# PropFuzz - An IT-Security Fuzzing Framework for Proprietary ICS Protocols

Matthias Niedermaier, Florian Fischer, Alexander von Bodisco
University of Applied Sciences - Hochschule Augsburg
Augsburg, Germany
{Matthias.Niedermaier, Florian.Fischer, Alexander.vonBodisco}@hs-augsburg.de

*Abstract*—**Programmable Logic Controllers are used for smart homes, in production processes or to control critical infrastructures. Modern industrial devices in the control level are often communicating over proprietary protocols on top of TCP/IP with each other and SCADA systems. The networks in which the controllers operate are usually considered as trustworthy and thereby they are not properly secured. Due to the growing connectivity caused by the Internet of Things (IoT) and Industry 4.0 the security risks are rising. Therefore, the demand of security assessment tools for industrial networks is high. In this paper, we introduce a new fuzzing framework called PropFuzz, which is capable to fuzz proprietary industrial control system protocols and monitor the behavior of the controller. Furthermore, we present first results of a security assessment with our framework.**

## I. INTRODUCTION

Programmable Logic Controllers (PLCs) are the basic components used in a wide variety of Industrial Control Systems (ICSs) for example to regulate production processes or to control critical infrastructures. Historically, these devices operated in a separated network, with no connection to the Internet or office areas. In the past decade, this separation changed due to the demand of highly connected systems in the IoT and the fourth industrial revolution. A typical company network with control systems consists of several hierarchical layers as illustrated in Figure 1. Nowadays, the communication on the higher levels (ERP, MES, SCADA and PLC) is mostly based on the Internet Protocol (IP) protocol.
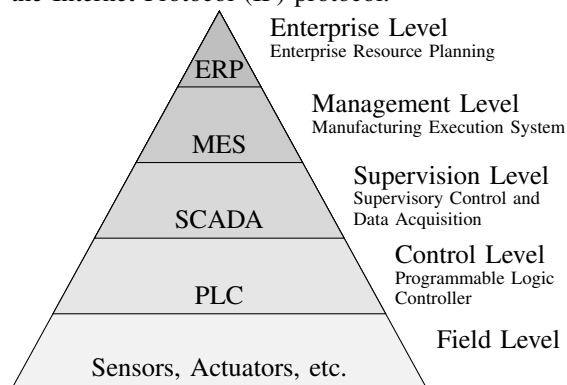


Fig. 1: Common Topology for Industrial Networks

Most PLCs offer the possibility to configure and program them via a proprietary TCP/IP connection. This simplification allows remote access to these devices, if there is no additional hardware restricting the communication. Thus, it is often possible for attackers to interact directly with the PLC. Therefore, it is necessary to analyze the communication between the control system and the Integrated Development Environment (IDE) with the aim of fuzzing proprietary industrial protocols to find security issues. The main problem concerning fuzzing these protocols is defining the data structure to be fuzzed.

In our fuzzing framework, the inputs are identified by using statistical computation to analyze the structure of proprietary industrial protocols. Popular fuzzing frameworks are introduced in Section II. Section III describes the methodology of fuzzing a proprietary PLC communication. Section IV explains our framework architecture. In Section V and VI first result, with possible attacks and recommendations are presented. Finally, an outlook and conclusion in Section VII is given.

## II. RELATED WORK AND MOTIVATION

Barton Miller, discovered a program crash caused by noise as a result of a lightning strike on his network connection during a thunderstorm [1]. This bug was triggered by a random input which is called fuzz-testing or fuzzing in the literature. Fuzzing could only trigger bugs, if the input is not rejected by a validation function of the Device under Test (DuT). A full automated fuzzing framework for ICSs include the process steps illustrated in Figure 2 [2].
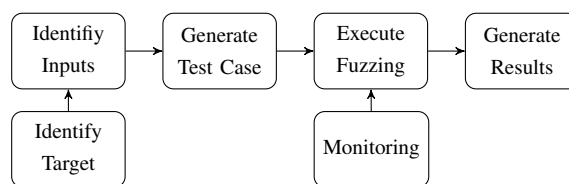


Fig. 2: Fuzzing Test Process

There are two elementary categories of fuzzers, based on how they create input for fuzzing. Generation-based fuzzers create input from scratch and thus require some knowledge of the protocol with corresponding data fields. With mutation fuzzers, samples of valid input are used to produce malformed input. A simple mutation fuzzer can modify a valid input sample and send it to the DuT.

- **Generation-Based Fuzzing** applies with generation rules to fuzz input. BooFuzz [3], a fork and successor of the Sulley [4] fuzzing framework, and Peach [5] are block-based fuzzers. These

kinds of fuzzers require a deep knowledge of the protocol structure and test case definition to generate inputs. Recent generation-based fuzzer like VUzzer [6] are able to automatically generate input test cases for basic communications.

- **Mutation Fuzzing** uses valid inputs and modifies them to create fuzzing input. Most of the frameworks analyze previously captured traffic, although there are fuzzers which allow live capturing. Radasma [7] is an input generation tool for basic protocols to identify field boundaries. LZFuzz framework [8] is an online fuzzer, which intercepts traffic directly, analyzes it with the LempelZiv compression algorithm and sends the manipulated packages to the device.

For fuzz-testing ICSs, these fuzzers and frameworks do not fulfill our requirements in automated input generation for proprietary protocols and electrical monitoring [8]. The documentation between the IDE and the PLC is mostly not public available and reverse engineering is highly time-consuming. Thus, it is necessary to have an automatic fuzzing framework for this kind of communication. More over in case of ICS it is necessary to monitor any behavior change of the device, where network monitoring is not enough.

## III. METHODOLOGY

Most of the modern PLCs are programmed with an IDE over TCP/IP. This communication is often open and not filtered. Figure 3 illustrates the minimal setup to interact with the ICS and the IDE.
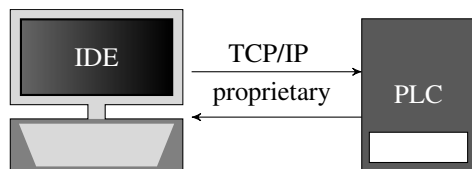


Fig. 3: Communication between IDE and PLC

The majority of these protocols are proprietary and have no public available documentation. Our framework allows a direct investigation of this communication. To start the data transfer between an IDE and a PLC a TCP/IP handshake is done. After that, there is often an additional proprietary handshake with a kind of challenge. Followed by this, the command and data transfer could be started. For a permanent connection, it could be essential to send keep-alive messages between the IDE and the PLC. This is not required for single command interaction. To make fuzzing feasible, it is necessary to perform the proprietary handshake and determine the protocol field, which should be fuzzed.

## IV. FRAMEWORK ARCHITECTURE

At a high-level view, illustrated in Figure 4, Prop-Fuzz is separated in three parts to fulfill the requirements of a full integrated fuzzing framework [9]. The **analyze** part splits the protocol and filtrates the information necessary to **fuzz** the DuT. In addition

to the network response monitoring the **monitor** part observes the PLC electrically.
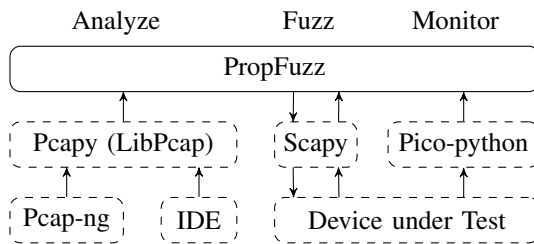


Fig. 4: Data-flow within PropFuzz Framework

A detailed view of the PropFuzz structure is shown in Figure 5, which illustrates the python modules, classes and configuration files within PropFuzz.
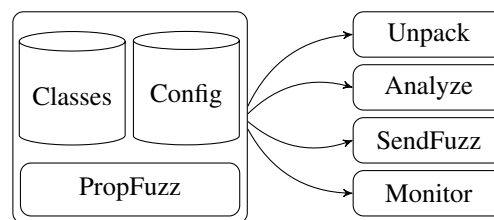


Fig. 5: PropFuzz Modularity

The implemented python modules are modular and could be extended or used within other frameworks.

### A. Unpack

The PropFuzz implementation provides two possibilities for package analysis. First a live capturing of the communication between DuT and IDE is possible. The second option is to read in existing pcap files. For live capturing Address Resolution Protocol (ARP) spoofing is needful, which could be prevented by some network devices. Once data acquisition is done the Unpack module splits up the information of the captured packages and stores them in objects. The module uses pcapy [10], a python implementation for LibPcap [11], for further examination of the collected packages.

### B. Analyze

Inside the Analyze process the messages of the proprietary protocol are interpreted. At the beginning a statistical analysis with the Ratcliff/Obershelp [12] pattern recognition algorithm of the created package objects is done. With these similarities, the proprietary handshake between the IDE and the PLC can be determined. After the detection of a handshake in the captured communication commands between the IDE and the PLC must be identified. Commands can be identified by comparing different captures containing a full match of the same command.

### C. SendFuzz

The SendFuzz module is responsible for sending and receiving packages inside the PropFuzz implementation. The gained information from the Analysis module is used to mimic the protocol handshake by sending

sniffed messages to the DuT. For constructing these messages the python module scapy [13] is used. After a successful established protocol handshake, further packages containing protocol-specific commands are sent to the DuT.

*D. Monitor*

Most fuzzing frameworks only observe the network connection during the test. Special about fuzzing ICSs is the monitoring of the process control [14]. To detect the effect of fuzz-testing with our framework, an output channel is monitored by an oscilloscope. Therefore, a square wave signal is generated on a certain output of the DuT, created with an alternating write to the output within the execution cycle of the PLC. Control commands sent to the PLC which e.g. stop or restart the device make an impact on the periodic signal of the output. This variation can then be monitored with the oscilloscope.
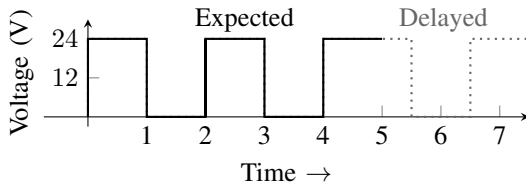
Fig. 6: Monitor PLC Output with Oscilloscope

Figure 6 illustrates the expected square wave (solid black) and an unexpected delayed output (dotted grey). This behavior can occur if a command causes high load on the controller. With our fuzzing framework, it is possible to trigger and monitor such changes. If the delay exceeds the allowed output jitter the process control is not feasible. This could result in unexpected behavior. For our framework, Pico-python [15] is used, to interact with the PicoScope 2208, which is a scriptable Universal Serial Bus (USB) oscilloscope.

## V. SECURITY ASSESSMENT EVALUATION

For testing PropFuzz two different PLCs, the "ILC 171 ETH" and the "ILC 150 ETH" from Phoenix Contacts, are used. According to the datasheet the devices support the protocols shown in Table 1. In this assessment the IDE "AUTOMATIONWORX Software Suite v1.83" from Phoenix Contacts is used.

TABLE 1: Phoenix Contacts Test Equipment

| DuT | ILC 171 | ILC 150 |
|---|---|---|
| Man. number | 2700975 | 2985330 |
| Profinet | ✓ | |
| Modbus | ✓ | |
| Proprietary | ✓ | ✓ |
| FTP | ✓ | ✓ |
| HTTP | ✓ | ✓ |
| HTTPS | ✓ | |
| SNTP | ✓ | ✓ |
| SNMP | ✓ | ✓ |
| SMTP | ✓ | ✓ |
| SQL | ✓ | ✓ |
| MySQL | ✓ | ✓ |

Three ports are open, if the factory default settings are applied. Table 2 shows the results of a nmap scan of the PLCs. A vulnerability in one of the protocols leads to high risks, caused by the remote exploitability.

TABLE 2: Factory Default Port Scan

| Port | Protocol | State | Service |
|---|---|---|---|
| 21 | TCP | open | FTP |
| 1962 | TCP | open | unknown |
| 41100 | TCP | open | unknown |

For our purpose the most interesting ports are the undocumented ones, which are used by the IDE to communicate with the PLC. Most commands are exchanged via port 1962. The connection establishment of this protocol is illustrated in Figure 7. The IDE sends a synchronize (SYN) request to the PLC. This should respond with a SYN acknowledgment (ACK). Consequentially the IDE complete the Transmission Control Protocol (TCP) handshake with an ACK.
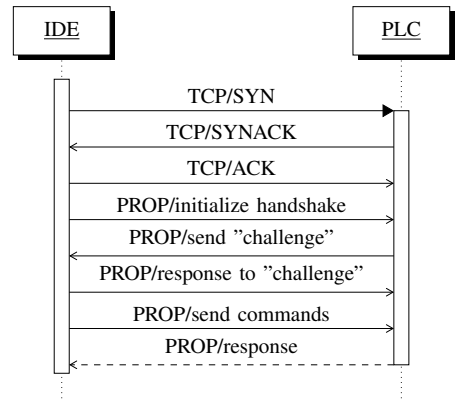
Fig. 7: Handshake between Phoenix PLC and IDE

After the TCP handshake with the PLC, a proprietary initializing sequence between the IDE and the Phoenix Contacts PLC is necessary. This starts with a request from the IDE to the controller:

```
0000  01 01 00 1a 00 00 00 80    ........
0008  64 15 00 03 00 0c 49 42    d.....IB
0010  45 54 48 30 31 4e 30 5f    ETH01N0_
0018  4d 00                      M.
```

This request is answered from the PLC, with an identifier (in this case 0x48) in it:

```
0000  81 01 00 14 00 00 00 01    ........
0008  00 00 00 00 00 02 00 00    ........
0010  00 48 00 00                .H..
```

This value must be send back from the IDE to the PLC. It could be seen as a simple session key, which does not change on a controller:

```
0000  01 05 00 16 00 01 00 00    ........
0008  e8 e9 00 48 00 00 00 1c    ...H....
0010  00 04 02 95 00 00          ......
```

For the proprietary protocol from Phoenix Contacts the same handshake for a specific DuT could be send, because it is a constant value for each device. Thus, the handshake is a simple replay for the Phoenix PLCs and

PropFuzz is able to detect the handshake by statistically comparing the start sequences of live captures or pcap-ng files. After the proprietary handshake, commands could be send to the controller. Below a reset command is shown, where the PLC performs a complete reboot.

```
0000  01 05 00 16 00 10 00 00     ........
0008  e8 c8 00 48 00 00 00 00     ...H....
0010  00 04 0a ba 00 00           ......
```

At this point scapy is used to fuzz different fields of the command. With the PropFuzz framework it was possible to identify different vulnerabilities in the session management and command handling of the PLCs. The vulnerabilities found in the Phoenix Contact products were reported to the manufacturer and customers were informed with an advisory (ICSA-16-313-01).

## VI. Possible Attacks & Recommendations

The identified security problems within the protocol make several remote attacks possible. Considering the usage of these controllers in critical infrastructures, the severity of potential attacks is classified high.

- A **replay attack** is a network attack in which a valid data transmission is repeated. The attacker needs few knowledge and can simple replay previous captures containing PLC commands.
- By **manipulating variables**, it is possible to change the sequence or process of a program on the PLC. This requires knowledge about the setup of the control system.
- With **changing the software or firmware** of an ICS the total control of it could be achieved, e.g. to create a BotNet.

These attacks demonstrate the severity of the found vulnerabilities with our framework. It is possible to remotely exploit effected PLCs without physical access to it.

The **recommendations** to avoid such vulnerabilities or to defend against possible attacks can be categorized by the responsible stake holders: manufacturer, system integrator and operator. Integrators and end users of ICSs should use a defense-in-depth architecture for their networks, considering the following:

- Devices should not be accessible public without the use of a Virtual Private Network (VPN)
- Firewalls should be used for network segmentation and controller isolation

Manufacturers of ICSs should develop products, which are secure by design, e.g.:

- Authentication for the communication
- Well implemented session and user management
- Secure and cryptographically protected protocols

## VII. Outlook & Conclusion

The PropFuzz framework has reached a stable testing state. We have already used our framework for fuzzing PLCs from other vendors. For similar protocols, PropFuzz is able to perform the handshake and start fuzzing without any adjustment to our framework. In the current implementation, complex protocols with a proper session management and cryptographic measures are not fuzzable. This functionality will be extended later, making it possible to fuzz a wider spectrum of protocols. Furthermore, to observe a PLC within a process it is necessary to virtually represent and observe all used in- and outputs of the controller. This must be done with high efforts for every simulated process itself, which is not feasible at this time.

In this paper, we presented a stable and extensible fuzzing framework for proprietary ICS protocols. Compared with the available software, PropFuzz is able to automatically analyze the communication between the IDE and the PLC and fuzz the DuT. In addition, PropFuzz is able to monitor the output and detect suspicious behavior.

We have demonstrated the abilities of our framework by fuzzing two PLCs and detected three critical vulnerabilities, which could be exploited remotely by attackers (Advisory ICSA-16-313-01). We have worked in a close cooperation with Phoenix Contacts to find solutions and fixes.

## References

[1] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of unix utilities," *Commun. ACM*, vol. 33, pp. 32–44, Dec. 1990.

[2] S. Kim, W. Jo, and T. Shon, "A novel vulnerability analysis approach to generate fuzzing test case in industrial control systems," in *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*, pp. 566–570, May 2016.

[3] J. Pereyda, "Boofuzz fuzzing framework." https://github.com/jtpereyda/boofuzz. Accessed: 24.02.2017.

[4] P. Amini, A. Portnoy, and R. Sears. https://github.com/OpenRCE/sulley. Accessed: 13.03.2017.

[5] M. Eddington, "Peach fuzzing framework." http://www.peachfuzzer.com/. Accessed: 23.02.2017.

[6] S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, "Vuzzer: Application-aware evolutionary fuzzing," 2017.

[7] A. Helin, "Radamsa fuzzing test case generator." https://github.com/aoh/radamsa. Accessed: 22.02.2017.

[8] R. Shapiro, S. Bratus, E. Rogers, and S. Smith, *Identifying Vulnerabilities in SCADA Systems via Fuzz-Testing*, pp. 57–72. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[9] S. Plaga, S. Tatschner, and T. Newe, "Logboat - a simulation framework enabling can security assessments," in *2016 International Conference on Applied Electronics (AE)*, pp. 215–218, Sept 2016.

[10] CORE Security, "Pcapy." https://github.com/CoreSecurity/pcapy. Accessed: 04.03.2017.

[11] V. Jacobson, C. Leres, and S. McCanne, "Libpcap." http://www.tcpdump.org/. Accessed: 04.03.2017.

[12] J. Ratcliff and D. Metzener, "Ratcliff-obershelp pattern recognition," 1998.

[13] P. Biondi, "Scapy packet manipulation." http://www.secdev.org/projects/scapy/. Accessed: 28.02.2017.

[14] H. Yoo and T. Shon, "Grammar-based adaptive fuzzing: Evaluation on scada modbus protocol," in *2016 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 557–563, Nov 2016.

[15] C. O'Flynn, "Pico-python." https://github.com/colinoflynn/pico-python. Accessed: 28.02.2017.

[16] "Riskviz." https://www.riskviz.de. Accessed: 28.02.2017.