# PhasorSec: Protocol Security Filters for Wide Area Measurement Systems

Prashant Anantharaman*†, Kartik Palani*§, Rafael Brantley†, Galen Brown†, Sergey Bratus†, Sean W. Smith†

* These authors contributed equally to this paper.
†Dartmouth College, Hanover, New Hampshire, USA
{pa, sergey, sws}@cs.dartmouth.edu
rafael.brantley@gmail.com, galen.brown.18@dartmouth.edu
§ University of Illinois at Champaign-Urbana, Champaign, Illinois, USA
palani2@illinois.edu

***Abstract*—The syntactic complexity of SCADA/ICS protocols along with the emphasis on availability has led to the increase in the prevalence of input-handling vulnerabilities in SCADA/ICS protocols. We present PhasorSec, a methodology to produce hardened implementations of the C37.118 protocol in Wide Area Measurement Systems. PhasorSec reduces latency by outsourcing input validation to a separate network filter that exhaustively validates C37.118 packets with respect to the specification of the protocol. We evaluate PhasorSec in terms of CPU-time and development effort and demonstrate its resilience to the state-of-the-art AFL fuzzer.**

## I. Introduction

Phasor measurement units (PMUs) are being widely deployed by power grid transmission owners and generator owners with the primary goal of improving situational awareness about the reliability of the grid. These measurement devices are part of a larger critical infrastructure: wide-area measurement systems (WAMS). WAMS include PMUs, phasor data concentrators (PDCs), central data concentrators (CDCs), GPS receivers and communication and networking infrastructure to facilitate better planning and operation of power systems. The measurements collected by the PMUs are sent upstream to PDCs over a wide area network where they are processed and used by applications. Every application has a desired timing bound on the freshness of data it requires. While some applications like transient stability require access to near real time data, others like postmortem analysis work with data that has been archived by the PDCs. Due to the real time abilities of PMUs, there are plans to integrate the infrastructure with the current control system of the grid, thus making security of the WAMS a high priority.

One of the primary tricks used by attackers to compromise devices is to find vulnerabilities in code that handles input. Regardless of how the code receives input, some processing on the input has to be done to make sure that the input is exactly as intended by the programmer. The lack of proper input recognition has commonly led to critical vulnerabilities like Heartbleed [1] and Shellshock [2].

In the past, our investigation of current supervisory control and data acquisition (SCADA) protocols like DNP3 [3] revealed vulnerabilities in several implementations of the protocol by various vendors. Recently, vulnerabilities were found in implementations of the IEEE C37.118 [4], the most commonly used WAMS communication protocol. These vulnerabilities could allow attackers to send deviously crafted malformed packets to PMUs, which might lead to either a compromised device or to the device crashing, thus affecting availability. With the increased use of PMU measurements in automating real-time protection, such as identifying a collapse of the electric system before-time, and taking corrective action, attackers could force unexpected behavior in the power grid if they were to take control of several PMUs.

This paper presents the initial design and implementation of PhasorSec, an input validation filter for WAMS. PhasorSec is an ingress-based network appliance that inspects packets in the WAMS network for potentially malformed inputs. PhasorSec is designed to filter out inputs that might compromise any WAMS devices by making use of Language-theoretic Security (LangSec) principles. Language-theoretic Security is an upcoming field of security that focuses on validating and handling input safely, using the principles of formal language theory [5]. Several challenges exist in developing such a filter. We describe them below.

First, the IEEE specification of the C37.118 protocol [6] includes plenty of verbose text, without a readable specification of the protocol. This makes it extremely hard and error-prone to implement a parser for the protocol. The protocol is also designed in a way that the PDC first receives a synchronization frame, and then receives a number of data frames that depend on the values in the synchronization frame. This dependency forces a validator to look at the state of the protocol stream, rather than look at the packets individually.

Second, most SCADA devices are comprised of proprietary software that cannot be modified by any means. This proprietary software is mostly optimized for the hardware, but even then, utmost importance is given to availability. Most SCADA protocols prescribe a maximum permitted latency in the communication. Meeting these latency requirements and at the same time validating the input when the synchrophasor data is collected at 60 to 240 measurements per second at the PDC becomes a challenging task.

Third, as we mentioned earlier, such a filter would need to maintain state over multiple flows. There is a possibility of state space explosion and hence a concern over how much memory is available for the filtering process to maintain context.

PhasorSec addresses the above mentioned challenges with several key ideas.

First, PhasorSec filters don't match against an attack signature nor compare to a pattern of acceptable behavior, like in traditional intrusion detection systems, but match each packet and its contents on a grammar, and on the context of the packet based on the protocol state. PhasorSec filters keep track of the actual values in the configuration frames, and use them to make sure that the subsequent values are well-formed as well.

Second, PhasorSec does not require any modification to the code on any of the PMUs and PDCs. Since our filters are in the substation, but do not require any alterations to the actual PDCs or PMUs, they require minimal effort to deploy. They can be directly placed in the network, and configured to receive all packets from the router, and would filter and forward only the valid packets.

Third, PhasorSec does not add any latency to the power grid devices, but a minimal overhead is added to the network. We discuss our overheads in Section 6. Availability is of utmost importance in the Smart Grid. Synchrophasor data is collected at the PDC present in a substation and the PMUs receive command frames that need to be validated. To achieve this, we introduce the PhasorSec placement tool that decides on the optimal placements of PhasorSec given the network topology.

We have implemented a prototype for PhasorSec in C. Our experience shows that our approach is promising: we were able to show that our parsers are resilient to the AFL fuzzer, and our placement tool reduces the latency introduced by parsing. We also implemented PhasorSec as a bump-in-the-wire, and evaluated the performance in terms of overhead added due to the input parsing operation.

Our paper is organized as follows. Section 2 puts PhasorSec in the context of the related work. Section 3 describes the overall architecture, the process of building the C37.118 parsers and our approach to deploy PhasorSec. Section 4 provides some evaluation results. Section 5 suggests several directions for future work and concludes.

## II. Related Work and Motivation

### A. Language-Theoretic Security

LangSec posits that the design of any software must begin with gathering the protocol state machine and the grammar of the protocol. In the past, we have looked at a similar approach to LangSec in the Internet of Things for the MQTT and XMPP protocols [7] and the SCADA/ICS DNP3 protocol [3].

The DNP3 implementation was built as a proxy, that would consume DNP3 packets asynchronously, and would run the parser on it offline. This design decision was made since the DNP3 parser induced latency. To avail the security benefits of having a parser protect a system from input handling vulnerabilities, the parsing has to be performed in real-time. Also, the DNP3 implementation looked at parsing separate packets, without regard for the protocol state. SCADA protocols are traditionally designed such that they rely on a previous packet for some information and context. The MQTT and XMPP implementations perform the parsing synchronously, but add significant overheads due to parsing.

PhasorSec overcomes these shortcomings in the DNP3 and IoT implementations of LangSec. We build a placement tool and outsource the parsing to a separate device on the substation network to serve as a filter for the entire network, while at the same time maintaining the state for each of the connections on the network.

### B. IEEE C37.118 protocol

The C37.118 protocol is used to gather synchronized phasor measurements to make better-informed decisions about the operation of the smart grid. If PMUs within a substation are compromised, and spoof wrong phasor measurements, it could lead to a destabilization of the power system, and the system could take incorrect corrective action. Figure 1 shows the various types of C37.118 protocol messages. The PDC sends commands to the PMU, and the PMU in turn sends a configuration frame specifying the format of the data frames that would follow after that. Header frames can contain any sort of human-readable information (i.e., plain-text).

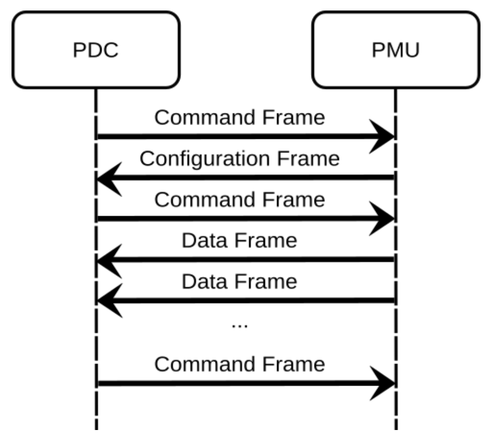


Fig. 1: Flow of messages for the C37.118 protocol.

In the initial state of the PMU, only Command frames are valid. Receiving a Command frame requesting a Header or Configuration packet causes the machine to temporarily accept a single packet of the appropriate type. Due to the complexities of simultaneous communication, this packet need not be the next one received. Instead, the machine will continue accepting valid packets of any type until it receives the one requested, then stop accepting those packets until a second Command packet requests them.

At any point, receiving a Command frame requesting data causes Data frames to be accepted as valid, until a Command frame ending data is received, at which point all Data frames are rejected.

### C. Security of Wide Area Measurement Systems

Intrusion detection in wide area measurement systems has been widely studied. Yang et al. propose an intrusion detection mechanism specific to the C37.118 protocol [8]. Their technique involved making use of a heterogeneous whitelist of known attacks, and a behavior-based approach to detect unknown ones. Stewart et al. provide a very comprehensive set of guidelines to describe what approach is to be taken by operators to ensure confidentiality, data integrity and availability of the grid [9]. Stewart et al. also note that the specifications of the C37.118 protocol do not include any security features, and that it is left to the network layers to enforce the security. The paper also provides several results demonstrating the effectiveness of the safeguards. Although the paper goes in depth about substation security and information security, Stewart et al. do not discuss the issue of input handling in the PMUs.

Coppolino et al. conducted a study of synchrophasor devices (PMU) and phasor data concentrators (PDC) [10]. Coppolino et al. note that telnet was being used to perform a lot of management tasks, and this is susceptible to man-in-the-middle attacks. Also, there was no input validation or sanitization in the PDC application that they examined, which was the open source OpenPDC application. The content of the messages were usually not verified, and the authors note that a host of input validation attacks and bugs are possible including SQL injection and buffer overflows. The authors recognize that the issue of input validation exists in the PDCs and PMUs, but do not talk about ways this could be addressed.

Another important source of motivation for our work is looking at past list of Common Vulnerabilities and Exposures (CVE) and understanding why these errors occurred. In the database we find three CVEs that are specific to the C37.118-2 communication system and PMUs [11], [12], [13]. CVE-2013-2800 and CVE-2013-2801 signify exactly the same problem we are trying to address in this paper. These bugs would expose the devices to a host of memory corruption and denial-of-service attacks. The most valid action would be to reject these messages since they are malformed, and don't satisfy the grammar of the protocol. Prior to the security investigation of DNP3 implementations by Chris Sistrunk and Adam Crain [3], the DNP3 CVE list had just one reported vulnerability. The researchers found over 30 vulnerabilities through their investigations in 2013-2014. We anticipate that a similar investigation of the C37.118 protocol would reveal several more of these vulnerabilities.

## III. Approach

We are using a principled approach to build a high-assurance input validator, and provide it in a way that operators of constrained edge devices do not have to bother about the performance and CPU time. We want to do this by placing these parsers optimally on a level higher on the hierarchical structure followed by wide area measurement systems, and sometimes in the edge device itself when the PMU is known to be sending and receiving critical data.

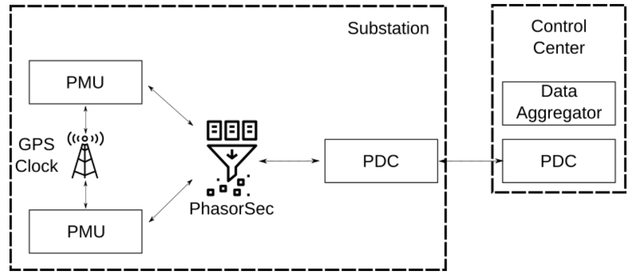

Fig. 2: Overall architecture of PhasorSec. A single substation contains multiple PMUs, which are synchronized using a GPS clock. These PMUs communicate with a single PDC at the substation, which then aggregates data to a PDC in the control center. This diagram shows the placement of PhasorSec in a substation network in the control center.

Our development effort of PhasorSec was divided into three broad parts. First, we build the LangSec parsers for the IEEE C37.118 protocol for the revision of 2011. Second, we introduce PhasorSec as a bump-in-the-wire on a substation network. Finally, we find the most optimal location to place PhasorSec based on the importance of the PMUs and the network topology. Figure 2 demonstrates the purpose of PhasorSec in a substation network.

### A. Designing PhasorSec

The LangSec methodology involves a critical reading of the protocol specification to be able to extract information that is needed to build parsers. We start by understanding what states follow in the protocol, and the messages that are to be accepted by each of these states. The architecture of our parsing methodology can be seen in Figure 3. A clear and direct correlation between the packet format diagram, the grammar and the code validating this grammar is necessary, and is the goal of LangSec, since it helps programmers and auditors alike.

We make use of the Hammer parser combinator toolkit to both describe our extracted grammar for each state of the C37.118 protocol, and at the same time implement a parser for the prootocol. Our coding style makes the C/C++ code more clean and concise, and more readable that the pointer arithmetic-based parsing code usually written.

**Extracting the session language:** To understand how the parser must be built, we need to understand the exact order in which the messages are sent and received. In some cases, messages that aren't supposed to appear one after the other or in a certain order may appear so, and the session language must be able to reject such messages. Figure 1 shows the communication between the PMU and the PDC from which we can extract the individual state machines for both the PMU and the PDC. Figure 2 also shows the session language or the protocol state machine for an individual PDC.

**Extracting grammar from specification:** Usually all protocol specifications have clear descriptions of what are the various packet formats. The specifications also include some

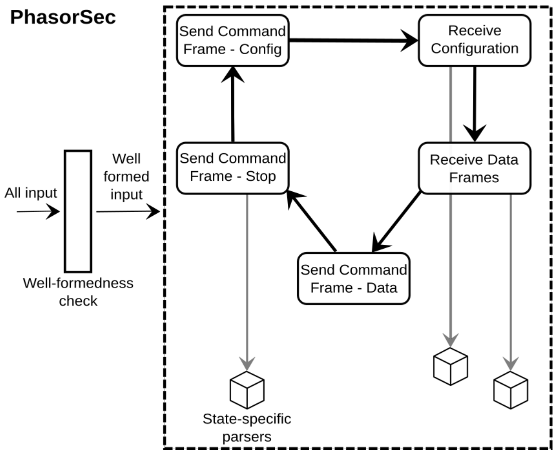


Fig. 3: Parsing methodology in PhasorSec. For protocols involving complex session languages and states, we would first perform a well-formedness check to make sure the message overall conforms to the specification of the protocol, and only then parse the message with respect to the current state of the system.

critical information such as what the boundaries of the various parts of the payload are, and which fields depend on other fields. Context-sensitivity must usually be avoided, and we advocate that we must stick to the easily recognizable regular and context-free languages. A LangSec analysis of the specification would reveal such issues as pitfalls in the specification of protocols [14]. Below is the grammar we were able to extract for the C37.118 protocol. A well-formed PMU frame can comprise any of the four frame types. One thing to note in the config_frame description is that the Station_config field repeats NUM_PMU times. This leads to a complexity in parsing, since this value NUM_PMU has to first be extracted, following which the rest of the packet has to be extracted. The same occurs for the fields PHUNIT, ANUNIT and DGUNIT which all depend on the previous values PHNMR, ANNMR and DGNMR for their lengths.

$$
\begin{aligned}
\text{config\_frame} &\rightarrow \text{Header NUM\_PMU Station\_config}^* \\
\text{Header} &\rightarrow \text{sync framesize idcode soc} \\
&\quad\ \text{fracsec time\_base} \\
\text{Station\_config} &\rightarrow \text{name id\_code FORMAT} \\
&\quad\ \text{PHNMR ANNMR DGNMR} \\
&\quad\ \text{PHUNIT ANUNIT DGUNIT options} \\
\text{options} &\rightarrow \epsilon \mid \text{options DATA\_RATE} \mid \\
&\quad\ \text{options CHKSUM}
\end{aligned}
$$

**Well-formedness check:** We build recognizers for the chosen protocol, and recognize the overall syntax of the message, without actually taking into consideration the session state the receiver is in. This check is really important, since there are some semantic actions involved in checking whether a receiver is in a certain state, and making sure the correct parser is being run on it. When a device is on the receiving end of a stream of

```
header = h_sequence(h_uint16(), h_uint16(), h_uint16(),
    h_uint32(), h_uint32(), h_uint32(), NULL);
station_config = h_sequence(name, id, format, phnmr, annmr,
      dgnmr, phunit, anunit, dgunit, options, NULL);
options = h_sequence(h_optional(h_uint16()),
    h_optional(h_uint16()), NULL);
initial_parse = h_sequence(header, h_uint16(), NULL);
next_parse = h_repeat_n(station_config, num_pmu);
```

Fig. 4: The Hammer-based code for handling the C37.118 configuration frame as described in the grammar above.

completely malformed and invalid messages, these messages get rejected directly at this step without being subject to any semantic actions hence saving CPU time and memory. The placement of the well-formedness check in the code can be seen in Figure 3.

**Parsing based on the session state:** Once the message is checked for overall syntactic validity in the well-formedness check, we check which state our system is in, and deploy that particular parser to be run on the given input to make sure that the message we received is not just structurally correct, but also valid with respect to the current state our system is in. In most protocols that are based on sessions, including the C37.118 protocol we study in depth in this paper, parsing based on the current state is important since these parsers need to be changed based on a set of variables that were set by previous messages. In the C37.118 protocol specifically, the data frame parsers need to be built based on the configuration frame that was received previously. Figure 3 describes how each state of the system has a parser of its own, that gets called if a message was received in that particular state. Figure 4 shows a snippet of how the grammar in the previous section translates directly to a parser in our code snippet written in C. As described, due to the dependency of the fields within the packet, partial parses are necessary.

**PhasorSec as a bump-in-the-wire.** One of the bigger goals of PhasorSec is to make sure that all the C37.118 packets in our scope goes through our parsers, and no device receives a packet that has not been parsed by one of our parsers. To introduce PhasorSec as a bump in the wire, we perform the following:

- The PhasorSec device performs an *arpspoof* on the PMUs telling them that it is the PDC, and the vice versa to the PDC.
- We use *scapy*[1] to recover the packets the PDC is sending or receiving.
- The state of each of the connection is maintained in the form of a finite state machine, and the correct parser is called.

In case of failure to parse a message, PhasorSec logs the message after dropping the packet. If the parsing is successful, PhasorSec forwards it on to the recipient.

## B. Deployment

Applications that depend on synchrophasor data often have real time requirements in the order of seconds. Meeting these

[1]Scapy helps us manipulate packets on the wire - https://scapy.net/

demands can be hard and hence security considerations are often ignored in the pursuit of performance requirements. In deploying PhasorSec, we hope to guarantee end to end delay requirements while providing input validation. Another constraint is the strict security budget that utilities have which makes it expensive to deploy and manage a filter per network link. In Section IV we show that PhasorSec adds a network overhead in the order of a few microseconds and hence even if we were to deploy a filter on every network link, the end to end delay requirements for even the most time critical of applications will not be exceeded. Hence, we define the problem of deploying PhasorSec as maximizing the number of links that are monitored by a filter while guaranteeing that the budget is met [15].

We represent the WAMS network as an undirected graph $G = (W \cup N, L)$, where W is the set of WAMS devices and N is the set of network infrastructure nodes. L is the set of links at which PhasorSec can be deployed.

**Objective function:** Not all network links are created equal i.e. some links carry data from PMUs that provide more valuable measurements than other PMUs. Thus, we can assign an importance to each network link. Let $w(l_i)$ be the importance of link $i$. The objective of PhasorSec deployment is then to maximizie coverage of the most important links.

**Deployment Cost:** The cost of deploying the network appliance is affected not just by the cost of the appliance itself but also the cost of installation which can vary significantly due to geographical location or accessibility of the substation. Let $c(S_j)$ be the cost of placing a filter on a set of links $S_j \subseteq L$.

Hence, the formal definition of deployment is given by the following equation where $x_i = 1$ if link $l_i$ is selected and $y_j = 1$ if a set $S_j$ of links is chosen.

$$\begin{aligned} &\text{maximize} && \sum w(l_i)x_i \\ &\text{subject to} && \sum C(S_j)y_j \leq B; \\ &x_i \in \{0,1\}; \\ &y_j \in \{0,1\}; \end{aligned}$$

Note that this is reformulation of the budgeted maximum coverage problem which is NP-hard [16]. There exists a $1 - \frac{1}{e}$ approximation algorithm to solve the problem [16]. It turns out that the greedy algorithm achieves the best approximation ratio.

## IV. Evaluation

We evaluate our parsers using three broad validation techniques. We performed a static analysis on our system, we fuzz-tested our system, wrote unit tests, and performed CPU-time analysis.

### A. Unit Testing, Static Analysis and Coverage

We make use of the *Infer* tool to perform a static analysis of our system [17]. Our implementation was found to not have any of the categories of errors found by *infer*, namely, null de-references, memory leaks, premature nil termination arguments and resource leaks.

| **Type** | Lines | Percentage |
|---|---|---|
| Line Coverage | 364/485 | 75.1% |
| Function Coverage | 31/34 | 91.2% |

TABLE I: Code coverage of the C37.118 parser using *gcov* and *lcov*. This shows the number of lines and functions that could be reached by our unit testing suite.

| **Parser** | **Total Run-time** | **Crashes** | **Hangs** | **Cycles** |
|---|---|---|---|---|
| Configuration Frame | 25 hours | 0 | 37 | 9452 |
| Data Frame | 25 hours | 0 | 42 | 10300 |

TABLE II: A summary of our AFL fuzz-testing results.

We used a set of unit tests to test our implementation of the C37.118 protocol. To validate our unit testing technique, we used *gcov* to assess the code coverage of our implementation. The results of our coverage are in Table I.

### B. Fuzzing

We make use of coverage-guided fuzz-testing using the *American Fuzzy Lop* fuzzer (AFL). Figure 5 shows that we ran AFL on the configuration frame parser. Our results show that after 25 hours of fuzzing, there were no crashes. Although there were several hangs, these were mostly due to the fact that our parser was stateful.

### C. Timing Analysis

The experiments were run on an Firefly Development Board with a Quad-core ARM Cortex-A17 processor and 2GB of RAM.

| **Frame** | **CPU Time** | **Lines of code** |
|---|---|---|
| Command Frame | 20 $\mu s$ | 29 |
| Configuration Frame | 13 $\mu s$ | 71 |
| Data Frame | 27 $\mu s$ | 56 |
| Header Frame | 80 $\mu s$ | 30 |

TABLE III: We compare the time taken to parse each of the frames and the number of lines of code in each of these parsers.

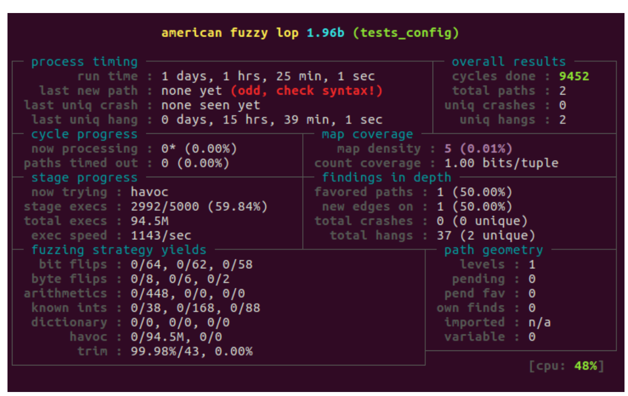


Fig. 5: AFL Fuzzer screenshot showing results of fuzz-testing of our configuration parser written in hammer.

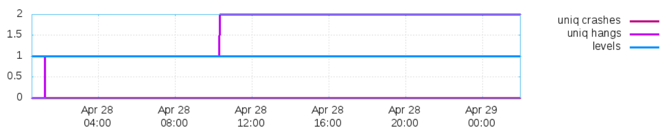


Fig. 6: AFL did not detect any crashes and had only two unique hangs after over 24 hours of testing for both the configuration frame and command frame parsers..

We perform CPU-time analysis on the individual parsers, to understand the overhead which would be introduced. The command frame and the header frame would have to be parsed at the PMU, and the configuration and data frames would be parsed at the PDC which aggregates the data from multiple PMUs. In Table III, we see that the overhead due to the addition of these parsers is very minimal (in the order of micro-seconds) considering that it prevents input-handling vulnerabilities and bugs. We also note that the complex parser we have written was for the configuration frame, which contains the context needed for the data frames to parse the data correctly. Despite performing stateful parsing, we note that we can construct these parsers in under 75 lines of code each.

## V. Conclusion

We showed that a context-aware parser can be built for the C37.118 parser, that is not only resilient to state-of-the-art fuzzing techniques such as AFL, but also does not add much overhead to the devices.

We started with a critical reading of the specification of the IEEE C37.118 protocol, understanding what devices are included in the protocol, and at the same time understanding the syntax and semantics of the messages. We implemented individual parsers for the different messages, that first look at the overall syntax of the message, and then look at the context of the device based on the previous messages.

Although our current design is inspired from our discussions with domain experts, we anticipate that the substation networks are moving slowly towards software-defined networking. In our future work, instead of using an ingress-based network appliance, we would instead like to move to software switches based on P4 [18]. To improve usability of *hammer*, we will also explore using domain specific languages to improve the usability of the parser building methodology.

## Code Availability

The source code of our system is available at https://github.com/Dartmouth-Trustlab/C37.118PMU.

## Acknowledgment

This material is based upon work supported by the 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 United States Government or any agency thereof.

The authors would like to thank Dr. David Nicol for his insightful ideas during the initial phases of the project. The authors would also like to thank Dr. Carl Hauser and Tim Yardley for their help and guidance in obtaining datasets for our research.

## References

[1] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman, "The matter of heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, ser. IMC '14. New York, NY, USA: ACM, 2014, pp. 475–488. [Online]. Available: http://doi.acm.org/10.1145/2663716.2663755

[2] C. Mary, "Shellshock attack on linux systems–bash," *International Research Journal of Engineering and Technology*, vol. 2, no. 8, pp. 1322–1325, 2015.

[3] 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*, ser. ICSS '16. New York, NY, USA: ACM, 2016, pp. 45–53. [Online]. Available: http://doi.acm.org/10.1145/3018981.3018985

[4] Industrial Control Systems Cyber Emergency Response Team, "OSIsoft Multiple Vulnerabilities," https://ics-cert.us-cert.gov/advisories/ICSA-13-225-02, 2015.

[5] M. Sipser, *Introduction to the Theory of Computation*. Thomson Course Technology Boston, 2006, vol. 2.

[6] "IEEE Standard for Synchrophasor Data Transfer for Power Systems," *IEEE Std C37.118.2-2011 (Revision of IEEE Std C37.118-2005)*, pp. 1–53, Dec 2011.

[7] P. Anantharaman, M. Locasto, G. F. Ciocarlie, and U. Lindqvist, "Building Hardened Internet-of-Things Clients with Language-theoretic Security," in *4th Language-theoretic Security Workshop at IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE, May 2017, pp. 120–126.

[8] Y. Yang, K. McLaughlin, S. Sezer, T. Littler, B. Pranggono, P. Brogan, and H. Wang, "Intrusion detection system for network security in synchrophasor systems," *IET Conference Proceedings*, pp. 246–252(6), January 2013. [Online]. Available: http://digital-library.theiet.org/content/conferences/10.1049/cp.2013.0059

[9] J. Stewart, T. Maufer, R. Smith, C. Anderson, and E. Ersonmez, "Synchrophasor security practices," *Tennessee Valley Authority Report*, 2011.

[10] L. Coppolino, S. D'Antonio, I. A. Elia, and L. Romano, *Security Analysis of Smart Grid Data Collection Technologies*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 143–156. [Online]. Available: https://doi.org/10.1007/978-3-642-24270-0_11

[11] Mitre, "Arbiter Power Sentinel Denial of Service Attack," http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-3012, 2012.

[12] ——, "OSIsoft PI Interface for IEEE C37.118 Memory Corruption and Consumption through crafted packets," http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-2800, 2012.

[13] ——, "OSIsoft PI Interface for IEEE C37.118 shutdown and data collection outages through crafted packets," http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-2801, 2012.

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

[15] P. Kansal and A. Bose, "Bandwidth and latency requirements for smart transmission grid applications," *IEEE Transactions on Smart Grid*, vol. 3, no. 3, pp. 1344–1352, 2012.

[16] S. Khuller, A. Mossy, and J. S. Naorz, "The budgeted maximum coverage problem."

[17] C. Calcagno, D. Distefano, J. Dubreil, D. Gabi, P. Hooimeijer, M. Luca, P. W. O'Hearn, I. Papakonstantinou, J. Purbrick, and D. Rodriguez, "Moving Fast with Software Verification." in *7th NASA Formal Methods International Symposium (NFM)*. Pasadena, CA, USA: Springer, 2015, pp. 3–6.

[18] N. Lopes, N. Bjørner, N. McKeown, A. Rybalchenko, D. Talayco, and G. Varghese, "Automatically verifying reachability and well-formedness in P4 Networks," Technical Report, Tech. Rep., 2016.