

# Enhancing ICS Security Diagnostics with Pseudo-Greybox Fuzzing During Maintenance Testing

Kazutaka Matsuzaki<sup>1</sup><sup>a</sup> and Shinichi Honiden<sup>2</sup><sup>b</sup>

<sup>1</sup>*Faculty of Global Informatics, Chuo University, Tokyo, Japan*

<sup>2</sup>*Faculty of Science and Engineering, Waseda University, Tokyo, Japan*

**Keywords:** Industrial Control Systems, Pseudo-Greybox Fuzzing, Maintenance Testing, Security Diagnostics, Stateful Protocol Fuzzer, Network Fuzzer, ICS Monitoring.

**Abstract:** This paper presents a novel Pseudo-Greybox Fuzzer (pseudo-GBF) methodology designed to improve the security diagnosis of Industrial Control Systems (ICS) during maintenance testing. The proposed method combines stateful protocol fuzzing, network fuzzing, and ICS monitoring to optimize the coverage of state transitions in the system under test (SUT) while operating within the constraints of on-site maintenance testing. Pseudo-GBF enhances security testing by utilizing replayable seeds to trigger specific state transitions, enabling efficient and practical testing. By incorporating Pseudo-Greybox Fuzzing during maintenance testing, the methodology addresses the challenges faced in ICS security diagnostics, leading to improved security and resilience of critical infrastructure systems. This paper provides a comprehensive overview of the system design, including integrating stateful protocol fuzzing, network fuzzing, and ICS monitoring, demonstrating its potential to advance ICS security testing.

## 1 INTRODUCTION

Industrial control systems (ICS) are vital components of modern critical infrastructures, responsible for managing and controlling critical processes across various industries. As these systems become increasingly interconnected and integrated with the broader IT infrastructure, the risk of cyber-attacks escalates, posing significant threats to both safety and security.


Fuzzing technology, a widely used method for discovering software vulnerabilities, has existed since the early 1990s. Fuzzing generates syntactically or semantically invalid inputs and monitors the program's output for anomalies. Attackers often employ fuzzing for exploit generation or intrusion testing. In recent years, network fuzzing has gained popularity in industries utilizing embedded devices in ICS.


Researchers have proposed various enhancements to fuzzing techniques, such as coverage-guided fuzzing to improve test coverage and the integration of machine learning for efficient testing. Particularly,

research on fuzzing technologies has centered on their application in industrial control systems. For example, conducting such tests before deploying ICS components in power management systems or smart grids can effectively identify unknown vulnerabilities, preventing attackers from exploiting them (Zhu 2023, Boehme 2021, Serpanos 2021).

However, ICS typically have long lifecycles and undergo partial renewals, necessitating long-term security maintenance. In practice, security measures are primarily implemented during the system's development phase, while security patches are applied to external devices like network equipment during operation. This approach can result in a vulnerable state that relies on perimeter defences.

This paper proposes a method that combines fuzzing technology with patching to broaden the application scope and extend lifecycle support for ICS security. The proposed method aims to address the challenges faced by ICS security measures by offering a comprehensive and practical approach that integrates fuzzing and patching techniques to ensure the robust security of industrial control systems.

<sup>a</sup> <https://orcid.org/0000-0003-2337-2686>

<sup>b</sup> <https://orcid.org/0000-0003-1385-3996>

Furthermore, by combining these strategies, we aim to enhance the resilience of ICS throughout their entire lifecycle, effectively mitigating potential vulnerabilities and reducing the risk of cyber-attacks.

The remainder of this paper is structured as follows: Section 2 provides the background of this paper, covering fuzzing techniques for network protocol implementations, fuzzing techniques for ICS security, and cybersecurity aspects of IEC 61850-based systems and distributed energy resources. Section 3 explains the challenge, and section 4 discusses the proposed method that combines fuzzing technology with patching to enhance the security of industrial control systems throughout their lifecycle. Section 5 shows the evaluation of the proposed methodology. Section 6 discusses several topics regarding pseudo-GBF, and section 7 shows related work. Finally, Section 8 concludes.

## 2 BACKGROUND

This section overviews the challenges and issues associated with patch management in industrial control systems and the complexities of multiple states in ICS protocols.

### 2.1 Patch Management of ICS

Patch management for ICS, as outlined in the international standard for control system security, IEC 62443-2-3, faces several challenges in ensuring security risks are adequately mitigated during maintenance. These challenges include:

- **Minimizing downtime:** In ICS, availability is of paramount importance. System downtime can result in high costs and operational disruptions. Patch management must balance maintaining system availability and addressing security vulnerabilities.
- **Limited time for vulnerability assessment:** Due to the need to minimize downtime, patch management must identify and address vulnerabilities within a narrow time frame, which can hinder the effectiveness of vulnerability assessments.
- **Independent verification of security:** Security certification schemes based on IEC 62443 and other standards require that an independent organization verify security measures. In practice, the burden of diagnosis by an independent organization is substantial, and the implemented measures may need to be revised to address the underlying security risks.

- **Inconsistent test quality:** While security certification schemes mandate the implementation of specific test items, application-specific tests are often overlooked. Conversely, penetration tests may include application-specific tests, but the test quality may not be uniform across different systems.
- **Focus on known vulnerabilities:** Given the practical constraints, the only feasible solution is often to confirm the absence of known vulnerabilities in the operating system or components that can be remotely exploited. However, this approach may overlook unknown or emerging threats, exposing systems to potential attacks.

Addressing these challenges in patch management for ICS is crucial for maintaining the security and resilience of industrial control systems against the ever-evolving landscape of cyber threats.

### 2.2 Multiple States in ICS Protocols

Modern ICS often employ stateful protocols, such as IEC 61850-MMS, RTPS, DDS, MQTT, etc., to efficiently synchronize various systems. These current protocols exhibit more complex state transitions than traditional protocols like IEC 60870, Modbus, and DNP3, which mainly focus on simple read and write values operations. In particular, the IEC 61850 protocol is an example of stateful protocols with intricate state transitions. The challenges associated with multiple states in current ICS protocols include the following:

**Stateful Protocol Analysis:** Given the stateful nature of current ICS protocols, more than traditional fuzzing techniques may be required for effective vulnerability identification. Fuzzing methods must consider various states and transitions in the protocol to generate meaningful test cases and uncover potential weaknesses.

**State-Aware Vulnerability Detection:** The complex state transitions in current ICS protocols necessitate state-aware vulnerability detection tools and techniques capable of recognizing and accounting for specific states and transitions that may introduce security risks.

Addressing the challenges multiple states pose in current ICS protocols is crucial for enhancing the security and resilience of industrial control systems.

### 3 CHALLENGES IN ICS SECURITY DIAGNOSE

This section focuses on the challenges and priorities associated with the constraints in developing security diagnostic frameworks for ICS. Maintaining system availability is of utmost importance in ICS, leading to severe restrictions on time allocated for on-site security testing during maintenance operations. Moreover, the inability to embed test code within the installed software presents additional difficulties.

#### 3.1 Constraints in Security Diagnostic Frameworks

Developing a security diagnostic framework for ICS faces two significant constraints:

1. The inability to embed test code in the installed software.
2. Limited time for diagnosis at the installation site.

These constraints introduce several technical challenges:

- Recompiling software is impossible, preventing the implementation of greybox fuzzing with coverage measurements.
- For stateful protocol systems, fuzzing test coverage is biased, making testing states reached after multiple message exchanges challenging. This issue can result in missed bugs and wasted time.

#### 3.2 Challenges in Advanced Fuzzing

Most fuzzing research can be conducted on a PC within a virtual environment, using a debugger to monitor the process under test. However, in ICS, advanced fuzzing is challenging in the field due to the need for hardware monitoring to support IT/OT convergence. This hardware integration complicates the fuzzing process and presents additional challenges in securing ICS effectively.

## 4 SYSTEM DESIGN

This paper proposes a methodology for incorporating security diagnostics during maintenance testing of ICS. The method, Pseudo-Greybox Fuzzing (pseudo-GBF), aims to enhance security testing during the system life cycle.

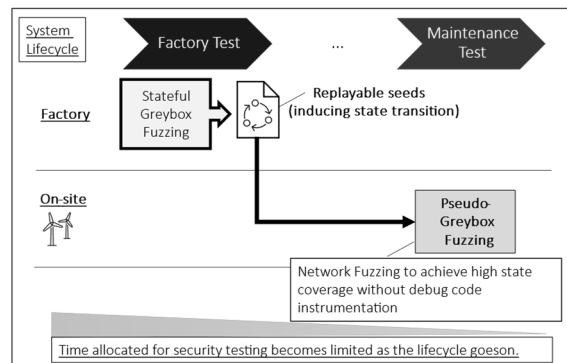


Figure 1: Overview of security testing for smart grid security diagnose.

#### 4.1 Pseudo-Greybox Fuzzing for Security Diagnosis in Maintenance Test

Pseudo-GBF involves performing security testing with an awareness of the state coverage of the test target. Since there is generally enough time for security testing during the factory test in the system life cycle, this method utilizes fuzzing to optimize the coverage of state transitions in the system under test.

The outcome of the fuzzing process is a set of replayable seeds capable of triggering specific state transitions in the system under test (SUT). Network fuzzing uses these replayable seeds to ensure efficient and effective security testing during maintenance.

Figure 1 illustrates the overall process of the proposed pseudo-GBF methodology. By incorporating pseudo-GBF during maintenance testing, the proposed method addresses the challenges and constraints faced in ICS security diagnostics, leading to improved security and resilience of critical infrastructure systems.

The pseudo-GBF methodology for security diagnosis in maintenance tests combines multiple components: *Stateful Fuzzer*, *Network Fuzzer*, and *ICS Monitoring*. These components work together to enable efficient and effective testing, even under on-site constraints.

##### 4.1.1 Stateful Protocol Fuzzer

Stateful protocol fuzzer considers various possible states, including state transitions defined in the protocol, state transitions modeling the interaction with the PUT as states, and state transitions modeling the flow structure of the program. Examples of these states include state transitions in protocols specified in IEC 61850, state transitions considered according to the sequence patterns of responses returned by TCP

and UDP as implemented in AFLNet (Pham 2020) and StateAFL (Roberto 2022), and state transitions tracking the internal structure of a PUT program as a model. In addition, general coverage-guided fuzzing is included in this category.

The functional requirement of the stateful protocol fuzzer is the ability to persist replayable seeds that can be retransmitted later. The performance requirement is to increase state coverage.

#### 4.1.2 Network Fuzzer

In maintenance testing, the PUT is either embedded in hardware, becoming a Device Under Test (DUT), or embedded in a system, becoming a System Under Test (SUT). Test equipment must be brought onsite and tested from the network interface. Since this is a production environment, operations like debug builds, which can introduce vulnerabilities, are not permitted, and embedding code is not allowed.

The functional requirements of the network fuzzer are the ability to retransmit over the network with replayable seeds as input and adjust the retransmission interval. If the retransmission speed is too fast, the DUT or SUT may be unable to process the retransmissions, rendering the test ineffective.

#### 4.1.3 ICS Monitoring

ICS Monitoring diagnoses security by receiving fuzzing feedback from the DUT and SUT. In addition, it checks for the maintenance of essential functions as defined in IEC 62443. This monitoring applies to digital outputs, analog outputs, serial outputs, heartbeats, LEDs, etc. The functional requirements of ICS Monitoring include having a physical monitoring interface and being able to verify the maintenance of essential functions.

Combining these components, the proposed pseudo-GBF methodology provides a comprehensive solution for effective security diagnosis during maintenance testing in ICS.

### 4.2 System Implementation

The following describes an example implementation of the proposed configuration detailed in the previous subsection. During the factory testing phase, we utilized the IEC 61850-MMS extension of AFLNet as the stateful protocol fuzzer within a container on a PC. For on-site testing, we implemented the network fuzzer function by integrating *aflnet-replay*, a

retransmission tool of AFLNet, with *afl-cov*<sup>3</sup>, a coverage measurement tool for AFL-based tools.

*Afl-cov* is employed to gather and parse the output of *afl-fuzz*, while *aflnet-replay* is used to transmit these outputs to the on-site DUT/SUT. Consequently, we utilized dedicated hardware to monitor digital signals and other relevant data. In this specific implementation, the Achilles Testing Platform (ATP)<sup>4</sup> was used.

## 5 EVALUATIONS

### 5.1 Experiment Setup

Table 1 presents the evaluation experiment combinations. The evaluation criteria are as follows:

1. The number of test cases and the time required.
2. The number of crashes detected by fuzzing.
3. The code coverage achieved by fuzzing.

We compared the performance of AFLNet, which was extended in this study, with pseudo-GBF, which utilizes AFLNet's replayable seeds. The evaluation targets were the libiec61850 application in versions 1.4.0 and 1.5.0, containing known vulnerabilities, and version 1.5.1, the latest version at the time of writing.

Message sequences containing control commands were provided as initial seeds for the target. The results of each experiment were averaged across three repetitions.

The figure illustrates the structure of the pseudo-GBF implementation. The upper half of the figure represents the *Factory*, while the lower half depicts the *On-Site* test.

In the *On-Site* case, physical monitoring is necessary, so *Fuzzing* and *Monitoring* are conducted on separate hosts. The environment for *Fuzzing* is the same as in the *Factory*. However, for *Monitoring*, the DUT/SUT ethernet ports and digital output are monitored by the *Test Device*. Although the monitoring environment is similar to the *Factory* setup, the ATP monitors the DUT/SUT's ethernet port and digital output.

The experiment was conducted using PCs with two vCPUs and 8 GB of memory for both fuzzer and DUT.

<sup>3</sup> <https://github.com/vanhauser-thc/afl-cov>

<sup>4</sup> <https://www.ge.com/digital/applications/achilles-communications-certified-products>

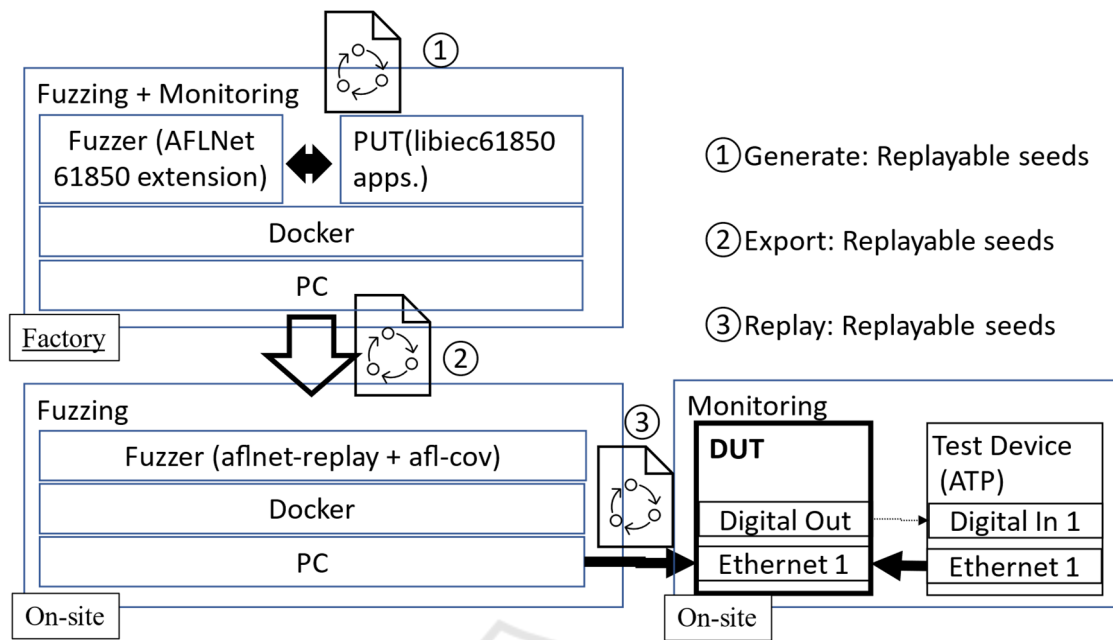


Figure 2: Experiment Settings for Evaluation.

Table 1: Combination table of evaluation experiments.

Test target		Proposed method	Base method
Lib version	App.	pseudo-GBF	AFLNet
v1.4.0	basic_	num. of testcases / time	
v1.5.0	io	num. of crashes found/ time	
v1.5.1		% of path coverage / time	

## 5.2 Number of Testcases

Evaluate the overall volume of test cases and the time it takes to execute them. Table shows the number of test cases executed.

In the Factory, 500,000 to 800,000 test cases were executed in 24 hours, and since the tool is based on AFL, it is difficult to determine that the entire test was covered in 24 hours.

On-site, about 1,000 replayable seeds were run. The time required was less than 1 hour, with a timeout setting of 5 seconds.

Table 2: Number of Testcases.

	v1.4.0	v1.5.0	v1.5.1
Factory(24h)	721,174	536,432	543,011
On-site	989	757	1,015

Table 3: Number of test cases for Comparison (Open sourced software and accredited software for security certification schemes).

	v1.4.0	v1.5.0	v1.5.1
boofuzz(61850-fuzzing)	12,800,000		
ATP	168,069	168,335	168,694

Table shows the values for the prominent example of open-source software (IEC 61850 MMS-compliant version of boofuzz<sup>5</sup>) and one of the prominent examples of an accredited tool (ATP) in certification schemes as a reference.

In the case of boofuzz, all numbers are determined by default values, but in this evaluation environment, only about 10% of all tests were performed in 24 hours.

In the case of accredited tools, pre-determined test cases are conducted. The variation in the number of cases is the effect of changing the behavior depending on the monitoring situation, but there is no significant difference between the versions.

## 5.3 Number of Crashes Detected

Table 3 shows the number of unique crashes detected. Both methods reproduced the same number of crashes. Strictly speaking, ensuring that the DUT is back to the state of listening on TCP port 102 before sending

<sup>5</sup> <https://github.com/fkie-cad/61850-fuzzing>

replayables is necessary. This needs to be done in conjunction with proper monitoring.

Table 4: Number of unique crashes.

	v1.4.0	v1.5.0	v1.5.1
Factory (24h)	27	20	0
On-site	27	20	0

In the IEC 61850 MMS-compliant version of boofuzz and ATP, no crashes were detected in 24 hours for boofuzz. The order of execution may have had an impact since we did not run all of the defined fuzz, although, in the case of ATP, the vulnerable versions (1.4.0, 1.5.0) correctly indicated a test failure. However, the test is terminated when the DUT response does not return within a set time (15 sec), and it cannot determine unique crashes.

## 5.4 Test Coverage

Table 4 and Table 5 present the test coverage (SLOC) for each version of libiec61850 and the overall coverage, including both the application and library components. Although these tables display the actual measured values, it is challenging to make pure comparisons due to significant differences caused by the tools used.

In the *Factory*, we used *afl-cov* to measure the coverage based on AFLNet runs. *On-site* measurements were conducted using *llvm-cov*. There is no substantial difference in coverage, and the results in Table 5 are considered sufficient. The lower values in Table 4 compared to Table 5 can be attributed to excluding code counts related to the application's end and function names.

The inability to conduct identical measurements stems from the fact that both AFL and LLVM perform instrumentation to insert functions into the generated code, which affects AFL results, and the incompatibility between *llvm-cov* and AFL.

Table 5: Test coverage (SLOC, Factory).

	v1.4.0	v1.5.0	v1.5.1
App.	23.9 % 92 SLOC	25.5 % 94 SLOC	22.5 % 111 SLOC
Lib.	7.9 % 34,413 SLOC	7.9 % 37,166 SLOC	8.0 % 38,105 SLOC

Table 6: Test coverage (SLOC, On-site).

	v1.4.0	v1.5.0	v1.5.1
App.	63.7 % 102 SLOC	67.9 % 105 SLOC	66.9 % 121 SLOC
Lib.	15.3 % 32,803 SLOC	15.5 % 35,672 SLOC	19.4 % 36,617 SLOC

## 6 DISCUSSIONS ON PSEUDO-GBF

This section discusses the application of pseudo-GBF, the framework presented in this paper, and compares it with other alternatives.

### Effects of Initial Seed on Fuzzing

Mutation-based fuzzing performance generally varies significantly depending on the initial seed. AFLNet uses real-world inputs as the initial seed. In this paper, we also use the minimum communication required for a program implementing IEC 61850 as the initial seed. This approach seems more efficient than 61850-fuzzing, which automatically generates semi-random inputs from the IEC 61850 protocol but requires more time and effort. Ideally, we would like to generate initial seeds automatically by recording test cases during normal functional tests and converting them into initial seeds. Although the problem of providing initial seeds remains, even when focusing on IEC 61850, the system is model-based, so it can be largely automated. Combining symbolic execution tools and fuzzers for providing seeds is a topic for future research.

### Pluggable Architecture

This test was realized by combining the IEC 61850-MMS extension of AFLNet with *afl-cov* and ATP. This combination is flexible, and that pseudo-GBF can be reproduced in principle, even with pioneering tools such as Polar (Luo 2019). However, in the case of a system that combines components from multiple manufacturers, real hurdles, like combining tests conducted at each factory, are high. Further research is needed to address such real-world use cases.

### Other Techniques That Can Be Used For CPS Security Diagnostics Other than Fuzzing

This paper assumes fuzzing to realize security diagnosis. However, it might be possible to automate minimum security control measures by performing vulnerability assessments. Combining vulnerability scanning with known vulnerability assessments of libraries and application-specific assessments can be beneficial. However, to reduce testing time, it would

be preferable to know whether vulnerabilities exist without conducting actual scans by managing the SBOM (software bill of materials).

#### **Current Tools and Practices may be Sufficient**

It might appear that the pseudo-GBF framework can be realized with current tools, such as the ATP referenced in the evaluation. However, a mechanism generating generic fuzz based on protocol definitions is similar to the current practice of conducting tests at the Factory and improving them to pass them. Increasing test coverage of the application-specific processing part is challenging, and the difference in the performance of greybox fuzzers, such as AFLNet, is likely to manifest itself as a difference in test accuracy.

## **7 RELATED WORK**

### **Fuzzing in CPS/IACS (for Systems)**

Several works focus on testing entire systems rather than individual devices, targeting CPS and industrial IoT systems, including ICS. In one study, an intelligent fuzzing approach is proposed to target combinations of possible parameters in the control system for fuzzing (Chen 2019). Although this approach has a high degree of abstraction, it differs from traditional fuzzing that targets software and individual devices. Renewable energy systems adopting IEC 61850 are also advancing fuzzing and security measures. In one case, fuzzing is performed on systems implemented with IEC 61850, and methods to confirm robustness have been presented and evaluated (Matsuzaki 2020).

### **Fuzzing in Control System Protocol Implementation (for Devices)**

Fuzzing in control systems has been extensively researched, particularly focusing on Programmable Logic Controllers (PLCs), one of the key components in such systems. ICSFuzz, for example, enables the fuzzing of control applications by rewriting binaries in situations with no source code (Tychalas 2021). Polar addresses the inefficiency of existing mutation-based fuzzing for ICS protocols with function codes by using static program analysis to create an abstract syntax tree for more efficient fuzzing (Luo 2019).

For IoT devices, Snipuzz presents a black-box fuzzing approach that targets the firmware by inferring message snippets and making mutations based on device reactions (Feng 2021). This technique addresses the challenges associated with firmware acquisition and emulation, which often

make black-box fuzzing less accurate and harder to optimize.

ICS3Fuzzer is a framework for discovering protocol implementation bugs in ICS supervisory software by fuzzing (Fang 2021). The prevalent use of proprietary protocols is an issue when fuzzing supervisory software. The framework provides a state-book mechanism to grasp state transitions from the execution trace of the supervisory software and the corresponding inputs, allowing it to focus fuzzing on the target state.

### **Fuzzing Stateful Protocol Implementations**

Fuzzing stateful protocol implementations presents unique challenges due to the need for multiple rounds of interaction with the test subject to reach the desired fuzzing points. Chen et al. explore effective fuzzing strategies for analyzing communication protocols, addressing the limitations of non-stateful greybox fuzzers (Chen 2019).

Several fuzzing tools have been proposed to extend AFLNet and support stateful protocols. SnapFuzz, for example, adds a snapshot mechanism to AFLNet, handling the overhead of recreating the target state (Chen 2019). The snapshot mechanism enables easy saving and restoring of the execution states of the target. StateAFL extends AFLNet and degenerates the state transition model to improve fuzzing efficiency when constructing the state transition model from the Device Under Test (DUT) response (Roberto 2022).

FairFuzz is another extension of AFL that focuses on increasing greybox fuzz testing coverage by employing a targeted mutation strategy (Caroline 2018). This approach shares similarities with pseudo-GBF, which utilizes AFLNet results and aims to cover rare branches by generating fuzz input using AFL.

## **8 CONCLUSION**

This paper has presented a novel Pseudo-Greybox Fuzzer (pseudo-GBF) methodology for enhancing the security diagnosis of ICS during maintenance testing. The proposed method addresses the challenges and constraints faced in ICS security diagnostics by optimizing the coverage of state transitions in the system under test (SUT) using replayable seeds.

The pseudo-GBF methodology integrates stateful protocol fuzzing, network fuzzing, and ICS monitoring, allowing for efficient and effective testing within the constraints of on-site maintenance testing.

Future research can expand on the presented methodology by exploring further optimization techniques for stateful protocol fuzzing, refining network fuzzing capabilities to work with a wider range of systems, and enhancing ICS monitoring for better detection of vulnerabilities and anomalies.

## REFERENCES

- Zhu, X., Wen, S., Camtepe, S., & Xiang, Y. (2022). Fuzzing: A Survey for Roadmap. *ACM Computing Surveys*, 54(11s), Article 230. <https://doi.org/10.1145/3512345>
- Boehme, M., Cadar, C., & Roychoudhury, A. (2021). Fuzzing: Challenges and Reflections. In *IEEE Software*, 38(3), pp. 79-86. doi: 10.1109/MS.2020.3016773
- Serpanos, D., & Katsigiannis, K. (2021). Fuzzing: Cyberphysical System Testing for Security and Dependability. In *Computer*, 54(9), pp. 86-89. doi: 10.1109/MC.2021.3092479
- V. -T. Pham, M. Böhme and A. Roychoudhury, "AFLNET: A Greybox Fuzzer for Network Protocols," 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), 2020, pp. 460-465, doi: 10.1109/ICST46399.2020.00062.
- Roberto Natella. 2022. StateAFL: Greybox fuzzing for stateful network servers. *Empirical Softw. Engg.* 27, 7 (Dec 2022). <https://doi.org/10.1007/s10664-022-10233-3>
- Luo, Z., Zuo, F., Jiang, Y., Gao, J., Jiao, X., & Sun, J. (2019). Polar: Function Code Aware Fuzz Testing of ICS Protocol. *ACM Trans. Embed. Comput. Syst.*, 18(5s), Article 93, 22 pages. <https://doi.org/10.1145/3358227>
- Chen, Y., Poskitt, C. M., Sun, J., Adepu, S., & Zhang, F. (2019). Learning-guided network fuzzing for testing cyber-physical system defences. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE '19)*. IEEE Press, 962–973. DOI: <https://doi.org/10.1109/ASE.2019.00093>
- Wilkerson, C., & Hariri, M. E. (2022). IEC 61850-Based Renewable Energy Systems: A Survey on Cybersecurity Aspects. In *2022 IEEE International Conference on Environment and Electrical Engineering and 2022 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*. Prague, Czech Republic, 2022, pp. 1-6, doi: 10.1109/EEEIC/ICPSEurope54979.2022.9854539
- Matsuzaki, K., Sawabe, N., Maeda, R., Suzuki, D., Matsuura, T., & Hamada, H. (2020). Cybersecurity Evaluation Methodology for Distributed Energy Resources: Industrial Demonstration. In *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*. Singapore, 2020, pp. 2169-2174, doi: 10.1109/IECON43393.2020.9254422
- Tychalas, D., Benkraouda, H., & Maniatakos, M. (2021). ICSFuzz: Manipulating I/Os and Repurposing Binary Code to Enable Instrumented Fuzzing in ICS Control Applications. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2847-2862.
- Feng, X., Sun, R., Zhu, X., Xue, M., Wen, S., Liu, D., Nepal, S., & Xiang, Y. (2021). Snipuzz: Black-box Fuzzing of IoT Firmware via Message Snippet Inference. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*, pp. 337-350. <https://doi.org/10.1145/3460120.3484543>
- Fang, D., Song, Z., Guan, L., Liu, P., Peng, A., Cheng, K., Zheng, Y., Liu, P., Zhu, H., & Sun, L. (2021). ICS3Fuzzer: A Framework for Discovering Protocol Implementation Bugs in ICS Supervisory Software by Fuzzing. In *Annual Computer Security Applications Conference (ACSAC '21)*, pp. 849-860. <https://doi.org/10.1145/3485832.3488028>
- Chen, Y., Lan, T., & Venkataramani, G. (2019). Exploring Effective Fuzzing Strategies to Analyze Communication Protocols. In *Proceedings of the 3rd ACM Workshop on Forming an Ecosystem Around Software Transformation (FEAST'19)*, pp. 17-23. <https://doi.org/10.1145/3338502.3359762>
- Caroline Lemieux and Koushik Sen. (2018). FairFuzz: a targeted mutation strategy for increasing greybox fuzz testing coverage. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18)*. Association for Computing Machinery, New York, NY, USA, 475–485. <https://doi.org/10.1145/3238147.3238176>.