

# **EFFECTIVE FUZZ TESTING FOR PROGRAMMABLE LOGIC CONTROLLERS VULNERABILITY RESEARCH TO ENSURE NUCLEAR SAFETY**

J. SUCHORAB

National Centre for Nuclear Research  
Otwock, Poland  
Email: Jakub.Suchorab@ncbj.gov.pl

K. STASZKIEWICZ

National Centre for Nuclear Research  
Otwock, Poland  
Email: Kinga.Staszkiwicz@ncbj.gov.pl

J. WALKIEWICZ

National Centre for Nuclear Research  
Otwock, Poland  
Email: Joanna.Walkiewicz@ncbj.gov.pl

M. DUDEK

National Centre for Nuclear Research  
Otwock, Poland  
Email: Marcin.Dudek@ncbj.gov.pl

## **Abstract**

The paper focuses on ensuring nuclear safety through vulnerability research of programmable logic controllers, which are key components of the operational technology employed in nuclear power plants. Being one of the most popular vulnerability discovery techniques, fuzz testing was chosen as a testing method. The aim of the study was to prove the effectiveness of fuzz testing in the search for vulnerabilities of programmable logic controllers. The research was undertaken in order to develop a specific fuzz testing methodology allowing to test the security of industrial protocols stack implementation in firmware of programmable logic controllers. A fuzzing laboratory testbed has been designed with the purpose of conducting various fuzzing tests. The paper describes fundamental components of a recommended testbed, with regards to hardware and software. As a result of using the developed methodology, several tests were conducted, producing diverse outcomes. The process of discovering and investigating a zero-day vulnerability in a Siemens S7-1500 series PLC is discussed in the paper. The research was carried out in the Nuclear Centre for Nuclear Research (Poland) as part of IAEA Coordinated Research Project J02008 on incident response in nuclear facilities.

## **1. INTRODUCTION**

Critical infrastructure, such as nuclear power plants (NPP), widely uses various Operational Technology (OT) solutions, such as Industrial Control Systems (ICS), in order to fulfill their functions. OT networks used to be logically and physically isolated from other business functions, but nowadays this is not always true. Along with the digitalization of such systems, they became interconnected and inter-networked. Thus new cybersecurity threats were introduced.

In the case of critical infrastructure, even the smallest disruption can cause undesirable, hazardous outcomes. An action as simple as changing a value of a single variable (e.g. temperature sensors readings) can affect the pump control or the whole cooling system. One of the key components of industrial control systems are programmable logic controllers (PLCs), which process information about the physical process in order to control it. Because PLCs are an inseparable part of 80% of ICS designs, their robustness has a direct, incontestable impact on the safety of the whole control systems [1]. Some of the publicly reported incidents involving PLCs are: a Stuxnet worm attack on Iranian nuclear facilities (2010) that reprogrammed PLCs to operate incorrectly resulting in failure of centrifuges, and an incident in Browns Ferry nuclear plant in Alabama, where a faulty PLC overloaded the network with excessive traffic [2][3]. Therefore, PLCs have been chosen as an object for our studies.

The security of various aspects should be tested: logical security of hardware components, firmware code, or its communication channels – both radio and physical connections. Because testing the security of hardware components is time-consuming on a large scale, and code review is often not possible as the code is not publicly available, testing communication channels was chosen for this research.

The security of communication channels can be tested using two approaches. The first one is a classic verification of whether mechanisms ensuring confidentiality, integrity and accountability of data or commands sent through a communication channel, are implemented. The second approach focuses on how well the messages sent through that channel are processed by the device itself. As the first approach often depends on the transmission medium or protocol used, security problems are usually easy to notice, whereas an analysis of whether the protocols are processed by a device in a secure way is much more difficult, especially without access to firmware source code.

One of the methods commonly used for that purpose is network-based fuzz testing. It allows testing of the implementation of communication protocols at various stack levels, without any prior knowledge about the device internals. What is more, it is relatively easy to implement, and provides great scalability.

In order to define an efficient fuzz testing methodology for PLCs, a specialized laboratory, consisting of several PLCs and a fuzzing tool, was created. Using this testbed, different models of Siemens PLCs were examined for robustness of different network protocols implementations. During the conducted research several vulnerabilities were found, including a zero-day vulnerability in Siemens S7-1500 PLC.

## 2. FUZZ TESTING

Fuzz testing (also known as fuzzing) is an automated technique for detecting vulnerabilities in software. The main foundation of this technique is to prepare numerous malformed inputs (Input Generation), deliver them to the tested target (Input) and monitor the target for any unexpected behavior (Instrumentation). A software performing these tasks is called a fuzzer. Depending on how the malformed inputs are created, fuzzers can be classified as one of the following types: mutational, generational or evolutionary. Mutational fuzzers randomly mutate samples of valid inputs in order to produce malformed ones. Generational fuzzers are more advanced, as they generate inputs from scratch, with full knowledge of protocol structure. Evolutionary fuzzers learn the protocol structure over time, using feedback from sent inputs as the reference.

In order to perform various tasks, PLCs need to communicate with other components, such as supervisory systems, Human Machine Interfaces, I/O islands or engineering workstations. For years, numerous communication protocols were designed for that purpose, such as Modbus, Profinet or S7. Nowadays, more and more industrial protocols are based on the IP protocol, known from the world of Information Technologies. On the one hand, it provides easier integration and cheaper solutions for customers, but on the other hand, it introduces risks, as manufacturers often use complex libraries to handle its implementation, without proper testing.

Network protocol fuzzing means that the target of testing is protocol-handling code. Firstly, network packets that are compatible with the chosen protocol rules have to be generated with a fuzzer. Secondly, the delivery mechanism has to be chosen, depending on whether the target is a server or a client. In the first case, the fuzzer acts as a client, though sending malformed requests instead of proper ones. In the second case, the fuzzer represents a server, listening to target requests and responding with malformed replies. Fuzzing can be applied in two architectures - end-to-end testing or pass-through testing. The end-to-end testing variant requires the target device to be directly connected to the fuzzer, whereas pass-through testing variant allows the target to only mediate in communication. Therefore, end-to-end testing is used for PLCs testing, while pass-through testing can be applied during investigation of components like firewalls.

In order to make good use of fuzz testing, it is essential to obtain feedback from the tested device (referred to as Instrumentation). Revealing and locating a vulnerability is the aim of fuzzing, which means that both checking for failure and reproducing the failure are equally important. That being said, there is a need to issue a verdict whether a failure has occurred or not. Depending on the tested system, instrumentation can be done using different methods. Human observation is the simplest of them. While in certain cases it is enough to simply look at a flashing diode, other alarm indicators or log files could be necessary in order to determine that an error has occurred. Although this approach is not efficient when testing a lot of targets for a long time, it may prove useful while testing a setup and conducting initial tests. Another fail detection method is the valid case

method, which is automated and does not rely on human observation. It involves sending a valid input following a certain number of malformed inputs and waiting for a valid response. One of its advantages is that it is usually easy to implement, in particular in the case of network protocol fuzzing. Resource monitoring is yet another method enabling recognition of an occurring malfunction. It is based on monitoring usage of processor, memory, disk space, and other parameters that may indicate failure of the target system. This method provides opportunity to discover even subtle errors, as some of them may not be critical enough to result in a lack of responsiveness of the target, but may only be revealed based on, for example, higher memory usage [4].

In order to ensure that the vulnerability discovery process is as effective as it can be, it is advised to use a fuzzer capable of logging both sent test cases and received responses. Apart from those logs, logs from the tested target are also beneficial to monitor parameters such as connection status, memory or CPU usage. Moreover, it is good practice to provide an external network sniffer that captures all the traffic between tested target and the fuzzer. It can be of value especially in the case of false positive signals from the fuzzer, since an analysis of recorded traffic can help to rule out an actual malfunction.

### 3. LABORATORY CONFIGURATION

The initial testbed of a fuzzing laboratory for PLC testing can be a simple one, although it can be extended if needed. Its fundamental hardware components are a computer for running the fuzzing software and a PLC to be tested. In terms of software, not only a fuzzer is required, but also software for the PLC configuration, which is usually provided by the PLC vendor. Depending on the specific laboratory configuration and environmental requirements, some additional network devices, such as switches or routers, may be useful.

The fuzzing laboratory formed at National Centre for Nuclear Research (NCBJ) consists of one laboratory stand. Its key components are presented in Table 1, while the laboratory scheme is illustrated in Fig. 1.

TABLE 1. HARDWARE AND SOFTWARE USED IN LABORATORY

	<b>Component</b>	<b>Importance</b>	<b>Specification</b>
<b>Hardware</b>	PLC	<i>required</i>	PLC Siemens S7-317 PLC Siemens S7-1511 PLC Siemens S7-1512
	Computer	<i>required</i>	1 fuzzer server 1 fuzzer license server
	Switch	<i>optional</i>	D-Link Web Smart Switch DGS-1224T
<b>Software</b>	Fuzzer	<i>required</i>	Defensics from Synopsys
	PLC configuration software	<i>required</i>	TIA Portal v14
	Network protocol analyzer	<i>recommended</i>	Wireshark
	SCADA/HMI	<i>optional</i>	WinCC

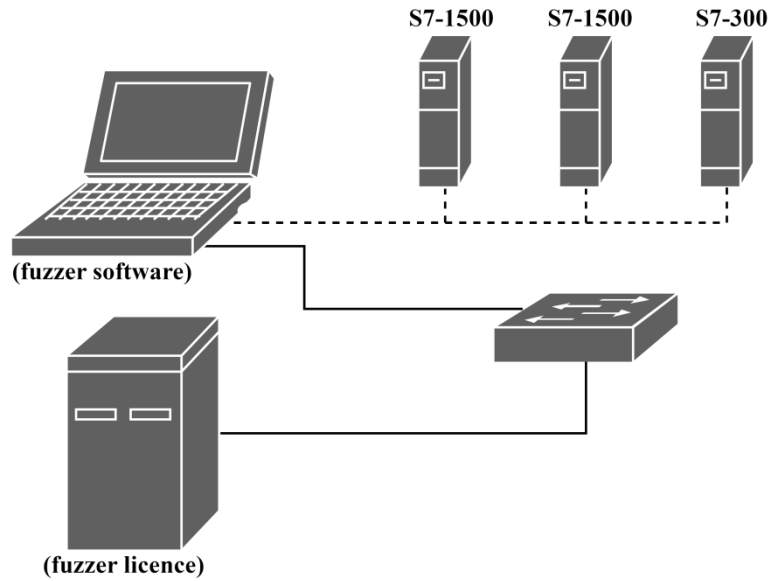


FIG. 1. Laboratory setup.

The tested devices were chosen based on an interview with NCBJ engineers who indicated popular choices in the case of nuclear applications. The manufacturer Siemens was chosen as a provider of PLCs to be tested because its products are believed to be widely used in industry [5]. Then two PLC series were selected as test subjects: S7-300 and S7-1500. One of the reasons for choosing the S7-300 series was the fact that these PLCs were targets of the Stuxnet worm attack in 2010 [6]. The S7-1500 series PLCs are the most modern Siemens PLCs already used in various applications and are expected to gain even more popularity throughout the industry.

Apart from PLCs, another required component of a fuzzing laboratory is a computer with the fuzzer. In the case of the discussed laboratory, two computers are used. One of them acts as a license server for the fuzzer software, while the other is equipped with the software itself. Both switch and router are used as a part of the laboratory's own local area network in order to provide connection between computers, i.e. between the fuzzer and the license server. In terms of industrial protocols fuzzing software, there are many available fuzzers, both free and commercial, such as ProFuzz [7], Profinet Set Fuzzer [8], Peach Fuzzer [9] and Defensics [10]. The latter was chosen for the presented setup, because it offers the most suitable capabilities for it. Defensics is a generational fuzzer with over 250 ready-made test suites, which are sets of configuration options and test cases for a given protocol. Several test suites compatible with protocols supported by the selected PLCs were purchased for the purposes of the planned testing.

As far as software is concerned, a PLC configuration software is essential in order to prepare PLCs for fuzz testing. Details such as IP address, ports options and other general settings should be set before the start of tests. Writing a simple program and downloading it to the tested PLC in order to detect a potential change in PLC operation caused by a test is also advised. Choosing the TIA Portal software for this research setup is self-explanatory, as it is designed specially for Siemens devices. The use of software that allows capturing and analyzing network traffic is recommended in order to have additional analysis capabilities in case an anomaly was detected. This can prove useful in particular to exclude false positive cases, when for example the fuzzer mistakenly marks a test as failed. Wireshark [11], being a popular sniffer, was chosen as network protocol analyzer for the described laboratory. Lastly, a SCADA or HMI can be used as an optional component of a fuzzing laboratory. Thanks to it, communication problems between the PLC and operators displays can be detected. Similarly to the PLC configuration software, the process of choosing a software for the discussed laboratory purposes was dictated by the chosen PLCs. Therefore, WinCC software provided by Siemens became part of the proposed fuzzing laboratory.

4. RESEARCH ON METHODOLOGY

4.1. Initial Tests

In order to see if the setup behaved correctly and the fuzzer could communicate with the tested PLC, some initial configuration steps were required. Available test suites were configured by providing the correct IP and MAC addresses of the tested PLC. Sample tests, consisting of 10% of all available test cases were run to see how the tested devices behaved and how the used fuzzer (Defensics) operated. The other settings were set to default. All the described tests were carried out on the S7-1512 PLC with firmware version 2.1.0. The PLC was tested only in server mode, as the target PLC was not configured to perform any actions and it would not send any queries without prior request from the fuzzer.

Table 2 shows the conducted tests with their results and comments. Erroneous behavior was observed for:

- IGMP test suite: the target did not respond to IGMP queries;
- Profinet DCP Server test suite: some of the test cases did not pass. Further investigation of this issue is described in the next subsection;
- TCP for IPv4 test suite: most payload types tested in this test suite (SSHv2, TELNET, BGP4) were not applicable to the PLC. Only TCP message with HTTP GET payload received any response. Since there is a separate test suite for HTTP protocol, no further tests were conducted using TCP for IPv4 suite.

TABLE 2. INITIAL TESTS RESULTS

Test suite name	No. of test cases	Duration [hh:mm:ss]	Verdict	Comment
ARP Server	139177	01:48:33	pass	---
HTTP Server	18170	00:24:32	pass	---
ICMPv4	474179	00:27:41	pass	---
IGMP	1380	00:00:22	n/a	Tested device did not respond to IGMP Queries
IPv4	44916	00:32:54	pass	---
Profinet DCP Server	5939	00:08:46	fail	Error investigation described in part 4.2 of the paper
Profinet PTCP Server	124426	1:23:00	pass	---
TCP for IPv4	interoperability check	00:00:00	inconclusive	Tested device did not respond to some types of payload

4.2. Profinet DCP Server Error Investigation

The Profinet DCP Server test on the S7-1512 PLC resulted in several fail verdicts. Some of the test cases required two instrumentation attempts to pass. The default value of max. instrumentation attempts is 1, so those test cases were marked as failed. In order to diagnose the problem, the connection between TIA portal and the PLC was checked. No connectivity problems were detected, so the question was why the device did not respond to all queries despite a working connection. To find the answer network traffic was captured from the test and analyzed using the Wireshark tool. A detailed traffic review revealed that the instrumentations marked as failed in fact received proper responses, and that therefore, the fail verdict might be caused by too long response times. In order to confirm this hypothesis, the default 1000ms value of maximum response time for instrumentation was first changed to 10000ms and then to -1 value (which is unlimited wait time).

Contrary to expectations, the error persisted, so the description of the issue, along with collected logs and traffic captures were sent to the Defensics technical support. After internal investigation, the support concluded that a bug in the Profinet DCP test suite caused a problem, and a fix was provided. After conducting tests using this new version, failures were no longer observed. Therefore, the connectivity problem was recognized as a false positive error.

### 4.3. Further Testing

Further testing included running full tests of all applicable test suites, including the new version of the Profinet DCP test suite. As presented in Table 3, all of the tests, except IPv4, passed. Investigation of that verdict is described in the next section.

TABLE 3. FULL TESTS STATISTICS

Test suite name	No. of test cases	Duration [hh:mm:ss]	Verdict	Comment
ARP Server	5722150	67:27:23	pass	---
HTTP Server	2003009	40:27:45	pass	---
ICMPv4	6239887	05:49:12	pass	---
IPv4	113516	17:06:45	fail	Test run interrupted due to PLC error
Profinet DCP Server	221432	01:33:33	pass	---
Profinet PTCP Server	4121092	45:54:22	pass	---

### 4.4. IPv4 Test Suite Fail Investigation

The failed verdict in the IPv4 test suite was caused by a continued lack of response after a specific test case. The default setting allows unlimited instrumentation attempts (the test is never interrupted). To aid the investigation, the traffic capture was started and the maximum instrumentation attempts value was set to 100 for the next tests, so that if the error happened again, the test would not have to be interrupted manually. Rerunning the full test showed repeatability of the error.

After inspecting the PLC visually and analyzing recorded traffic, it was found that connectivity between the PLC and the computer running the test was lost. Additionally, the connectivity was tested using TIA Portal and a ping command from the connected laptop, with the same results. Also, a simple program switching output values in cycle was uploaded to the device and it was observed to continue uninterrupted. Communication with the PLC could only be restored after a hard manual reset of the device.

As it was not clear whether one test case or a sequence of test cases had caused the error, multiple tests, as shown in Table 4, were conducted. Firstly, the individual test case that was run directly preceding the break in communication was sent, but did not cause a failure (*test 01* in Table 4). Test cases are grouped by type of their modification and it was decided to run *test 02* containing all test cases from the group in which the error is triggered. This approach resulted in a fail verdict. The next step was to run only a fragment of the group (narrowing the malformation type) in *test 03*. It triggered the error in about 33 minutes. Next, in order to confirm repeatability of the fail conditions after about the same time, the same test was rerun (*tests 04-08*). To further narrow the group, it was decided to run *test 09* consisting a very small fragment of the previous test sequence. As this did not result in a fail verdict, *tests 10-16* were run with larger and larger fragments, up to the point when *test 16* successfully caused an error. To exclude a coincidence, *test 16* was repeated (*tests 17,18*).

Based on the results of the described tests, it was observed that after rerunning the test several times with test sequences starting from a different test case, but all from the same group, every time the error occurred after a similar amount of time (about 33 minutes). This suggested that the number of sent test cases from the group was a decisive factor, rather than the specific sequence or a single malformed packet. To confirm this hypothesis, several test cases from the group were randomly selected and sent in an infinite loop (*test 19*). Eventually, this triggered the error after a similar amount of time to the previous tests. Going further, subsequent

single test cases were rerun in loop. Some of them are presented as *tests 20-23*. It was determined that the majority of the looped test cases from the group caused an error after about the same time.

TABLE 4. ERROR INVESTIGATION TESTS RESULTS

Test ID	Test cases sequence	Duration [hh:mm:ss]	Failed instr.	Verdict	Comment
01	386263	00:00:00	0	pass	a single test case
02	0-355119	04:19:32	6	fail	group tested up to failure
03	327644-355119	00:33:53	6	fail	a fragment of tested group
04	327644-355119	00:32:42	2	fail	rerun of above
05	327644-355119	00:32:17	2	fail	rerun of above
06	327644-355119	00:32:04	2	fail	rerun of above
07	327644-355119	00:32:01	3	fail	rerun of above
08	327644-355119	00:33:32	5	fail	rerun of above
09	365000-368263	00:02:34	0	pass	a small fragment of tested group
10	360000-368263	00:06:29	0	pass	---
11	355000-368263	00:10:27	0	pass	---
12	350000-368263	00:14:22	0	pass	---
13	345000-368263	00:18:20	0	pass	---
14	340000-368263	00:22:17	0	pass	---
15	335000-368263	00:25:45	0	pass	---
16	330000-368263	00:30:11	2	fail	---
17	330000-368263	00:29:53	1	fail	rerun of above
18	330000-368263	00:30:09	2	fail	rerun of above
19	365186-368263	00:35:00	6	fail	a small fragment in a 35 minute loop
20	368235	00:35:00	6	fail	a single test case in a 35 minute loop
21	368258	00:35:00	9	fail	a single test case in a 35 minute loop
22	2682247	00:35:00	0	pass	a single test case in a 35 minute loop
23	368249	00:35:00	8	fail	a single test case in a 35 minute loop

In the next step, the network traffic from the last rerun (*test 23*) was recorded to extract the exact IP packet that triggered an error when sent in loop. It was recognized to be a packet with incorrect data in one of the rarely used header fields. A proof of concept script using python with scapy library [12] was created. In order to add the monitoring functionality to the script, the python-snap7 [13] library was used. This enabled a verification of connectivity after one or more manipulated packets were sent. Several tests using the prepared exploit were performed, every one of them resulting in triggering the vulnerability. The script was sending packets much faster than the fuzzer, and therefore, it caused an error in under a minute, after sending the malformed packets approximately 33000 times. This confirmed that a vulnerability was found and was triggered by sending a certain amount of modified IP packets.

As all the above tests were conducted on S7-1512 with firmware version 2.1.0, other firmware versions from 2.0.0 to 2.5.0 on S7-1512 and S7-1511 were examined using the proof of concept script in order to check the scope of the vulnerability. Both tested PLCs behaved alike, however, the attack was successful only on firmware versions preceding 2.5.0. This could have indicated that Siemens was aware of the problem and solved it in new firmware releases.

To confirm these findings, vulnerabilities for S7-1500 PLCs, published by Siemens CERT were reviewed [14]. No known vulnerabilities were found that corresponded to the one discovered during this work, so, in accordance with the responsible disclosure policy, proof of concept was sent directly to the Siemens CERT. Eventually, the vulnerability was classified as not previously known and an appropriate Siemens Security Advisory [15] was published. The code CVE-2018-13805 was assigned to the vulnerability.

## 5. RESULTS

The conducted research led to the creation of a methodology of fuzz testing based on practical experience. Fig. 2 presents the developed procedure for fuzz testing PLCs in a systematic and intuitive way.

In order to begin testing, a list of protocols supported by the tested device should be prepared. This task can be accomplished using one or both of following approaches. The first approach is to thoroughly study the documentation of the tested PLC to check which protocols are supported by the device. Another way of determining the list of protocols to be tested is to scan the PLC looking for open ports and available network services. This approach allows to create a list of protocols that needs to be tested, as any available service is a potential attack vector.

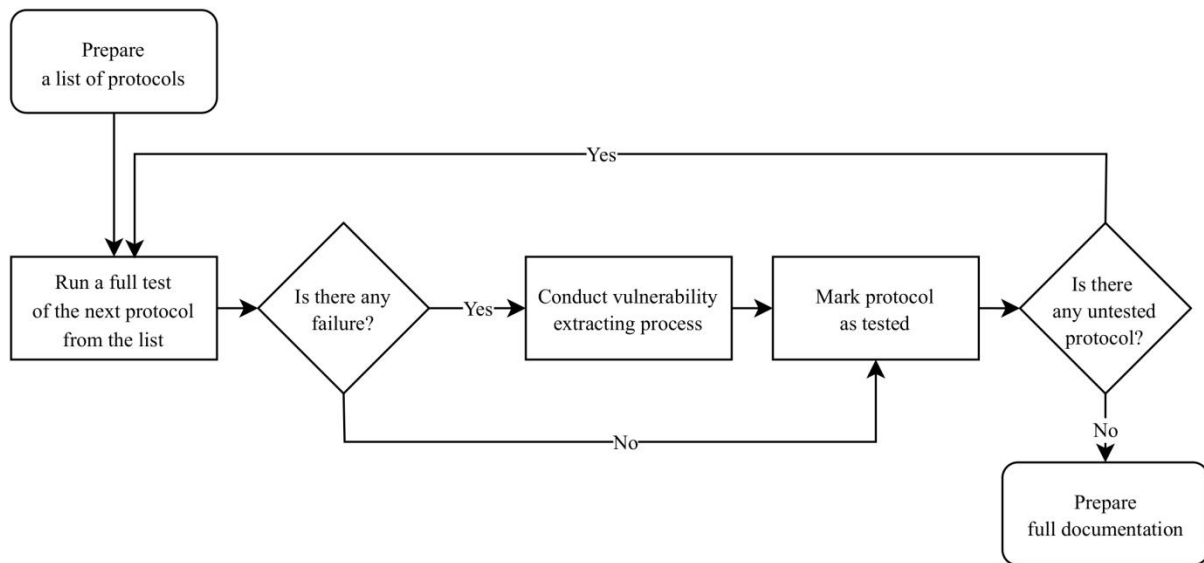


FIG. 2. Diagram presenting developed fuzzing methodology.

Once the list of protocols is complete, the main part of testing can be started. The first step is to run a full test of the protocol selected from the list. A full test means that during the test, all available test cases from the selected test suite should be used. It is recommended to log both valid and failed cases. However, using traffic capture is inadvisable, as it could use excessive amounts of disk space. At the same time, run control interval times should be carefully adapted to ensure that they are not unnecessarily long. This is especially crucial during a full test run, since even one needless millisecond per test case can result in a noticeable time difference, given the sheer number of them.

A full test run results in either a failure or a pass verdict. Depending on the received result, there are two possibilities of further actions. If no failure is detected, the protocol should be marked as tested, and a full test of the next protocol from the list should be carried out. The second scenario is a fail verdict of the test. In order to further investigate the issue, a vulnerability extraction process should be conducted. Generally speaking, the aim is to confirm the correctness of the verdict and determine the particular sequence of test cases that triggers the vulnerability. Fig. 3 shows the proposed steps of this process.



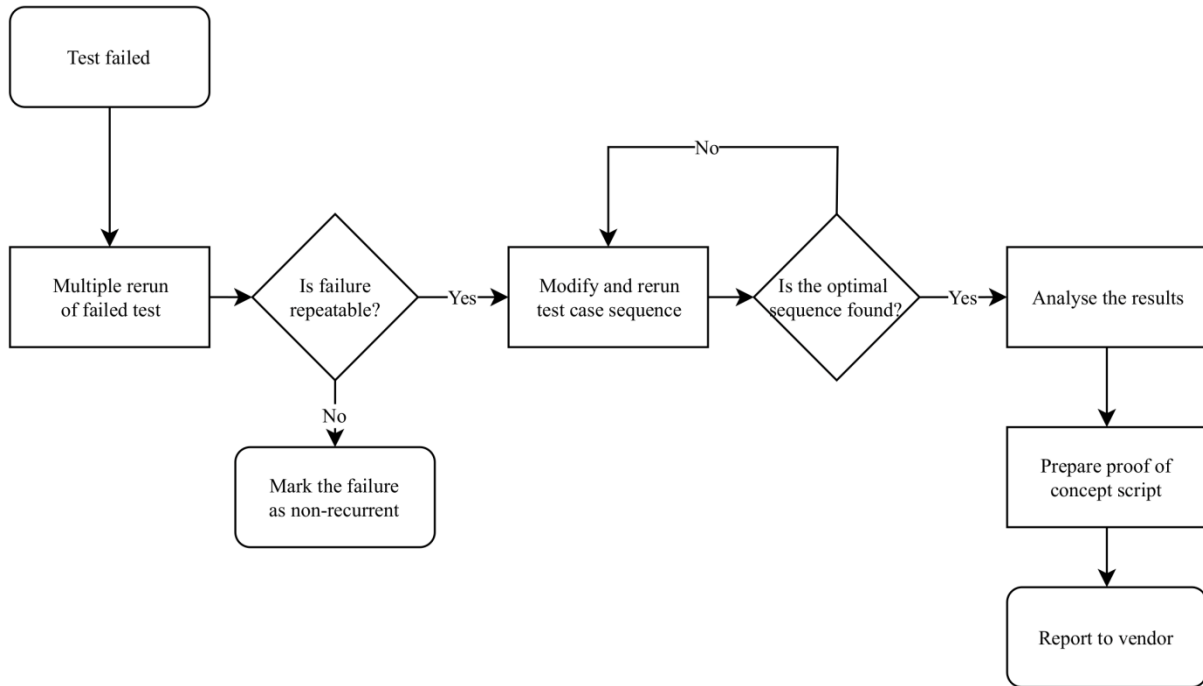


FIG. 3. Diagram presenting vulnerability extracting process.

In order to analyze the potential vulnerability, it is crucial to determine if a fail verdict is repeatable. When it is not, it should be marked as a false positive and no further conclusions can be drawn. To check repeatability of the verdict, failed cases should be rerun. In order to avoid false positives, additional diagnostics should be used, such as traffic analysis and constant connectivity monitoring. Assuming that the scenario is reproducible, two subsequent approaches to investigation of the optimal test case sequence that triggers the vulnerability were defined.

It is recommended to start with short tests which are highly likely to trigger a failure. One should begin with a rerun of a single test case that was sent directly before the failure occurred. If the failure is not reproduced, it is advised to rerun several additional test cases and finally, the whole test group. If such a trivial approach has no effect, the second approach can be applied, the main idea of which is to start using the whole sequence and then decreasing the number of test cases in it. The first rerun consists of all test cases preceding the erroneous state. In the next steps the test sequence should be narrowed down using the bisection algorithm. That means halving the sequence after every rerun, up to the point when the failure stops occurring, and then similarly reducing the sequence in the range between the last sequence that still triggers the error and the first sequence that does not.

In addition to decreasing the number of test cases in the attack sequence, less typical sequences should be tested. For instance, looping the last test case that was sent before the crash or looping the whole group from which the test case originates. This approach is necessary to trigger errors like buffer overflow.

The final goal of investigating a vulnerability is fixing the bug. All discovered vulnerabilities should be reported to the vendor in a responsible manner. In order to prove its existence and speed up verification process, a proof of concept can be prepared and provided to the vendor of the faulty device or software.

The last step of the systematic approach to PLC fuzz testing is to prepare a full documentation that consists of a description of the scope of testing and detailed results for each test suite. This approach can help enhance existing fuzzing procedures and aid further tests.

## 6. SUMMARY AND OUTLOOK

The proposed systematic approach to fuzz testing of programmable logic controllers allows the examination of new and existing devices in order to find vulnerabilities associated with incorrect protocol processing. Such testing increases security of critical systems where industrial devices, such as PLCs, are widely used. It is worth emphasizing that the testing effectiveness of testing strongly depends on the quality of the fuzzing tools used. A fuzzer generating as many different inputs as possible should increase probability of finding a vulnerability. It should be noted that a tested PLC presenting no failures does not mean that it is flawless or that there are no hidden vulnerabilities contained in it. No testing can cover every single possible combination of test cases and external, concurrent circumstances that may result in unexpected event causing

device failure. The goal is only to find as many vulnerabilities as possible in a systematic way, thus reducing a threat of undetected ones.

Additionally, the research shows the effectiveness of the fuzz testing method. It was proven that it is possible to find a new vulnerability in a popular device thanks to systematic searching through millions of combinations in an automatic way.

The presented laboratory could be extended in several ways. There is a need to equip the fuzzer with additional different protocols used in the automation industry. Next, more PLCs and different industrial devices could be tested using the presented systematic approach. In order to investigate how exploiting vulnerabilities (such as the CVE-2018-13805 discovered in this work) affects whole critical systems, attack scenari could be created and analyzed.

## REFERENCES

- [1] CHEN, X., LI, Q., “Research on industrial control devices flaw discovery technology”, International Conference on Advances in Mechanical Engineering and Industrial Informatics (Proc. Int. Conf. Zhengzhou, China, 2015), Atlantic Press, (2015).
- [2] CHERDANTSEVA, Y., BURNAP, P., BLYTH, A., EDEN, J., JONES, K., SOULSBY, H., STODDART, K., A review of cyber security risk assessment methods for SCADA systems, *Computer&Security* 56 (2016) 1-27.
- [3] LEMOS, R., *SecurityFocus* (2007), [www.securityfocus.com/news/11465](http://www.securityfocus.com/news/11465).
- [4] What is Fuzzing: The Poet, the Courier, and the Oracle, whitepaper, Synopsys, San Francisco, USA, 2017.
- [5] DAWSON, T., *Interact Analysis* (2018), <https://www.interactanalysis.com/who-were-the-leading-vendors-of-industrial-controls-plcs-and-dcs-in-2017/>
- [6] DE FALCO, M., *Stuxnet Facts Report: A Technical and Strategic Analysis*, NATO CCDCOE, Tallinn, Estonia, 2012.
- [7] LEITENMAIER, T., MAYER, D., SOLOVJOVS, D., *ProFuzz* (2012), [www.github.com/HSASec/ProFuzz](http://www.github.com/HSASec/ProFuzz).
- [8] ATIMORIN, *Profinet Set Fuzzer* (2014), [www.github.com/atimorin/scada-tools](http://www.github.com/atimorin/scada-tools).
- [9] DÉJÀ VU SECURITY, *Peach Fuzzer* (2014), [www.community.peachfuzzer.com/WhatIsPeach.html](http://www.community.peachfuzzer.com/WhatIsPeach.html).
- [10] SYNOPSYS, *Defensics Fuzz Testing*, [www.synopsys.com/software-integrity/security-testing/fuzz-testing.html](http://www.synopsys.com/software-integrity/security-testing/fuzz-testing.html).
- [11] WIRESHARK FOUNDATION, *Wireshark Home Page* (2019), [www.wireshark.org](http://www.wireshark.org).
- [12] DIONDI, P., *Scapy* (2019), [www.scapy.net](http://www.scapy.net).
- [13] MOLENAAR, G., PREEKER, S., *A Python Wrapper For the snap7 PLC Communication Library* (2019), [www.github.com/gijzelaerr/python-snap7](http://www.github.com/gijzelaerr/python-snap7).
- [14] SIEMENS CERT, *Siemens Security Advisories* (2019), [new.siemens.com/global/en/products/services/cert.html#SecurityPublications](http://new.siemens.com/global/en/products/services/cert.html#SecurityPublications).
- [15] SIEMENS ProductCERT, *SSA-347726* (2019), <https://cert-portal.siemens.com/productcert/pdf/ssa-347726.pdf>.