CVD: A Communications Validity Detector for SCADA Networks *

Prashant Anantharaman[1], Anmol Chachra[1], Shikhar Sinha[1], Michael C. Millian[1], Bogdan Copos[2], Sean W. Smith[1], Michael E. Locasto[2]

[1] Dartmouth College, Hanover, NH
[2] SRI International, New York, NY

Abstract. Supervisory Control and Data Acquisition (SCADA) systems are a lucrative attack target due to physical impacts. A large percentage of these attacks are crafted input attacks. Buffer overflows, a relatively common form of crafted input attacks, are still common in SCADA systems and the Internet on the whole. Attackers can use such vulnerabilities to take over SCADA systems or force them to crash using vulnerabilities in software. These compromised devices could be used to issue SCADA commands to the other devices on the network and perform malicious actions.
We present *CVD*, a novel SCADA forensics tool to help operators detect crafted input attacks and monitor a SCADA substation for harmful actions. CVD includes various Language-Theoretic Security-compliant parsers to ensure the syntactic validity of the SCADA communication, hence detecting many crafted packet zero days. CVD detects attacks triggered using legacy protocols widely used in SCADA networks such as Telnet, Web interfaces, or DNP3 protocols. CVD also includes command-line tools, GUIs, and tools to compare network traffic against various configuration files. To evaluate CVD, we first ran our parsers on an extensive collection of valid packets for all the SCADA protocols we support. Next, to ensure that our parsers were resilient to random data, we fuzz-tested our parsers against *AFL++* and *python-fuzz*. To ensure that our network interfaces are resilient, we fuzz-tested the TCP Server endpoints using *fuzzotron*. Last, we also constructed various attack scenarios using malformed packets and invalid configurations and CVD was able to detect and visualize these attacks successfully.

1 Introduction

Supervisory Control and Data Acquisition (SCADA) protocols are being used to make the Power Grid *smart* and *automated*. In such a modernized grid, substations are increasingly unstaffed and controlled from control centers via SCADA protocols such as DNP3 and IEC 61850 MMS.

Such critical-infrastructure systems usually boast of an IT-OT air-gap—physical separation of the critical infrastructure (OT) and IT systems using non-routable interfaces and legacy hardware. However, this air-gap is disappearing, given the need for remote access to control these devices [1].

* This material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-16-C-0179. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force and DARPA.

In recent times, this air-gap has not helped either. Attackers were able to make centrifuges malfunction in the case of Stuxnet by taking control of PLCs [2]. These PLCs were attacked using multiple zero-day attacks using compromised USB sticks. We had no prior knowledge of these vulnerabilities and held no signatures of them. We also witnessed the use of such zero days in the Ukraine power grid attack [3]. However, these zero-day attacks have one thing in common: they are mostly input-handling vulnerabilities such as stack- or heap-based buffer-overflows [4].

Securing our SCADA Systems has been a challenge for various reasons. First, since any communication received by a SCADA device can have a physical effect, availability and timeliness are of paramount importance. The devices are usually too constrained, and security schemes often fail to meet some of the real-time guarantees required by Smart Grid networks [5].

Second, Anomaly and Intrusion detection schemes proposed for SCADA systems are either targeted at detecting anomalies using the physics of these systems or using the communication patterns. Such schemes cannot detect crafted-input attacks such as those used in the Ukraine attack or the Stuxnet attacks.

Hence, any forensic tool used to investigate cyberattacks must detect attacks that use invalid communications. In SCADA systems, invalid communications can stem from various causes. First, communications can exploit weaknesses in programs due to **insufficient syntax checking**. Programs often fail to implement communication protocols correctly, leading to vulnerabilities. An attacker can exploit this vulnerability to crash the program or gain complete access to the device running the program.

Second, forensic tools must detect syntactically valid but **semantically invalid communications**. For each device in a SCADA network, the SCADA operator holds a specification document listing all the IP addresses and the endpoints of that device. For protocols such as DNP3 and MMS, the device only supports a specific set of requests known as setpoints. A SCADA operator holds specification documents showing what setpoints each device supports. Communication violating any of these network or setpoint configurations is semantically invalid.

Finally, forensic tools must also help detect **communications triggered by a malicious program** or device, not a human. Differentiating between human actions and a malicious program is a difficult problem. SCADA forensic tools must provide visual feedback or confirmation on any human-triggered actions or evil actions.

To address these challenges, we present *CVD*, a communications validity detector. Our contributions are as follows:

- Our CVD implementation detects malformed packets in a wide range of protocols. CVD detects any packet that does not conform to the protocol specification, detecting potential zero-day attacks.
- CVD detects various Web-based, Telnet-based, and DNP3 actions that attackers take. Although these protocols are known to be insecure, they are used extensively in SCADA networks.
- CVD can detect various configuration and communication mismatches within a Smart Grid substation.
- With CVD, we propose a new forensics paradigm to permanently place our CVD devices in substations and control centers to monitor traffic and detect attacks early.

The paper is organized as follows. Section II introduces the required background and related work for the paper. Section III introduces the design of CVD. We evaluate CVD in Section IV using various datasets and fuzzers. We discuss lessons learned from building and evaluating CVD in Section V. Section VI concludes the paper.

2 Background and Related Work

2.1 SCADA Systems

Supervisory Control and Data Acquisition (SCADA) systems are deeply embedded in critical infrastructure systems [6]. SCADA systems are used to automate several critical infrastructures such as railways, aircrafts, nuclear powerplants, power grids, water plants, and oil refineries.

SCADA systems in the electric grid comprise two components: control centers and substations. Substations usually span large geographic areas, and many substations communicate with a single control center. A control center includes Real-Time Automation Controllers (RTAC), Human Machine Interfaces (HMIs), and Master Technical Units (MTUs). In contrast, substations include Intelligent Electronic Devices (IEDs), relays, Remote Terminal Units (RTUs), and Programmable Logic Controllers (PLCs). These devices have physical impacts and can open or close circuits.

Devices in substations such as PLCs and RTUs communicate with the HMI and RTAC in the control center. The control center aggregates various data such as phasor information and the state of all the relays in the substation. Operators use HMIs to send commands to various relays via the RTUs.

SCADA systems use several communication protocols for various purposes. This paper focuses on Distributed Network Protocol 3 (DNP3), IEC 61850 Manufacturing Message Specification (MMS), IEEE C37.118, SEL Fast Message, and

SES92. Of these protocols, DNP3, MMS, and SES92 can be used interchangeably to poll RTUs and send commands from HMIs.

C37.118 and SEL Fast Message are used to send phasor measurements from phasor measurement units (PMUs) to phasor data concentrators (PDCs). Phasor measurements are useful for estimating a power system's state, detecting wide-area power events, or monitoring power flows. Some stations use dedicated PMUs and PDCs, but often these features are included in relays.

SCADA systems or Operational Technology (OT) Networks have some characteristics that are distinct from Information Technology (IT) networks [7]. First, OT networks are hard real-time [4]. Any packets received after a deadline are useless and can have adverse effects on a power system's state. For example, in IEC 61850 GOOSE protocol, this deadline to receive packets is set at 4 ms. Any packets received beyond this deadline are ignored.

Furthermore, off-site devices in substations are often not touched for several years at a time. These devices operate for decades. Most of these devices run real-time operating systems (RTOS). Such operating systems handle memory differently than conventional operating systems—memory is often not separated between kernel and user memory. This feature in RTOSs makes them a prime target for buffer overflow attacks.

Some SCADA devices do not support any IP-based protocols but support serial connections. Using IP-based sniffers from routers and switches are mostly ineffective due to this. Also, most attacks targeting IT systems exploit various features of IP-based protocol stacks. These devices are inert to such attacks but are still prone to buffer-overflow or crafted packet attacks.

Finally, most of these SCADA protocols do not support encryption. Even if the specification specifies encryption protocols, they are left unutilized since encryption adds latency. These SCADA protocols also mostly do not support authentication mechanisms. Devices can be easily spoofed.

2.2 Attacks on SCADA Networks

For a long time, SCADA networks used proprietary protocols and devices that were air-gapped away from companies' other components. Today, this air-gap is slowly disappearing with corporate and cloud networks used to control SCADA networks.[3] Without the air-gap, attackers can find easier ways to make their way into SCADA networks.

Attackers did precisely this in Ukraine in 2015 and 2016 [3]. Hackers sent spearfishing emails with malicious Word documents to grid operators. These hackers could now observe the operators' actions and collected virtual private network

[3] https://www.infosecurity-magazine.com/infosec/air-gap-between-it-and-ot-1-1-1-1/

(VPN) credentials to access the SCADA network. Attackers took several substations offline by opening several relays remotely.

One of the earliest recorded attacks on a SCADA system was in 1982 [8]. A trans-Siberian gas pipeline used software that included a trojan horse. This malware was executed when the pipeline operators were conducting a routine pressure test on the pipeline. The malware increased the pressure in the pipeline, causing an explosion.

In 2017, a Saudi Arabian Oil and Gas company suffered a cyberattack that used the TRITON malware [9]. In a similar fashion to the Ukraine attack, attackers jumped from the IT network to the OT network to infect engineering workstations. This malware reprogrammed the Triconex Safety Instrumentation System leading to an automatic shutdown. Stuxnet, in contrast, entered an air-gapped OT network using infected USB drives [10]. In 2010, it was reported that Stuxnet ruined almost one-fifth of Iran's centrifuges. Stuxnet injected a rootkit into a Siemens PLC that would send unexpected commands to the centrifuges while reporting typical values back to users.

Apart from these large scale attacks carried out on various critical infrastructure, researchers have discovered several vulnerabilities that could have been exploited by attackers. For example, Lee et al. [11] demonstrated several simulated attacks on DNP3 systems. Sistrunk and Crain also found several issues in DNP3 vendor implementations [12].

Buffer overflow weaknesses are widely prevalent in SCADA systems [4]. Apart from the buffer overflows Sistrunk and Crain found, researchers also found heap-based buffer overflows in the WellinTech KingView servers widely used in China. In 2020, Triangle Microworks [13] reported buffer overflows in their DNP3 library that could be exploited using crafted packets. Any forensic tool must be able to detect and log such crafted packet attempts.

2.3 Language-Theoretic Security

Language-Theoretic Security (LangSec) is a programming paradigm stating that all input to be received by a program must be treated as a formal language, such as a context-free grammar or a regular grammar. And, any input received must be validated by a recognizer for the formal language before being acted on.

Most protocol specifications include long pages of verbose text and no machine-readable grammar. Developers need to read through the entire specification and implement code that conforms to this grammar. Developers often leave certain features unimplemented or do not implement features correctly. For example, a stack-overflow bug was found recently in the Triangle Microworks DNP3 library [13]. The CVE description of this bug acknowledges that it was due to "poor validation of user-supplied data."

To follow the LangSec methodology, we convert the protocol specifications to a formal grammar. We then use a parser-combinator toolkit such as Hammer [14] to implement the parser for the grammar. Parser combinators make it easy to implement these grammars within a programming language.

Several attempts have been made to implement parsers for various SCADA protocols. First, Bratus et al. [15] implemented a parser for the DNP3 protocol. They found that the Hammer toolkit constructs were insufficient to implement the protocol and had to add additional constructs. This DNP3 parser is a part of the CVD now.

Similarly, Anantharaman et al. [16] implemented PhasorSec, a tool to filter invalid and malformed IEEE C37.118 packets using a LangSec parser. We have incorporated this parser in CVD with a caveat: that the parser only detects anomalous traffic, and does not perform filtering. When we detect malformed packets, we only send alerts since availability is of paramount importance. If a packet gets filtered as a false negative, and causing the SCADA device to not perform a time-critical task, the consequences can be devastating.

Millian et al. [17] demonstrated how an entire power grid utility network could be converted to use LangSec-compliant protocol implementations. They explored the steps it would take to include LangSec filters, i.e., software that would not allow any malformed traffic through. In contrast with that work, given the risk of false negatives, we employ a technique to allow all traffic through, and alert users on any malformed input.

2.4 SCADA Forensics

After a cyberattack occurs, investigators use various forensic tools to gather evidence to retrace the steps of the attackers and to prevent such attacks in the future. Wright et al. [18] proposed a model to investigate cyberattacks against SCADA systems. Their model included four sequential steps: an examination of various evidence sources, identification of an attack, collection of evidence, and documentation of evidence. CVD follows these stages of forensic analysis. It gathers evidence from network packets and identifies packets that violate specifications that could be crafted packets. CVD then logs these evidences in locally kept databases.

Valli et al. [19] proposed a framework to create Snort signatures for various known vulnerabilities. They examined multiple vulnerabilities in SCADA protocols such as DNP3 and Modbus to create these signatures. They re-created the attacks in a test environment and built a system to generate Snort rules from packet captures. CVD differs from this framework in terms of approach: we do not create rules for previously known attacks. Instead, we develop parsers for various SCADA protocols that would validate all input in the network.

Ahmed et al. [20] discuss various challenges in investigating SCADA cyberattacks. First, they point out that operators would rather keep the SCADA system online than turn it off for evidence gathering; SCADA systems need live forensics tools [21,22]. Second, intrusion detection tools based on prior data or a set of rules may be too strict for forensic tools. A forensics gathering tool may set off several false alarms. Third, substation devices are often too resource-constrained. Storing forensic data on these devices may not be an option.

CVD addresses these challenges in forensic tools in various ways. First, CVD connects to a live network tap interface. The router or switch duplicates all network traffic and sends them to the tap interface. CVD then exhaustively validates the packets. Second, since CVD runs its analysis, it ignores packets it creates, reducing false alarms. Finally, CVD includes a PostgreSQL database to store forensics locally.

2.5 Anomaly Detection

Anomaly detection techniques are usually specification-based [23], signature-based, or learning-based [24, 25]. Signature-based approaches look for packets that seem to be replicating a known attack [26]. In learning-based techniques, there is a learning phase where the system learns what routine operation or an abnormal operation is. And later, the system uses this learned data to decide if an operation is malicious.

We can use these anomaly detection techniques in conjunction with CVD to gather more forensic data. Although such methods are useful in identifying fuzzing attacks, they are not effective against a wide range of zero-days, i.e., protection against future attacks we have not seen before. CVD can detect attacks that we have not seen earlier while still enabling detecting malicious SCADA commands.

CVD can also be used in conjunction with various Device Fingerprinting techniques such as [27], to prevent forgery attacks. Similarly, Physics-based defenses such as [28] could be used in conjunction with CVD. CVD does not fingerprint devices, nor does it look at the underlying physics of these SCADA devices; instead, it focuses on SCADA network protocols.

Berthier et al. [23] use specification-based intrusion detection, where they specify what the security properties of the network are of the transport layer, network layer, as well as the application layer. They demonstrate their technique on metering infrastructure protocols. Hong et al. [29] combine host- and network-based intrusion detection to get better coverage. Neither of these techniques can capture zero-day attacks such as crafted packets.

EDMAND [30] categorizes smart grid network traffic into three categories: transport, operations, and content. They then go on to build multi-level anomaly detection frameworks. Our technique uses some techniques similar to EDMAND

while differing in the core approach. EDMAND relies on the IDS tool, Zeek (formerly Bro), to provide attack alerts, whereas we built LangSec-compliant parsers to detect attacks. The attacks our parsers detect are significantly different from the ones Zeek can detect.

We wish to highlight that Zeek is an IDS that should be used in conjunction with CVD. Zeek can detect brute-force attacks and SQL injection attacks, but CVD employs a vastly different set of detectors and is designed to detect misconfigurations and crafted input attacks specifically for power protocols.

3 System Design

To build a useful forensic gathering tool for SCADA networks, we set the following technical goals for CVD:

- **Live Forensics**: CVD devices must be connected to SCADA traffic to continuously monitor and collect forensic information. In case there are any suspicious actions in a network, operators can view CVD's insights.
- **Detecting syntactically invalid messages**: CVD must detect messages that violate protocol specifications.
- **Detecting semantically incorrect packets**: Based on SCADA operator configuration files, CVD must detect communication flows violations.
- **Help in detecting non-human-triggered actions**: Most critical SCADA actions with physical effects such as opening or closing breakers and relays are human-triggered. If a compromised device communicates these SCADA commands over the network, CVD must detect and visualize all SCADA actions with physical effects.

To ensure that CVD is a tool that is extensible and usable, we set the following design goals for CVD:

- **Adaptive.** We must not depend on existing attack scenarios and heuristics, but be able to detect crafted packet attacks. Since zero-day attacks exploit patterns and vulnerabilities that we have never seen earlier, we do not want to rely on previously seen patterns but instead, use the LangSec paradigm to detect new attacks.
- **Scalable.** We need to support a wide range of Smart Grid protocols with an easy API to support future protocols. CVD detects syntactically malformed packets for protocols we implement. CVD must hold a flexible architecture so that in case a SCADA network supports protocols we do not support, developers can add parsers for this protocol with minimal effort.
- **Distributed.** Given that there is a large amount of data generated in every substation, we want to perform as much analysis as possible within the same

substation. In the control center, we only want to perform certain aggregated operations. This goal is vital to not add too much network overhead to the SCADA network due to our CVD devices.
– **Usable.** It must be able to provide alerts in a usable and visual way while not overwhelming users with information.

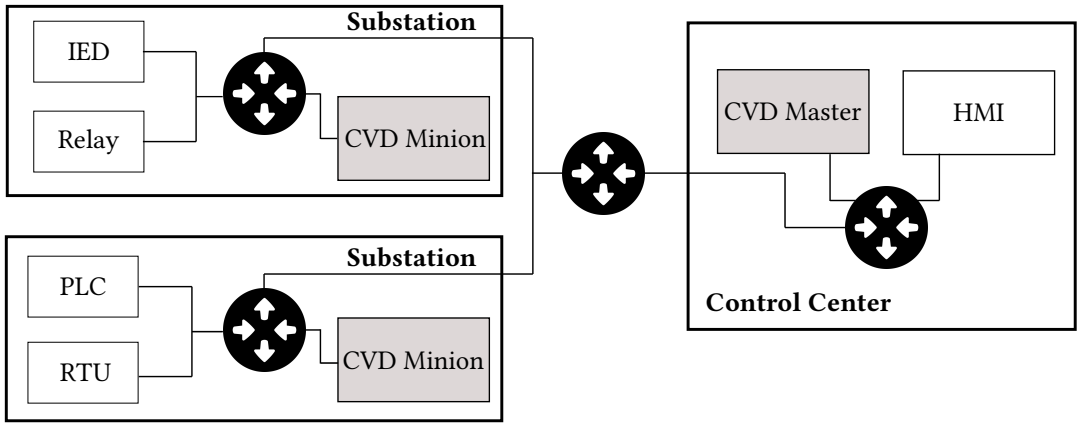

Fig. 1. *CVD Minions* are connected to the routers within the substation, and collect traffic from two interfaces. Whereas, the *CVD Master* is present in the Control Center. The shaded boxes show our CVD devices.

3.1 Design

To realize our design goals, CVD uses various techniques. First, CVD uses a comprehensive set of Language-Theoretic Security-compliant parsers for different protocols commonly used in Smart Grid substations. Some of the protocols that CVD includes parsers for are DNP3, IEC 61850, and IEEE C37.118. Our parsers are **adaptive**, not relying on previous attack samples to detect crafted input attacks. We will discuss these parsers in more detail in Section 3.5.

Second, CVD uses a producer-consumer model within each implementation. This design using Apache Kafka makes CVD **scalable**. Any future protocol additions only require adding consumers. The producers extract the payload portions of the packets and broadcast them to all the consumers simultaneously.

Third, CVD uses a **distributed** *master–minion* system, where the *CVD minions* are placed in every substation, whereas the *CVD master* is present in a control center. This design is shown in Figure 1. The CVD master performs only data aggregation and correlation, whereas the data collection and our parser checks happen in the minions.

Finally, CVD includes strong visual components in terms of various Web UIs and a command-line interface. The Web UIs present Smart Grid operators with multiple alerts and a visual representation of specific traffic, which gives operators an ability to monitor the network for traffic that may be well-formed but not sent by the operators. For example, attackers could then use a particular relay device compromised via a side-channel attack to send DNP3 commands to other breakers. Operators can easily detect such attacks via our visual component that displays various DNP3 protocol actions.

3.2 Continuous Data Collection and Monitoring Paradigm

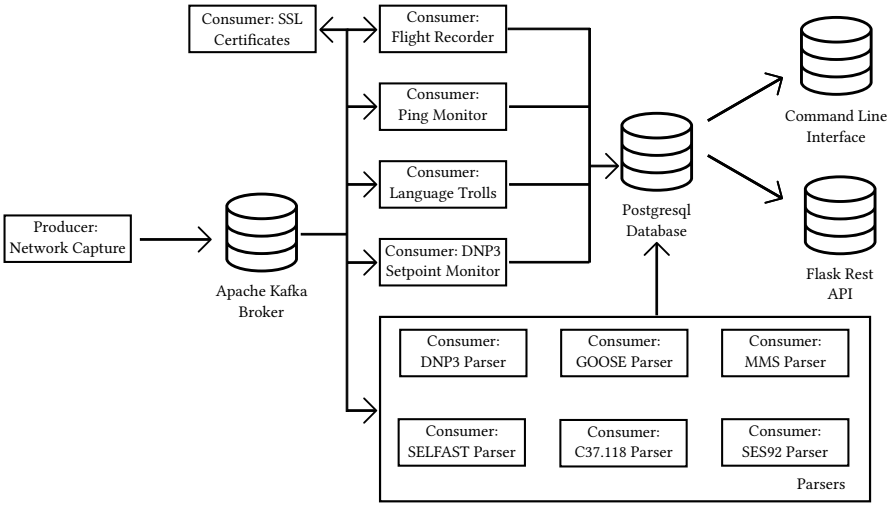

Fig. 2. Overall Architecture of the CVD Minion. The producers process network packets and forward them to various consumers via the Apache Kafka Broker. The various consumers perform their analysis and store the results in a PostgreSQL database. Operators can consume information from the PostgreSQL database via interactive Web UIs or a command-line interface.

CVD proposes a novel paradigm of continuous data collection and monitoring. Most forensic investigation tools are only deployed after a bulk of the attackers' actions are complete. We deploy CVD in SCADA networks to continuously monitor network traffic. In case of a suspicion of a cyberattack, operators can retrace the attackers' steps using the CVD database.

We require a live network tap interface that can be created by duplicating all the traffic going through a router. CVD processes all the packets the router forwards asynchronously, generating alerts on all suspicious packets. We continuously push these alerts to our databases along with all network traffic metadata observed. This paradigm of leaving CVD connected to a SCADA network tap before any

cyberattack to aid in forensics allows operators to reproduce the steps taken by attackers rapidly.

3.3 Distributed Data Collection

As seen in Figure 1, data is collected by CVD Minions present in every substation. These minions collect and store data within the substation, as well as across them. This data can include mostly various SCADA commands, but could also include other routing behavior such as ARP, NTP, and DNS.

Since the CVD Minion and Master are also a part of the substation networks, network traffic originates from these devices as well. However, this data does not contribute to any forensics gathered by our parsers since we whitelist all traffic originating from CVD.

3.4 CVD Minion Publish-Subscribe Model

One of the goals of CVD is to run the same packet received from the network, through our various LangSec parsers in parallel to check if the packet conforms to any of those packets. To do this, we adopt the publish-subscribe model.

We show this model in Figure 2. In this model, we have producers that capture network traffic on various interfaces and send them to the Apache Kafka Broker. The Kafka Broker then broadcasts it to all the consumers, that then decide what to do with these packets. We chose the publish-subscribe model since it gives us the *flexibility* to add future protocols and other analyzers easily without much effort.

Producers. We designed our Kafka producers to accept input in two formats. They can take packet capture files or attach to live network interfaces. In either case, our producers read each packet, and convert them to a `string` and then send them over to all the consumers. They do not do any pre-processing since each consumer analyzes the packet at a different layer of the protocol stack.

Consumers. Our Kafka consumers receive all the packets from the producers and process them in various ways. In this section, we describe the various consumers present in CVD. As seen in Figure 2, all our consumers store the results of their analysis in the PostgreSQL Database.

3.5 Detecting Syntactically Invalid Packets

We implemented LangSec-compliant parsers for the protocols shown in Table 1. To build these parsers, we purchased or procured the specifications of all SCADA protocols of interest. After carefully reading these specifications, we extracted the protocol state machine and the message formats in these protocols.

The protocol state machines specify what the correct sequences of packets must be and what sequences are prohibited. In contrast, message formats determine how a packet conforming to a specific protocol must look like. We converted these message formats to formal grammars.

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

Fig. 3. Code showing a portion of our IEC 61850 GOOSE parser. `h.sequence()` is a function provided by the Hammer parser-combinator toolkit.

We use parser-combinators to convert these formal grammars to code. Parser-combinators such as Hammer allow us to write parsers in code. As seen in the code snippet Figure 3, the parser-combinators help us write parsers that visually resemble the formal grammar.

Our LangSec-compliant parsers detect packets that violate the formal specifications of a protocol. Although we cannot find or see specific semantic bugs, where well-formed packets crash the application, state-of-the-art fuzzers should find such semantic bugs. Our parsers or syntax validators cannot detect other types of attacks. However, based on our prior research, most new zero-days discovered are crafted input attacks such as buffer overflows that we prevent [31].

Our parsers are Kafka consumers that accept a raw byte string. We run our parsers on this byte string and decide whether the packet is safe or not. Often, packets may conform to protocols we are yet to support. Based on the packet metadata (first two bytes of the payload and the Ethernet frame), we first shortlist packets to check if it may conform to a protocol we support.

These consumers run on Docker containers to ensure functional separation. Our parsers would return a parsed object if the parse were successful or `NULL` if the parse failed. The parsed object is essentially an abstract syntax tree (AST). Our parser interacts with the AST to store various information. Based on data extracted by our parsers, we keep all instances of failed parses due to malformed packets in our CVD database. After ascertaining if a particular packet is making any setpoint changes, we store these new setpoint values in our local CVD database.

Table 1. Various parsers included in CVD. The other LangSec parsers will also be made available soon.

Protocol	Language	Code Availability
DNP3	C/C++	Yes [15]
C37.118	C/C++	Yes [16]
SES92	C/C++	Not yet
GOOSE	Python	Not yet
MMS	Python	Not yet
SEL Fast Message	Python	Not yet

3.6 Setpoint Monitors

Operators of SCADA devices control the devices using HMIs. The HMI communicates with the SCADA devices using one of the many SCADA protocols, triggering setpoint changes.

Our setpoint monitoring consumer records these setpoints transmitted over HTTP as well as DNP3 to these devices. Along with the setpoints and values, it also stores the source and destination MAC addresses, IP addresses, and ports. This consumer adds these records to the database, and these setpoints our visualized in our timeline, shown in Figure 7.

3.7 Detecting Semantically Incorrect Packets

SCADA operators usually hold various configuration files. These files specify a setpoint mapping from point numbers to a human-readable format. Configuration files also specify IP addresses and MAC addresses of all devices on the SCADA network. To ensure high availability, SCADA operators do not strictly enforce these configurations on routers and switches filtering traffic.

We tap into these configuration files in the setup process of CVD. Operators upload these files using CVD's web UI. Once CVD stores these configurations in the database, we use the configurations in our consumers to make semantic decisions. If these configurations change during the SCADA network's routine operation, CVD allows operators to modify details on the UI manually or re-upload a new file.

CVD detects two semantic violations. First, CVD can find devices that generate network traffic but are not present in network configurations. These devices could be rogue devices introduced by adversaries. Adversarial code on devices can also change their network interfaces so that CVD detects it as a new device. We check the IP address and MAC address of each device on our network.

Second, CVD detects setpoint communications that violate the setpoint mappings. Each SCADA device has a fixed list of points configured and the type of communication (such as digital or analog). Our parsers extract the setpoint information. For each setpoint, we check the configuration files to ensure the setpoint

was valid. We log any violations as semantic violations and alert the user. Such misconfigurations can be the result of malicious code on devices or human error. We aid operators in detecting such semantic error conditions.

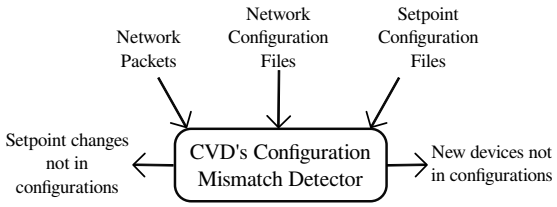

Fig. 4. CVD's Configuration Mismatch Detector. This module consumes network packets, configuration files, and setpoint configuration files, and files various mismatches in these files.

3.8 User Interfaces

Flask-based Web UI. One of the most critical abilities in a SCADA forensic tool is to visualize various actions while also logging them. CVD includes a robust visual component to aid operators in detecting problems within their network. We built a Python Flask-based UI to enable operators to perform various actions.

First, we aid operators in setting up CVD for their network. Users can upload various network and setpoint configuration files to CVD from this UI. Once operators start capturing traffic using CVD, we seamlessly use these configuration files to provide more insight to operators on misconfigurations or missing devices.

Second, we provide a timeline interface for operators to check their DNP3 actions. Although most DNP3 operations are automated, some critical functions such as OPERATE, DIRECT-OPERATE, and WRITE are human-triggered. Operators can continuously monitor these visuals to ensure that all critical functions were initiated by them and not a malicious program in the network. Likewise, operators can also monitor network interface visuals to check if any device hasn't communicated in some time or unidentified devices.

Finally, we demonstrate the overall state of a crank path using one-line diagrams. Using various codes, we denote whether substations are clean and have CVD running on them. Such a UI that runs on CVD Master provides a single place to monitor CVD instances running on various devices. Operators can pinpoint network issues and CVD issues in one location.

Command-Line Interface. To enable users to use CVD's various functionalities, we built two interfaces. Our command-line toolkit allows users to initialize CVD and specify a live capture network interface to run CVD. We run packets from that interface through all our producers and consumers. Our command-line

interface also supports restarting individual producers and consumers. We support various commands to query multiple portions of the database, such as setpoints, telnet commands, and malformed packets for any protocol. Table 2 shows some of the commands supported by CVD.

Command	Feature
`map`	Starts CVD against a packet capture file or a network interface.
`dnppoints`	Prints the DNP3 setpoints observed. This command has several options to query the database based on various timeframes and to query the database for only certain types of DNP3 packets such as OPERATE or DIRECT-OPERATE commands.
`cmds`	Prints the Telnet commands observed using CVD. Often, cyberattacks originate using Telnet interfaces on various SCADA devices.
`malformed`	Prints the malformed packets observed across all protocols. Provides an overall summary or protocol-specific summary with the bytes observed in protocols.
`flows`	All TCP and UDP connection within the substation, entering the substation, and leaving the substation.
`layer2`	MAC addresses of all the devices observed.
`certs`	Any SSL certificates observed.
`roles`	CVD inferred roles for devices on SCADA network. Based on communication patterns and MAC addresses we infer the function of various devices.
`stop`	Stop CVD and stop all the producers and consumers launched via CVD.

Table 2. A non-exhaustive list of CVD commands supported by the command-line interface.

4 Evaluation

To evaluate CVD, we try to answer the following questions:

- Are the LangSec parsers CVD implements correct?
- Are the LangSec parsers in CVD resilient to fuzzers?
- Are CVD's network interfaces resilient to fuzzers?
- Can CVD detect crafted packet attacks for various protocols?
- Can CVD handle the high rate of traffic in SCADA networks?
- Can CVD visualize various human actions?

We ran our experiments on an Intel Xeon E31245 3.30 GHz processor with four cores and 16GB RAM. We used Apache Kafka version 0.10 and Hammer toolkit version 1.0-rc3.

4.1 Running our LangSec parsers through a large dataset

To ensure that our LangSec parsers cover a wide range of features in various SCADA protocols, we ran data collected from a SCADA tested through our parsers. Apart from running live network captures, CVD can also replay and process previously captured packet captures (PCAP files).

Dataset. We used a dataset provided by Yardley et al. [32]. They collected data from 20 substations and three control centers. These substations and control centers were not provided by an actual industry utility, but they were otherwise realistic, simplified physical substations with both SCADA communications and power equipment in a field-deployed testbed. These were experimental substations spread over two square miles representing three independent crankpaths and fed by three generators connected by real overhead and underground wires. There were also high-voltage substations handling electricity at 13 kV.

Table 3. Our parser correctness experiments. We ran our parsers through a large dataset of SCADA traffic provided by Yardley et al. [32].

Protocol	Number of Substations	Packets parsed Successfully	Total number of packets	Percentage of packets correctly parsed
DNP3	25	1888861	2007277	94.5%
MMS	1	35635	36262	98.2%
C37.118	2	1619479	1619582	99.9%
GOOSE	1	4501	4511	99.7%
SEL Fast Message	1	45802	46737	98.1%
SES-92	2	488147	503244	97%

Each substation included at least four relays and an RTU. The three control centers had RTACs and HMIs. These devices were from at least four different manufacturers. Our dataset includes five hours of traffic. Although most of these substations and control centers ran DNP3, each of the following protocols—SES-92, SEL Fast Message, IEEE C37.118, IEC 61850 GOOSE, and IEC 61850 MMS—was running in one substation each.

The results of our experiment are in Table 3. Our parsers ran with a minimum accuracy of 94%, and most parsers had an accuracy of at least 98%. Our experiments demonstrate that our parsers cover an extensive feature set of these SCADA protocols successfully. Many practical DNP3 implementations support experimental and error-prone features that we are yet to support.

4.2 Fuzzing our parsers

Fuzzing SCADA devices can lead to crashes and denial-of-service attacks if the parsers are vulnerable [33]. As explained in Section III-D, CVD includes a set of

LangSec parsers to ensure the syntactic validity of the SCADA network packets. Since CVD is designed to detect crafted-packet attacks for SCADA protocols, we want to ensure that fuzzing approaches do not crash CVD itself. Fuzzing CVD serves two primary purposes: (i) Parser resilience: ensuring that our parsers do not crash on any input—well-formed, malformed, or random, (ii) Network resilience: testing the network interfaces of our consumers to check if they can handle the high volume of traffic in SCADA networks.

Table 4. Results of our fuzzing experiment with our LangSec parsers.

	Parser Resilience			
Protocol	**Number of packets**	**Unique paths**	**Crashes**	**Hangs**
DNP3	3.62 Million	623	0	0
SES92	637 Million	6	0	0
C37.118	112 Million	5	0	0
GOOSE	1.2 Million	254	0	0
MMS	2.2 Million	13	0	0
SEL Fast	1 Million	6	0	0

Parser Resilience. We fuzzed our C/C++ parsers using AFL++ [34] and our python parsers using Python Fuzz [35]; both coverage-guided fuzzers. To create a fuzzing target for AFL++, we created additional C files called our parsers while supporting the fuzzing tools. AFL++ requires us to compile the program with instrumentation using an `afl-cc` compiler. We then run `afl-fuzz` on the binaries generated with a seed folder. We created a corpus of valid packets for our seed.

Our fuzzing results are in Table 4 and Figure 5. We ran each of our fuzzers for 48 hours. None of our fuzzing executions led to any crashes or unresponsive parsers. Each of our python-fuzz targets ran at least one million permutations through our parser targets. In comparison, our AFL++ targets ran a minimum of three million executions through our parsers.

Table 5. Results of our fuzzing experiment with our LangSec parser network interfaces. We fuzzed our interfaces using *fuzzotron* while setting a maximum limit on number of tries.

	Network Resilience	
Protocol	**Number of connections**	**Crashes**
DNP3	900000	120
SES92	900000	6
C37.118	900000	0
GOOSE	900000	0
MMS	900000	0
SEL Fast	900000	5

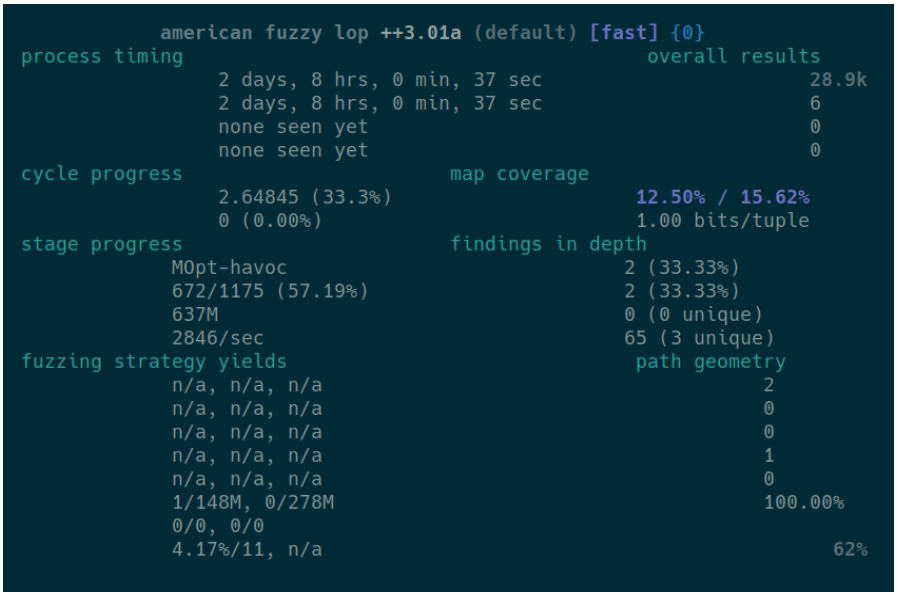

Fig. 5. Fuzzing our SES-92 parser using AFL++.

Network resilience. We implemented all our CVD parsers as Apache Kafka consumers. These consumers receive raw bytes from the producers that they parse and decide if they are safe or no. Since our parsers also include network interfaces, we fuzzed our parsers using fuzzotron [36]. This fuzzing exercise aims to see if our network interfaces are resilient, and they can withstand several connections and drops in connections every second.

We specify what ports each CVD consumer uses and then run fuzzotron targeting those ports. Fuzzotron creates and drops connections to these ports, often violating the TCP state machine. Once the connections are established, fuzzotron also communicates random bytes on the open TCP ports.

Table 5 shows the total number of connections attempted to the CVD consumer by fuzzotron. Using fuzzotron, we detect the number of network timeouts or crashes. We found that three of our consumers encountered no timeouts or crashes, whereas with the other three consumers, at most 0.1% of the packets caused crashes. We observed that we could not reproduce any of these crashes; most of these were introduced due to heavy network loads.

4.3 CVD versus Crafted Packets

To test CVD against crafted packets, we collected malformed DNP3 packets from the Aegis fuzzer [37]. Our sample consisted of 198 malformed DNP3 packets that were generated by mutating well-formed DNP3 packets. These malformed packets

used some of the DNP3 vulnerabilities identified by Crain and Sistrunk [12]. Most of these vulnerabilities were structural or syntactic vulnerabilities.

Since Aegis only supports Modbus and DNP3, we mutated well-formed packets for SEL Fast Message, GOOSE, MMS, IEEE C37.118, and SES92 protocols. We generated 198 packets for each of these protocols. We mutated packets so that they mostly conform to the protocol but violate the specification in specific locations.

We fed these generated packets to our CVD producers, which passed on these packets to the respective CVD consumers. We found that CVD was able to detect all the mutated packets as malformed. Also, none of these malformed packets led to any crashes on any of CVD's parsers.

4.4 Performance of CVD

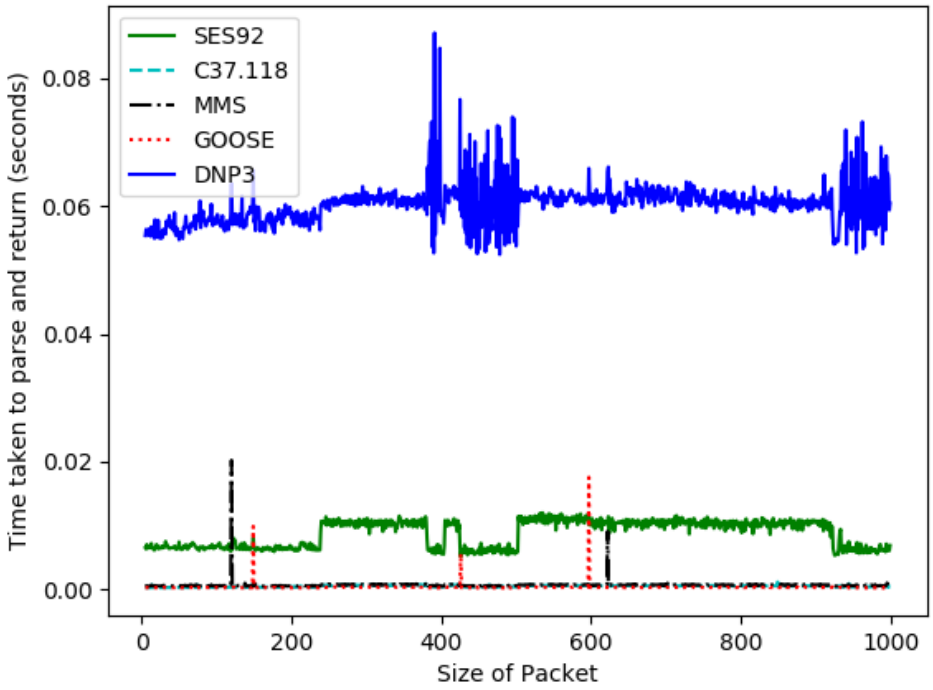

Fig. 6. Most of our parsers show that the amount of time is mostly constant, with minor variations. These times are in the order of micro-seconds for C37.118, MMS, and GOOSE, whereas our DNP3 and SES92 parsers took in the order of milliseconds to run and decide the validity of packets.

To measure our various CVD parsers' performance, we wanted to measure the time it takes for our different parsers to decide whether a packet is well-formed or malformed. We used the same corpus as in Section 4.1.

The results of the above experiment are summarized in Figure 6. We see that our parsers take in the order of milliseconds to decide the safety of packets. We also found that the time taken does not directly depend on the size of packets. There was an added latency due to the publish-subscribe model as well.

We also found that our python-based parsers performed far superior to our C implementations. In CVD, even in our C implementations, we used some Python code to ensure seamless interoperability across all the containers. We found that this feature added additional latency.

4.5 CVD's Visualization Capabilities

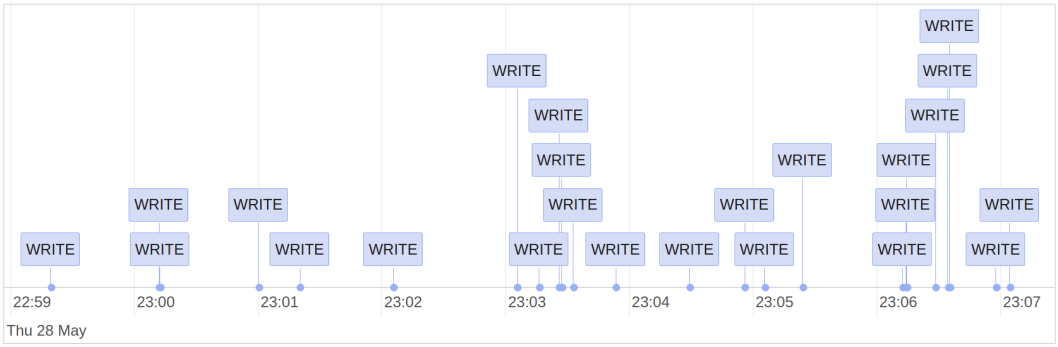

Fig. 7. CVD's Web UI displaying the DNP3 Write operations captured within a ten minute window.

One of CVD's core features is an added visual component so operators can confirm user actions. To validate our various UIs, we constructed several scenarios. We created network configurations for our test network with three relays, two RTUs, and one RTAC from three different manufacturers. This network was a SCADA-only network. Our relays were not controlling any live power settings.

We removed various devices from our network configuration and ran CVD on the network. We found that CVD was successfully able to detect network devices not present in the configuration. Our network configuration mismatch analyzer triggered alerts to alert the user with the network diagram in Figure 8.

Next, to evaluate our DNP3 timeline UI, we crafted a scenario where a malicious program injected various DNP3 WRITE commands in the network. Operators must monitor our DNP3 timeline to ensure no malicious WRITE or OPERATE commands enter the network. Only users can trigger these commands.

When we injected DNP3 WRITE commands and ran CVD, we found that CVD generated various alerts and generated the timeline UI in Figure timeline. If an operator notices the timeline UI in such a state, it could imply that one of the network devices could be sending malicious commands.

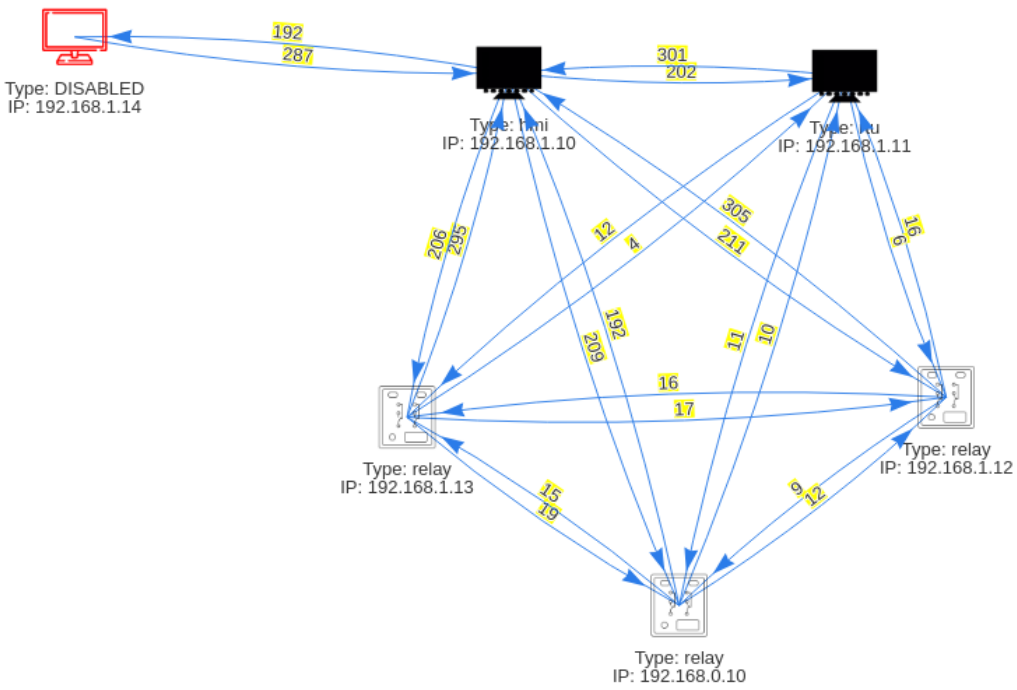

Fig. 8. CVD's network UI. Devices not identified are shown in red.

5 Discussion

Overwhelming operators and users with alerts can always mean that a large chunk of them are false positives. CVD filters packets to decide if they match the headers of a specific protocol and then run the parser on the packet. In case the parser fails, we log this result. This approach is effective for most protocols, but it proved challenging for SES-92 and SEL Fast Message protocols.

Both these protocols do not start with a fixed set of bytes designated to be the header. Instead, we need to run our parsers to decide if the packets conform to one of the two protocols. Although our parser may detect several crafted packet attacks for these two protocols, we may also miss several packets.

Let us suppose we run a single packet through DNP3, SES-92, and MMS, for example. Should we log this packet as an invalid SES-92 packet when the header does not match MMS and DNP3, and also our SES-92 parser cannot successfully parse this packet? This packet could be any protocol such as DNS or NTP that we have not written a parser for.

In this paper, we depended on the Hammer parser-combinator toolkit to write parsers for various SCADA protocols. Hammer does provide functions to free the

parsed AST but does not provide any mechanism to free the parser objects. We found this issue during our fuzzing efforts since our fuzzers reached their memory limits much sooner than anticipated. We found a workaround for this problem and sidestepped this issue. We are in touch with the Hammer development team to add functions to free parser objects to ease fuzzing efforts.

Formally verifying our LangSec parsers is yet another challenge. There are currently no formal parsing algorithms available to handle the context-sensitive languages used in our SCADA communications. There have been some recent efforts to build parsers that extend beyond regular grammars or context-free grammars. Our parser-combinator tool, Hammer, provides bindings to parse these languages with context-sensitive properties. However, Hammer has not been verified. We have fuzzed our implementations extensively to ensure we have not introduced any bugs in our toolkit.

Although we designed CVD as a forensics analysis toolkit to gather data from SCADA networks, it can also be deployed as a network intrusion detection or prevention system (NIDS/IPS). One of the primary reasons we did not follow this direction was that we were introducing latency in the order of a few milliseconds given our tool limitations. Some protocols, such as Goose, specify that latency must not exceed 4 ms. We would violate some of this latency requirement if CVD was deployed as an intrusion prevention system in its current state. With significant engineering improvements and parsers running directly on the hardware using FPGAs, we believe CVD can be re-purposed for this use.

TCP Reassembly poses yet another challenge for parsing efforts. Although most SCADA packets are small enough not to be fragmented, DNP3 and MMS send large packets in rare conditions. In its current state, CVD cannot handle fragmented packets. Introducing a TCP reassembly engine in our parsers requires significant overhead. It would require us to maintain the TCP state as well as buffer packets before decisions are made. We plan to introduce a TCP reassembly engine in the next iteration of CVD with parsers implemented in FPGAs.

6 Conclusions

This paper presented CVD, a novel, distributed tool to monitor SCADA communications networks for crafted input attacks and malicious operations. We presented our novel paradigm of continuous monitoring and data collection to gather forensics from the SCADA network. We showed how we use LangSec-compliant parsers to detect crafted input attacks and malformed packets.

We demonstrated CVD Minion's parsers on a wide range of SCADA protocols and a large dataset of valid SCADA traffic. We also built a GUI that would enable SCADA operators to make security decisions. We fuzzed CVD to show that it is resilient and can detect various SCADA misconfigurations.

Our future work takes us in various directions. We would like to formally verify our parsers and build a new toolkit to generate verified parsers. To do this, we are working on new parsing algorithms for context-sensitive grammars. We are creating highly parallel versions of these parsing algorithms to implement on FPGAs. Finally, we need better user studies to understand SCADA operators' needs and what tools can be useful for them to detect various attacks in their networks.

References

1. J. F. Brenner, "Eyes Wide Shut: The Growing Threat of Cyber Attacks on Industrial Control Systems," *Bulletin of the Atomic Scientists*, vol. 69, no. 5, pp. 15–20, 2013. [Online]. Available: https://doi.org/10.1177/0096340213501372
2. N. Falliere, L. O. Murchu, and E. Chien, "W32.Stuxnet Dossier," *Symantec Security Response*, vol. 1, no. 1, pp. 1–64, 2010.
3. D. U. Case, "Analysis of the cyber attack on the ukrainian power grid," *Electricity Information Sharing and Analysis Center (E-ISAC)*, vol. 388, 2016.
4. B. Zhu, A. Joseph, and S. Sastry, "A taxonomy of cyber attacks on scada systems," in *2011 International conference on internet of things and 4th international conference on cyber, physical and social computing*. IEEE, 2011, pp. 380–388.
5. 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 (SCS)*, vol. 11, 2010, p. 7.
6. P. Eden, A. Blyth, P. Burnap, Y. Cherdantseva, K. Jones, and H. Soulsby, "A forensic taxonomy of scada systems and approach to incident response," in *3rd International Symposium for ICS & SCADA Cyber Security Research 2015 (ICS-CSR 2015) 3*, 2015, pp. 42–51.
7. R. M. van der Knijff, "Control systems/scada forensics, what's the difference?" *Digital Investigation*, vol. 11, no. 3, pp. 160–174, 2014.
8. E. J. Byres and P. Eng, "Cyber security and the pipeline control system," *Lantzville, BC, Canada*, 2009.
9. N. Perlroth and C. Krauss, "A Cyberattack in Saudi Arabia Had a Deadly Goal. Experts Fear Another Try." https://www.nytimes.com/2018/03/15/technology/saudi-arabia-hacks-cyberattacks.html, 2018.
10. N. Falliere, L. O. Murchu, and E. Chien, "W32. stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, no. 6, p. 29, 2011.
11. D. Lee, H. Kim, K. Kim, and P. D. Yoo, "Simulated attack on dnp3 protocol in scada system," in *Proceedings of the 31th Symposium on Cryptography and Information Security, Kagoshima, Japan*, 2014, pp. 21–24.
12. C. Sistrunk and A. Crain, "Project Robus: Serial Killer," https://dale-peterson.com/2014/01/23/s4x14-video-crain-sistrunk-project-robus-master-serial-killer/, 2014.
13. Mitre CVE Repository, "Triangle Microworks DNP3 Errors," http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-10613.
14. M. Patterson, "Hammer Parser Combinator," https://gitlab.special-circumstanc.es/hammer/hammer.
15. S. Bratus, A. J. Crain, S. M. Hallberg, D. P. Hirsch, M. L. Patterson, M. Koo, and S. W. Smith, "Implementing a Vertically Hardened DNP3 Control Stack for Power Applications," in *Proceedings of the 2nd Annual Industrial Control System Security Workshop*, 2016, pp. 45–53.
16. P. Anantharaman, K. Palani, R. Brantley, G. Brown, S. Bratus, and S. W. Smith, "PhasorSec: Protocol Security Filters for Wide Area Measurement Systems," in *IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE, 2018, pp. 1–6.
17. 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. Cham: Springer International Publishing, 2019, pp. 73–85.

18. C. Wright, "Forensics management," *Handbook of SCADA/Control Systems Security*, p. 173, 2013.
19. C. Valli, "Scada forensics with snort ids," 2009.
20. I. Ahmed, S. Obermeier, M. Naedele, and G. G. Richard III, "Scada systems: Challenges for forensic investigators," *Computer*, vol. 45, no. 12, pp. 44–51, 2012.
21. M. Naedele, "Addressing it security for critical control systems," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, 2007, pp. 115–115.
22. F. Adelstein, "Live forensics: Diagnosing your system without killing it first," *Commun. ACM*, vol. 49, no. 2, p. 63–66, Feb. 2006. [Online]. Available: https://doi.org/10.1145/1113034.1113070
23. R. Berthier and W. H. Sanders, "Specification-based intrusion detection for advanced metering infrastructures," in *2011 IEEE 17th Pacific Rim International Symposium on Dependable Computing*. IEEE, 2011, pp. 184–193.
24. L. A. Maglaras and J. Jiang, "Intrusion detection in scada systems using machine learning techniques," in *2014 Science and Information Conference*. IEEE, 2014, pp. 626–631.
25. V. Narayanan and R. B. Bobba, "Learning based anomaly detection for industrial arm applications," in *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*, ser. CPS-SPC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 13–23. [Online]. Available: https://doi.org/10.1145/3264888.3264894
26. J. Verba and M. Milvich, "Idaho National Laboratory Supervisory Control and Data Acquisition Intrusion Detection System (SCADA IDS)," in *IEEE Conference on Technologies for Homeland Security*. IEEE, 2008, pp. 469–473.
27. D. Formby, P. Srinivasan, A. Leonard, J. Rogers, and R. A. Beyah, "Who's in Control of Your Control System? Device Fingerprinting for Cyber-Physical Systems," in *NDSS*, 2016.
28. D. I. Urbina, J. A. Giraldo, A. A. Cardenas, N. O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, and H. Sandberg, "Limiting the Impact of Stealthy Attacks on Industrial Control Systems," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1092–1105.
29. J. Hong, C. Liu, and M. Govindarasu, "Integrated Anomaly Detection for Cyber Security of the Substations," *IEEE Transactions on Smart Grid*, vol. 5, no. 4, pp. 1643–1653, 2014.
30. W. Ren, T. Yardley, and K. Nahrstedt, "EDMAND: Edge-Based Multi-Level Anomaly Detection for SCADA Networks," in *IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE, 2018, pp. 1–7.
31. P. Anantharaman, V. Kothari, J. P. Brady, I. R. Jenkins, S. Ali, M. C. Millian, R. Koppel, J. Blythe, S. Bratus, and S. W. Smith, "Mismorphism: The heart of the weird machine," in *Security Protocols XXVII*, J. Anderson, F. Stajano, B. Christianson, and V. Matyáš, Eds. Cham: Springer International Publishing, 2020, pp. 113–124.
32. T. Yardley, "Building A Physical Testbed For Blackstart Restoration," https://www.youtube.com/watch?v=lGPDmOjaOO4.
33. F. Tacliad, T. D. Nguyen, and M. Gondree, "DoS Exploitation of Allen-Bradley's Legacy Protocol through Fuzz Testing," in *Proceedings of the 3rd Annual Industrial Control System Security Workshop*, 2017, pp. 24–31.
34. A. Fioraldi, D. Maier, H. Eißfeldt, and M. Heuse, "AFL++: Combining incremental steps of fuzzing research," in *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, Aug. 2020.
35. Y. Pats, "Pythonfuzz: Coverage-Guided Fuzz Testing for Python," https://gitlab.com/gitlab-org/security-products/analyzers/fuzzers/pythonfuzz, 2020.
36. D. Denandz, "Fuzzotron: A Simple Network Fuzzer," https://github.com/denandz/fuzzotron, 2018.
37. A. Crain, "Aegis Fuzzer," https://stepfunc.io/products/aegis-fuzzer/.