Chapter 8

A COMMUNICATIONS VALIDITY DETECTOR FOR SCADA NETWORKS

Prashant Anantharaman, Anmol Chachra, Shikhar Sinha, Michael Millian, Bogdan Copos, Sean Smith and Michael Locasto

Abstract Supervisory control and data acquisition systems (SCADA) are attractive targets due to their widespread use in the critical infrastructure. A large percentage of attacks involve crafted inputs. Buffer overflows, a form of crafted input attack, are still common. These attacks can be used to take over SCADA systems or force them to crash. The compromised systems could be leveraged to issue commands to other devices in a SCADA network and cause harm.

This chapter presents a novel forensic tool that enables operators to detect crafted input attacks and monitor SCADA systems and networks for harmful actions. The tool incorporates several language-theoretic security-compliant parsers to ensure the syntactic validity of communications, enabling the detection of zero-day attacks that leverage crafted packets. The tool also detects attacks triggered using legacy protocols and includes graphical user interfaces, command-line interfaces and tools for comparing network traffic against configuration files to detect malicious activities. Experimental evaluations of the parsers using a large SCADA network traffic dataset demonstrate their efficacy. Fuzzing experiments demonstrate the resilience of the parsers as well as the tool itself.

Keywords: SCADA systems, language-theoretic security, forensics

1. Introduction

Supervisory control and data acquisition (SCADA) protocols are used throughout modern power grids for automation and "smart" operation. Substations in power grids are increasingly unstaffed and are managed from control centers using SCADA protocols such as Distributed Network Protocol 3 (DNP3) and IEC 61850 Manufacturing Message Spec-

Published by Springer Nature Switzerland AG 2022
J. Staggs and S. Shenoi (Eds.): Critical Infrastructure Protection XV, IFIP AICT 636, pp. 155–183, 2022.
https://doi.org/10.1007/978-3-030-93511-5_8

ification (MMS). These critical infrastructure assets have historically separated operational technology systems from information technology systems using non-routable interfaces and legacy hardware. However, the air gaps are disappearing as the assets are increasingly being operated remotely [8].

The interconnectivity of SCADA systems has also increased their attack surfaces. In December 2015 and 2016, attackers leveraged spearphishing email to access computer systems of electric utility operators in Kiev, Ukraine [15], eventually taking several substations offline by opening electrical relays remotely. Iran's uranium hexafluoride centrifuges in Natanz were targeted by multiple zero-day attacks using a USB drive to breach the air gap [16].

Securing SCADA systems is challenging. Commands received by SCADA devices often have physical effects, and availability and timeliness are of paramount importance. Additionally, the devices are usually resource-constrained and security schemes often do not meet the strict real-time guarantees [41].

Meanwhile, anomaly and intrusion detection schemes for SCADA systems rely on the physics of the systems and/or communications patterns. Such schemes cannot detect crafted input attacks and zero-day attacks that are of increasing concern.

Forensic tools for SCADA systems must detect attacks that leverage invalid communications. In SCADA systems, invalid communications exploit weaknesses in programs due to insufficient syntax checking. Programs often fail to implement communications protocols correctly, leading to vulnerabilities. An attacker can exploit these vulnerabilities to crash programs or gain access to the devices that execute the programs.

Forensic tools must detect syntactically-valid but semantically-invalid communications. For each device in a SCADA network, an operator has a specifications document that lists all the IP addresses and endpoints. For SCADA protocols such as DNP3 and IEC 61850 MMS, a device only supports a specific set of requests known as setpoints. A SCADA operator also maintains documents showing the setpoints supported by SCADA devices. Communications that violate any of the network or setpoint configurations are semantically invalid.

SCADA forensic tools must also help detect communications triggered by malicious programs and devices. Differentiating between human actions and malicious program/device actions is a difficult problem. SCADA forensic tools must provide visual feedback or confirmation of human-triggered actions and malicious actions.

To address these challenges, this chapter presents a communications validity detection tool for SCADA networks. The tool detects malformed

packets in a wide range of SCADA protocols. Packets that do not conform to the protocol specifications are flagged, providing the ability to detect potential zero-day attacks. The tool detects various web-based, Telnet-based and DNP3 commands leveraged in attacks. Also, it detects configuration and communications mismatches in SCADA networks. Indeed, this work heralds a new forensic paradigm that permanently positions devices across a large SCADA network to monitor traffic and provide early warnings of cyber attacks.

2. Background and Related Work

This section presents background information and related research on SCADA systems, SCADA network attacks, language-theoretic security, SCADA system forensics, software-defined networks and anomaly detection.

2.1 SCADA Systems

SCADA systems are deeply embedded in critical infrastructure assets [13]. These systems are used to monitor and control assets such as railroads, aircraft, nuclear power plants, electric power grids, water treatment plants and petrochemical refineries.

SCADA systems in the electric grid are housed in two principal types of facilities, substations and control centers. Substations usually span large geographic areas. Multiple substations are typically managed by a single control center. A control center houses real-time automation controllers (RTACs), master terminal units (MTUs) and human-machine interfaces (HMIs). Substations house intelligent electronic devices (IEDs), relays, programmable logic controllers (PLCs) and remote terminal units (RTUs). These devices are responsible for physical tasks such as opening and closing circuits at substations, among others.

Devices in a substation such as programmable logic controllers and remote terminal units communicate with the real-time automation controllers and human-machine interfaces at the control center. The control center aggregates data such as phasor information and the states of all the relays in the substation. Operators use the human-machine interfaces to send commands to relays via the remote terminal units.

SCADA systems use various communications protocols. This research focuses on the DNP3, IEEE C37.118, IEC 61850 MMS, IEC 61850 GOOSE, SEL Fast Message and SES-92 protocols. Of these protocols, DNP3, IEC 61850 MMS and SES-92 can be used interchangeably to poll remote terminal units and send commands from human-machine interfaces.

The IEEE C37.118 and SEL Fast Message protocols are used to send phasor measurements from phasor measurement units to phasor data concentrators. Phasor measurements are useful for estimating the state of a power system, detecting wide-area power events and monitoring power flows. Some substations use dedicated phasor measurement units and phasor data concentrators, but these features are often incorporated in relays.

SCADA systems reside in operational technology networks that have distinct characteristics from conventional information technology networks [36]. First, operational technology networks operate under hard real-time constraints [40]. Network packets received after deadlines are often useless and may have adverse effects on the power system state. For example, the IEC 61850 GOOSE protocol has a packet reception deadline of 4 ms. Any packets received after this time are ignored.

Furthermore, devices in substations often operate continuously for long periods of time. Most of the devices run real-time operating systems, which handle memory differently from conventional operating systems. In particular, the operating system kernel and user memory are not separated. This feature of real-time operating systems renders them prime targets for buffer overflow attacks.

Some SCADA devices only support serial protocols. As a result, IP-based sniffers on routers and switches are mostly ineffective for these devices. Also, most attacks that target information technology systems exploit various features of IP-based protocol stacks. SCADA devices may be immune to such attacks, but they are still prone to buffer overflow and crafted packet attacks.

Finally, most SCADA protocols do not support encryption. Even when the implementations support protocol encryption, communications are unencrypted because encryption adds latency. Most SCADA protocols also do not support authentication mechanisms, which enables SCADA devices to be spoofed easily.

2.2 SCADA Network Attacks

Historically, SCADA networks have used proprietary protocols and devices that were separated from other components. The air gaps are disappearing as corporate and cloud networks increasingly connect to SCADA networks [6]. The absence of air gaps enables attackers who enter corporate or cloud networks to make their way into SCADA networks.

In December 2015 and 2016, attackers leveraged spear-phishing email with malicious Word documents to access computer systems of electric

utility operators in Kiev, Ukraine [15]. The attackers observed operator actions and collected virtual private network (VPN) credentials to access the SCADA networks. The attackers then took several substations offline by opening electrical relays remotely.

One of the earliest recorded SCADA system attacks was in 1982 [9]. A trans-Siberian gas pipeline ran software that incorporated a Trojan horse. The malware executed when pipeline operators were conducting a routine pressure test, increasing the pipeline pressure and causing an explosion.

In 2017, a Saudi Arabian petrochemical plant was targeted by the Triton malware [29]. As in the Ukraine attacks, the attackers pivoted from the information technology network to the operational technology network to infect engineering workstations. The Triton malware reprogrammed the Triconex industrial safety system to induce an automatic shutdown.

In contrast, the Stuxnet malware entered a highly-secure air-gapped operational technology network in Natanz, Iran via an infected USB drive [16]. The malware injected a rootkit into a Siemens programmable logic controller that sent malicious commands to uranium hexafluoride centrifuges while reporting normal readings to plant operators. In 2010, it was reported that Stuxnet destroyed almost 20% of Iran's centrifuges.

Meanwhile, researchers have discovered several vulnerabilities that could have been exploited by attackers. For example, Lee et al. [21] have demonstrated several simulated attacks on DNP3 systems. Crain and Sistrunk [11] have discovered several vulnerabilities in DNP3 vendor implementations.

SCADA systems often have buffer overflow weaknesses [40]. In addition to the buffer overflows discovered by Crain and Sistrunk [11], researchers have found heap-based buffer overflows in WellinTech KingView servers that are widely used in China. Triangle Microworks [26] reported buffer overflows in their DNP3 library that could be exploited using crafted packets.

2.3 Language-Theoretic Security

Language-theoretic security is a programming paradigm that mandates that all inputs received by a program must be treated as sentences in a formal language, such as one generated by a regular or context-free grammar. Moreover, all inputs must be validated by a recognizer for the formal language before they are processed.

Most protocol specifications constitute pages of verbose text without machine-recognizable grammars. Developers have to read through

the entire specifications and implement code that conforms to the undocumented grammars. As a result, they often leave certain features unimplemented and/or do not implement features correctly. For example, a stack overflow bug was found in the Triangle Microworks DNP3 library [26]. The CVE description of the bug acknowledges that it was due to "poor validation of user-supplied data."

In a language-theoretic security methodology, a protocol specification is expressed in terms of a formal grammar. A parser-combinator toolkit such as Hammer [28] is used to implement a parser based on the grammar. Parser combinators make it easy to implement grammars using programming languages.

Several attempts have been made to implement parsers for SCADA protocols. Bratus et al. [7] were the first to implement a parser for the popular DNP3 protocol. However, they discovered that the Hammer toolkit constructs were inadequate to implement the protocol and had to incorporate additional constructs. The resulting DNP3 parser is incorporated in the communications validity detection tool described in this work.

Anantharaman et al. [4] have developed a tool that filters invalid and malformed IEEE C37.118 packets using a language-theoretic security parser. This parser is also incorporated in the communications validity detection tool with the caveat that the parser is only used to detect anomalous traffic and not perform filtering. When malformed packets are detected, only alerts are sent because availability in SCADA systems is of paramount importance. Specifically, a packet that is filtered as a false negative could prevent a SCADA device from performing a time-critical task with adverse consequences.

Millian et al. [23] have demonstrated how a power grid utility network could be converted to use language-theoretic security-compliant protocol implementations. They explored the steps it would take to include language-theoretic security filters, i.e., software that would not allow any malformed traffic to pass. In contrast, given the risk posed by false negatives, this work allows all traffic to pass while operators are alerted to malformed inputs.

2.4 SCADA System Forensics

After a cyber attack occurs, digital forensic practitioners apply various techniques and tools to gather evidence to retrace the attack steps and prevent similar attacks in the future. Wright et al. [38] have proposed a model for investigating cyber attacks on SCADA systems. The model has four sequential steps: examination of evidence sources, identifica-

tion of an attack, collection of evidence and documentation of evidence. The communications validity detection tool described in this research follows these forensic analysis steps. The tool gathers evidence from network packets and identifies packets that violate specifications as potential crafted packets. This evidence is logged in a local database.

Valli et al. [35] have proposed a framework for creating Snort signatures for various vulnerabilities. They examined multiple vulnerabilities in SCADA protocols such as DNP3 and Modbus to create signatures. They performed attacks in a test environment and constructed a system to generate Snort rules from packet captures. The communications validity detection tool differs from this framework in a fundamental manner. Rules are not created for known attacks; instead, parsers that validate all inputs are developed for SCADA protocols.

Ahmed et al. [2] have discussed the challenges encountered when investigating attacks on SCADA systems. They point out that operators would rather keep SCADA systems online than turn them off for evidence gathering. Therefore, SCADA systems need live forensic tools [1, 24]. Additionally, intrusion detection tools based on prior data or rule sets may be too strict for forensic tools, causing the tools to raise too many false alarms. Also, substation devices are often very resource-constrained and storing forensic data on the devices may not be an option.

The communications validity detection tool described in this research addresses these challenges in various ways. First, the tool connects to a live network tap interface. A router or switch duplicates all network traffic and sends the duplicated traffic to the tap interface, enabling the tool to validate the packets. Second, since the tool runs its own analysis, it ignores the packets it creates, which reduces false alarms. Finally, the tool employs a PostgreSQL database to store forensic data.

2.5 Software-Defined Networks

Software-defined networks (SDNs) allow for packet processing, forwarding and filtering at virtual switches [14]. These networks disassociate network forwarding rules in a data plane and routing rules in a control plane. In the case of an operational technology network, a software-defined network would manage network access control and Ethernet forwarding for SCADA devices. Kalra et al. [20] describe a software-defined operational technology network that meets the network performance and cyber security requirements. However, their cyber security requirements only overlap with a portion of the semantic errors considered in this research, namely, issues with missing and newly-added devices.

Urias et al. [34] explore how software-defined operational technology networks can be used to deceive attackers and learn about their techniques. However, the work described in this chapter is closest to the approaches of Chang et al. [10] and Kalra et al. [20]. Chang and colleagues use software-defined operational technology networks to collect network situational awareness information. They gather system logs and event logs from operational technology networks in near real time using software-defined networks. However, this technique can only detect network configuration issues and malicious devices in the networks. The work described in this chapter goes beyond the systems proposed by Chang et al. and Kalra et al. by providing situational awareness about SCADA-system-specific semantic issues and the syntactic validity of SCADA protocols.

Since software-defined networks traditionally support only network-header-based filtering, they are conducive to providing situational awareness via event logging. More research is needed to understand how to build on existing software-defined networking technologies to validate the syntactic correctness of SCADA protocol packets.

2.6 Anomaly Detection

Anomaly detection techniques are classified as specification-based [5], signature-based [37] or learning-based [22, 25]. Specification-based techniques employ manually-specified behavior to detect anomalies. Signature-based techniques look for packets that appear to be replicating known attacks. Learning-based techniques use statistical or machine learning techniques to identify normal and abnormal operations.

These anomaly detection techniques can be used in conjunction with the communications validity detection tool to gather forensic data. Although the anomaly detection techniques can identify fuzzing attacks, they are ineffective against zero-days because they cannot protect against attacks they have not encountered before. In contrast, the communications validity detection tool can identify attacks that have not been seen previously while also detecting malicious SCADA commands.

The communications validity detection tool can be used in conjunction with device fingerprinting techniques [18] to prevent forgery attacks. Physics-based defenses [33] can also be used with the tool. In fact, the communications validity detection tool neither fingerprints SCADA devices nor considers the underlying physics of the devices; instead, it focuses on SCADA network protocols.

Berthier et al. [5] use specification-based intrusion detection, which analyzes the security properties at the transport, network and applica-

tion layers; the technique is demonstrated using metering infrastructure protocols. Hong et al. [19] combine host-based and network-based intrusion detection to obtain better detection coverage. However, neither technique can identify zero-day attacks that employ novel crafted packets.

Ren et al. [30] categorize smart grid network traffic into transport, operations and content traffic, and proceed to construct multilevel anomaly detection frameworks. The communications validity detection tool uses some similar techniques while differing in the core approach. Ren and colleagues rely on the Zeek (formerly Bro) intrusion detection system to provide attack alerts whereas the communications validity detection tool employs language-theoretic security-compliant parsers to detect attacks. The attacks detected by the parsers are significantly different from those detected by Zeek.

Of course, the Zeek intrusion detection system could be used in conjunction with the communications validity detection tool. Zeek would detect brute-force attacks and SQL injection attacks whereas the communications validity detection tool would detect misconfigurations and crafted input attacks that target electric power sector protocols.

3. Tool Design

The following technical goals were set for the communications validity detection tool to support forensic data collection in SCADA networks:

- **Live Forensics:** The tool must be connected to SCADA network traffic to continuously monitor and collect forensic data. When suspicious network activities are encountered, operators can review the collected data.

- **Syntactically-Invalid Message Detection:** The tool must detect messages that violate protocol specifications.

- **Semantically-Incorrect Packet Detection:** The tool must detect communications flow violations based on SCADA network configuration files.

- **Non-Human-Triggered Action Detection:** Most SCADA system commands with physical effects such as opening and closing breakers and relays are triggered by human operators. Since a compromised device can send these SCADA commands over a network, the tool must detect and visualize all commands with physical effects.

The following design goals were set to ensure that the communications validity detection tool would be extensible and usable:

- **Adaptive:** The tool must not depend on attack scenarios and heuristics, but should detect crafted packet attacks. Since zero-day attacks exploit patterns and vulnerabilities that have not been seen before, the language-theoretic security paradigm is leveraged to detect new attacks.
- **Scalable:** The tool must support a range of smart grid protocols and application program interfaces to support future protocols. The tool must detect syntactically-malformed packets for all the supported protocols. The tool must have a flexible architecture so that unsupported SCADA protocols can be handled by adding parsers with minimal effort.
- **Distributed:** Since copious amounts of data are generated at each substation, as much data analysis as possible should be performed locally at the substations. The control center would primarily perform aggregated operations. This is vital to reduce SCADA network overhead during tool operation.
- **Usable:** The tool must provide alerts in a useful and visually-appealing manner while not overwhelming operators with information.

3.1 Design Techniques

Several techniques are leveraged to realize the design goals of the communications validity detection tool. First, the tool uses a comprehensive set of language-theoretic security-compliant parsers for the protocols commonly used in smart grid substations. The supported protocols include DNP3, IEEE C37.118 and IEC 61850. The parsers are adaptive in that they do not rely on previous attack samples to detect crafted input attacks.

Second, the tool employs a producer-consumer model in each implementation. This design, which uses an Apache Kafka broker, renders the tool highly scalable. Future protocol support merely involves adding consumers. The producers extract the payload portions of the packets and broadcast them to all the consumers simultaneously.

Third, the tool uses a distributed master-minion system, where minions are placed at substations and the master is positioned at the control center. Figure 1 shows the distributed master-minion system. Minions, which are connected to substation routers, collect traffic from two interfaces. The shaded boxes show the communications validity detection

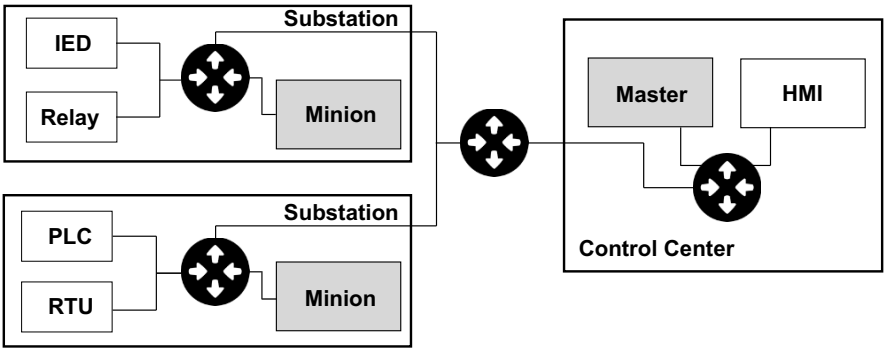

Figure 1. Distributed master-minion system.

devices. Note that the master performs data aggregation and correlation whereas the minions perform data collection and parser checks.

Finally, the tool incorporates strong visual components such as web user interfaces and command-line interfaces. The web user interfaces present smart grid operators with alerts and visual representations of specific traffic, giving operators the ability to monitor network traffic that may be well-formed but not sent by operators. For example, an attacker could use a relay device compromised via a side-channel attack to send DNP3 commands to other circuit breakers. Operators can easily detect such attacks using the visual component that displays DNP3 protocol commands.

3.2 Continuous Data Collection and Monitoring

The communications validity detection tool engages a novel paradigm of continuous data collection and monitoring. Most forensic investigation tools are deployed after attacker actions are complete. Instead, the communications validity detection tool is deployed in SCADA networks to continuously monitor network traffic. If a cyber attack is suspected, an operator can retrace the attacker's actions using the tool database.

A live network tap interface must be created by duplicating all the traffic going through a router. The duplication ensures that the communications validity detection tool does not add significant overhead to the network. The tool asynchronously processes all the packets forwarded by the router, alerting to suspicious packets. The alerts are continuously pushed to the database along with observed network traffic metadata. Leaving the tool connected to a SCADA network tap before a cyber attack to support forensic investigations enables operators to reproduce attacker steps rapidly.

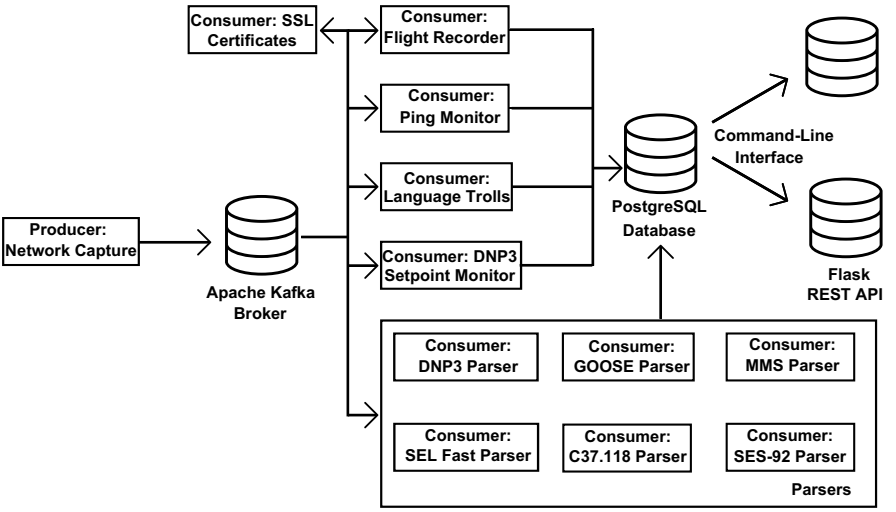

Figure 2. Publish-subscribe minion model.

3.3 Distributed Data Collection

Data is collected by minions at the substations. The minions collect and store data at the substations as well as across them. The data mostly includes SCADA commands, but could also include ARP, NTP and DNS routing data.

Since the minions and master are part of substation networks, network traffic originates from these devices as well. However, the data does not contribute to the forensic data gathered by the parsers because all traffic originating from the tool is whitelisted.

3.4 Publish-Subscribe Minion Model

A key goal is to run a given packet received from the network through the parsers in parallel to check if the packet conforms to any of the supported protocols. This is accomplished using a publish-subscribe model for minions.

Figure 2 shows the publish-subscribe minion model. The producers process network packets and forward them to the Apache Kafka broker, which broadcasts them to all the consumers. The consumers perform their analyses and store the results in a PostgreSQL database. Operators can consume information from the PostgreSQL database using web user interfaces or command-line interfaces. This publish-subscribe

Table 1. Parsers included in the current tool version.

Protocol	Language	Availability
DNP3	C/C++	Yes [7]
IEEE C37.118	C/C++	Yes [4]
IEC 61850 MMS	Python	Not yet
IEC 61850 GOOSE	Python	Not yet
SEL Fast Message	Python	Not yet
SES-92	C/C++	Not yet

model provides the flexibility to add future protocols and other analyzers without much effort.

Producers. The Apache Kafka producers accept inputs in two formats. They accept packet capture files or attach directly to live network interfaces. In either case, the producers read each packet, convert it to a string and send it to all the consumers. The producers do not do pre-processing because each consumer analyzes the packet at a different layer of the protocol stack.

Consumers. The Apache Kafka consumers receive all the packets from the producers and process them in various ways as described below. As shown in Figure 2, the consumers store the results of their analyses in a PostgreSQL database.

3.5 Detecting Syntactically-Invalid Packets

Table 1 shows the language-theoretic security-compliant parsers currently deployed in the tool; the remaining parsers will be available in the near future. Parser construction required the specifications of all the SCADA protocols of interest to be procured or purchased.

After carefully analyzing the specifications, protocol state machines and message formats corresponding to the protocols were created. The protocol state machines specify the correct sequences of packets as well as prohibited sequences. The message formats specify the structures of packets conforming to the protocols. The message formats were subsequently converted to formal grammars.

Parser-combinators, such as Hammer, were used to convert the formal grammars to code that visually resembles the formal grammars. Figure 3 shows a code snippet in the IEC 61850 GOOSE protocol parser. Note that `h.sequence()` is a function provided by the Hammer parser-combinator toolkit.

```
IECGoosePdu = h.sequence(gocbRef, timeAllowedToLive, datSet, goID, T,
                         stNum, sqNum, simulation, confRev, ndsCom,
                         numDatSetEntries, allData)
```

Figure 3. Code snippet in the IEC 61850 GOOSE protocol parser.

The language-theoretic security-compliant parsers detect packets that violate the formal protocol specifications. Although it is difficult to identify specific semantic bugs where well-formed packets crash applications, state-of-the-art fuzzers should be able to find such bugs. The parsers or syntax validators cannot detect other types of attacks. However, previous research has determined that most zero-day attacks are crafted input attacks such as buffer overflows that are handled by the parsers [3].

The parsers are Apache Kafka consumers that accept raw byte strings. The parsers process the byte strings and decide whether or not the corresponding packets are safe. Often, the packets may conform to protocols that are not yet supported. However, packets can be shortlisted by checking the packet metadata (first two bytes of payloads and Ethernet frames) to see if they conform to a supported protocol.

The consumers run in Docker containers to ensure functional separation. A parser returns the parsed object if the parse is successful and `null` if the parse has failed. The parsed object is essentially an abstract syntax tree. The parser interacts with the abstract syntax tree to store information. Based on the data extracted by the parsers, all instances of failed parses due to malformed packets are saved in a local database. After ascertaining if a particular packet is making a setpoint change, the new setpoint value is also stored in the database.

3.6 Setpoint Monitors

Operators use human-machine interfaces to interact with SCADA devices. The human-machine interfaces communicate with SCADA devices using various protocols.

Monitoring and changing setpoints are important actions performed by operators. The setpoint monitoring consumer of the communications validity detection tool records the setpoints transmitted over HTTP and DNP3 to SCADA devices. Along with the setpoints and their values, the setpoint monitoring consumer also stores the source and destination MAC addresses, IP addresses and ports. The consumer adds these records to the database and the setpoints can be visualized on a timeline.

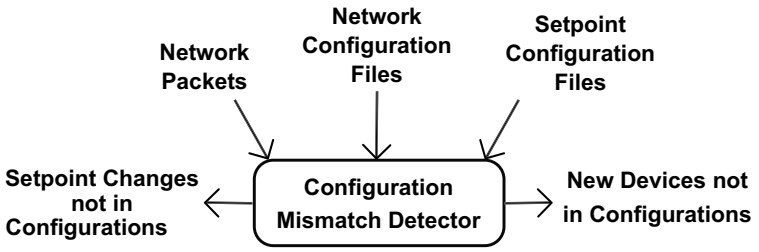

Figure 4. Configuration mismatch detection.

3.7 Detecting Semantically-Incorrect Packets

SCADA operators maintain several configuration files. The files specify setpoint mappings from point numbers in a human-readable format. Configuration files also specify IP and MAC addresses of all the SCADA network devices. To ensure availability, SCADA operators do not strictly enforce the configurations on routers and switches that filter traffic.

The configuration files are accessed during communications validity detection tool setup. Operators upload the files using a web user interface. The tool stores the configurations in the database, enabling the consumers to use them to make semantic decisions. When the configurations change, the tool enables operators to make modifications manually via the web user interface or by uploading new files.

The communications validity detection tool detects two types of semantic violations. The first type of semantic violations are due to devices that generate network traffic, but are not present in the network configurations. These could be rogue devices introduced by attackers. Adversarial code on devices can also change their network interfaces, so the tool detects them as new devices. The tool checks the IP and MAC addresses of every device in the SCADA network.

The second type of semantic violations involve setpoint mappings. Each SCADA device has a list of configured points and the types of communications (digital or analog). The parsers extract the setpoint information. The configuration files are checked to ensure that each setpoint is valid. Semantic violations are logged and alerts are sent to operators. The misconfigurations can be due to malicious code executing on SCADA devices or human error. The communications validity detection tool assists operators in detecting such semantic error conditions.

Figure 4 shows the configuration mismatch detection module. The module consumes network packets, configuration files and setpoint configuration files, and finds mismatches in the files.

3.8 User Interfaces

The communications validity detection tool provides two types of user interfaces, web interfaces and command-line interfaces:

- **Flask-Based Web User Interfaces:** Important features of a SCADA system forensic tool are supporting the visualization of actions and logging them. The communications validity detection tool incorporates a robust visual component to aid operators in detecting problems in a SCADA network. This is provided by Python Flask-based web user interfaces.

  The web user interfaces assist operators in setting up the communications validity detection tool in a SCADA network. Operators may use the interfaces to upload various network and setpoint configuration files to the tool. After operators start capturing traffic using the tool, the configuration files provide insights into misconfigurations and missing devices.

  Timeline interfaces are also provided to enable operators to check DNP3 protocol actions. Although most DNP3 actions are automated, some critical functions such as OPERATE, DIRECT-OPERATE and WRITE are triggered by humans. Operators can continuously monitor the user interfaces to ensure that all the critical functions were initiated by them and not by malicious programs. Operators can also monitor network interface visuals to check if devices have not communicated in some time and to pinpoint unidentified devices.

  The overall states of crank paths are presented as one-line diagrams. Multiple substations are connected using crank paths. During a power failure, crank paths enable troubleshooters to restore power to one substation at a time. Using various codes, the user interfaces identify whether substations are clean and have the communications validity detection tool running on them. A web user interface running on a master provides a single location for monitoring tool instances running on SCADA devices. The same interface helps pinpoint network and communications validity detection tool problems.

- **Command-Line Interfaces:** The command-line interfaces enable operators to initialize the communications validity detection tool and specify live network capture interfaces for running the tool. Packets from the interfaces are provided to all the producers and consumers. The command-line interfaces also enable individual producers and consumers to be restarted. Several commands

Table 2. Sampling of commands supported by command-line interfaces.

Command	Description
`map`	Start the communications validity detection tool against a packet capture file or network interface.
`dnppoints`	Print the observed DNP3 setpoints. The command provides several options for querying the database based on various timeframes and for DNP3 commands such as OPERATE and DIRECT-OPERATE.
`cmds`	Print the Telnet commands observed using the communications validity detection tool. Cyber attacks often leverage Telnet interfaces on SCADA devices to gain entry.
`malformed`	Print the observed malformed packets for all protocols. The command provides an overall summary or protocol-specific summary with the bytes observed in the protocols.
`flows`	Print the TCP and UDP connections entering and leaving a substation.
`layer2`	Print the MAC addresses of the observed devices.
`certs`	Print the observed SSL certificates.
`roles`	Print the SCADA device roles inferred by the communications validity detection tool. The device roles are inferred based on communications patterns and MAC addresses.
`stop`	Stop the communications validity detection tool and all the producers and consumers launched by the tool.

are available for querying multiple portions of the database, such as setpoints, Telnet commands and malformed packets for any protocol. Table 2 shows a sampling of commands supported by the command-line interfaces.

4. Tool Evaluation

Experiments were conducted to evaluate the communications validity detection tool. The experiments were conducted on a workstation with an Intel Xeon E31245 3.30 GHz processor with four cores and 16 GB RAM. Apache Kafka version 0.10 and Hammer toolkit version 1.0-rc3 were employed in the experiments.

The following metrics were employed to evaluate the communications validity detection tool:

- Correctness of the implemented language-theoretic security parsers.

- Resilience of the implemented language-theoretic security parsers to fuzzing.
- Resilience of the network interfaces of the tool to fuzzing.
- Detection of crafted packet attacks for the supported protocols.
- Ability to handle high SCADA network traffic rates.
- Visualization of operator actions.

4.1 Parser Correctness

To verify that the parsers cover a wide range of features in SCADA protocols, data collected from a SCADA tested was input to the parsers. In addition to running live network captures, the communications validity detection tool can replay and process packet capture (PCAP) files.

A dataset provided by Yardley [39] was used. The dataset contained data from 20 substations and three control centers. The substations and control centers did not belong to an actual utility, but they were otherwise realistic, simplified physical substations with SCADA communications and power equipment in a field-deployed testbed.

The experimental substations were spread over two square miles representing three independent crank paths. The substations were fed by three generators connected by real overhead and underground cables. Also included were high-voltage substations handling electricity at 13 kV. Each substation had at least four relays and a remote terminal unit. The three control centers had real-time automation controllers and human-machine interfaces from at least four manufacturers.

The dataset comprised five hours of traffic. Although most of the substations and control centers ran the DNP3 protocol. At least one substation ran each of the other SCADA protocols: IEEE C37.118, IEC 61850 MMS, IEC 61850 GOOSE, SEL Fast Message and SES-92.

Table 3 shows the parser correctness results. The parsers ran with a minimum accuracy of 94.5% and most of them had accuracies of 98% or higher. The experiments demonstrate that the parsers successfully cover an extensive feature set of the six SCADA protocols. However, several practical DNP3 implementations provide experimental and error-prone features that are not supported at this time.

4.2 Resilience to Fuzzing

Fuzzing SCADA devices can lead to crashes and denial-of-service attacks if the parsers are vulnerable [32]. The communications validity detection tool includes a set of parsers to ensure the syntactic validity

Table 3. Parser correctness results.

Protocol	Number of Substations	Packets Parsed Successfully	Total Number of Packets	Packets Parsed Correctly
DNP3	25	1,888,861	2,007,277	94.5%
IEEE C37.118	2	1,619,479	1,619,582	99.9%
IEC 61850 MMS	1	35,635	36,262	98.2%
IEC 61850 GOOSE	1	4,501	4,511	99.7%
SEL Fast Message	1	45,802	46,737	98.1%
SES-92	2	488,147	503,244	97.0%

of SCADA network packets. Since the tool is designed to detect crafted packet attacks on SCADA protocols, it was necessary to ensure that fuzzing does not crash the tool itself.

Fuzzing the communications validity detection tool was intended to serve two purposes. The first is parser resilience in that the parsers do not crash on any input (well-formed, malformed or random). The second is network resilience in that the network interfaces of the consumers can handle large volumes of SCADA network traffic.

Parser Resilience. In this set of experiments, separate fuzzers had to be used for the C/C++ and Python parsers. The C/C++ parsers were fuzzed using AFL++ [17] and the Python parsers were fuzzed using `pythonfuzz` [27], both coverage-guided fuzzers. To create a fuzzing target for AFL++, additional C files were created that invoked the parsers. AFL++ required the programs with instrumentation to be compiled using an `afl-cc` compiler. Next, `afl-fuzz` was executed on the binaries generated with a seed folder. A corpus of valid packets was created for the seed.

Each fuzzer was executed for 48 hours. Table 4 shows the parser fuzzing results. None of the fuzzing executions led to any crashes or unresponsive parsers. Each `pythonfuzz` target ran at least one million permutations through the parsers. In comparison, the AFL++ targets ran a minimum of three million executions through the parsers.

Network Resilience. In the experiments, the parsers were implemented as Apache Kafka consumers. The consumers received raw bytes from the producers that they parsed to decide if they were safe or not. Since the parsers included network interfaces, they were fuzzed using `fuzzotron` [12]. The fuzzing experiments sought to determine if the net-

Table 4. Parser resilience results.

Protocol	Parser Resilience			
	Number of Packets	Unique Paths	Crashes	Hangs
DNP3	3.62 million	623	0	0
IEEE C37.118	112 million	5	0	0
IEC 61850 MMS	2.2 million	13	0	0
IEC 61850 GOOSE	1.2 million	254	0	0
SEL Fast Message	1 million	6	0	0
SES-92	637 million	6	0	0

work interfaces were resilient and could withstand several connections and drops in connections every second.

The `fuzzotron` tool targeted the ports used by each consumer. Specifically, it created and dropped connections to the ports, often violating the TCP state machines. Once connections were established, `fuzzotron` sent random bytes on the open TCP ports.

Table 5. Network resilience results.

Protocol	Network Resilience	
	Number of Connections	Crashes
DNP3	900,000	120
IEEE C37.118	900,000	0
IEC 61850 MMS	900,000	0
IEC 61850 GOOSE	900,000	0
SEL Fast Message	900,000	5
SES-92	900,000	6

The network interfaces were fuzzed using `fuzzotron` while setting a maximum limit on the number of attempts. Table 5 shows the total numbers of connections that `fuzzotron` attempted to establish with the consumers. The numbers of network timeouts or crashes were recorded. Three consumers encountered no timeouts or crashes. The other three consumers had at most 0.1% of the packets cause crashes. Most of the crashes were due to heavy network loads. None of the crashes could be reproduced.

4.3 Crafted Packet Detection

Malformed DNP3 packets from the Aegis fuzzer [31] were employed to evaluate the communications validity detection tool against crafted packets. The dataset comprised 198 malformed DNP3 packets that were generated by mutating well-formed DNP3 packets. The malformed packets leveraged some of the DNP3 vulnerabilities identified by Crain and Sistrunk [11]. Most of the vulnerabilities were structural or syntactic.

Since the Aegis fuzzer only supports the Modbus and DNP3 protocols, mutations of well-formed packets corresponding to the IEEE C37.118, IEC 61850 MMS, IEC 61850 GOOSE, SEL Fast Message and SES-92 protocols were created. The mutated packets mostly conformed to the protocols, but violated the specifications in certain locations. A dataset comprising 198 packets was generated for each protocol.

The mutated packets were fed to the producers, which passed the packets on to their consumers. The communications validity detection tool was able to detect all the mutated packets as malformed. Also, none of the malformed packets caused any of the parsers to crash.

4.4 Parser Performance

Parser performance was assessed by measuring the time taken to decide whether packets are well-formed or malformed. The same dataset used to evaluate parser correctness was employed.

Figure 5 shows the parser performance results. The times required by most of the parsers was constant with minor variations. The times are in the order of microseconds for the IEEE C37.118, IEC 61850 MMS and IEC 61850 GOOSE parsers whereas they are in the order of milliseconds for the DNP3 and SES-92 parsers. The time taken does not directly depend on packet size. Also, latency is introduced by the publish-subscribe model.

The Python-based parsers performed much better than the parsers implemented in C. The tool incorporated Python code even for the C implementations to ensure seamless interoperability across the containers. However, this feature was found to add latency.

4.5 Visualization Capabilities

One of the core features of the communications validity detection tool is the visual component for operators to confirm actions. Several scenarios were developed to validate the operator interfaces. Network configurations were created for the test network using three relays, two remote terminal units and one real-time automation controller from three man-

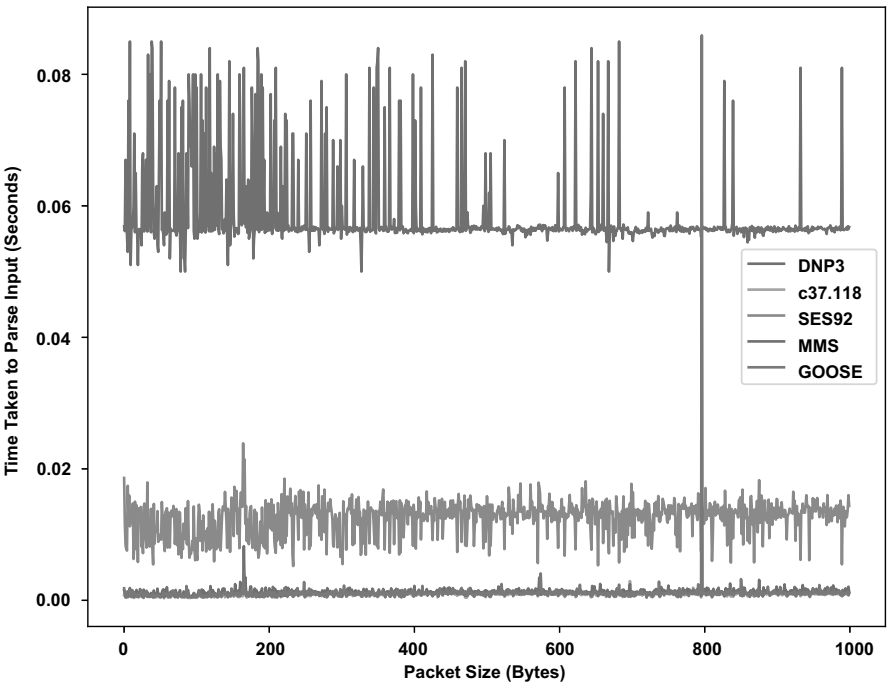

Figure 5. Parser performance results.

ufacturers. This was exclusively a SCADA system network because the relays did not control any live power settings.

Figure 6 shows the network user interface. Unidentified devices are shaded and the numbers at the edges indicate the numbers of packets observed. Various devices were removed from the network configuration and the communications validity detection tool was applied to the network. The tool was successfully able to detect all the network devices not present in the configurations. Additionally, the network configuration mismatch analyzer triggered the appropriate alerts.

The DNP3 timeline user interface was evaluated by crafting a scenario where a malicious program injects DNP3 WRITE commands in the network. In this scenario, operators must monitor the DNP3 timeline to ensure that no malicious WRITE or OPERATE commands enter the network (only human users should trigger these commands).

When the DNP3 WRITE commands were injected, the communications validity detection tool raised alerts and generated the timeline shown in Figure 7. An operator viewing the timeline on a web user inter-

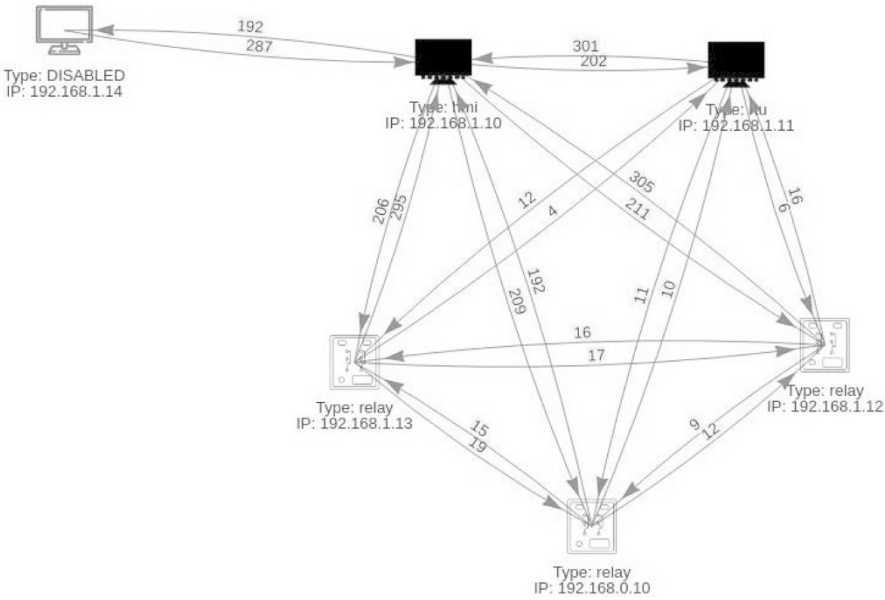

Figure 6. Network user interface.

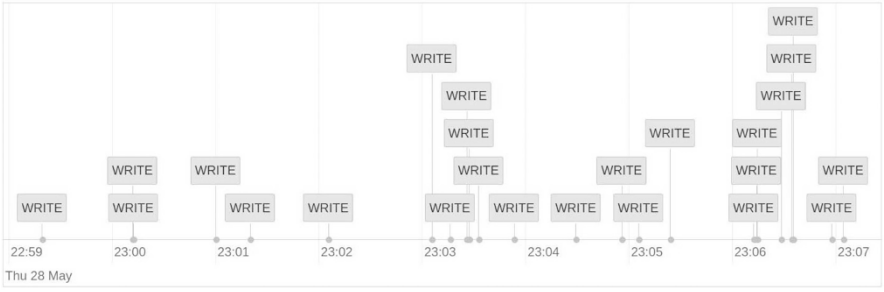

Figure 7. Web user interface with DNP3 WRITE commands in a 10-minute window.

face would see immediately that a network device is sending malicious commands.

5. Discussion

The communications validity detection tool filters packets to decide if they match the headers of a specific protocol and then run the associated parser on the packets. If the parser fails, the result is logged. This

approach is effective for most protocols, but it proved challenging for the SEL Fast Message and SES-92 protocols. These protocols do not have a fixed set of bytes designated as the header. Instead, the parsers have to be executed to determine if the packets conform to one of the two protocols. Although the parsers may detect several crafted packet attacks for the two protocols, several packets could be missed.

For example, suppose a single packet is processed by the DNP3, IEC 61850 MMS and SES-92 parsers. Then, the question is, if the packet header does not match the DNP3 and IEC 61850 MMS protocols and the SES-92 parser does not parse the packet successfully, whether the packet should be as logged an invalid SES-92 packet. This is problematic because the packet could correspond to any protocol, such as DNS or NTP, for which a parser has not been developed.

The Hammer parser-combinator toolkit was employed to construct parsers for SCADA protocols. Hammer uses two data structures, parser objects to define a parser and a parsed abstract syntax tree to provide the parsed output for a specific input. Hammer provides functions to free the parsed abstract syntax tree but does not provide a mechanism to free parser objects. This issue was discovered during the fuzzing efforts because some of the fuzzers reached their memory limits much earlier than anticipated. Discussions are underway with the Hammer development team about facilitating fuzzing by adding functions to free parser objects.

Formally verifying the language-theoretic parsers is another challenge. No formal parsing algorithms are available to handle the context-sensitive languages used in SCADA communications. Efforts have been made to construct parsers that extend beyond regular grammars and context-free grammars. The Hammer parser-combinator tool provides bindings to parse languages with context-sensitive properties. However, Hammer has not been verified. Therefore, the parser implementations were fuzzed extensively to ensure that no bugs had been introduced.

Although the communications validity detection tool was designed as a forensic analysis toolkit to gather data from SCADA networks, it can also be deployed for network intrusion detection and prevention. The primary reason for not pursuing this direction is that latency of a few milliseconds was introduced due to tool limitations. Protocols such as IEC 61850 GOOSE specify that the latency must not exceed 4 ms. This latency requirement would be violated if the communications validity detection tool in its current state were to be used as an intrusion prevention system. However, the tool could be repurposed for such applications with engineering improvements and parsers running on field programmable gate array (FPGA) hardware.

TCP reassembly poses another challenge to parsing. Although most SCADA packets are small enough not to be fragmented, the DNP3 and IEC 61850 MMS protocols have large packets in rare conditions. In its current state, the communications validity detection tool cannot handle fragmented packets. However, introducing a TCP reassembly engine in the parsers would introduce significant overhead. It would require maintaining the TCP state as well as buffering packets before decisions can be made. Nevertheless, future plans involve creating a TCP reassembly engine to go with the proposed FPGA-based parser implementations.

6. Conclusions

The communications validity detection tool presented in this chapter is designed to monitor SCADA networks for crafted input and malformed packet attacks. It incorporates language-theoretic security-compliant parsers and continuous monitoring and data collection to gather forensic data from SCADA networks.

The parsers incorporated in the communications validity detection tool cover a range of SCADA protocols and experimental evaluations using a large dataset of SCADA network traffic demonstrate their efficacy. Fuzzing experiments with the tool demonstrate the resilience of the parsers and the use of the tool in SCADA networks. Graphical user interfaces are also provided to facilitate security decision making by SCADA system operators.

Future research will formally verify the parsers and build a new toolkit to generate verified parsers. Other research will focus on new parsing algorithms for context-sensitive grammars and well as implementing highly-parallelized parsing algorithms on FPGAs to reduce latency. Additionally, attempts will be made to understand SCADA operator needs with regard to the tools needed to detect attacks on SCADA networks.

Any opinions, findings and conclusions or recommendations expressed in this chapter are those of the authors and do not necessarily reflect the views of the U.S. Air Force or DARPA.

Acknowledgement

This research was supported by the U.S. Air Force and by DARPA under Contract no. FA8750-16-C-0179.

References

[1] F. Adelstein, Live forensics: Diagnosing your system without killing it first, *Communications of the ACM*, vol. 49(2), pp. 63–66, 2006.

[2] I. Ahmed, S. Obermeier, M. Naedele and G. Richard, SCADA systems: Challenges for forensics investigators, *IEEE Computer*, vol. 45(12), pp. 44–51, 2012.

[3] P. Anantharaman, V. Kothari, J. Brady, I. Jenkins, S. Ali, M. Millian, R. Koppel, J. Blythe, S. Bratus and S. Smith, Mismorphism: The heart of the weird machine, in *Security Protocols XXVII*, J. Anderson, F. Stajano, B. Christianson and V. Matyas (Eds.), Springer, Cham, Switzerland, pp. 113–124, 2020.

[4] P. Anantharaman, K. Palani, R. Brantley, G. Brown, S. Bratus and S. Smith, PhasorSec: Protocol security filters for wide-area measurement systems, *Proceedings of the IEEE International Conference on Communications, Control and Computing Technologies for Smart Grids*, 2018.

[5] R. Berthier and W. Sanders, Specification-based intrusion detection for advanced metering infrastructures, *Proceedings of the Seventeenth IEEE Pacific Rim International Symposium on Dependable Computing*, pp. 184–193, 2011.

[6] D. Bradbury, The "air gap" between IT and OT is disappearing, and we're not ready to manage the risk, *Infosecurity Magazine*, February 26, 2019.

[7] S. Bratus, A. Crain, S. Hallberg, D. Hirsch, M. Patterson, M. Koo and S. Smith, Implementing a vertically-hardened DNP3 control stack for power applications, *Proceedings of the Second Annual Industrial Control System Security Workshop*, pp. 45–53, 2016.

[8] J. Brenner, Eyes wide shut: The growing threat of cyber attacks on industrial control systems, *Bulletin of the Atomic Scientists*, vol. 69(5), pp. 15–20, 2013.

[9] E. Byers, Cyber security and the pipeline control system, *Pipeline and Gas Journal*, pp. 58–59, February 2009.

[10] L. Chang, T. Bryan, A. McKinnon and M. Hadley, Enabling situational awareness in operational technology environments through software-defined networking, *Journal of Information Warfare*, vol. 18(4), pp. 156–166, 2019.

[11] A. Crain and C. Sistrunk, S4x14 Video: Crain/Sistrunk – Project Robus, Master Serial Killer, Digital Bond, Sunrise, Florida (`dale-peterson.com/2014/01/23/s4x14-video-crain-sistrunk-project-robus-master-serial-killer`), 2014.

[12] `denandz`, `fuzzotron`: A TCP/UDP Based Network Daemon Fuzzer, GitHub (`github.com/denandz/fuzzotron`), 2018.

[13] P. Eden, A. Blyth, P. Burnap, Y. Cherdantseva, K. Jones and H. Soulsby, A forensic taxonomy of SCADA systems and approach to incident response, *Proceedings of the Third International Symposium on ICS and SCADA Cyber Security Research*, pp. 42–51, 2015.

[14] M. Ehrlich, D. Krummacker, C. Fischer, R. Guillaume, S. Perez Olaya, A. Frimpong, H. de Meer, M. Wollschlaeger, H. Schotten and J. Jasperneite, Software-defined networking as an enabler for future industrial network management, *Proceedings of the Twenty-Third IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 1109–1112, 2018.

[15] Electricity Information Sharing and Analysis Center, Analysis of the Cyber Attack on the Ukrainian Power Grid, Washington, DC, 2016.

[16] N. Falliere, L. O'Murchu and E. Chien, W32.Stuxnet Dossier, Version 1.4, Symantec, Mountain View, California, 2011.

[17] A. Fioraldi, D. Maier, H. Eissfeldt and M. Heuse, AFL++: Combining incremental steps of fuzzing research, *Proceedings of the Fourteenth USENIX Workshop on Offensive Technologies*, 2020.

[18] D. Formby, P. Srinivasan, A. Leonard, J. Rogers and R. Beyah, Who's in control of your control system? Device fingerprinting for cyber-physical systems, *Proceedings of the Twenty-Third Annual Network Distributed System Security Symposium*, 2016.

[19] J. Hong, C. Liu and M. Govindarasu, Integrated anomaly detection for cyber security of substations, *IEEE Transactions on Smart Grid*, vol. 5(4), pp. 1643–1653, 2014.

[20] A. Kalra, D. Dolezilek, J. Mathew, R. Raju, R. Meine and D. Pawar, Using software-defined networking to build modern, secure IEC 61850 based substation automation systems, *Proceedings of the Fifteenth International Conference on Developments in Power System Protection*, 2020.

[21] D. Lee, H. Kim, K. Kim and P. Yoo, Simulated attack on the DNP3 protocol in SCADA systems, *Proceedings of the Thirty-First Symposium on Cryptography and Information Security*, 2014.

[22] L. Maglaras and J. Jiang, Intrusion detection in SCADA systems using machine learning techniques, *Proceedings of the Science and Information Conference*, pp. 626–631, 2014.

[23] M. Millian, P. Anantharaman, S. Bratus, S. Smith and M. Locasto, Converting an electric power utility network to defend against crafted inputs, in *Critical Infrastructure Protection XIII*, J. Staggs and S. Shenoi (Eds.), Springer, Cham, Switzerland, pp. 73–85, 2019.

[24] M. Naedele, Addressing IT security for critical control systems, *Proceedings of the Fortieth Annual Hawaii International Conference on System Sciences*, 2007.

[25] V. Narayanan and R. Bobba, Learning-based anomaly detection for industrial arm applications, *Proceedings of the Workshop on Cyber-Physical Systems Security and Privacy*, pp. 13–23, 2018.

[26] National Institute of Standards and Technology, CVE-2020-10613 Detail, National Vulnerability Database, Gathersburg, Maryland (`nvd.nist.gov/vuln/detail/CVE-2020-10613`), April 22, 2020.

[27] Y. Pats, `pythonfuzz`: Coverage-Guided Fuzz Testing for Python, GitLab (`gitlab.com/gitlab-org/security-products/analyzers/fuzzers/pythonfuzz`), 2020.

[28] M. Patterson, Hammer, GitLab (`gitlab.special-circumstanc.es/hammer/hammer`), 2020.

[29] N. Perlroth and C. Krauss, A cyberattack in Saudi Arabia had a deadly goal. Experts fear another try, *New York Times*, March 15, 2018.

[30] W. Ren, T. Yardley and K. Nahrstedt, EDMAND: Edge-based multilevel anomaly detection for SCADA networks, *Proceedings of the IEEE International Conference on Communications, Control and Computing Technologies for Smart Grids*, 2018.

[31] Step Function I/O, Aegis Fuzzer, Bend, Oregon (`stepfunc.io/products/aegis-fuzzer`), 2021.

[32] F. Tacliad, T. Nguyen and M. Gondree, DoS exploitation of Allen-Bradley's legacy protocol through fuzz testing, *Proceedings of the Third Annual Industrial Control System Security Workshop*, pp. 24–31, 2017.

[33] D. Urbina, J. Giraldo, A. Cardenas, N. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell and H. Sandberg, Limiting the impact of stealthy attacks on industrial control systems, *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 1092–1105, 2016.

[34] V. Urias, W. Stout and B. Van Leeuwen, On the feasibility of generating deception environments for industrial control systems, presented at the *IEEE International Symposium on Technologies for Homeland Security*, 2018.

[35] C. Valli, SCADA forensics with Snort IDS, *Proceedings of the International Conference on Security and Management*, pp. 618–621, 2009.

[36] R. van der Knijff, Control systems/SCADA forensics, what's the difference? *Digital Investigation*, vol. 11(3), pp. 160–174, 2014.

[37] J. Verba and M. Milvich, Idaho National Laboratory Supervisory Control and Data Acquisition Intrusion Detection System (SCADA IDS), *Proceedings of the IEEE Conference on Technologies for Homeland Security*, pp. 469–473, 2008.

[38] C. Wright, Forensics management, in *Handbook of SCADA/Control Systems Security*, R. Radvanovsky and J. Brodsky (Eds.), CRC Press, Boca Raton, Florida, pp. 169–200, 2016.

[39] T. Yardley, Building a physical testbed for black start restoration, presented at the *ICS Village at DEF CON Safe Mode*, 2020.

[40] B. Zhu, A. Joseph and S. Sastry, A taxonomy of cyber attacks on SCADA systems, *Proceedings of the International Conference on Internet of Things and Fourth International Conference on Cyber, Physical and Social Computing*, pp. 380–388, 2011.

[41] B. Zhu and S. Sastry, SCADA-specific intrusion detection/prevention systems: A survey and taxonomy, *Proceedings of the First Workshop on Secure Control Systems*, 2010.