2019 IEEE/IFIP International Conference on Dependable Systems and Networks Industry Track

A Parser for Deep Packet Inspection of IEC-104: A Practical Solution for Industrial Applications

Justyna Chromik
University of Twente
Enschede, the Netherlands
j.j.chromik@utwente.nl

Anne Remke
Westfälische Wilhelms-Universität
Münster, Germany
anne.remke@wwu.de

Boudewijn R. Haverkort
Tilburg University
Tilburg, the Netherlands
b.r.h.m.haverkort@uvt.nl

Gerard Geist
Coteq Netbeheer
Almelo, the Netherlands
g.geist@coteq.nl

***Abstract*—We present a practical solution for deep packet inspection for IEC-104 SCADA traffic, which can be used in monitoring approaches to ensure the dependable operation of critical systems. We re-implement an outdated parser and extend it to also parse the content of individual IEC-104 packets and to extract information relevant for monitoring and securing the physical processes being controlled. The deep packet inspection framework *Spicy* was used for the implementation, which allows for easy extensibility in the future. To illustrate the feasibility of the proposed solution, the throughput obtained when using the parser in combination with the monitoring tool Zeek[1] has been evaluated for traces of different lengths. The traces have been captured in an operating electrical distribution field station with a single RTU.**

***Keywords*-parser; IEC-104; SCADA; IDS; Zeek;**

I. Introduction

SCADA (Supervisory Control and Data Acquisition) systems are used to monitor and control large physical infrastructures, like electricity transmission and distribution systems. In such systems, traditionally, little attention was paid to security. Today, however, these systems are connected to the Internet to provide remote control capabilities, which makes them vulnerable to adversary parties, which aim at disrupting the controlled process; we have seen various examples of this recently [1]. Due to this increased attack surface on SCADA systems, many packet inspection and monitoring mechanisms have been proposed [2]–[6]. Deep packet inspection requires the capability to read and interpret protocol-specific information from captured packets in real-time. However, many SCADA protocols are not yet supported by adequate parsers providing real-time monitoring capabilities, as is the case for IEC-60870-5-104 (also named, for short, IEC-104). This protocol is often used in electrical industries in Europe [7]. With recent malware developed to abuse the lack of security mechanisms in IEC-104 [1], it is of utmost importance to include real-time monitoring capabilities for this protocol.

This paper presents a dedicated IEC-104 parser, implemented using the Spicy deep packet inspection framework [8], and its connection to the Zeek[1] network monitoring tool [9]. We extend the parser proposed by Udd et al. [10] by implementing 34 IEC-104 functions (referenced by Type IDs), and extracting relevant process variables from the IEC-104 Data Units. By making the parser available, we hope to share a practical solution for deep packet inspection, which will enable further research w.r.t. monitoring, and intrusion detection of the IEC-104 protocol to enhance the dependability of the controlled system.

Related work: Intrusion detection techniques for critical infrastructures require dedicated approaches. Standard signature-based approaches such as whitelisting [11] and anomaly-based approaches [2] help to solve only part of the problem. IDS for SCADA systems can be improved when also information about the physical process is taken into consideration [3]–[6]. Zeek is a network monitoring tool with a modular structure that allows using so-called policies to analyze system traffic; it supports the easy inclusion of new parsers, and is often used in SCADA systems [5], [6], [10]. However, the parsing of the very important protocol IEC-104 is currently not supported. To the best of our knowledge, only one parser has been proposed for IEC-104, using a compiler-assisted tool called BinPac++ [10]. However, BinPac++ is no longer in use (it was replaced by Spicy [8]) which limits its current usability. Moreover, Udd et al. [10] implemented only 5 different Type IDs and was limited to parsing only the control information w.r.t. the connection from the IEC-104 packets, and not the contents of the packets. In contrast, the solution presented here allows for parsing the content of the most relevant Application Service Data Units (ASDU) for monitoring individual substations.

This paper is further organized as follows. Section II explains our implementation of the IEC-104 parser and its use in process-aware policies, Section III evaluates its performance on traces of various lengths, and Section IV concludes the paper with pointers to future work.

II. IEC-104 protocol analyzer

This section presents the main components of the IEC-104 parser and illustrates its use within an Intrusion Detection System implemented in Zeek [9].

[1]The Bro Network Monitor has recently been renamed into Zeek; we adhere to the new name.

Table I
IEC-104 Protocol Type IDs

Number	Type ID	Number	Type ID
1	M_SP_NA_1	49	C_SE_NB_1
2	M_SP_TA_1	50	C_SE_NC_1
3	M_DP_NA_1	51	C_BO_NA_1
5	M_ST_NA_1	58	C_SC_TA_1
7	M_BO_NA_1	59	C_DC_TA_1
9	M_ME_NA_1	60	C_RC_TA_1
11	M_ME_NB_1	61	C_SE_TA_1
13	M_ME_NC_1	62	C_SE_TB_1
21	M_ME_ND_1	63	C_SE_TC_1
30	M_SP_TB_1	64	C_BO_TA_1
31	M_DP_TB_1	70	M_EI_NA_1
32	M_ST_TB_1	100	C_IC_NA_1
33	M_BO_TB_1	101	C_CI_NA_1
34	M_ME_TD_1	102	C_RD_NA_1
35	M_ME_TE_1	103	C_CS_NA_1
36	M_ME_TF_1	107	C_TS_TA_1
45	C_SC_NA_1	142	proprietary
46	C_DC_NA_1	143	proprietary
47	C_RC_NA_1	200	proprietary
48	C_SE_NA_1		

A. *IEC-104 protocol and parser*

Every IEC-104 packet, a so-called Application Protocol Data Unit (APDU), contains a header called Application Protocol Control Information (APCI). S-frames (for numbered supervisory functions) and U-frames (for unnumbered control functions) are built from only the APCI. I-frames (used for information transfer), consist additionally of Application Service Data Units (ASDUs). ASDUs determine what kind of function (the so-called Type ID) they carry. They can contain up to 127 Information Objects (IOs), referring to different addresses on the RTU that is being controlled.

Zeek uses a set of protocol parsers to process network data, and the information generated is analyzed, e.g., to detect intrusions. For currently unsupported protocols, a parser can be built using the deep packet inspection framework called Spicy [8]. It provides a format specification language and a toolchain that compiles the protocol specification into a robust and efficient native code protocol parser. We extend previous work of Udd et al. [10] to define the syntax of the IEC-104 protocol and implement 34 protocol Type IDs, as listed in Table I. The five Type IDs highlighted in orange were implemented already in [10]. To summarize the applicability of the proposed solution, we characterize the implemented Type IDs below.

The measurement information from field stations to the control room is transported using data types with Type IDs beginning with 'M_' (monitoring direction). The controlling functions sent from the control room to field stations are sent using Type IDs with names beginning with 'C_' (control direction). Process values are sent using Type IDs and data format defined by the operator. For example, measurement of the current could be sent either using a normalized value with Type ID 9 or 34, a scaled value with Type ID 11 or 35, or as a floating point using Type ID 13 or 36. Type IDs 1, 3, 5, 7, 9, 11 and 13 transport various data types without time tags. They transport single point information, double point information, step position, bit string, normalized or scaled measured values and floating points, respectively. Type IDs 30–36 transport the same data types but with time tag of format CP56Time2a as defined in the IEC-60870-5-101 standard. Type IDs 45–51 refer to commands of setting values for the same data types as listed before, without time tag, while Type IDs 58–64 do the same and include the time tag of format CP56Time2a. Type ID 2 is not supported by IEC-104, as it contains a single point information with an outdated time tag CP24Time2a, defined in IEC-101. Type IDs 100 and 101 are the interrogation and counter-interrogation commands, respectively. They are always sent after establishing a connection between the control room and the field station, and when the control room requests the most up-to-date measurements in the field station.

With the Type IDs described above, it is possible to parse traffic containing the most commonly used functions. The parser is available on Github[2] and can be tested using exemplary IEC-104 traffic[3].

B. *Connecting the parser and Zeek*

Having specified the grammar of the IEC-104 protocol and compiled a parser, it can be connected to Zeek to parse network traffic. When processing pre-defined objects within the Spicy code, events are generated and evaluated within Zeek. For example, processing a control function-related U-frame triggers an event as follows:

```
on T104::Apci if (self.ctrl.mode == 3) ->
       event t104::u($conn);
```

Here, if parser `T104` encounters an `Apci` object with property `ctrl.mode` equal to 3, a `t104::u` event is created with argument `$conn`, denoting the connection information.

Depending on the desired level of detail, various events can be created. For example, in order to analyze the values from a single Information Object within an ASDU carrying multiple objects, it may be necessary to create events for every created object. For example, for analyzing the content of an ASDU containing Type ID M_ME_TD_1 (nr. 34), i.e., containing the monitoring information for a measured value in normalized format with time tag CP56Time2a, one would need to export the type for that Type ID and create the following event:

```
on T104::Measured_Val_Normalized_Val_TTCP56 ->
       event t104::m_me_td_1s
       ($conn, T104::bro_m_me_td_1(self));
```

The event generated above has two arguments: `$conn` is the connection information, while `T104::bro_m_me_td_1()` is a function call that returns `self` (ASDU information). This function is located in the *.spicy file and defines the tuple passed to Zeek as an argument. For the above Type ID with measured values in normalized format and time tag, we obtain the Object's address and the normalized value measured for that Object.

[2]Available at https://github.com/jjchromik/hilti-104.
[3]https://wiki.wireshark.org/SampleCaptures\#IEC_60870-5-104

```
tuple <uint64, double>
bro_m_me_td_1 (asdu :
    Measured_Val_Normalized_Val_TTCP56) {
        return (asdu.info_obj_addr,
        asdu.normalized_value); }
```

This function allows us to access the normalized value of the Information Object in Zeek, based on which, detection policies can be implemented, as explained next.

C. Policies for IEC-104

The enhanced parser allows defining network-level policies, like whitelists [11], and include them in Zeek. In the following, we will illustrate how policies accessing ASDU fields can be specified for process monitoring, as proposed in [4], [12].

policy = rule $\rightarrow$ **action**
rule = condition $(OR|AND)$ **condition** $(OR|AND)$ ...
condition = state[ioa] $\bowtie$ **reference_value**

$\bowtie \in \{<, \leq, >, \geq, =, \neq\}$
action $\in$ { block, log, display alert }

The policy consists of a rule that has to be satisfied in order to trigger one of the three policy actions. The monitoring tool can either block the traffic, log the event, or display an alert. Every rule consists of several conditions that can be connected via an OR or an AND operator. A condition compares a state value to a predefined reference value.

When the monitoring tool processes the traffic in the network and evaluates the policies in real time, an event is triggered when a specific Type ID instance is parsed within a packet. Upon an event, depending on the Type ID, the monitoring tool may pre-calculate or update the system state, or idle, before the execution of the policy is triggered.

event **type_id** $\rightarrow$ **preparation**
$\rightarrow$ **policy**

preparation $\in$ { precalculation, update, $\varnothing$ }

We distinguish three sets of Type IDs: (i) the unused Type IDs (**typeid_un**), (ii) the Type IDs in the monitoring direction (**typeid_m**) and (iii) the Type IDs in the control direction (**typeid_c**), as summarized below.

typeid $\in$ { **typeid_un**, **typeid_m**, **typeid_c** }
event **typeid_un** $\rightarrow$ **action**
event **typeid_m** $\rightarrow$ **state_m[ioa] = value**
$\rightarrow$ **policy(state_m)**
event **typeid_c** $\rightarrow$ **state_c[ioa] = value**
$\rightarrow$ **state_c =** f (**state_c[ioa]**, **state_m**)
$\rightarrow$ **policy(state_c)**

ioa $\in$ [0; 65535] (IOA range is defined in the standard [13])

In case of an unused Type ID, the monitoring tool directly performs an action, e.g., displays a warning. For Type IDs in the monitoring direction, i.e. readings, beginning with ‘M_’ (c.f. Table I), the measured values is extracted and stored in state_m. The policy evaluates that state.

If Type IDs in the control direction appear (commands), e.g., numbers 45–48 in Table I), the information about the requested change to the system is extracted and stored in state_c[ioa]. Additionally, if a physical relationship f like consistency or safety [14] has been defined for elements of the system state, it can be re-calculated based on previously measured state values. Finally, the last action triggered by a typeid_c event is the evaluation of state_c w.r.t. the policy.

III. Applying the Parser in Practice

To illustrate the feasibility of the proposed parser, we evaluated its throughput for varying flow lengths. We obtained an IEC-104 traffic trace on March 2, 2018 at a Dutch power distribution station, operated by our industrial partner Coteq. SCADA networks in general have a relatively low throughput, in this capture the IEC-104 throughput was around 0.1 packets per second (pps). To stress-test the monitoring tool linked to the parser, we used the Scapy Python library to create four groups of IEC-104 traffic traces, each with a different number of packets per TCP flow, i.e., 1048, 10130, 50043 and 100233 packets per flow. From these, we created different traffic traces, with total lengths ranging between 20000 and 500000 packets. These traces have each been processed ten times on an Oracle VM VirtualBox with two logical processors with 2.9 GHz core and 4 GB RAM. The resulting processing time (in seconds) and throughput (in pps) are shown in Figures 1 and 2, together with 95% confidence intervals.

Figure 1 shows that the processing time for each TCP flow length increases linearly with the total trace length. However, the longer a single TCP flow is, the faster the processing time increases. The throughput as presented in Figure 2, however, shows that for different TCP flow lengths, the throughput of the monitoring tool reaches a constant (albeit different) value. Moreover, for the shorter TCP flows (1000 and 10000 packets), the throughput is initially higher and then decreases until that constant value is reached.

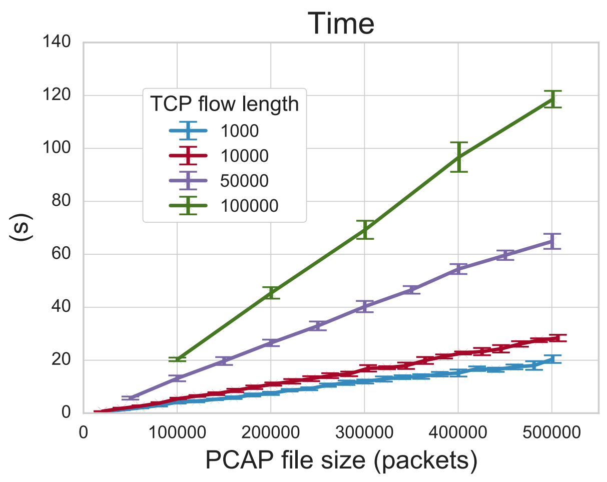

Figure 1. Processing time for multiple TCP flow lengths.

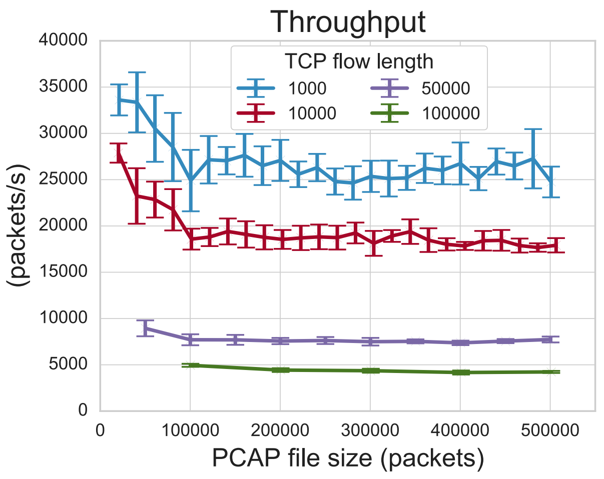

Figure 2. Throughput as function of file size for multiple TCP flow lengths.

The processing times have been measured within Zeek and the measurements were only started after the initialization phase of Zeek was finished. The measurements demonstrate that the performance of Zeek is strongly influenced by the total flow length in the trace. As indicated on the Zeek website[4], its performance highly depends on various parameters and currently no performance benchmarks are available for comparison. As SCADA traffic usually consists of long TCP flows [11], it is important to be aware of the throughput drop for such long TCP flows. The obtained results are based on a single trace only, hence, may not be generic enough to fully evaluate the performance of our parser. However, we do believe that the proposed parser in combination with Zeek is fast enough to be used for real-time monitoring of SCADA traffic on single RTUs (as also proposed in related work [14]), since such networks have a relatively low throughput.

IV. Conclusion

Monitoring SCADA traffic can help to ensure a dependable operation of critical systems, such as power distribution. The parser presented here for IEC-104, builds on the Spicy framework and is able to extract traffic- and control-relevant information by deep packet inspection. Based on a real IEC-104 trace, taken at a Dutch power distribution station, we have created IEC-104 traces with variable length. We have shown that the realized throughput is sufficient to employ the parser in real-time with Zeek in real SCADA systems. The evaluation also shows that the performance of Zeek highly depends on the offered load. We demonstrated that the proposed parser provides a practical solution for inspecting IEC-104 packets in real-time.

[4]https://www.zeek.org/development/projects/benchmark.html

Acknowledgment

Funded by NWO MOSES project, grant nr. 628.001.012.

References

[1] ICS-CERT, "Alert (TA17-163A) CrashOverride Malware," released June 12, 2017, available online: https://www.us-cert.gov/ncas/alerts/TA17-163A. Accessed on: 24 Apr 2018.

[2] M. Caselli, E. Zambon, and F. Kargl, "Sequence-aware intrusion detection in industrial control systems," in *1st ACM Work. on CPSS*. ACM, 2015, pp. 13–24.

[3] J. J. Chromik, A. Remke, and B. R. Haverkort, "Improving SCADA security of a local process with a power grid model," in *4th Int. Symp. for ICS-CSR*. BCS, 2016, pp. 114–123.

[4] I. N. Fovino, A. Carcano, T. D. L. Murel, A. Trombetta, and M. Masera, "Modbus/DNP3 state-based intrusion detection system," in *24th IEEE Int. Conf. on Advanced Information Networking and Applications*. IEEE, 2010, pp. 729–736.

[5] H. Lin, A. Slagell, C. Di Martino, Z. Kalbarczyk, and R. K. Iyer, "Adapting Bro into SCADA: building a specification-based intrusion detection system for the DNP3 protocol," in *8th Annual Cyber Security and Information Intelligence Research Workshop*. ACM, 2013, p. 5.

[6] J. Nivethan and M. Papa, "A SCADA Intrusion Detection Framework that Incorporates Process Semantics," in *11th Annual Cyber and Information Security Research Conference*. ACM, 2016, p. 6.

[7] G. R. Clarke, D. Reynders, and E. Wright, *Practical modern SCADA protocols: DNP3, 60870.5 and related systems*. Newnes, 2004.

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

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

[10] R. Udd, M. Asplund, S. Nadjm-Tehrani, M. Kazemtabrizi, and M. Ekstedt, "Exploiting bro for intrusion detection in a SCADA system," in *2nd ACM Int. Work. on CPSS*. ACM, 2016, pp. 44–51.

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

[12] J. J. Chromik, A. Remke, and B. R. Haverkort, "Bro in SCADA: dynamic intrusion detection policies based on a system model," in *5th International Symposium for ICS-CSR 2018*. BCS, 2018, pp. 112–121.

[13] IEC-104, "IEC TS 60870-5-7:2013," TC 57 - Power systems management and associated information exchange, Geneva, Technical Specification, 2013.

[14] J. J. Chromik, A. Remke, and B. R. Haverkort, "An integrated testbed for locally monitoring SCADA systems in smart grids," *Energy Informatics*, pp. 1–29, 2018.