks (27.5%), but with low FP (12.3%) because most modern attack is similar to benign.
based detection had more FPs because it flagged many benign missing signals, showing that missing signals are not effective, which we can mitigate by raising its detection threshold.

E. Quantifying Manual Work to use SCAPHY in Practice

We quantify the amount of manual work that may be required to apply SCAPHY’s pipeline in practice. For example, although our SCADA platform provides interfaces to automatically export FBDS to STL, other platforms may require manual export via the GUI. In addition, automatically matching and tracing OPC tags in the STL require SCADA and OPC to use the same namespace (e.g., "BRK" in OPC is also "BRK" in STL). This was the case in our testbed and in integrated platforms such as Siemens’ TIA. However, in plants using third-party tools such that STL and OPC have different namespace (e.g., "BRK" v. "BR"), an STL-to-OPC element matching can be achieved semi-automatically by manually seeding a regular expression script to perform the matching. Table VI summaries the time to perform each step and repetitions needed.

| SCAPHY Pipeline Requirement | Where Used | Manual Step | Avg. Time under 1 per scenario | Times repeated under 1 per scenario |
|-----------------------------|-----------|-------------|-------------------------------|-----------------------------------|
| Exporting FBDS to STL JSON  | Element   | Tracing     | Severe deviations in SCADA GUI | 1 per hour                        |
| Matching STL to OPC tags    | Element   | Tracing     | Seed a regular expression with match parameter | 1 per 10 minutes per scenario |
| Setting up SCADA ENV        | Impact    | Derivation  | Combining tested networking & devices | 1 per 1 hour per scenario |

TABLE VI: Quantifying amount of manual work to apply SCAPHY in practice

VIII. DISCUSSIONS: SCAPHY’S PRACTICALITY

Run-time Overhead. SCAPHY leverages LibVMi state-of-the-art VM analysis tool in Xen Baremetal hypervisor, which enables detection of malicious APIs in SCADA VMs around 9 seconds, as shown in Section VII-A. LibVMi achieves near-native speed access of guest VM memory pages when run on Baremetal Hypervisors, and now used in production [91–96]. This makes SCAPHY a practical detection technique. In contrast, existing work based on sensor data [16, 17], must observe several sensor deviations before making a sound decision, which pushes their time-to-detect to the minute range [16, 17].

Real vs. Emulated State Switching. As switching real processes can be dangerous, we used emulated processes. However, the switched states were informed by deployed OPC server in the plant. Real-world gaps exist if poor emulation is used. To reduce this practicality gap, SCAPHY leveraged state-of-the-art Simulink and PowerWorld ICS emulation.

ICS Attack Dev. Difficulty. Unlike IT, developing ICS attacks is hard due to finding their SCADA and physical target. As such, we spent months developing many modern attack scenarios, and tested more attacks than existing work.

Robustness of PHYSICS Constraints. PHYSICS constraints are generated per ICS scenario and SCADA. This is practical because plants rarely update physical domain logic, since they are based on immutable laws of physics.

Limitations and Future Work. SCAPHY cannot detect attacks that originate outside of SCADA such as side channels and device hardware. We will investigate similar execution phase-based analysis these ICS threat models. Further, although SCAPHY can rely on Window’s KPP to detect rootkits evasion, with time new rootkits may bypass KPP. We will investigate SCADA-specific defenses for rootkit such as integrating Call Stack analysis. This shows promise given that SCADA process-control have well-defined Call Stack behavior (Fig 2) which can be robust (but expensive) than API calls.

IX. RELATED WORKS

What differentiates ICS from IT is that physical tasks follow immutable laws of physics [16, 17], which can be learned to build prediction models [97]. Existing work build sensor behavior models [16–19] to predict when behaviors deviate from expected. However, in practice, such models are not always available [19, 20], and raise false alarms due to noise and config changes [19]. Offensive ways such as Harvey [58] can MITM sensors and present “false” data. [60–62] proposed ways to address MITM. [63] uses non-PLC diode gateways to avoid MITM. Reinforcement and Deep Learning [39–42] uses game-theory to learn normal and attack behaviors, but requires a high-interaction system, known attacks, and expert reward function, which may limit its use in diverse ICS practice.

Statistical analysis of ICS traffic [3, 8–15] are effective for noisy and abnormal traffic such as illegal protocols and scans. However, modern attacks evade them using benign protocols and knowledge of parameters to cause specific (not noisy) attacks [1, 2, 5]. Flow-based approaches [8, 10, 11, 37] analyze abnormal function codes/channels, such as shown in [98, 99], but are evade by attacks such as Industroyer, which uses legitimate HMIs. Timing analysis [3, 9, 38] analyze anomalous round trip time delays and inter-arrival times. However, they are effective for signals that are “chatty” [9] such as attacker scans, not modern attacks that are targeted. Lee [36] only monitors for host DLL injection, which may not happen in ICS attacks. In contrast, SCAPHY deems API calls as anomalous when executed in the wrong SCADA execution phase. Side channel defenses such as EMF and power [55, 56] may require close proximity (motherboard-level) to controllers.

Pattern-based analysis [10, 76, 100] detects anomalous signals such as isolated signals but suffers from model’s high sensitivity, which causes false alarms due to slight config changes (e.g., addition of new devices) [10, 76]. State-based tools [76] detects critical states in ICS but requires manual rules, which do not scale. Further, because they analyze all state transitions, can suffer from state explosion when parameters increase. SCAPHY is stateless and only analyze current and impactful element states. Process-aware tools [18, 19, 43–47] analyze sensor data unique to specific process functions, which may reduce ambiguity in detection, but requires experts to specify safety threshold violations per ICS-domain. This results in a tradeoff of being manual or domain specific. For example, [43, 45] uses manual BRO rules (e.g., heat level must not exceed 20). [44] uses expert-software to predict power flows. [46] uses pre-defined power flow rules to analyze deviations in measurements. In contrast, SCAPHY’s process-aware physical model automatically detects physical anomalies by analyzing the disruptive “impact” of control signals. Further, unlike the above techniques, SCAPHY correlates physical “impact” with behaviors in SCADA (i.e., anomalous API calls in atypical execution phase) for contextual “end-to-end” attack detection.
X. CONCLUSION

We present SCAPHY to detect ICS attacks by leveraging unique execution phases of SCADA to identify the limited set of benign behaviors to control the physical world in different phases, which differentiates from attacker’s activities. To do this, SCAPHY first leverages OPC to build a physical model, which enables it to detect physical anomalies. SCAPHY then uses this model to inform a physical process-aware dynamic analysis, whereby SCADA is induced to reveal API calls unique to process-control. Through this, SCAPHY detects attack behaviors that violates process-control phase. SCAPHY achieved high accuracy and outperformed existing work.

XI. ACKNOWLEDGEMENT

We thank the anonymous reviewers for their helpful and informative feedback. We also thank Marshall Daniels for his guidance. This material was supported in part by Sandia National Laboratories, the Office of Naval Research (ONR) under grants N00014-19-1-2179, N00014-17-1-2895, and N00014-18-1-2662, the Defense Advanced Research Projects Agency (DARPA) under contracts HR00112190031 and HR00112190087, Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy National Nuclear Security Administration under contract DE-NA0003525. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

REFERENCES

[1] WIN32/INDUSTROYER: A new threat for industrial control systems. URL: https://www.welivesecurity.com/wp-content/uploads/2017/06/Win32_Industroyer.pdf.
[2] Ruimin Sun et al. “SoK: Attacks on Industrial Control Logic and Formal Verification-Based Defenses”. In: arXiv preprint arXiv:2006.04806 (2020).
[3] Stanislav Ponomarev. Intrusion Detection System of industrial control networks using network telemetry. Louisiana Tech University, 2015.
[4] Oldsmar Water Treatment Facility Cyber Attack. URL: https://www.dragos.com/blog/industry-news/recommendations-following-the-oldsmar-water-treatment-facility-cyber-attack/.
[5] Defense Use Case. “Analysis of the cyber attack on the Ukrainian power grid”, In: Electricity Information Sharing and Analysis Center (E-ISAC) (2016).
[6] W32.Stuxnet Dossier. URL: https://www.wired.com/images_blogs/thr3atlevel/2011/02/Symantec-Stuxnet-Update-Feb-2011.pdf.
[7] Ransomware Attack leads to shutdown of Major U.S. Pipeline System. URL: https://www.washingt onpost.com/business/2021/05/08/cyber-attack-colonial-pipeline/.
[8] 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.
[9] 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.

[10] 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.
[11] James Halvorsen and Julian L Rruhni. “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.
[12] 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).
[13] 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.
[14] 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).
[15] 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.
[16] Hamid Reza Ghaemini 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.
[17] 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.
[18] Dina Hadzisumanovic 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.
[19] Wissam Aoudi, Mikel Ibarue, 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.
[20] Istvan Kiss, Bela Genge, and Pirosha 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.
[21] OPC Foundation, OPC Unified Architecture. URL: https://opcfoundation.org/about/opc-technologies/opc-ua/.
[22] Mai Son and Myeong-Jae Yi. “A study on OPC specifications: Perspective and challenges”. In: International Forum on Strategic Technology 2010. IEEE. 2010, pp. 193–197.
[23] Wolfgang Mahnke. “2.4 OPC Unified Architecture”. In: OPFCalibration Process Automation Systems on Strategic Technology 2010. IEEE. 2010, pp. 193–197.
[24] Next Gen PLC Training. URL: https://factoryio.com.
[25] ACTIVSg2000: 2000-bus synthetic grid on footprint of Texas. URL: https://electricgrids.engr.tamu.edu/electric-grid-test-cases/activsg2000/.
[26] 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.
[27] Secure Water Treatment (SwAT). URL: https://itrust.sutd.edu.sg/itrust-labs-home/itrust-labs_swat/.
[28] 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.
[29] Thomas Morris et al. “A control system testbed to validate critical infrastructure protection concepts”. In: International
Dangerous Stuff: Hackers Tried to Poison Water Supply of WinSPS-S7 Programming AND simulation tool for Siemens Rockwell Automation. 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.

John Mulder et al. “WeaseIBoard: zero-day exploit detection for programmable logic controllers”. In: Sandia report SAND2013-8274, Sandia national laboratories (2013).

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.

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

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.

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.

Mehmet Necip Kurt et al. “Online cyber-attack detection in smart grid: A reinforcement learning approach”. In: IEEE Transactions on Smart Grid 10.5 (2018), pp. 5174–5185.

Kai Zhong et al. “An Efficient Parallel Reinforcement Learning Approach to Cross-Layer Defense Mechanism in Industrial Control Systems”. In: IEEE Transactions on Parallel and Distributed Systems (2021).

Martina Panfili et al. “A game-theoretical approach to cybersecurity of critical infrastructures based on multi-agent reinforcement learning”. In: 2018 26th Mediterranean Conference on Control and Automation (MED). IEEE, 2018, pp. 460–465.

David Wilson et al. “Deep learning-aided cyber-attack detection in power transmission systems”. In: 2018 IEEE Power & Energy Society General Meeting (PESGM). IEEE, 2018, pp. 1–5.

Justyna J Chromik, Anne Remke, and Boudewijn R Haverkort. “Bro in SCADA: dynamic intrusion detection policies based on a system model”. In: 5th International Symposium for ICS & SCADA Cyber Security Research 2018 5. 2018, pp. 112–121.

Hui Lin et al. “Runtime semantic security analysis to detect and mitigate control-related attacks in power grids”. In: IEEE Transactions on Smart Grid 9.1 (2016), pp. 163–178.

Jeyasingam Nivelhan and Mauricio Papa. “A SCADA intrusion detection framework that incorporates process semantics”. In: Proceedings of the 11th Annual Cyber and Information Security Research Conference. 2016, pp. 1–5.

Justyna J Chromik, Anne Remke, and Boudewijn R Haverkort. “What’s under the hood? Improving SCADA security with process awareness”. In: 2016 Joint Workshop on Cyber-Physical Security and Resilience in Smart Grids (CPSR-SG). IEEE, 2016, pp. 1–6.

Helge Janicke et al. “Runtime-monitoring for industrial control systems”. In: Electronics 4.4 (2015), pp. 995–1017.
APPENDIX

//STL-Source - Generated with:
//WinSPS-S7
//Version=V6.05
//
NETWORK
TITLE= Reset
//Detect when Factory I/O starts running, re

STL Logic
Arithmetic
(per line)

A I 1.5;
FP M 2.5;
R M 0.0;
R R M 0.1;
R R M 0.2;
R M 0.3;
R R M 0.4;
R C 1;  // Process Logic evaluates
        // to Element M-0.5, which
        // specifies M-0.5 in the
        // Terminal element
R C 2;

= M 0.5;

Fig. 9: Our Example Statement List (STL) TEXT FILE Exported from WinSPS-S7 Function Block Diagram. Annotated to show the different parts

Fig. 10: Our Example ARFF Header showing fields (Element states) we used for specific element in the Sorting Station Scenario

Fig. 11: Sample ARFF-formatted Element State reporting every 100 scan cycles

Fig. 12: Example PDIG Model for the Level Control Process of the FL Attack Incident

Fig. 13: Complete water treatment plant based on [26, 27]: Showing the chemical dosing operation in reference to the FL water poisoning attack