Architecture and prototype implementation for process-aware intrusion detection in electrical grids

Robert Flosbach[†]
r.flosbach@gmx.de

Justyna Chromik[*]
j.j.chromik@utwente.nl

Anne Remke[†*]
anne.remke@wwu.de

[†]*University of Münster*, Münster, Germany
[*]*University of Twente*, Enschede, the Netherlands

***Abstract*—Supervisory Control and Data Acquisition (SCADA) systems monitor and control electric power distribution. Recent history has shown that cyber-attacks pose a tremendous risk for the economy and safety of modern countries. This paper introduces an architecture and a prototype implementation for a process-aware, network-based Intrusion Detection System (IDS) to secure control networks in the domain of power distribution. Based on a recently proposed process model, the system continuously assesses the local physical process and all control commands with regard to consistency and safety of the underlying physical process. Its detection capabilities focus on process-based attacks like manipulated control commands, which appear legitimate to traditional IDS but might nevertheless have devastating effects on the power distribution. The architecture separates the evaluation part from the traffic processing, which ensures extensibility and scalability. The developed implementation has been successfully tested at a Dutch power distribution substation. Its detection performance is characterized by a very low miss rate and high precision.**

***Index Terms*—SCADA, Intrusion detection, power distribution, Zeek, process-aware**

I. Introduction

In 2015 unknown hackers compromised control networks of multiple Ukrainian electricity distribution companies and were able to shut down the power supply of approximately 225,000 customers for up to 6 hours. Unauthorized commands were sent to open circuit breakers at over fifty medium- and low-voltage distribution stations [1]–[3]. Best practices, such as security policies and identity management, make it harder for *outsiders* to access control networks. Intrusion Detection Systems (IDS) can identify anomolous activities *inside* the control networks and detect cyber-attacks on Industrial Control Systems (ICS). Most modern ICS and SCADA systems rely on well-known and standardized communication protocols, which often operate over TCP/IP.

Given the large-scale distribution of the electrical grid, remote control (e.g., over the Internet) is essential. This provides, however, a vast attack surface for malicious outsiders [4]. In the power grid, a hacker could either physically compromise substations or place malware on workstations within a company. Once in the network, the hacker might be able to intercept and modify, replay or forge commands that have detrimental physical effects on the electrical grid [5].

The main contribution of the paper is the implementation and evaluation of a prototype for process-aware intrusion detection, which can be placed locally at remote terminal units (RTU). We propose a novel and protocol-independent architecture, separating the traffic parsing with Zeek[1] from the knowledge of the process, which is at the heart of process-aware intrusion detection.

The underlying process model for electrical grids (c.f. [6]–[9]) distinguishes between a *static* system topology describing physical properties of the system, and a *variable* system state, which is updated upon sensor readings. The physical process variables are evaluated w.r.t. consistency and safety rules when commands, measurements and configuration changes are sent to the RTU. In contrast to earlier work, the proposed tool is able to automatically match the relevant consistency and safety rules, for a given topology.

The feasibility of the approach was validated with real IEC-104 traffic obtained at a Dutch Distribution System Operator on August 15, 2018. We evaluated the intrusion detection capabilities and practical issues of applicability in distribution stations, as well as potential limitations.

a) Related Work: Many intrusion detection approaches for SCADA systems focus solely on traffic patterns [10]–[18], which are rather regular. Work on anomalies in protocol usage [16], [19]–[24] is able to detect traditional network attacks and more general SCADA-specific attacks like the usage of malicious or normally unused function codes. However, they are not able to detect *semantic attacks*, which refer to malicious commands that are syntactically correct. These attacks can only be detected, if the physical process is taken into consideration, as, e.g., proposed in [25].

Specification-based approaches detect malicious commands, e.g., issued to local protective relays [26] or combine different locations to identify measurement and command anomalies [27]–[29]. [30] executes a power flow analysis on process values to estimate the effect of control commands. These approaches all require process and configuration specifications, while other approaches infer a general process model automatically [25], [31]. Many approaches have not been tested on real SCADA networks but only in (simulation) testbeds,

[1]Bro network monitor was renamed to Zeek in October 2018.

2575-8462/19/$31.00 ©2019 IEEE
DOI 10.1109/SRDS47363.2019.00015

as criticized by [16], [32], [33]. In contrast, we propose a prototype IDS based on [6]–[9], who propose a local and process-aware intrusion detection model for electrical grids, which is evaluated on real traffic.

b) Outline: Section II discusses the overall architecture. Section III evaluates the prototype on a real-life traffic capture and Section IV discusses feasibility and practicality of the proposed approach. Finally, Section V concludes the paper.

II. Architecture

The proposed architecture ensures extensibility by separating the evaluation engine and the packet inspection part. This leads to a protocol independent prototype and allows for easy inclusion of new safety rules and/or new grid components.

Section II-A provides an overview of the architecture. Section II-B describes the implementation in more detail. The connection to the IDS Zeek, the traffic parsing unit and the event engine are addressed in Section II-C.

A. Overview of architecture

The prototype is split into two main components as depicted in Figure 1, which both run as completely separate processes.

The left-hand side of the figure, marked with number 1, shows the electrical grid model and the intrusion detection evaluation, which are written in Python. This component is split into three major subcomponents: the state management, the topology data structures, and the rule evaluation logic. Furthermore, an event engine, which offers an interface to receive process measurements and commands, connects this first component to its input source, e.g., Zeek. Event engine triggers the evaluation of rules on different occasions. Note, that this component works protocol-independent.

The second component, marked with number 2 in Figure 1, is the extension of the IDS Zeek. It encompasses the protocol parser, the protocol-specific events and a component for translating and converting the packets' raw contents into interpreted and converted values from the physical process. Zeek is an open-source, passive network traffic analyzer with

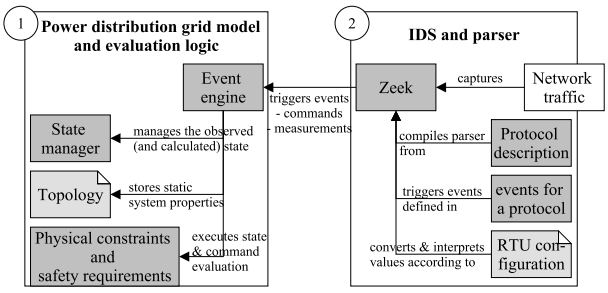

Fig. 1. Overview of the prototype's architecture. Network traffic is the evaluated input, illustrated as white box. Topology and RTU configuration are directly based on inputs from the power operator, and are used to generate rules. They are marked as bright gray boxes.

TABLE I
Physical constraints and safety requirements

id	explanation
P1	Kirchhoff's current law: the incoming current at a bus must equal the outgoing current
P2	all reported voltages at the bus are equal
P3	if a switch is open, then the current on the line equals 0
P4	the current and the voltage equal at the beginning and end of each power line
P5	the power law formula $P = I \cdot V$ holds true for all generators and consumers
P6	transformation ratio at transformers is consistent with measurements for both current and voltage
R1	the current does not exceed the defined safety threshold
R2	the voltage level is within its allowed bounds
R3	all fuses and protective relays are functional (i.e., connected)
R4	the current measured on the power line with a fuse / protective relay does not exceed the cutting current of that device
R5	the secondary voltage value at a transformer does is within the allowed reference voltage bounds
R6	all consumers are connected to power
R7	the total generated power should equal the power consumed in the electrical grid
R8	set points are in a proper interval around the physical threshold values
R9	the systems interlocks (mutually dependent switch states) are not violated

many intrusion detection capabilities. It offers a capable event-based scripting language for user-defined extensions [34]–[36]. Furthermore, Zeek's protocol parser capabilities can be extended with parsers written and compiled in the open source Spicy framework, which makes it the perfect choice for parsing network traffic also for protocols that are not yet supported [37], [38].

B. Electrical grid model and evaluation logic

Component 1 of the prototype strictly separates the system state from the topology representation, which contains the static components of the grid and their connections. While the topology is set up once at the initialization of the IDS, the state is continuously updated every time new measurements are detected in the traffic.

1) Topology representation: Every component of the electrical grid, such as power lines, buses, consumers, transformers, sensors, is implemented as a class with its properties defined as attributes. An example of an attribute of a power line is the maximum allowed current. Specific components are instances of the corresponding class with their values defined according to the information about topology[2]. State-dependent information, such as currently measured value of current or voltage, or other process variables are not saved in the objects themselves but are referenced by so-called tags, which act like a key to retrieve the value from a state object. Those tags are also saved in attributes of the objects.

[2]A class diagram for the data structures that compose a topology can be accessed at https://github.com/jjchromik/RuleGeneratorSCADA/blob/master/img/ClassDiagram.pdf.

The evaluation engine supports all components and properties of the electrical grid model defined in [8]. The rules evaluated by the tool are described in Table I. Classes define a function for every safety and consistency rule that is applicable to that component type. Abstract parent classes may be used if the evaluation applies to multiple component types.

Rule evaluating functions always return *False* if there is a violation and *True* otherwise. The evaluation of a component can be either called isolated (e.g., by `checkR1(state)`) or by functions which bundle all consistency and safety rules for that component (e.g., by `executeSafetyCheck(state)`). A rule evaluation of an observed or calculated state of the topology is done recursively. For each tested RTU in a topology those bundled functions are called on each connected node component.

All rules are implemented in a fail-safe way, which means that all exceptions and errors that occur during the evaluation process are caught and do not propagate. All comparisons of floating-point numbers are done with a special function which takes either an allowed relative or absolute error margin as a parameter for its precision.

2) Managing system state: If a command (for example, a switch opening command) is captured, the tool should be able to evaluate, whether this command could lead the system into an unsafe state. To achieve this, it calculates new possible state ST_c based on the previously observed state ST_o and evaluates its rules over many states of the same immutable topology.

On the implementation's side, the state of the electrical grid is saved in a dictionary. Every time a new measured or reported value is seen on the traffic, the previous value is updated in the observed state dictionary. Every state depending property is referenced in the dictionary by a globally unique key $u \in U$. As described before, this key is part of the immutable topology. For example, the key which references the measured voltage is saved in the attribute `voltageKey` of the class `Meter`. A state property $(v, t, i) \in V$ contains the observed value v, the time of observation t, which is important to judge its freshness, and its validity. Mathematically speaking a state ST is defined as a function. Let U be the set of all keys defined in the topology, $V \in \mathbb{Q} \times \mathbb{Q}_{\geq 0} \times \{0, 1\}$ and $V^\epsilon = V \cup \{(\epsilon, \epsilon, 0)\}$: $ST : U \to V^\epsilon$

$$u \mapsto \begin{cases} (v, t, 1) & \text{if value } v \text{ of property } u \text{ was seen most recently at time } t \text{ and still holds.} \\ (v, t, 0) & \text{if value } v \text{ of property } u \text{ was seen most recently at time } t \text{ and is invalidated.} \\ (\epsilon, \epsilon, 0) & \text{otherwise.} \end{cases}$$

For example, if the key $u \in U$ references the measured voltage of a specific meter, then $ST(u)_1$ would return the last measured voltage value, which has been seen on traffic at time $ST(u)_2$. The second element contains the elapsed time in seconds. Implementation-wise it uses UNIX epoch time. Milliseconds are represented by the decimals of this number. $ST(u)_3$ indicates if this value is still considered valid. A more recent change in switching or transformer configuration could have invalidated the value. If there has been no corresponding measurement anytime before, then its state is unknown and $ST(u) = (\epsilon, \epsilon, 0)$. This state is considered as invalidated data when evaluating the rules. One more task of the state manager is the management of a history of all measured values for offline analyses.

3) Event engine and intrusion detection execution: The event engine offers the interface for process data input. It acts in an event-based manner. If connected to Zeek, it uses the Python bindings from the *Bro Client Communications Library*, called "broccoli" [39], which allows to receive and send events and data between a Python application and a running Zeek instance in a client-server fashion. Every time a process variable has been parsed from the traffic, Zeek triggers an event in the state manager (written in Python) containing the measured value, its physical context and the time. The same applies to commands sent over the network.

The evaluation of the intrusion detection rules are triggered by different events:

(i) **Measurements:** periodical batch-processing assessment of the observed state. Ideally each time a new measurement is parsed, all rules are immediately checked. In real deployment this approach might provoke performance, concurrency and consistency issues. This is due to the fact that the measurements can arrive in different packets (with delays), and it can take up to few seconds until new values are reported (e.g., after switching). To mitigate these issues, the evaluation process is triggered periodically every x seconds if there has not been a state update in the last y seconds. If there was an update within this time, the evaluation is postponed until the next point in time for which there was no update in the last y seconds. To prevent denial of service attacks on the IDS, the evaluation is forced after z seconds of delay. These values depend on the configuration of the system at hand, but for our case study $x = 5$, $y = 1$, $z = 10$ proved as good values. (ii) **Switching and tap position changes:** real-time and continuous assessment of their predicted effect. When commands like circuit breaks are parsed by the IDS, it might be desired to calculate their effect on the observed state. Depending on the information available - or the lack thereof - an accurate or a worst-case prediction is represented as a calculated state. The safety of this state is subsequently analyzed to assess the goodness of the command. (iii) **Set point changes:** real-time assessment, whether they are safe or not. (iv) **Manual:** the assessment of the observed state can be triggered manually. This option allows testing and debugging the intrusion detection process.

C. Connection to IDS Zeek

All described components discussed so far work independently from the network traffic. This section presents the usage of Zeek to extract process information from IEC-104 network traffic. Section II-C1 briefly describes the parser and its connection to the event engine. Then, Section II-C2 outlines the conversion of raw traffic content into values of the physical process variables.

RTU	ioa_m	ioa_c	Description	TagName	Min	Max
101	501	1001	Current	RTU1_BUS1_M11_I	-2	2
101	502	1002	Voltage	RTU1_BUS1_M11_V	-10	10
101	531	1031	Switch	RTU1_BUS1_SW11_ST		

TABLE II
EXCERPT FROM THE RTU CONFIGURATION OF A POSSIBLE RTU1.

1) Protocol parser and observed events: Zeek itself does not support the parsing of IEC-104 traffic, but the developers offer a seamless integration of custom protocol parsers written in Spicy language [35], [37], [38]. The actual process information in the Application Service Data Unit (ASDU) part of the Application Protocol Data Unit (APDU) was initially only supported for a small set of only six functions [16]. We extended the parser to trigger events for all IEC-104 function codes used at the Dutch substation [40].

Zeek offers an event-based scripting engine, which calls user-defined scripts written in Zeek language upon different events. With Spicy's integration into Zeek it is possible to define new Zeek events for parsing events, for example, upon a successfully parsed ASDU. This is utilized to trigger Zeek events for every parsed measured or commanded value. The user-defined event handler also has knowledge about the actual parsed raw value and its info object address, which are both passed as parameters to user-defined Zeek code.

2) Value and physical context interpretation: Event handlers for all relevant IEC-104 function types of measured and commanded variables have been developed. They convert the parsed raw values into the actual physical value and its context. At first, the common address, which refers to a unique SCADA device like the local RTU, and the info object address, which refers to a memory register on that device, are translated into the physical process context. Depending on the context and IEC-104 function, the raw value is subsequently converted into the right data type and right value scale according to its normalization properties.

All necessary information for this preprocessing can be found in an RTU configuration. The RTU configuration is a table which contains information about what is stored in different memory addresses on the RTU. Table II shows an excerpt of three configuration rows and the most important columns from a (fictional) RTU configuration. The first column, `RTU`, refers to the common address of the local device. The information object addresses (`IOA_M`, `IOA_C`) refer to the memory location at the local device. Please note, that they both refer to the same underlying memory location as IEC-104 uses different virtual address spaces for measurement and command functions.[3] Then, a description of the physical process context and other information like its value dimension (not depicted in this excerpt) is given. The fifth column, `TagName`, is a globally unique identifier for the process variable. As described earlier, the grid model and the state manager do not work with info object addresses as they are protocol-specific. Instead, they use this tag in the topology as a key to reference the corresponding value in the state.

The state manager expects a tuple $(tag, value)$ as its input format for process variables. Hence, Zeek must do a conversion of the protocol-specific unique address tuple (RTU, IOA) to its protocol-independent tag. Depending on the IEC-104 function used, the process values are not transmitted as floating-point numbers but normalized values instead. The normalization process takes a value and a normalization interval and outputs the value's relative position within this interval as a value in $[-1; +1[$ [41]. For example, if a current of $1000A$ is measured at M_{11} and the normalization interval is $[-2000, 2000]$ as in the depicted table, the transmitted normalized value is $+0.5$. A parsed raw value of $+0.75$ can hence be interpreted as $1500A$. The normalization interval is saved in the last two columns of the RTU configuration and is needed for the interpretation, because the real process variable cannot be inferred without this interval from traffic alone.

To enhance Zeek with the capabilities to map the protocol-specific and RTU-depending $(RTU, IOA, rawValue)$ tuple to an independent $(tag, value)$ tuple, a script was developed which reads an RTU configuration and automatically generates Zeek code for the conversion process. Subsequently, the $(tag, value)$ tuple is passed as input into the state manager. The state manager uses "broccoli" (see Section II-B3) and is connected to Zeek as a client, whereas Zeek acts as a server. Hence, Zeek can trigger events remotely in the state manager. Each parsed measurement and command trigger a corresponding event in the state manager and pass the $(tag, value)$ tuple with the current time as an argument.

III. EVALUATION ON A REAL DISTRIBUTION STATION

The prototype was used to analyze real SCADA traffic at a Dutch distribution substation, as described in Section III-A. Section III-B describes the captured data traffic and Section III-C the designed scenarios and their evaluation.

A. Topology of distribution station

Figure 2 shows an excerpt of the grid topology at the distribution station, as described by the file provided by the operator. The four power lines: L_2, L_5, L_{10} and L_{13} are feeders which are represented as incoming power lines. All other lines are outgoing. Their allowed maximal current threshold is depicted in the figure. All power lines have a meter and a switch and are guarded by a protective relay. Those components are denoted as M_X, S_X and R_X, where X is the number of the corresponding power line. Feeders L_5, L_{10} and L_{13} are directly connected to the same source bus.

B. Test setup and description of the captured data

The tests were conducted on August 15th, 2018 at a Dutch power distribution substation implementing the IEC-104 protocol. For the purpose of the tests, a hub was used to copy the incoming packets not only to the local RTU but also to the monitoring device. The monitoring device could therefore analyze the incoming commands without interrupting the

[3]In the real RTU configuration of the case study, even four different info object addresses are configured for every process variable.

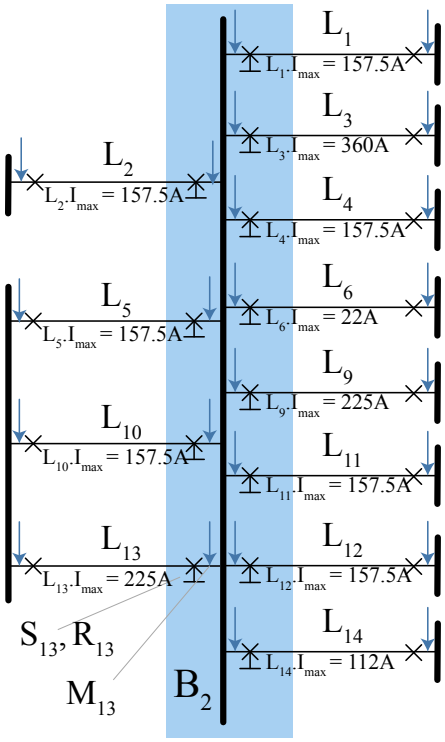

Fig. 2. Excerpt from the electrical grid topology at the case study substation.

Test	Length	Packet count		IEC-104 Events	
	[min:s]	Total	IEC-104	Commands	Measurements
Baseline	31:34	2663	300	6	4796
Set points	8:02	804	150	4	2789
Switching	4:29	479	137	4	2351
Relay	6:29	706	132	2	2369

TABLE III
SUMMARIZED INFORMATION ABOUT THE CAPTURED TRAFFIC.

working of the RTU and the physical system. The monitoring was performed on a laptop running a Docker image[4] with all needed packages and configurations for deployment and testing of the prototype mentioned in Section II.

In total four different files containing the network traffic were generated. Table III summarizes the captured traffic: the general packet count and the number of IEC-104 packets. The first file is a baseline test and includes the whole deployment process, four safe switching and two safe set point commands to ensure that the tool works properly. The other tests are shorter (all below 9 minutes), as described in Section III-C.

C. Scenarios

The baseline test was performed to confirm the proper working of the network connection, the parser and the intrusion detection model. Later tests are based on a slightly modified topology, imitating a "weaker" system, e.g., with smaller capacity and more constraints. The real system thus was never put at risk, but the IDS was still able to detect correct safety violations. A general interrogation was conducted to update the observed state before and after each command. All state changes caused by commands (i.e., changes in set points and switch positions) were reversed at the end of each scenario.

Below the four scenarios are analyzed in the following categories: *description* explains the contents of the test; *objectives* define the goal of the test; *precondition* describes the necessary initial state of the system; *state evaluation* and *command evaluation* analyses the state of the system and reflects on the effect of respective commands on the system; the rules from Table I that were violated when performing this test are listed in *violated rules*.

*1) **Scenario 1. The baseline scenario**:*

*a) **Description**:* The sequence diagram in Figure 3 visualizes all actions performed for the baseline test. After connecting the laptop running the Docker container with Zeek, the IDS was started and connected to Zeek. Then, a safe current set point was configured for L_{10} and four safe switching commands (opening and closing of S_9 and S_{10}, respectively), were given. Then, the set point was changed to its original value. Between each action a general interrogation command was issued to update the observed state.

*b) **Objectives**:* Test the functionality of the traffic parser and IDS prototype in a real-world application, validation of the used topology information and RTU configuration.

*c) **Precondition**:* Switches at L_9 and L_{10} are closed.

*d) **State evaluation**:* The initial state yielded not safe and not consistent. The reason was twofold: (i) the set point for L_{13} was not updated in the operator's reference file after a physical power line was upgraded by the network operator before, and (ii) the bad precision of the measurements resulted in around 3,5% error in the Kirchhoff's law ($P1$). After adjusting the reference file to reflect the change and adjusting the rule $P1$ to include the imprecision error, the initial state was both safe and consistent. Because of the topology, only consistency rules $P1$ (Kirchhoff's current law), $P2$ (equal voltage levels) and $P3$ (zero current on a disconnected line) are relevant and evaluated. The sum of incoming ($332.58A$) and outgoing current ($330.23A$) differs by approximately 1%, which is below the allowed relative error margin and can thus be explained by minor precision issues in traffic and meters. Over the course of the baseline tests, there are two unexpected consistency alerts. After opening S_{10}, $L_{10}.I$ is invalidated. However, at the next general interrogation command M_{10} still reports current on the disconnected line 16 seconds later. It takes 22 seconds until the correct new current measurement of $0A$ is reported. This delay causes inconsistency: $P3$ is violated as S_{10} is open and the current is not equal to 0. The same problem occurs when connecting the switch - the current value for L_{10} arrives 30 seconds earlier than those for L_{13}. As a result of this delay, Kirchhoff's law ($P1$) is violated.

*e) **Command evaluation**:* The first command changed the set point of L_{10} to $154A$, which is correctly classified as safe. Also, both commands opening and closing S_9 are

[4]The evaluation tool is available at: https://github.com/jjchromik/RuleGeneratorSCADA

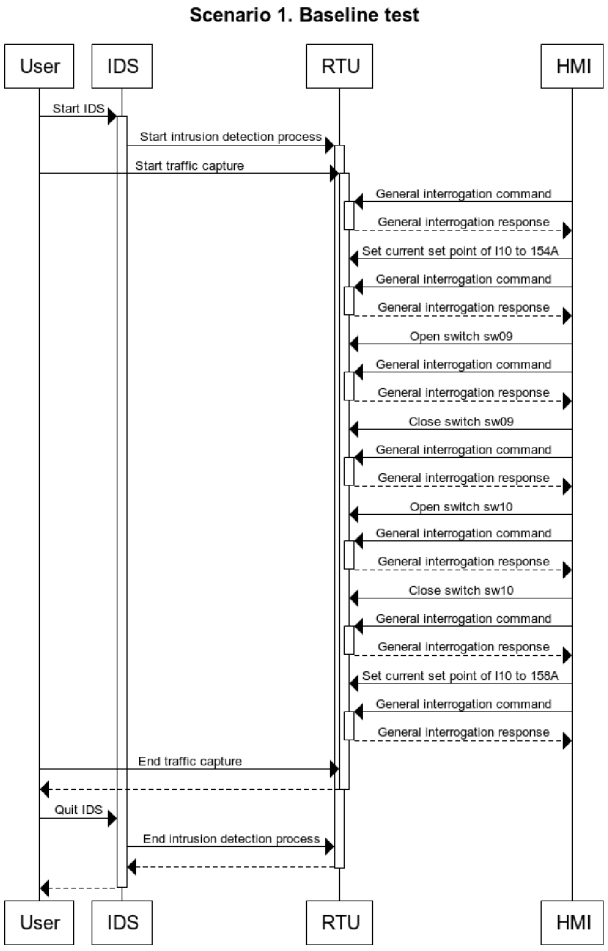

Fig. 3. Sequence diagram visualizing the baseline test of the case study.

considered safe as L_9 does not carry any current and thus the command has no effect on the power flow.

However, disconnecting the feeder L_{10} is unexpectedly flagged as unsafe. L_{10} is connected to the same bus as L_5 and L_{13}, which must then compensate the missing $92A$ of a disconnected L_{10}. It is predicted that L_5 carries $114A + \frac{92A}{2} = 160A$, which is above $L_5.I_{max} = 157.5$. The calculated current on L_{13} is $101A + \frac{92A}{2} = 147A < L_{13}.I_{max} = 225A$. This is a false positive, albeit quite a narrow one as the next general interrogation shows, that $L_5.I = 157A$ and $L_{13}.I = 142A$. The total current passing the bus seems to decrease. This is an indication that non-local feeders, which also connect power sources with consumers, experience a slight increase in current. Finally, the command that connects L_{10} again is correctly predicted as a safe one.

*f) **Violated rules**:* Violating $R5$ is a false positive. Also $P1$ and $P3$ are incorrectly violated due to inconsistent meter measurements.

*2) **Scenario 2. Unsafe set point changing commands and unsafe voltage measurements**:*

*a) **Description**:* At first, the topology information of line L_6 is changed: V_{ref} is set to 6000. After the start of the IDS three set point changes for line L_{10} are done. The first command is safe, the second set point is too low and the third one is too high.

*b) **Objectives**:* Validation of safety rules, evaluation of state calculation after set point changing command.

*c) **Preconditions**:* Set point for L_{10} equals its I_{max}.

*d) **State evaluation**:* One objective for this scenario is the validation of voltage boundary violations. $L_6.V_{ref}$ has been set to $6000V$ and thus every state evaluation correctly identifies two rule violations. Rule $R2$ is violated because the measured voltage is always far above $6000V$. Also, rule $R8$ triggers an alert, because the actual voltage set point is far too high for the modified topology.

Rule $R8$ is violated between the second and fourth command (described below), as the current set point of L_{10} is not safe. The state is considered consistent at all times.

*e) **Command evaluation**:* All four set point changes are correctly classified. The first one sets the current set point for L_{10} to $154A$ which is safe. Both the second ($126A$) and third ($190A$) command are correctly considered unsafe. Finally, the set point is restored to its original and safe value of $157.5A = L_{10}.I_{max}$.

*f) **Violated Rules**:* $R2$, $R8$ are violated.

*3) **Scenario 3. Switch commands violating an interlock**:*

*a) **Description**:* The topology is updated and a static interlock is added for lines L_9 and L_{10}. Furthermore, the value for I_{max} of line L_{13} is lowered to a value which is slightly above the current at the time of the measurement. In this test, first, switch S_9, and then, switch S_{10}, is opened.

*b) **Objectives**:* Evaluation of state calculation after switching commands.

*c) **Preconditions**:* Switches at L_9 and L_{10} are closed.

*d) **State evaluation**:* For this scenario $L_{13}.I_{max}$ was reduced to $120A$. This violates safety rule $R8$ as the set points are based on the real topology, where the I_{max} equals $225A$. Beside this violation, the state is considered both consistent and safe until shortly after the opening switch L_{10}. As expected, the current exceeds the safety threshold and violates $R1$. Additionally, the switch positions lead to interlock violations ($R9$) until S_{10} is closed again.

Once again, $P1$ and $P3$ are violated for a few seconds until the meters refresh their measurement (see scenario 1). No other consistency rule was violated.

*e) **Command evaluation**:* The first two commands, which open and close S_9, are both correctly classified as safe. Here, again, L_9 does not carry any current and thus this command has no effect on the power flow at all. The third command, which disconnects feeder L_{10}, is correctly classified as unsafe. Four violations are detected: both interlocks are violated, which results in violations of $R9$ (static interlocks and dynamic interlocks). Furthermore, two violations of $R1$ are expected: the calculated current for L_5 is $170A$ and the

one for L_{13} is $155A$, which are both above their corresponding I_{max} thresholds. The next measurement shows that the real current values on those lines are indeed $167A$ and $150A$. However, it was neither expected nor intended that the current of feeder L_5 exceeds its real current threshold by $10A$ for around a minute.[5] Fortunately, the protective relay for L_5 triggers at $300A$. The artificial violation of L_{13} was expected and has not violated the far higher real current threshold.

f) ***Violated Rules****:* $R1$, $R8$, $R9$ are violated. Again, alerts based on $P1$ and $P3$ were produced by measurement inconsistencies.

4) ***Scenario 4. Protective relay test****:*

a) ***Description****:* The protective relay information for R_{13} is changed in the topology. Its I_{cut} is set to a lower value and t_{cut} is set to 10 seconds. Afterwards, the switch on line L_{10} is opened. The current on line L_{13} then exceeds the cutting current of the protective relay.

b) ***Objectives****:* Validation of safety rules.

c) ***Preconditions****:* Switch at L_{10} is closed.

d) ***State evaluation****:* As expected, after disconnecting L_{10} the current of L_{13} exceeds the pre-configured cutting threshold of the protective relay. Thus, $R4$ is violated and reports that the cutting current of the protective relay is exceeded. At first, the alert states that the current has exceeded its threshold very recently and was still safe 10 seconds ago (which corresponds to the cutting time t_{cut}). Later, the alert changes and it attests a broken protective relay, which should have triggered after 10 seconds of exceeding current.
No other violations were expected. Like in the previous scenario, disconnecting line L_{10} led the real system into an unsafe state though. The current at L_5 exceeds its real threshold by approximately $5A$ and so $R1$ is violated. Again, delayed arrivals of measurements led to very few consistency violations of $P1$ and $P3$. At the beginning and in the end the state is considered both consistent and safe.

e) ***Command evaluation****:* Although it was intended, that all commands are safe, the switch command leads the real system into an unsafe state. This was correctly predicted by the command evaluation. For power line L_5 a current of $165A$ was predicted and shortly afterwards $162A$ was measured. The command evaluation proved to be very accurate again.

f) ***Violated Rules****:* $R1$ and $R4$ are violated and detected. Once again, $P1$ and $P3$ are violated due to measurement inconsistencies.

IV. Practical considerations

A real-world deployment of the theoretical intrusion detection model must be adjustable, since not all theoretical assumptions of the process model hold in practice. The representation of the physical process is further limited by the availability, freshness and precision of local measurements. Section IV-A introduces the concept of data unavailability in local control networks. Section IV-B investigates the handling of data that should be available but is, in fact, unknown. Then, Section IV-C analyses the consequences of measurements that are taken at different points in time. Section IV-D examines reasons and mitigation for imprecise data. Finally, Section IV-E studies the effect of those adjustments on the rate of false positives and false negatives.

A. Global and local view on the system

The theoretical model assumes a complete view of the state of the electrical grid, which could only be achieved by capturing the traffic of the whole control network. As the proposed prototype should primarily run on local devices at substations, the availability of state-dependent values is limited to data that is only locally available.

Rules that cannot be evaluated due to unavailable data will not generate an alert in order to reduce the amount of false positives. In the formal model, we need to decide whether a rule can be evaluated partially, in the case of missing data. If a rule does not contain at least one logical connection via $\wedge$ or $\vee$, the IDS cannot evaluate that rule if values are missing. Hence, no alerts are generated due to missing values. However, if a rule contains a logical *and* or a logical *or*, the rule can be partially evaluated on the available information.

Hence, all rules that contain logical conjunctions ($\wedge$) are defined in a way that allows the substitution of operands with $True$ if their evaluation is not possible due to missing data. All logical disjunctions ($\vee$) are defined in a way such that non-evaluable operands can be replaced with $False$ without causing false positives. As a result, the local IDS evaluates only a small subset of all rule expressions and under the assumption of a consistent state that corresponds to the physical process, no false positives are caused by unavailable data. False negatives may occur for all sub-expressions that cannot be evaluated due to missing values, these are however not detectable due to the unavailable information.

Some rules (e.g., the rule about total power production and consumption) need more globally dispersed state information than others. Table IV classifies all rules by the locality of required values. The letter l stands for local data at one node and g for global data from at least two different nodes. The extension $(+)$ indicates that the scope of the rules is extended if global data is known, but the rule still works in a limited evaluation scope otherwise. The consistency rules $P1, P2, P5, P6$, and the safety rules $R3, R4, R5, R8, R9$ can be fully evaluated with only local knowledge. Some rules, like $P3, R1, R2, R6$, require information of neighboring nodes. If this information is unavailable, the rules can be evaluated in a restricted manner by the reduction method described above. Some rules ($P4, R7$) cannot be reduced as they require measurements from at least two different nodes.

B. Availability

Local data may also be unavailable if local measurements have not been parsed from the traffic since the start of the intrusion detection process. Some values, for example switch states, are not reported to the RTU regularly but only after

[5] A later discussion with the grid operator revealed that the safety thresholds are chosen very conservative. A slightly exceeded current value does not really harm the system at hand.

TABLE IV
AN OVERVIEW OF RULES THAT ARE EVALUATED FOR EACH TOPOLOGY COMPONENT

component	P1	P2	P3	P4	P5	P6	R1	R2	R3	R4	R5	R6	R7	R8	R9
bus	L	L		G			L+	L+						L*	
transformer				G		L	L+	L+			L			L*	
power source/ consumer				G	L		L+	L+					G	L*	
switch			L+									L+			L*
fuse/PR			L						L	L					

"L" denotes that only local information is needed for the rule to be evaluated, "G" requires knowledge of more than one RTU. Symbol "+" means that the rule can be evaluated more thoroughly if information from more nodes is available and "*" indicates that the rule is applicable if the RTU has that aspect configured.

changes or an explicit interrogation command. Also in this case, rules might be reduced to allow partial evaluation.

To tackle this issue, immediately after the start of the IDS the state information can be initialized by a general interrogation command issued to the RTU. This command is part of the IEC-104 protocol suite and requests the report of all known process variables of an RTU. The IDS then captures the traffic and obtains knowledge of all present process variables. If the general interrogation command is not available, e.g., if other protocols are used or the RTU does not support the manual issuing of this command, detection capabilities are severely reduced until all values are received. The time of limited evaluation can be quite long as some values (e.g., the switch states) are reported only very infrequently in real systems. They are often only updated when changes occur, which can be a matter of days or even weeks in case of switches. Hence, a one-time general interrogation command for initialization is highly desirable.

C. Freshness

Data may also be outdated. The freshness of data can be impaired for three different reasons:

(i) **Changes in energy flow.** After a change in the connection of power lines (e.g., by a triggered fuse, protective relay or changed switch state) or a modification of the transformer tap position, all previous current and voltage measurements must be considered *invalid*, as they do not represent the real energy flow anymore. (ii) **Different times of measurement.** More gradual changes in energy flow are caused by the changing total power consumption over a day. Comparing older measurements then might lead to problems as they represent the real process at different time instances. (iii) **Network and parsing delay.** Values are not updated immediately due to transmission and propagation delays. Also parsing and processing takes time, which is however only a matter of milliseconds at local substations.

Outdated measurements can lead to false positive violations of consistency rules. For example, if the measurement of a power line is older than the information that the line was disconnected, a false-positive violation of $P3$ is triggered until a new measurement is obtained. Hence, all meter measurements affected are invalidated after switching or a tap position change, i.e., rules should not be evaluated with invalid information. This procedure can only be done for local events as the IDS normally does not learn about remote switch and transformer changes. Furthermore, measurements should only be used if they are relatively young. The maximal age of values that may be used for evaluation is defined as the *freshness period*. More constant data like switch states must have a longer freshness period than other, more fluctuating process variables, like meter measurements. Values that are older than the freshness period are invalidated as they are not reliable.

To overcome these challenges, an IDS should be able to actively interact with the SCADA system at hand to allow, e.g., issuing general interrogation commands or requesting individual values updates. General interrogation commands introduce traffic bursts into the control network, which might conflict with the hard real-time property of SCADA systems. However, when requesting only a subset of values, every value is separately requested and answered, the average protocol overhead is considerably increased. This trade-off needs to be carefully analyzed for a specific instance of the IDS.

D. Precision

Lastly, received values do never fully represent the physical process due to limitations in precision. The use of exact equality is only feasible in the abstract model. In reality the measurements are subject to inaccuracies caused by a) meter quality, b) precision loss due to data types in transit and c) by the data representation in the IDS. Especially the precision loss in transit is strong for IEC-104 as floating-point numbers are represented by only 2 bytes, which is very imprecise compared with even the lowest precision data types of common programming languages which usually reserve 4 bytes for *float* and 8 bytes for *double* variables. This imprecision affects all supported data types like normalized, scaled and floating-point values [42]. The effect intensifies by comparing values of different orders of magnitude given the underlying floating-point number format. Hence, the prototype compares values with an allowed relative error margin.

E. Effect on false positives and false negatives

Increasing the relative error margins for comparisons will likely lead to fewer false positives and more false negatives as the rule evaluation becomes less tight. The increased number of false negatives is not desirable, but necessary in order not

to flood the operator with too many false alerts. Increasing the freshness period will likely increase the number of alerts, since some rules would previously not be evaluated due to missing values. The freshness period should be large enough to significantly to ensure that not too many rules are skipped and small enough to ensure a reasonable precision.

Evaluating the freshness period is tightly connected to the relative error margin in comparisons. If the first increases, the latter must also be increased due to the stronger effect of value changes caused by gradual load changes. The influence on false positives (and precision) is hence not clear. It is expected that a reasonable increase in error margin prevents a surge of false positives due to gradual load differences. If this increase is too big though, a huge raise in the number of false negatives is expected. This is because the error margin should only compensate for relatively small differences caused by imprecision and gradual load changes. Long freshness periods will trigger many false positives due to load changes. Smaller periods will however lead to potentially dangerous false negatives, as many rules cannot evaluate.

We do not aim at indicating reasonable choices of error margins and freshness periods as they strongly context-dependent.

F. Quality aspects

Other crucial properties of the IDS are robustness, performance and scalability, as discussed in the following.

1) Robustness: The prototype possesses an extensive exception handling to catch errors and exceptions early to prevent, e.g., Denial of Service attacks. The only source of potentially malicious input is network traffic. Common ways to provoke a denial of service are malformed input, exploitation of logic errors and flooding [43]. The latter two offer very little attack surface: the Zeek scripts for the interpretation and conversion of values are very simple and do not contain looping control structures. Flooding the prototype with IEC-104 packets is unlikely to affect its performance and would be detected by a conventional behavior-based IDS anyway. Zeek already implements several techniques to prevent attacks in the form of overload and crashing attacks [34]. We tested traffic (i) with valid packet structure, but invalid contents and (ii) invalid malformed packets. Measurements or commands with unknown common address, info object addresses or raw values only cause an error message. The second type of packets might cause an irreversible crash of the parser by a segmentation fault, which requires a restart of the prototype. Note that this could be prevented by adapting the used parser framework.

2) Performance: The evaluation of state at each RTU is the most complex task of the state manager, as it executes all rules for all connected components. For the baseline scenario and the required knowledge of the values, state evaluation takes less than $0.002s$ on the test machine[6]. Command evaluation is even faster and takes less than $0.001s$ for any command in the test scenarios. The complexity of rule evaluation is linear in the number of nodes and power lines. Therefore, the state evaluation can likely be executed regularly on a huge electrical grid without performance issues. For evaluations, which operate on local knowledge, the evaluation is clearly less complex and hence even faster.

The performance of Zeek and its parser is analyzed using synthetic traffic of only IEC-104 packets at a constant packet rate. To simulate the worst case, all nodes have knowledge of all connected components, and both command and state evaluations are enabled. The prototype can handle $0.9Mbps$ (approx. 1500 measurements and 20 commands per second) over a longer period of time without delays. The broccoli event engine seems to be the bottleneck. When increasing traffic, first the evaluation is delayed before events are dropped due to a limited event buffer.

Replaying the captured traffic of the baseline test from the case study, Table III indicates, that traffic from real control networks is not evenly distributed, but rather occurs in bursts and contains more protocols. A speed multiplicator of up to 500 ($0.74Mbps$) was used to replay half an hour of real traffic in less than four seconds, without exhausting Zeeks internal buffer size at traffic bursts. Larger bandwidths may lead to packet drops though. The evaluation demonstrates that the prototype's performance is suitable for deployment in real substations. It is expected that traffic at larger substations also does not inflict any performance issues.

3) Scalability: Scalability is considered w.r.t. global electrical grid size and w.r.t. local substation size. As the present prototype does currently not share information with remote substations, its scalability is independent of the size of the electrical grid. A large number of components connected to the observed RTU leads to an linear increase in evaluation complexity, and should hence not result in performance issues even at large field stations.

V. Conclusion

Recent cyber-attacks on electrical grids demonstrated the vulnerability of SCADA systems controlling them. Critical infrastructures are at risk, because of outdated and vulnerable devices and protocols, and the increasing interconnectedness of control systems.

This paper shows the feasibility of process-aware intrusion detection in electrical grids. Based on a theoretical grid model, a defense-in-depth approach has been implemented into a prototype. The tool uses a novel architecture, that separates the traffic processing from the evaluation of the physical model. Given a description of the physical topology, it automatically matches the relevant rules that need to be evaluated. A case study conducted at a local distribution station of a Dutch distribution grid operator confirms the real-world feasibility. We evaluated the detection capabilities of the prototype in different scenarios and our results indicate that a network-based and process-aware IDS can strongly increase the safety and security of electrical grids.

Future work will investigate the required resolution of the underlying grid model and define response strategies to alerts.

[6]Intel Core i5 6500, 4x 3.20GHz, 128kB/102kB/6144kB L1/L2/L3 Cache, 8GB RAM

REFERENCES

[1] Homeland Security and Industrial Control Systems Cyber Emergency Response Team, "Cyber-Attack Against Ukrainian Critical Infrastructure," 2016. [Online]. Available: https://ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01

[2] R. M. Lee, M. J. Assante, and T. Conway, "Analysis of the cyber attack on the Ukrainian power grid," SANS Industrial Control Systems, Tech. Rep., 2016.

[3] National Cybersecurity and Communications Integration Center, "IR-ALERT-H-16-056-01 NCCIC / ICS-Cert Incident Alert: Cyber-Attack against Ukrainian critical infrastructure," 2016, from: https://ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01 accessed on 5 Dec 2018.

[4] K. Stouffer and K. Scarfone, "Guide to Industrial Control Systems Security Recommendations of the National Institute of Standards and Technology," 2011.

[5] R. F. Dacey, "Critical Infrastructure Protection: Challenges and Efforts to Secure Control Systems Typical Components of a Control System," 2004.

[6] J. J. Chromik, A. Remke, and B. R. Haverkort, "What's under the hood? Improving SCADA security with process awareness," in *IEEE Proceedings of the 2016 Joint Workshop on Cyber-Physical Security and Resilience in Smart Grids*, 2016.

[7] ——, "Improving SCADA security of a local process with a power grid model," in *Proceedings of the 4th International Symposium for ICS & SCADA Cyber Security Research 2016*, 2016, pp. 114–123.

[8] ——, "An Integrated Testbed for Locally Monitoring SCADA Systems in Smart Grids," *Energy-Open*, 2017.

[9] ——, "Bro in SCADA : dynamic intrusion detection policies based on a system model," in *Proceedings of the 5th International Symposium for ICS & SCADA Cyber Security Research 2018*, 2018, pp. 112—-121.

[10] D. Yang, A. Usynin, and J. W. Hines, "Anomaly-based intrusion detection for SCADA systems," in *5th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies*, 2005, pp. 12–16.

[11] R. R. R. Barbosa, "Anomaly Detection in SCADA Systems: A network based approach," Ph.D. dissertation, University of Twente, Enschede, 2014.

[12] R. R. R. Barbosa and A. Pras, "Intrusion detection in SCADA networks," in *Proceedings of the Mechanisms for Autonomous Management of Networks and Services, and 4th International Conference on Autonomous Infrastructure, Management and Security*, 2010, pp. 163–166.

[13] R. R. R. Barbosa, R. Sadre, and A. Pras, "Flow whitelisting in SCADA networks," *International Journal of Critical Infrastructure Protection*, vol. 6, no. 3-4, pp. 150–158, 2013.

[14] O. Linda, T. Vollmer, and M. Manic, "Neural Network based Intrusion Detection System for critical infrastructures," in *2009 International Joint Conference on Neural Networks*. IEEE, 2009, pp. 1827–1834.

[15] A. Valdes and S. Cheung, "Communication pattern anomaly detection in process control systems," in *Technologies for Homeland Security*. IEEE, 2009, pp. 22–29.

[16] R. Udd, M. Asplund, M. Kazemtabrizi, S. Nadjm-Tehrani, and M. Ekstedt, "Exploiting Bro for Intrusion Detection in a SCADA System," in *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, 2016, pp. 44–51.

[17] N. Goldenberg and A. Wool, "Accurate Modeling of Modbus / TCP for Intrusion Detection in 1 Introduction," *International Journal of Critical Infrastructure Protection*, vol. 6, no. 2, pp. 63–75, 2013.

[18] A. Mahmood, C. Leckie, J. Hu, Z. Tari, and M. Atiquzzaman, *Network traffic analysis and SCADA security*. Springer, 2010.

[19] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes, "Using Model-based Intrusion Detection for SCADA Networks," in *Proceedings of the SCADA security scientific symposium*, 2007.

[20] U. K. Premaratne, J. Samarabandu, T. S. Sidhu, R. Beresh, and J. C. Tan, "An intrusion detection system for IEC61850 automated substations," *IEEE Transactions on Power Delivery*, vol. 25, no. 4, pp. 2376–2383, 2010.

[21] J. Nivethan and M. Papa, "Dynamic rule generation for SCADA intrusion detection," in *2016 IEEE Symposium on Technologies for Homeland Security*, 2016.

[22] H. Lin, A. Slagell, C. Di Martino, Z. Kalbarczyk, and R. K. Iyer, "Adapting Bro into SCADA: building a specification-based intrusion detection system for the DNP3 protocol," in *Proceedings of the 8th Annual Cyber Security and Information Intelligence Research Workshop*, 2013.

[23] Y. Yang, K. McLaughlin, T. Littler, S. Sezer, and B. Pranggono, "Intrusion Detection System for IEC 60870-5-104 based SCADA networks," in *Power and Energy Society General Meeting*. IEEE, 2013.

[24] W. Gao, T. Morris, B. Reaves, and D. Richey, "On SCADA control system command and response injection and intrusion detection," *eCrime Researchers Summit*, 2010.

[25] M. Caselli, E. Zambon, and F. Kargl, "Sequence-aware Intrusion Detection in Industrial Control Systems," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, 2015, pp. 13–24.

[26] G. Koutsandria, V. Muthukumar, M. Parvania, S. Peisert, C. McParlan, and A. Scaglione, "A Hybrid Network IDS for Protective Digital Reays in the Power Transmission Grid," in *Proceedings of Smart Grid Communications*, 2014, pp. 908–913.

[27] J. L. Rrushi and R. H. Campbell, "Detecting Attacks in Power Plant Interfacing Substations through Probabilistic Validation of Attack Effect Bindings," in *Proceedings of the SCADA Security Scientific Symposium*, 2008.

[28] S. Pan, T. Morris, and U. Adhikari, "A specification-based intrusion detection framework for cyber-physical environment in electric power system," *International Journal of Network Security*, vol. 17, no. 2, pp. 174–188, 2015.

[29] Y. Wang, Z. Xu, J. Zhang, L. Xu, H. Wang, and G. Gu, "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.

[30] H. Lin, A. Slagell, Z. Kalbarczyk, P. Sauer, and R. K. Iyer, "Runtime Semantic Security Analysis to Detect and Mitigate Control-related Attacks in Power Grids," *IEEE Transactions on Smart Grid*, vol. 9, no. 1, pp. 163–178, 2018.

[31] D. Hadziosmanovic, R. Sommer, E. Zambon, and P. Hartel, "Through the Eye of the PLC: Towards Semantic Security Monitoring for Industrial Control Systems," in *Proceedings of the 30th Annual Computer Security Applications Conference*. New Orleans, Louisiana, USA: International Computer Science Institute, 2014, pp. 126–135.

[32] B. Zhu and S. Sastry, "SCADA-specific Intrusion Detection/Prevention Systems: A Survey and Taxonomy," in *Proceedings of the 1st Workshop on Secure Control Systems*, 2010.

[33] J. Giraldo, D. Urbina, A. Cardenas, J. Valente, M. Faisal, J. Ruths, N. O. Tippenhauer, R. Candell, N. O. Tippenhauer, and H. Sandberg, "A Survey of Physics-Based Attack Detection in Cyber-Physical Systems," *ACM Computing Surveys*, vol. 51, no. 1, p. 76, 2018.

[34] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23, pp. 2435–2463, 1999.

[35] The Bro Network Security Monitor, "Bro Introduction," 2018. [Online]. Available: https://www.bro.org/sphinx/intro/index.html

[36] P. Mehra, "A brief study and comparison of Snort and Bro Open Source Network Intrusion Detection Systems," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 1, no. 6, pp. 383–386, 2012.

[37] R. Sommer, J. Amann, and S. Hall, "Spicy: a unified deep packet inspection framework for safely dissecting all your data." in *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, 2016, pp. 558–569.

[38] International Computer Science Institute and Networking and Security Group, "About Spicy," 2018. [Online]. Available: http://www.icir.org/hilti/

[39] The Bro Network Security Monitor, "Broccoli: The Bro Client Communications Library," 2018. [Online]. Available: https://www.bro.org/sphinx/components/broccoli/broccoli-manual.html

[40] J. J. Chromik, A. Remke, B. R. Haverkort, and G. Geist, "A Parser for Deep Packet Inspection of IEC-104: A Practical Solution for Industrial Applications." in *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2019.

[41] G. Clarke, D. Reynders, and E. Wright, *Practical Modern SCADA Protocols: DNP3, 60870.5 and Related Systems*. Elsevier, 2004.

[42] International Electrotechnical Commission, "IEC 60870-5-104: Telecontrol equipment and systems - Part 5-104: Transmission protocols - Network access for IEC 60870-5-101 using standard transport profiles," 2006.

[43] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet Denial of Service: Attack and Defense Mechanisms*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.