Chapter 1

A ROADMAP FOR CONVERTING AN ELECTRIC POWER UTILITY NETWORK TO DEFEND AGAINST CRAFTED INPUT

Michael C. Millian, Prashant Anantharaman, Sergey Bratus, Sean W. Smith, and Michael E. Locasto

Abstract In this paper, we propose a concrete roadmap to eliminate the possibility of input-handling vulnerabilities in the OT side of an ICS network by using secure parsing. ICS utilities are responsible for maintaining the integrity of the power grid in the US. A complex communications network is the backbone of these systems. Communication on ICS networks is must be processed correctly and can't crash devices or allow attackers access to devices. Language-Theoretic Security (LangSec) is the practice of secure input handling via hardened parsers. Secure parsers improve ICS network security. Our previous work covers the implementation details of various ICS protocols. Here, we show that the existing collection of LangSec parsers for SCADA protocols offers coverage for the communication needs of an ICS network. We demonstrate a high degree of communications coverage on a network model, discuss the merits of a network guarded by LangSec parsers, and propose a triage procedure to implement such a network. Furthermore, we collect a summary of security benefits and lessons learned.

Keywords: LangSec, parsers, ICS, network

1. Introduction

Devices in Industrial Control System (ICS) networks have supported network communication for the past few decades. Connection to the internet, either directly or via connection to internet-connected devices, has become increasingly commonplace. ICS protocols ultimately control actuators and sensors that affect the operation of the power grid in some way. The cyber-physical nature of these devices paired with network

connection makes the security of network communication a concern with very high priority.

Our goal is to eradicate input-handling vulnerabilities from an ICS network. To motivate this goal, we will discuss in Section 2 how input-handling vulnerabilities are a non-trivial class of vulnerabilities with a long history and many modern examples [5, 6, 8, 19]. For example, our previous work has shown that ICS networks are not immune. Between 2013 and 2014 over 30 input-handling vulnerabilities were discovered in implementations of the DNP3 protocol in devices in ICS networks [2].

Our goal presents various challenges. First, while our Language-theoretic Security approach to secure parsers and protocols has been used to build implementations of several ICS protocols, our results have thus far been constrained mainly to academic domains. Adoption of these protocol implementations in real systems hasn't been high. This work aims to address the discrepancy by making clear the benefits and explaining how to use secure parsers.

We present a notional architecture for an ICS network employing LangSec parsers to secure its communications. Our notional architecture is a general network model that contains the components found in an ICS network. The model loosely maps a network that represents the real world without being tied to a single utility. We show that the LangSec parsers we developed over the past few years offer the necessary coverage of the communication edges on this model. Therefore, we can guard all communication with LangSec parsers.

Second, ICS networks operate a large variety of protocols. Securing a protocol implementation requires a careful examination of a protocol specification. Quite often, device manufacturers subset or fork existing protocols, which results in new protocols that must be analyzed. Other times device manufacturers build proprietary protocols, which makes this step harder. To address this problem, we discuss our proposal for best practice both for creating a new parser and for subsetting or forking an existing protocol.

Third, updating ICS devices offers unique challenges. Because ICS devices perform high priority operations, taking these devices offline or interrupting their ability to communicate is not an option. Nevertheless, some protocols contain inherently unsafe features, and these features must be removed in order to meet LangSec security standards. Therefore we must design a system that allows all devices to continue network operations during the transition. We offer a triage procedure that addresses this need.

The core of our approach relies on *subsetting* existing protocols. A subset of a protocol is just that protocol with particular messages not

allowed. As a trivial example, a certain opcode may be removed if the payload for that opcode is not safe (we define safety in Section 1.2.1). We note that we never add features to a protocol; we only remove unsafe features. As a result, all devices capable of understanding a protocol can understand our safe subset of that protocol.

In summary, this paper makes the following contributions:

- We define and outline a notional architecture for a model of an ICS utility where all communication is gated by LangSec-compliant parsers.
- We offer an analysis of this architecture, focusing on three points: the degree of network coverage we currently offer, trade-offs, and benefits.
- We discuss the details concerning a real ICS utility, e.g., an electric distribution utility, that wants to implement the proposed architecture.

The rest of the paper is organized as follows. Section 2 presents the required background and prior work, Section 3 discusses the architecture of our ICS network, Section 4 provides an analysis of our design, Section 5 discusses the procedure to augment an existing network to be LangSec-compliant, and Section 6 concludes.

2. Background and Prior Work

Input-handling vulnerabilities have plagued networked systems since their creation. Several popular bugs such as Heartbleed [6], Shellshock [17], Rosetta Flash [16], and Apple's goto bug [5] are all input-handling vulnerabilities. Any program that accepts input must validate the input holistically to ensure the input complies with the specifications of the protocol. Input-handling vulnerabilities stem from a violation of the protocol in some way. Typically this is due to a programmer error, such as forgetting to check a condition, but sometimes it is not a violation of the protocol *per se*, but a deeper flaw in the nature of the protocol.

Many bugs and exploits are parsing errors at their root. Some work has been done to show this fact in specific domains, such as USB [11], but there has not been a large-scale effort to label all parsing bugs as such. Another domain-specific work found over 30 input-handling vulnerabilities in various implementations of the DNP3 protocol [2]: Chris Sistrunk and Adam Crain found only a few implementations that were free of vulnerabilities. The immune devices were similar because each deployed a very constrained subset of the DNP3 protocol, which reduced

the attack surface drastically. This finding supports our position that using language subsets is a good way to eliminate input-handling vulnerabilities.

The impact of input-handling vulnerabilities ranges from devices crashing to attackers gaining access to the device. Heartbleed saw attackers exfiltrate data, and Apple's goto bug allowed man-in-the-middle attacks. Credentials could be gathered from these attacks to allow attackers access to systems. Shellshock directly allowed attackers access to systems. Given the range of systems with input-handling vulnerabilities, there is no reason to think that gaining access to ICS devices using ICS protocols is uniquely not possible. Even the case of merely crashing devices cannot be allowed for critical ICS infrastructure. We must protect against input-handling vulnerabilities.

2.1 Language-theoretic Security

Language-theoretic security (LangSec) postulates that all input received by a program must be *validated in entirety* by a parser written based on the *formal grammar* of the input *before any use* by program internals. When a program receives an unanticipated input, it drives the program into a state that the developers did not anticipate. LangSec hardened parsers ensure that the input validation code is explicitly and clearly based on such a grammar, that it is logically separate from the code performing processing on the input, and that a program can never operate on input that has not been exhaustively verified. There is no room for input that is "almost correct" because such input cannot be meaningfully distinct from malicious, crafted input.

Parsers. For the entirety of this paper, we use the following definitions. A *language* is a set of allowed inputs. A *protocol* is a grammar—a list of production rules for a language. A *parser* is an implementation of a protocol in code.

To construct a parser in a way that clearly and explicitly represents the protocol, we use a tool called a *parser combinator*. A parser combinator is a toolkit or framework that allows writing code which visually resembles the formal grammar rather than a collection of `if` statements to check all necessary conditions. Parser combinators dramatically reduces the possibility of programmer error, e.g., forgetting to check a condition. We implement parsers with a parser combinator tool called Hammer [15]. Hammer was developed with a security focus. Safety is measured against the Chomsky hierarchy. The Chomsky hierarchy is a set of classes used to indicate the complexity of a language [3]. These classes range from regular languages (think regular expressions[1])

to recursively enumerable (think Turing machines). Grammars that can be expressed as deterministic-context-free or simpler are considered safe: this limit has been discussed in prior work [13]. Parser combinator toolkits enable developers to build parsers for binary protocols, and specify byte-level constraints about languages. Parser combinators provide a way of representing top-down grammars. Hammer will parse the input into an Abstract Syntax Tree (AST).

2.2 Industrial Control System Security

Industrial Control Systems differ from traditional IT systems and require appropriately different security approaches. ICS networks ultimately control physical devices such as pumps or sensors using short messages with extremely low latency. On the other hand, IT networks are concerned with transferring data using much larger packets and longer latencies. Additionally, ICS networks are typically deeper than IT networks (see Figure 1). Much prior work has been done to ensure the security of these systems. The prevailing security paradigm is defense-in-depth, meaning adding security features and tools at each layer of the network to compound protection against external threats [9].

Secure Design of ICS devices Our work on secure parsers complements the defense-in-depth model. Industrial Control Systems were initially designed with analog equipment for isolated, local use. Over time, ICS networks have been augmented to allow automation and remote access. New connections and capabilities pose new security concerns that the systems were not designed to handle. Our approach is fundamentally about ensuring message security at the protocol design phase. This approach may require some modification of existing protocols if they do not meet the complexity-limitation requirements for security. Because we only restrict protocol complexity, our methods are compatible with current defense-in-depth strategies. We do not require replacing existing security measures and can be used in conjunction with them. Our approach can be used at every level of the defense-in-depth model to increase the security claims at that level and between levels.

3. A Notional Architecture

In this section we discuss the design our notional architecture for a LangSec-compliant ICS, that is, a utility which utilizes parsers developed under a LangSec framework. Our notional architecture contains the general elements and components of a real world network in an abstract representation that is not tied to a single utility.

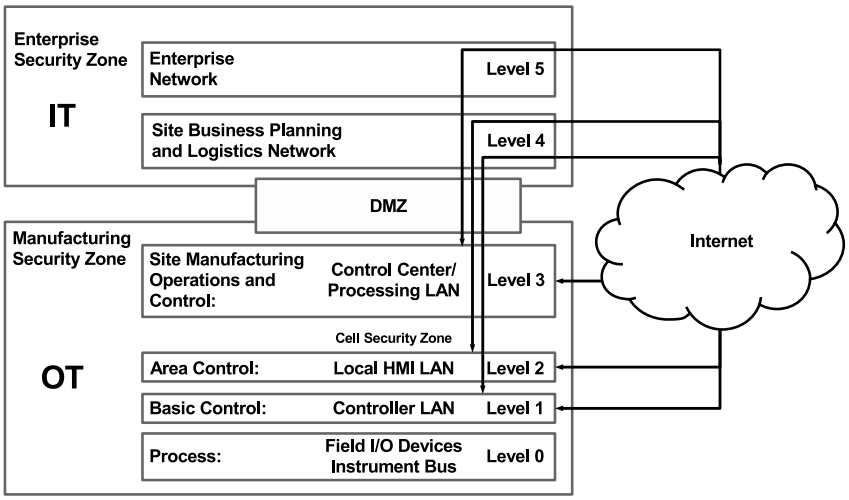

Figure 1. A summary of the Purdue model adapted from [18] annotated with various paths an attacker might use to access the ICS network.

We first enumerate the types of devices found at a utility, which of these are expected to communicate directly, and what protocols are used for each edge of communication. Afterwards, we discuss the list of existing LangSec parsers developed in our lab over the past few years and show that they offer coverage of the communication needs in our model so that all communication may be guarded by LangSec parsers.

To design our notional architecture, we begin with the Purdue model, shown in Figure 1. The Purdue model names six levels in ICS architecture—Level 5: Enterprise Network, Level 4: Business Planning and Logistics Network, Level 3: Site Manufacturing and Operations Control, Level 2: Area Control, Level 1: Basic Control, and Level 0: Process Devices [18]. These levels are divided into several zones, where a zone corresponds to large inter-connectivity. Implementing clear boundaries between zones is considered best practice for enforcing multiple layers of defense.

For the scope of this paper, we focus mainly on Level 2 through Level 0, called the cell security zone or the SCADA zone. This zone is made up of devices that can be found at a substation or directly involved in managing a substation. Level 2 is concerned with monitoring and controlling physical devices. Devices at this level include control center operation workstations, Human Machine Interfaces (HMI), engineering workstations, security event collectors, operations alarm systems, communications front ends, data historians, and network/application ad-

ministrator workstations. Level 1 is concerned with sensing and manipulating physics devices. Devices at this level include dedicated operator workstations, Programmable Logic Controllers (PLCs), control processors, programmable relays, Remote Terminal Units (RTUs), and process specific microcontrollers. Level 0 contains physical devices such as sensors, actuators, motors, process specific automation machinery, and field instrumentation devices [12].

In our work, we had the opportunity to see several real and development network architectures for the SCADA zone. These networks are considered critical infrastructure so the specifics of network topology is protected. Furthermore, there is a large variance in device types and layout from station to station. These constraints move us to develop a notional architecture that is not based on any given utility. We use an architecture that is as generic as possible while still being useful. This notional architecture, shown in Fig. 2 is derived from models described in previous work [20, 10, 18].

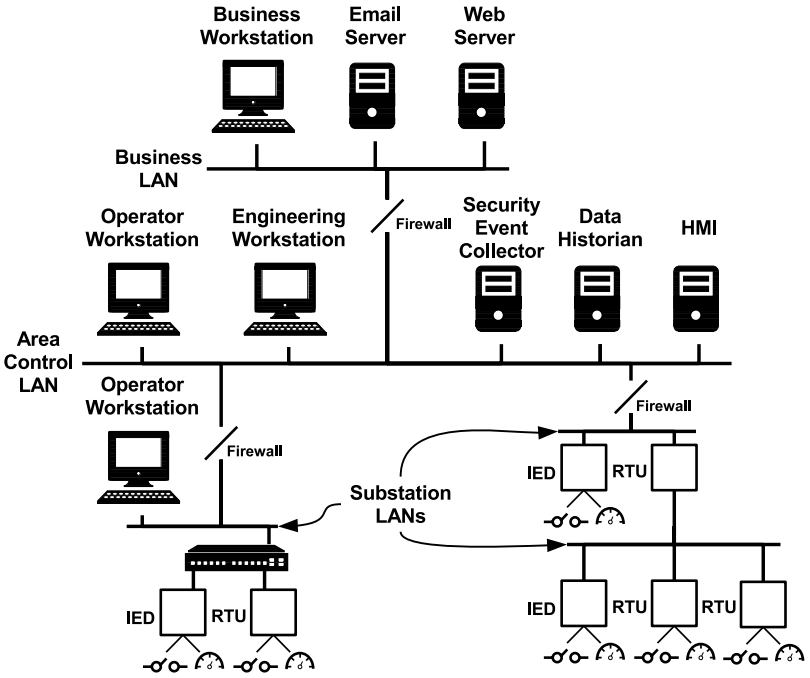

Figure 2. Visual representation of our notional architecture

This generic architecture allows us to express coverage by focusing on a small set of protocols being spoken on edges—e.g., RTU-RTU, RTU-HMI, or Control Center-Substation—without too much concern for the actual device model. What we mean by this is that while there are many protocols for a given edge, e.g., RTU-HMI, our notion of coverage is that we handle at least one of these protocols, therefore, it is feasible add this protection. Vendor-specific protocols exist, but many vendors provide devices capable of speaking a set of standard languages, so this concept of coverage is practical. Using popular protocols allows easier integration into any existing ecosystem. The set of popular protocols following is determined by our own informal surveys and by existing surveys [10].

4. Analysis

4.1 Coverage

To date, our lab has developed secure input handling for the protocols DNP3, MMS, Modbus, 61850-8-1 (GOOSE), IEEE C37.118 [1], SEL Fast Message, HTTP, and Telnet. This section discusses how this selection of protocols offers adaquate coverage of an ICS's communication needs.

DNP3, Manufacturing Message Specification (MMS) and Modbus are considered the de-facto industry standards for communcation needs. These protocols allow for communication between a master station's Human Machine Interface (HMI) and other Remote Terminal Units (RTU), Programmable Logic Controllers (PLC), and other Intelligent Electronic Devices (IED). SEL Fast Message is a vendor-specific protocol for SEL devices that handles much of the same communication. GOOSE is used to broadcast or multicast event data fast and reliably in a substation; GOOSE messages have a maximum latency of 4ms. For Wide Area Networks (WANS), phasor data can be transmitted used IEEE C37.118. To allow communication between workstations and to configure devices, HTTP, FTP, and Telnet may be used.

To reiterate, we cover communication from Level 2 downwards via the popular DNP3, MMS, and Modbus protocols as well as the vendor-specific SEL FastMessage. We cover Level 1 substation-to-physical devices via GOOSE and IEEE C37.118. Finally, we can cover workstation-to-workstation communication over HTTP and Telnet. By implementing parsers for these ICS protocols, we offer a large degree of protection for the majority of low level (on the Purdue Model) OT traffic on most networks. In particular, we offer secure parsing for the protocols responsible for manipulating physical devices, a task which has a very high priority.

The process to add more coverage is discussed in Section 5.

4.2 Benefits

In this section, we discuss the benefits our secure input handling offers.

The largest benefit for using a parser combinator tool is the possibility of provably correct code. The programmer implementing a parser should not have to worry about the correctness of the combinators in the same way that most programmers should not have to worry about the correctness of the compiler.

While the work to prove the correctness of each combinator in Hammer still needs to be done, there are only two possibilities from this work: either each combinator is correct or there is a bug in some combinator. If a bug is found in any individual combinator during this process, it can be corrected without needing to rewrite any parsers built using that combinator (though they will have to be recompiled using the updated combinator library). This is because each combinator performs a function that is fully understood from formal computational theory, so the function signature of each combinator is set, only the internals may change. Once this proof work is complete, every LangSec parser built with combinations of these combinators immediately gets the full benefit of provable safety.

The other benefit of the parser combinator is that it reduces the work of the programmer working on a parser. A key observation of LangSec is that attention to match the complexity of the parser to the complexity of the protocol must be baked in during the development and implementation process. Traditional parser programming resembles a series of `if` statements checking conditions. We have already discussed how it is easy to miss a condition, as in Heartbleed and in Apple's goto bug, but even when a fix is provided, it is still difficult to compare the new parser to the protocol to show they match completely [13].

Using a parser combinator simplifies this comparison task and thus decreases the likelihood of errors and the ease of fixes should errors occur. A parser combinator tool produces code that visually matches the structure of the grammar in order to make verification of equality trivial. Furthermore, a tool like Hammer does not have combinators that would allow programming too-complex constructions like Turing machines. If a programmer cannot implement a protocol feature using a parser combinator, it is an indication that perhaps that feature is not safe and a subset of the protocol without that feature should be used. Ideally, if we see the practice of subsetting protocols to remove unsafe features, this will lead to more protocols designed without unsafe features to begin with.

The goal, and the end result of using a parser combinator, is a parser that accepts all only messages in the protocol specification. The work of implementing protocols safely can then be properly pulled apart from both the work of designing protocols and the work of designing the parser combinator tool.

Our prior work with DNP3 demonstrates this approach in practice for ICS protocols. Implementing the DNP3 parser revealed that the specification indicated that the transport layer payload contain at least 1 byte, but "zero-length APDU (application layer message) would cause unhandled exceptions in certain implementations" [2]. Each protocol we implemented contained such features that are usually handled by an `if` check in the parser. The LangSec approach to parser building considers such a feature of the packets structure as a primary feature when writing the parser. This approach greatly decreases the chance that such a check can be left out.

4.3 Trade-offs

The major trade-off that comes with LangSec parsers is the occasional need to subset the protocol when inherently unsafe features are found in the protocol. The cost associated with this modification is the possibility that existing network devices regularly or occasionally emit messages using the unsafe features. In our experience, these messages represented a small fraction, if any fraction, of actual traffic. However this may not be representative, and the trade-off may be greater depending on use case.

We offer a warning about keeping unsafe features. Unsafe features are much more often about the format of the message than the content of the message, especially when it comes to the kind of data used in ICS networks. We understand the need to send certain kinds of messages and the development cost involved in any change, but the cost is the risk of an attacker crashing devices, exfiltrating data, or taking control of devices, as we have seen many times in many different systems.

5. A Triage Procedure

In this section we discuss the roadmap we have developed to get LangSec parsers into ICS networks so that utilities may realize the security benefits from our architecture. The roadmap is described as a three step plan to engage with utilities and vendors. First, we write LangSec parsers and incorporate them into devices on a per-device basis on a lab bench. Second, we create a virtual substation in a lab. Finally, we work with utilities and vendors to replace parser implementations in device

firmware on the existing product refresh cycle. We close this section with the report of where we currently are on this roadmap.

5.1 Protocols and Devices

The first step is to write and test parsers for the range of protocols used in ICS networks. We have made headway on this task by implementing parsers for eight protocols, as noted earlier. Accomplishing this task in full requires collecting a complete list of protocols used.

For each protocol, we find the protocol specification and then write and test a LangSec parser. At first, this testing is implemented as a bump-in-the-wire. We must ensure that the messages passed by the LangSec parsers allow the normal operation of the devices behind them.

There are difficulties inherent in this procedure to be aware of. First, obtaining documentation for SCADA protocols can be difficult. Many of the protocols specifications we dealt with were not free; costs ranged from a few hundred dollars to several thousand dollars. Second, the protocol specification obtained may not include the complete protocol; some protocols import other protocols to utilize existing work and offset the design burden, e.g., the data encoding format or protocol data unit (PDU). We found these embedded protocols might also not be free, incurring an additional monetary cost. Third, there is neither uniformity nor good practice when it comes to describing the protocol. Some specifications are all prose, and the developer must extract the structure of the grammar. Some protocols are more helpful and include state machines or grammars. Possibly worse though, some include state machines or grammars but modify the function of those machines in prose [2]. This causes divergent implementations depending on how closely the programmer reads the documentation. Until protocol-specification writing improves, a close reading of the specification is necessary. Fourth, sometimes the protocols themselves include unsafe features, and a LangSec parser must subset the protocol before implementing the parser. An example of an unsafe feature is nested length fields. Inclusion of nested length fields requires the inner length agree, e.g., the inner length does not exceed the outer length. This relation cannot be described purely in terms of the structure of the packets using a context-free language since it requires fully parsing both the outer and inner fields to determine agreement. If adherence to the protocol is not maintained in the structure of the packet but left to a check after-the-fact, it is too common for some check or other to be forgotten [5, 6].

After the parsers are written and tested as a bump-in-the-wire to ensure that devices can operate as needed, we must replace native parsers

with LangSec parsers on a by-device basis. This action is required because ICS protocols have maximum latency requirements, and parsing every message twice is not something we can afford. Incorporating LangSec parsers as the native parser also explains one way we provide security benefits beyond traditional Intrusion Detection Systems. IDS programs have difficulty providing insight into encrypted messages, but every message must be decrypted and parsed by a device. Thus, incorporating LangSec parsers as the only native parser in a device adds precise security properties to devices.

5.2 A Virtual Substation

After implementing a full range of parsers for ICS protocols and incorporating them into devices, the next step is to create and operate a virtual substation containing LangSec-hardened devices in a lab. Before deploying LangSec parsers into critical infrastructure, we want guarantees that not only will individual devices will work, but that we understand the consequences of a network of such devices operating under normal and stress conditions.

The virtual substation should be a fully-functioning substation that runs parallel to real-world networks but which does not affect real-world networks. It can ingest either real-time data or replayed captures, and it can operate either real or simulated devices. Developers can analyze the system to ensure correct operation of the virtual substation with no risk to the larger network.

The first step can motivate LangSec-hardened devices through a list of vulnerabilities that LangSec parsers prevent. This step will provide the data that LangSec-hardened devices are operationally viable, which we can present to utilities and vendors.

5.3 Deployment

Our last step involves putting LangSec-hardened devices in the field. This step has all the real-world constraints not found in the lab tests of the first two steps. In particular, ICS networks are slow to incorporate changes, and these changes are designed to last decades. We believe we can take advantage of the existing refresh cycle to push LangSec parsers to devices as a firmware update.

5.4 Where We Are

We are currently in step one on our roadmap. This paper outlines the work we have done writing eight LangSec parsers on a per-protocol basis. We have tested these protocols as bump-in-the-wire tools in confidential

field trials [14]. We have also put LangSec parsers for a proprietary JSON-based protocol into devices at GE [7].

We want to make these parsers publicly available, either as open source or under a similar license. It should not be the case that every programmer needs to implement a parser to read input in a known format. We aim to move towards a world where a standard library exists for each parser. We would love the crypto-idiom "don't roll your own crypto" to extend to parsers - "don't roll your parsers". The number of vulnerabilities we see due to poor parser code makes us feel our motivation here is justified.

Currently, the code for our DNP3 parser and our C37.118 parser is available on GitHub [4]. We are working on getting the rest of the parsers to live under the same master repository.

6. Conclusions

This paper presented the design and implementation of an ICS that used only LangSec-compliant implementations of their protocols.

This paper helps prevent input-handling vulnerabilities in an ICS that could lead to denial of service attacks and cause remote code execution. We achieved this by implementing LangSec-compliant parsers for all the communication protocols in an ICS. We presented the overall architecture of the power utility communications and described various ways utilities could install these parsers in their networks.

Acknowledgement

This material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-16-C-0179 and Department of Energy under Award Number DE-OE0000780.

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, DARPA, United States Government or any agency thereof.

References

[1] Anantharaman, P., Palani, K., Brantley, R., Brown, G., Bratus, S., Smith, S.W.: PhasorSec: Protocol security filters for wide area measurement systems. In: 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm). pp. 1–6 (2018)

[2] Bratus, S., Crain, A.J., Hallberg, S.M., Hirsch, D.P., Patterson, M.L., Koo, M., Smith, S.W.: Implementing a vertically hardened DNP3 control stack for power applications. In: Proceedings of the Second Annual Industrial Control System Security Workshop. pp. 45–53 (2016)

[3] Chomsky, N.: Three models for the description of language. IRE Transactions on information theory **2**(3), 113–124 (1956)

[4] at Dartmouth College, S.S.G.: Dartmouth's PKI/trust lab, `github.com/Dartmouth-Trustlab`

[5] Ducklin, P.: Anatomy of a "goto fail" Apple's SSL bug explained, plus an unofficial patch for OS X!, `nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/`

[6] Durumeric, Z., Li, F., Kasten, J., Amann, J., Beekman, J., Payer, M., Weaver, N., Adrian, D., Paxson, V., Bailey, M., et al.: The matter of heartbleed. In: Proceedings of the 2014 conference on internet measurement conference. pp. 475–488 (2014)

[7] for Energy Delivery Systems (CEDS) R&D Program, C.: From innovation to practice: Re-designing energy delivery systems to survive cyber attacks (2018)

[8] Freeman, J.: Exploit (& fix) Android "master key", `www.saurik.com/id/17`

[9] Galloway, B., Hancke, G.P.: Introduction to industrial control networks. IEEE Communications Surveys & Tutorials **15**, 860–880 (2013)

[10] Hurd, C.M., McCarty, M.V.: A survey of security tools for the industrial control system environment. Tech. Rep. INL/EXT-17-42229, Idaho National Lab (2017)

[11] Johnson, P.C., Bratus, S., Smith, S.W.: Protecting against malicious bits on the wire: Automatically generating a USB protocol parser for a production kernel. In: Proceedings of the Thirty-Third Annual Computer Security Applications Conference. pp. 528–541 (2017)

[12] Lee, R.M.: Detecting the Siemens S7 worm and similar capabilities (2016), `blogs.sans.org/industrial-control-systems/2016/05/`

[13] Momot, F., Bratus, S., Hallberg, S.M., Patterson, M.L.: The seven turrets of babel: A taxonomy of LangSec errors and how to expunge them. In: Cybersecurity Development (SecDev), IEEE. pp. 45–52 (2016)

[14] Newman, L.H.: The hail mary plan to restart a hacked US electric grid. Wired (2018)

[15] Patterson, M.: Parser combinators for binary formats, in C, `github.com/UpstandingHackers/hammer`

[16] Spagnuolo, M.: Abusing JSONP with Rosetta Flash, `miki.it/blog/2014/7/8/abusing-jsonp-with-rosetta-flash/`

[17] Symantec Security Response Team: ShellShock: All you need to know about the bash bug vulnerability, `www.symantec.com/connect/blogs/shellshock-all-you-need-know-about-bash-bug-vulnerability`

[18] Team, I.C.S.C.E.R.: Recommended practice: Improving industrial control system cybersecurity with defense-in-depth strategies. Tech. rep., Department of Homeland Security (2016)

[19] Torpey, K.: The DAO disaster illustrates differing philosophies in Bitcoin and Ethereum, `www.coingecko.com/buzz/dao-disaster-differing-philosophies-bitcoin-ethereum`

[20] Veitch, C.K., Henry, J.M., Richardson, B.T., Hart, D.H.: Microgrid cyber security reference architecture. Tech. Rep. SAND2013-5472, Sandia National Laboratories (2013)