



Fuzzing of SCADA Protocols used in Smart Grids

Eugen Winter | +(49) 228 50212-648 | eugen.winter@fkie.fraunhofer.de

Michael Rademacher | +(49) 228 50212-623 | michael.rademacher@fkie.fraunhofer.de

Motivation and Objective

Today's energy grids are on the verge of becoming smart. The main difference between a common grid widely used today and a smart grid, aimed at in the near future, is the high level of digitization, which helps to handle more complex tasks efficiently and effectively [1]. The gathering of information from diverse components in the smart grid using well-defined communication protocols is imperative. Consequently, several protocols have been proposed, such as IEC 61850, IEC 60870-5-104, or Modbus.

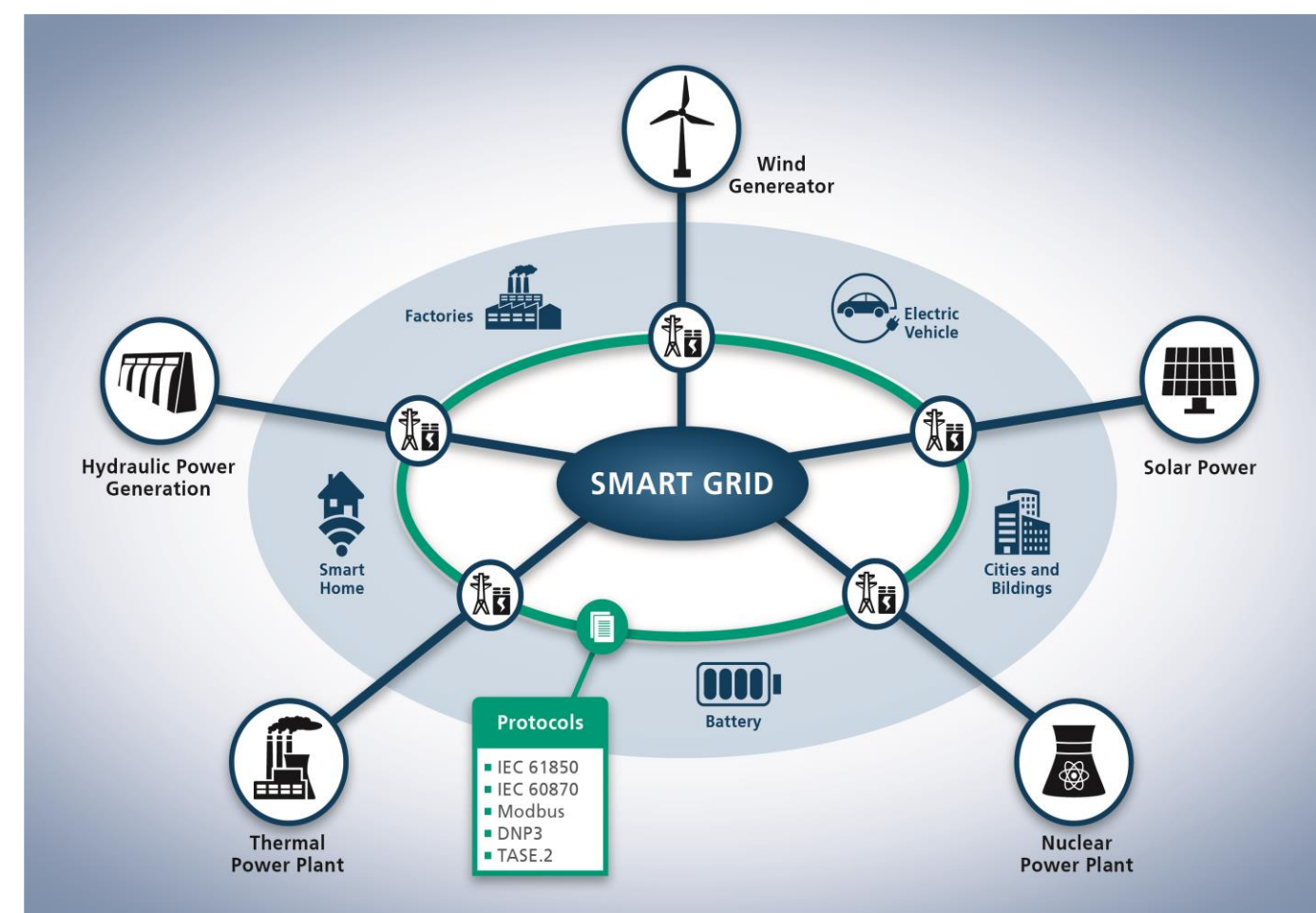


Figure 1: Smart Grid

A long-lasting blackout can have a severe impact on our daily lives. Therefore, the reliable operation of an energy grid is crucial. However, the increasing digitization of smart grids imposes a new threat — attacks on the digital infrastructure. Thus, information technology security for smart grids is becoming an important factor.

This work explores the security of the IEC 61850 protocol standard using a technique called fuzzing. Several studies deal with fuzzing of the IEC 61850 standard [2, 3, 4, 5, 6]. However, to the best of our knowledge, the fuzzing tools of these studies were not published. Therefore, we or other researchers are unable to reproduce or build upon the previous results.

In this work, we use fuzzing to reveal further unknown weaknesses in a frequently used implementation of the IEC 61850 protocol standard. The fuzzer we developed will be made publicly available to improve future security analysis of communication protocols used in the digital energy sector. Besides, we describe our methodology to facilitate further experimentation.

Methodology

The generative fuzzing approach we propose is applied to an existing open-source library of the IEC 61850 standard [7]. This library offers implementations of a Manufacturing Messaging Specification (MMS) server and Generic Object Oriented Substation Events (GOOSE) and Sampled Values (SV) subscribers. Since the library is publicly accessible, regularly maintained, and a commercial license with support is offered, we assume that it is used in real hardware and commercial projects.

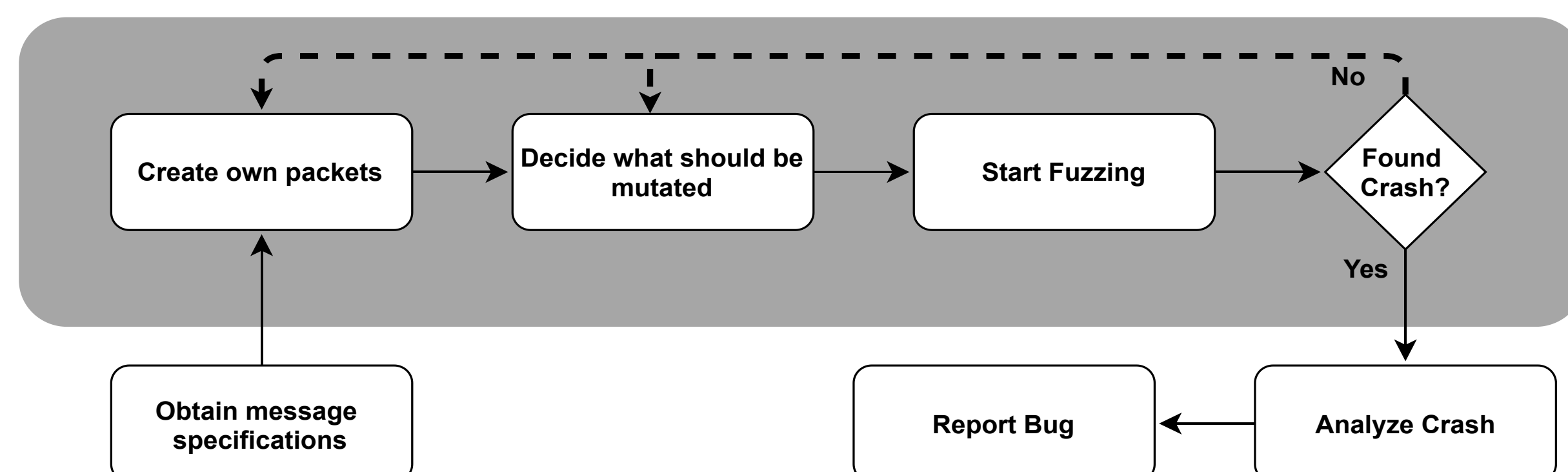


Figure 2: General Procedure for Protocol Fuzzing

A protocol fuzzing procedure can be divided into several steps, as illustrated in Figure 2. These steps can be used as a simple guideline to lower the entry barrier for other researchers outside the security domain.

Obtain Message Specifications

When fuzzing protocol implementations, the fuzzer must generate packets that comply with the protocol specifications. There are two general possibilities to determine these specifications: studying the official documentation of the protocol or reading and analyzing packets using a network protocol analyzer like Wireshark [8]. We use Wireshark to analyze the structure, fields, and contents of IEC 61850 packets and to check whether the packets generated by our fuzzer match this format. Figure 3 presents the details of an SV message, as it is shown in Wireshark.

Frame 1: 115 bytes on wire (920 bits), 115 bytes captured (920 bits) on interface 0			
Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: Iec-Tc57_01:00:01 (01:0c:cd:01:00:01)			
802.1Q Virtual LAN, PRI: 4, DEI: 0, ID: 0			
IEC61850 Sampled Values			
0000	01 0c cd 01 00 01 00 00	00 00 00 00 81 00 80 00	
0010	88 ba 40 00 00 61 00 00	00 00 60 57 80 01 02 a2	..@..a...W....
0020	52 30 27 80 06 73 76 70	75 62 31 82 02 00 01 83	R0'..svp ub1....
0030	04 00 00 00 01 85 01 00	87 10 44 9a 52 2b 3d fc	D R+..
0040	d3 5b 5e 3a 91 59 65 a1	ca 00 30 27 00 06 73 76	.[A:Ye...0'.sv
0050	70 75 62 32 82 02 00 01	83 04 00 00 00 01 85 01	pub2.....
0060	00 87 10 45 1a 52 2b 3e	7c d3 5b 5e 3a 91 59 65	...E R+> . [A:Ye
0070	a1 ca 00		...

Figure 3: Wireshark view of a sampled values message

Create own packets

To create artificial packets we use Boofuzz, which is a well-known open-source network protocol fuzzing framework [9]. We copy the message in hex representation from Wireshark and paste it into the fuzzing script. Afterwards, the message is divided according to the individual fields.

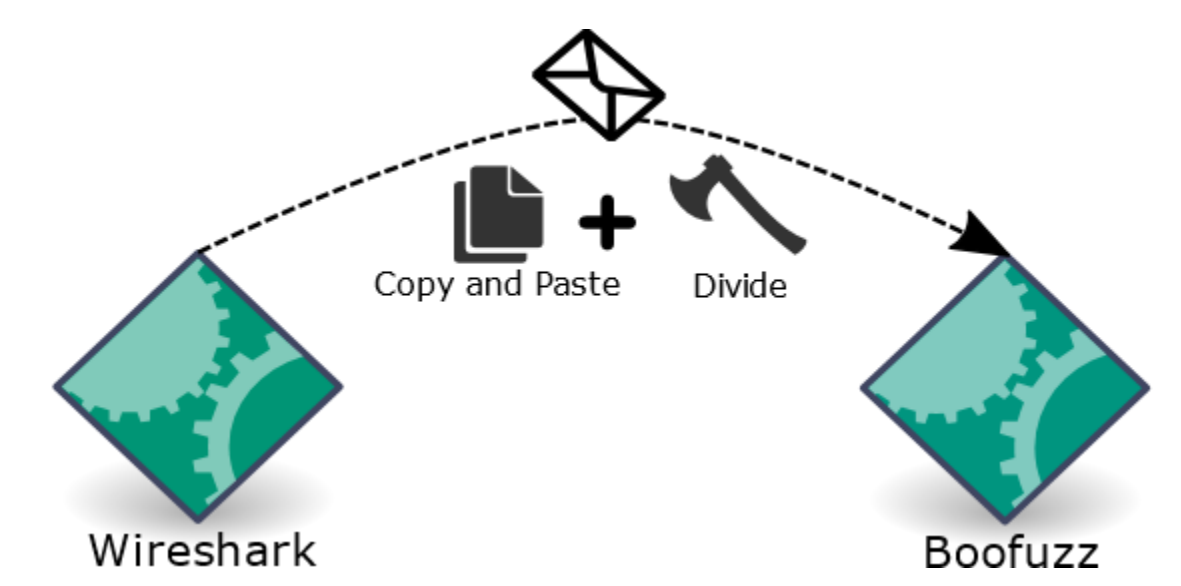


Figure 4: Basic Packet Creation

Decide what should be mutated

To realize the mutation, we use the `s_random` function of Boofuzz. The decision which fields of the message should be mutated is incumbent to the researcher. In this work, we mutate almost every field. Only essential attributes like destination, source, or ethertype remain unchanged.

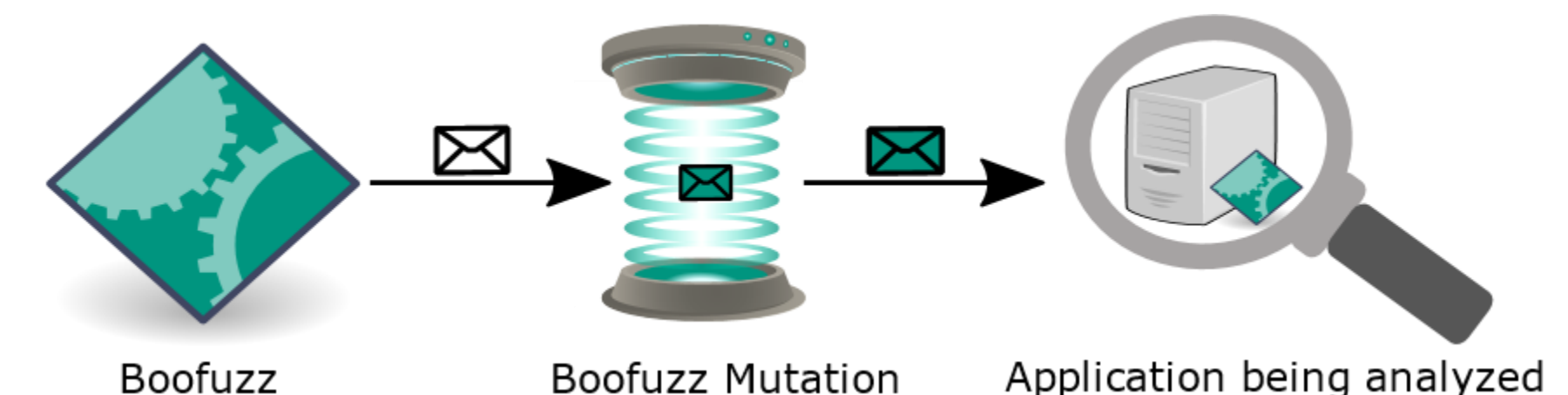


Figure 5: Fuzzing Process

Start fuzzing

Before starting the actual fuzzing, it has to be ensured that the program being fuzzed is observed and checked for crashes. For this purpose, Boofuzz offers a process monitor that must be started before the fuzzing process.

Found Crash?

If the fuzzing does not lead to any crashes, it should be double-checked if the packets sent by the fuzzer correspond to the required format. Also, it is advisable to mutate other fields or to create additional types of messages.

Analyze Crash

When a crash occurs the source code of the fuzzing target must be analyzed to determine whether the crash is unique and what caused it.

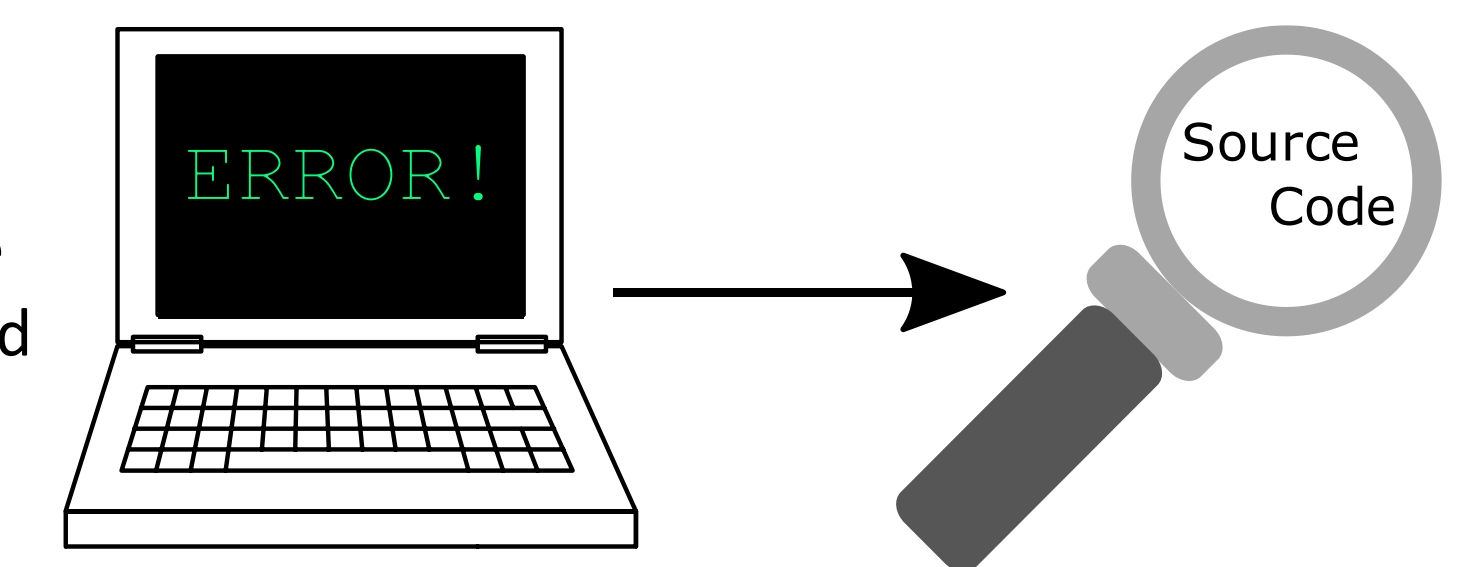


Figure 6: A found Crash requires a Source Code Analysis

Report Bug

A found crash that can be reproduced and explained should be reported to the software developer or maintainer. This reporting ensures that the bug can be fixed before attackers can find and exploit it.

Contributions and Results

```
ASAN:DEADLYSIGNAL
=====
==27504==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000000 (pc 0x562ebb3e9619 bp 0x7ff32b8fec00 sp 0x7ff32b8fec00 T1)
==27504==The signal is caused by a READ memory access.
==27504==Hint: address points to the zero page.
#0 0x562ebb3e9618 in memcpy_reverse src/sampled_values/sv_subscriber.c:683
#1 0x562ebb3e9666 in SVSubscriber_ASOU_getConfRev src/sampled_values/sv_subscriber.c:692
#2 0x562ebb3e96d7 in svUpdateListener /home/eugen/Projects/offsec/libiec61850/examples/sv_subscriber/sv_subscriber_example.c:34
#3 0x562ebb3e9eb6 in parseASDU src/sampled_values/sv_subscriber.c:352
#4 0x562ebb3e9f64 in parseSequenceOfASDU src/sampled_values/sv_subscriber.c:374
#5 0x562ebb3e9079 in parseSVPayload src/sampled_values/sv_subscriber.c:415
#6 0x562ebb3e92c2 in parseSVMessage src/sampled_values/sv_subscriber.c:535
#7 0x562ebb3e9316 in SVReceiver_tick src/sampled_values/sv_subscriber.c:548
#8 0x562ebb3e9a7e in svReceiverLoop src/sampled_values/sv_subscriber.c:164
#9 0x562ebb3e9de1 in destroyAutomaticThread hal/thread/linux/thread_linux.c:90
#10 0x7ff32f14b6da in start_thread (/lib/x86_64-linux-gnu/libpthread.so.0+0x76da)
#11 0x7ff32ee74a3e in __clone (/lib/x86_64-linux-gnu/libc.so.6+0x121a3e)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV src/sampled_values/sv_subscriber.c:683 in memcpy_reverse
Thread T1 created by T0 here:
#0 0x7ff32f39ad2f in __interceptor_pthread_create (/usr/lib/x86_64-linux-gnu/libasan.so.4+0x37d2f)
#1 0x562ebb3e9e2f in Thread_start hal/thread/linux/thread_linux.c:101
#2 0x562ebb3e9af1 in SVReceiver_start src/sampled_values/sv_subscriber.c:182
#3 0x562ebb3e884c in main /home/eugen/Projects/offsec/libiec61850/examples/sv_subscriber/sv_subscriber_example.c:76
#4 0x7ff32ed74b96 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21b96)
```

Figure 7: AddressSanitizer's report on a crash of the SV subscriber

Using the process illustrated in this work, we found crashes that could be traced back to four different errors — one in the MMS server, one in the GOOSE subscriber, and two in the SV subscriber. All four errors led to segmentation faults in the form of illegal read memory access, where the program tries to read from the zero page. They could be exploited by an attacker to conduct a Denial-of-Service attack, and thus should be fixed urgently. Figure 7 presents an AddressSanitizer report for one of these errors.

All bugs found during our research were reported to the project maintainer and will hopefully be fixed shortly.

References

- EU Commission Task Force for Smart Grids: Expert Group 1: Functionalities of smart grids and smart meters (2010)
- Yang, Y., Jiang, H., McLaughlin, K., Gao, L., Yuan, Y., Huang, W., Sezer, S.: Cybersecurity test-bed for iec 61850 based smart substations. In: IEEE Power & Energy Society General Meeting, pp. 1-5 (2015). IEEE
- Jiwen, C., Shanmei, L.: Cyber security vulnerability assessment for smart substations. In: IEEE PES Asia-Pacific Power and Energy Engineering Conference, pp. 1368-1373 (2016). IEEE
- Tilario, F., Berges, M.G.: Iec 61850 industrial communication standards under test. In: Proceedings of the 14th International Conference on Accelerator & Large Experimental Physics Control Systems (2014)
- Tu, T., Zhang, H., Qin, B., Chen, Z.: A vulnerability mining system based on fuzzing for iec 61850 protocol. In: 5th International Conference on Frontiers of Manufacturing Science and Measuring Technology (2017). Atlantis Press
- Kim, S., Jo, W., Shon, T.: A novel vulnerability analysis approach to generate fuzzing test case in industrial control systems. In: IEEE Information Technology, Networking, Electronic and Automation Control Conference, pp. 566-570 (2016). IEEE
- mz-automation: libiec61850. GitHub (2020)
- Orebaugh, A., Ramirez, G., Beale, J.: Wireshark & Ethereal Network Protocol Analyzer Toolkit, 1st edn. Elsevier Science, Amsterdam (2006)
- Pereyda, J.: boofuzz: Network Protocol Fuzzing for Humans. GitHub (2020)
- google: AddressSanitizer. GitHub (2019)

Further material including the fuzzer is available at <https://github.com/fkie-cad/61850-fuzzing>.