A Stateful Fuzzer for the TCP/IP Stack of the Real-Time Operating System Zephyr

Valentin Huber

at Cyber Defence Campus

and Institute of Computer Science at ZHAW

contact@valentinhuber.me

January 15, 2025

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Keywords: Software Testing, Fuzzing, Stateful Fuzzing, Zephyr, LibAFL.

Contents

1 Introduction

- Fuzzing is a widely used strategy to find software defects
- Overview: How does fuzzing work?

1.1 Zephyr

- Zephyr is an open-source RTOS developed by the Linux Foundation.
- Overview: Where is Zephyr used?
- The TCP/IP stack in Zephyr is particularly critical for the security of many products Zephyr is used in.

1.2 Fuzzing Network Stacks Is Hard

1.2.1 Deep integration with OS

- Environment/Hardware modelling necessary
- Usually, whole system needs to be run, which is a performance issue
- Instrumentation (Coverage, Sanitizers) is hard

1.2.2 Network packets are highly structured

- Dependencies between data (length fields, checksums, etc.)
- Dependencies between packages (state in TCP)

1.2.3 TCP Stacks Have an Internal State

- A certain packet results in different behavior depending on previous activity

1.3 Quotes, TODO: Integrate

Daniele *et al.* introduce a taxonomy of stateful fuzzing in which they define a stateful system as “a system that takes a sequence of messages as input, producing outputs along the way,and where each input may result in an internal state change” [1].

“To avoid confusion, we reserve the term message or input message for the individual input that the System Under Test (SUT) consumes at each step and the term trace for a sequence of such messages that make up the entire input.” [1]

“The input language of a stateful system consists of two levels: (1) the language of the individual messages, which we will refer to as the message format, and (2) the language of traces, built on top of that. A description or specification of such an input language will usually come in two parts, one for each of the levels: for example, a context-free grammar for the message format and a finite state machine describing sequences of these messages. We will call the latter the state model or, if it is described as a state machine, the protocol state machine.” [1]

“the internal state changes increase the state space that we try to explore: it may be hard for a fuzzer to reach ’deeper’ states. Indeed, fuzzing stateful systems is listed as one of the challenges in fuzzing by Boehme *et al.* [2]” [1].

1.3.1 TCP/IP

Checksums such as in TCP/IP are challenging. [3] Approaches to deal with this include removing the program sections from the target program [4].

1.4 Contributions of this Project

A fuzzer that targets the TCP/IP stack of Zephyr, which

- employs a framework that allows running Zephyr as a native executable, removing the necessity of an emulation or translation layer,
- introduces a custom ethernet driver based on shared memory to make fuzzing parallelizable and performant,
- guides mutation based on coverage information and a heuristic to infer the state of the TCP state machine based on flags in the TCP header of responses from the server

This project further

- evaluates how different ways of using the feedback described above influence fuzzer performance,
- evaluates different ways of modelling the packets passed to the fuzzer,
- evaluates different approaches to mutating those, and
- introduces various performance optimizations to LibAFL

2 Related Works

- Focus on Network Protocol Fuzzing

2.1 Protocol Fuzzing

2.1.1 Observing State

- Manual annotation
- Automatic annotation
- Other greybox approaches
- Heuristics

2.1.2 Use of State Information

- Feedback/Maximization
- Scheduling

2.1.3 Input Modelling and Mutation

2.2 Snapshotting

2.3 Data, TODO: Integrate

- *Stateful Greybox Fuzzing* [5]: "In this paper, we argue that protocols are often explicitly encoded using state variables that are assigned and compared to named constants [...] More specifically, using pattern matching, we identify state variables using enumerated types (enums). An enumerated type is a group of named constants that specifies all possible values for a variable of that type. Our instrumentation injects a call to our runtime at every program location where a state variable is assigned to a new value. Our runtime efficiently constructs the state transition tree (STT). The STT captures the sequence of values assigned to state variables across all fuzzer-generated input sequences, and as a global data structure, it is shared with the fuzzer." [5] Built on LibFuzzer

- *StateAFL: Greybox fuzzing for stateful network servers* [6]: compile-time probes observing memory allocation and I/O operations; state inference based on fuzzy hashing of long-lived memory areas.

- *Ijon: Exploring Deep State Spaces via Fuzzing* [7]: Manual annotations of code, to manually add entries to an AFL-style map (set/inc at calculated offset), include state information (variable values) in how edge coverage is calculated, and store the max value a certain variable reaches during execution for the fuzzer to then maximize.

- *SandPuppy: Deep-State Fuzzing Guided by Automatic Detection of State-Representative Variables* [8]: Ijon [7], but automatic (initial run capturing variable-value traces, analyze along with source code, add Ijon-style instrumentation, repeat during fuzzing)

- *The Use of Likely Invariants as Feedback for Fuzzers* [9]: run for 24 hours, record variable values and relationships between them, then add a feedback that rewards when the generated assertions are violated

- *Ankou: guiding grey-box fuzzing towards combinatorial difference* [10]: take combination of executed branches into consideration, reduce to manageable adaptive fitness function

- *FuzzFactory: domain-specific fuzzing with waypoints* [11]: framework to add custom feedbacks like number of basic blocks executed, amount of memory allocated, etc.

- *ParmeSan: Sanitizer-guided Greybox Fuzzing* [12]: Use sanitizers checks as fuzzing targets

- *Fuzzing with Data Dependency Information* [13]: Use execution of new data dependencies as feedback

- *StateFuzz: System Call-Based State-Aware Linux Driver Fuzzing* [14]: Find state variables (long-lived, can be updated by users, change control flow or memory access) using static analysis, use that to guide fuzzing (new coverage, new value-range, new extreme value). (Talk: Good Example of why coverage-guided alone is insufficient). Check value ranges instead of all values (static symbex!). 4-digit number of state varia les in linux kernel and Qualcomm MSM kernel (Google Pixel).

- *ProFuzzBench: a benchmark for stateful protocol fuzzing* [15]: Suite of 10 protocols and 11 open-source implementations of those to be tested. TCP is notably absent from this list. Certain protocols (like FTP) already return HTTP status codes, others are patched to do so. Dockerized. The authors note that configuration is not taken into account and multiparty (≥ 3) protocols can not be fuzzed right now. Non-determinism in the programs make feedback (like code coverage) less predictable and thus fuzzing less performant because it introduces non-differentiable duplicate entries into the corpus. Speed is another issue, where complex setup-processes, costly network operations (resp. synchronization for me), and long multipart-inputs contribute. Finally, state identification is only superficially handled.

- *AFLNET: A Greybox Fuzzer for Network Protocols* [16]: FTP and RTSP as targets, state transition (+coverage) feedback, corpus from traces, mutation on random entry (havoc and insert/delete/duplicate etc.), corpus scheduling based on statistics about each state.

- *TCP-Fuzz: Detecting Memory and Semantic Bugs in TCP Stacks with Fuzzing* [17]: TCP targets, differential fuzzing, input: syscalls + packets, complex mutators including intelligent dependencies between the current and all previous packets and syscalls in the input, feedback: inter-package diff in coverage.

- *FitM: Binary-Only Coverage-Guided Fuzzing for Stateful Network Protocols* [18]: Use persistent snapshots of userspace processes (CRIU), fuzzer-in-the-middle, multiple strategies such as input deduplication and resynchronization necessary because of the approach

- *Autofuzz: Automated network protocol fuzzing framework* [19]: Man-in-the-middle, learn protocol by constructing a FSA and packet syntax using a bioinformatics technique. Fuzz server and client.

- *EPF: An Evolutionary, Protocol-Aware, and Coverage-Guided Network Fuzzing Framework* [20]: Target SCADA, uses population-based simulated annealing to schedule which packet type to add/mutate next in conjunction with coverage feedback. Requires Scapy-compatible implementation of packet types to ensure packet structure.

- *A model-based approach to security flaw detection of network protocol implementations* [21]: Build FSM and minimize it, then use it to schedule basic mutations.

- *GANFuzz: a GAN-based industrial network protocol fuzzing framework* [22]: Learn GAN to generate next inputs. Targets Modbus-TCP.

- *Fuzzers for Stateful Systems: Survey and Research Directions* [1]: Provides taxonomy of components and categorizes stateful fuzzers, compares approaches and lists challenges and future directions.

- *Automated Attack Discovery in TCP Congestion Control Using a Model-guided Approach* [23]: Manual FSM from RFC to generate abstract attack models against congestion control implementations, which are then transformed to concrete attacks strategies and tested on actual implementations.

- *A Modbus/TCP Fuzzer for testing internetworked industrial systems* [24] is a Modbus fuzzer, that only seems to use TCP as transport. Generation of packets happens only for Modbus itself. *MTF-Storm: a High Performance Fuzzer for Modbus/TCP* [25] is follow-up work that does the same more systematically.

- *A survey on fuzz testing technologies for industrial control protocols* [26] review industrial control protocol. Notably, Modbus/TCP fuzzing seems common, but TCP is only used as transport layer, not target.

- *MTA Fuzzer: A low-repetition rate Modbus TCP fuzzing method based on Transformer and Mutation Target Adaptation* [27] use machine learning models to generate Modbus/TCP packets, but target Modbus and again only use TCP as a transportation layer.

- *A vulnerability detecting method for Modbus-TCP based on smart fuzzing mechanism* [28] is another Modbus/TCP fuzzer that does not fuzz the TCP stack. See also [29].

- A similar approach using a Scapy-based fuzzer was implemented for the industrial protocol EtherNet/IP, where TCP was declared as out-of-scope [30].

- *Fuzz Testing of Protocols Based on Protocol Process State Machines* [31] calculate a directed graph from the measured state variables, and schedule mutations based on a formula incorporating state depth, coverage, number of transitions, and number of mutations based on this state.

2.3.1 TODO: Read

- *A Survey of Protocol Fuzzing* [32]

3 Implementation

3.1 NativeSim

3.2 Network Interfaces

- Custom ethernet driver

- No changes to code, just added code, and includes in config/build system
- Shared memory layout/inner workings
- Manual Responses

3.3 Coverage Information

- LLVM pass: `trace-pc-guard`
- Added code/includes to build system
- Reset before input execution to improve reliability: Only code change

3.4 State Inference Heuristic

- Based on responses by server
- Parsing, different non-TCP responses classified
- On TCP packets, the header flags are used

3.5 Input Modelling and Mutation

3.5.1 Mutation Target

- Random (Multipart)
- ReplayingStateful
- Seeding

3.5.2 Input Modelling

- Byte array vs. parsed structure

3.5.3 Mutators

- `havoc_mutations()`
- TCP only, certain fixed values
- Full stack, certain fixed values
- Full stack, fixing mutators
- Appending

3.6 Implementation Details

3.6.1 LibAFL Structure

- Input Scheduling Algorithm
- Mutation Scheduling Algorithm
- Oracles
- MapObserver/-Feedback
- Executor

3.6.2 Helper Functionality

- Manual connection
- PCAP
- Scripting

3.7 LibAFL

3.7.1 Overcommit

3.7.2 Numeric Mutators

3.7.3 Mapping Mutators

3.7.4 TODO: others

4 Results

4.1 Throughput and Overcommit

4.2 Input Modelling and Mutation

With fixed feedback:

- `MultipartInput` and `havoc_mutations()`
- `ReplayingStatefulInput` and `havoc_mutations()`
- `ReplayingStatefulInput` and parsed structure with different mutators

4.3 Feedback

- Coverage (repeatability!)
- State marking
- State diff marking

5 Discussion

5.1 Feedback

- How effective do different improvements seem to be

5.2 Input Modelling and Mutation

- How effective do different improvements seem to be

5.3 Future Work

5.3.1 Improved Performance

- This was not prioritized further because memory was the limiting factor
- Semaphores instead of busy waiting

5.3.2 Alternate Targets

- Other OS network stacks
- Userland network stacks

5.3.3 Improved Oracles

- e.g. Differential Fuzzing
- Sanitizers

5.3.4 Improved Scheduling Based on State Feedback

5.3.5 Generilazation of Techniques

5.3.6 Extended Evaluation

- System calls
- IPv6
- Other network protocols
- Comparison to other fuzzers (not done because none were applicable without extensive engineering work to ensure compatibility with target)

5.4 Contributions

- Summary of paper

In the interest of open science, the source code of this project is publicly available and released under an open-source license. During development, thousands of lines of code have been introduced to multiple upstream projects.
All artifacts produced for this project are available at

github.com/riesentoaster/fuzzing-zephyr-network-stack.

Bibliography

[1] C. Daniele, S. B. Andarzian, and E. Poll, "Fuzzers for stateful systems: Survey and research directions," *ACM Comput. Surv.*, vol. 56, no. 9, Apr. 2024, ISSN: 0360-0300. DOI: `10.1145/3648468`. [Online]. Available: `https://doi.org/10.1145/3648468`.

[2] M. Boehme, C. Cadar, and A. ROYCHOUDHURY, "Fuzzing: Challenges and reflections," *IEEE Software*, vol. 38, no. 3, pp. 79–86, 2021. DOI: `10.1109/MS.2020.3016773`.

[3] H. Liang, X. Pei, X. Jia, W. Shen, and J. Zhang, "Fuzzing: State of the art," *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 1199–1218, 2018. DOI: `10.1109/TR.2018.2834476`.

[4] H. Peng, Y. Shoshitaishvili, and M. Payer, "T-fuzz: Fuzzing by program transformation," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 697–710. DOI: `10.1109/SP.2018.00056`.

[5] J. Ba, M. Böhme, Z. Mirzamomen, and A. Roychoudhury, "Stateful greybox fuzzing," in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, pp. 3255–3272, ISBN: 978-1-939133-31-1. [Online]. Available: `https://www.usenix.org/conference/usenixsecurity22/presentation/ba`.

[6] R. Natella, "Stateafl: Greybox fuzzing for stateful network servers," *Empirical Software Engineering*, vol. 27, no. 7, p. 191, Oct. 2022, ISSN: 1573-7616. DOI: `10.1007/s10664-022-10233-3`. [Online]. Available: `https://doi.org/10.1007/s10664-022-10233-3`.

[7] C. Aschermann, S. Schumilo, A. Abbasi, and T. Holz, "Ijon: Exploring deep state spaces via fuzzing," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1597–1612. DOI: `10.1109/SP40000.2020.00117`.

[8] V. Paliath, E. Trickel, T. Bao, R. Wang, A. Doupé, and Y. Shoshitaishvili, "Sandpuppy: Deep-state fuzzing guided by automatic detection of state-representative variables," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, F. Maggi, M. Egele, M. Payer, and M. Carminati, Eds., Cham: Springer Nature Switzerland, 2024, pp. 227–250, ISBN: 978-3-031-64171-8.

[9] A. Fioraldi, D. C. D'Elia, and D. Balzarotti, "The use of likely invariants as feedback for fuzzers," in *30th USENIX Security Symposium (USENIX Security 21)*, USENIX Association, Aug. 2021, pp. 2829–2846, ISBN: 978-1-939133-24-3. [Online]. Available: `https://`

www . usenix . org / conference / usenixsecurity21 / presentation/fioraldi.

[10] V. J. M. Manès, S. Kim, and S. K. Cha, “Ankou: Guiding grey-box fuzzing towards combinatorial difference,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE ’20, Seoul, South Korea: Association for Computing Machinery, 2020, pp. 1024–1036, ISBN: 9781450371216. DOI: 10.1145/3377811.3380421. [Online]. Available: https://doi.org/10.1145/3377811.3380421.

[11] R. Padhye, C. Lemieux, K. Sen, L. Simon, and H. Vijayakumar, “Fuzzfactory: Domain-specific fuzzing with waypoints,” *Proc. ACM Program. Lang.*, vol. 3, no. OOPSLA, Oct. 2019. DOI: 10.1145/3360600. [Online]. Available: https://doi.org/10.1145/3360600.

[12] S. Österlund, K. Razavi, H. Bos, and C. Giuffrida, “ParmeSan: Sanitizer-guided greybox fuzzing,” in *29th USENIX Security Symposium (USENIX Security 20)*, USENIX Association, Aug. 2020, pp. 2289–2306, ISBN: 978-1-939133-17-5. [Online]. Available: https :// www . usenix . org / conference / usenixsecurity20 / presentation/osterlund.

[13] A. Mantovani, A. Fioraldi, and D. Balzarotti, “Fuzzing with data dependency information,” in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, 2022, pp. 286–302. DOI: 10.1109/EuroSP53844.2022.00026.

[14] B. Zhao, Z. Li, S. Qin, *et al.*, “StateFuzz: System Call-Based State-Aware linux driver fuzzing,” in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, pp. 3273–3289, ISBN: 978-1-939133-31-1. [Online]. Available: https : / / www . usenix . org / conference / usenixsecurity22/presentation/zhao-bodong.

[15] R. Natella and V.-T. Pham, “Profuzzbench: A benchmark for stateful protocol fuzzing,” in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2021, Virtual, Denmark: Association for Computing Machinery, 2021, pp. 662–665, ISBN: 9781450384599. DOI: 10.1145/3460319.3469077. [Online]. Available: https://doi.org/10.1145/3460319.3469077.

[16] V.-T. Pham, M. Böhme, and A. Roychoudhury, “Aflnet: A greybox fuzzer for network protocols,” in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, 2020, pp. 460–465. DOI: 10.1109/ICST46399.2020.00062.

[17] Y.-H. Zou, J.-J. Bai, J. Zhou, J. Tan, C. Qin, and S.-M. Hu, “TCP-Fuzz: Detecting memory and semantic bugs in TCP stacks with fuzzing,” in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, USENIX Association, Jul. 2021, pp. 489–502, ISBN: 978-1-939133-23-6. [Online]. Available: https : / / www . usenix . org / conference/atc21/presentation/zou.

[18] D. Maier, O. Bittner, J. Beier, and M. Munier, “Fitm: Binary-only coverage-guided fuzzing for stateful network protocols,” Workshop on Binary Analysis Research, Internet Society, 2022. DOI: 10 . 14722 / bar . 2022 . 23008. [Online]. Available: http://dx.doi.org/10.14722/bar.2022.23008.

[19] S. Gorbunov and A. Rosenbloom, “Autofuzz: Automated network protocol fuzzing framework,” *Ijcsns*, vol. 10, no. 8, p. 239, 2010.

[20] R. Helmke, E. Winter, and M. Rademacher, “Epf: An evolutionary, protocol-aware, and coverage-guided network fuzzing framework,” in *2021 18th International Conference on Privacy, Security and Trust (PST)*, 2021, pp. 1–7. DOI: 10.1109/PST52912.2021.9647801.

[21] Y. Hsu, G. Shu, and D. Lee, “A model-based approach to security flaw detection of network protocol implementations,” in *2008 IEEE International Conference on Network Protocols*, 2008, pp. 114–123. DOI: 10.1109/ICNP.2008.4697030.

[22] Z. Hu, J. Shi, Y. Huang, J. Xiong, and X. Bu, “Ganfuzz: A gan-based industrial network protocol fuzzing framework,” in *Proceedings of the 15th ACM International Conference on Computing Frontiers*, ser. CF ’18, Ischia, Italy: Association for Computing Machinery, 2018, pp. 138–145, ISBN: 9781450357616. DOI: 10.1145/3203217.3203241. [Online]. Available: https://doi.org/10.1145/3203217.3203241.

[23] S. Jero, E. Hoque, D. Choffnes, A. Mislove, and C. Nita-Rotaru, “Automated attack discovery in tcp congestion control using a model-guided approach,” in *Proceedings of the 2018 Applied Networking Research Workshop*, ser. ANRW ’18, Montreal, QC, Canada: Association for Computing Machinery, 2018, p. 95, ISBN: 9781450355858. DOI: 10 . 1145 / 3232755 . 3232769. [Online]. Available: https://doi.org/10.1145/3232755.3232769.

[24] A. G. Voyiatzis, K. Katsigiannis, and S. Koubias, “A modbus/tcp fuzzer for testing internetworked industrial systems,” in *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, 2015, pp. 1–6. DOI: 10.1109/ETFA.2015.7301400.

[25] K. Katsigiannis and D. Serpanos, “Mtf-storm: A high performance fuzzer for modbus/tcp,” in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 926–931. DOI: 10.1109/ETFA.2018.8502600.

[26] X. Wei, Z. Yan, and X. Liang, “A survey on fuzz testing technologies for industrial control protocols,” *Journal of Network and Computer Applications*, vol. 232, p. 104 020, 2024, ISSN: 1084-8045. DOI: https : / / doi . org/10.1016/j.jnca.2024.104020. [Online]. Available: https : / / www . sciencedirect . com / science / article / pii/S1084804524001978.

[27] W. Wang, Z. Chen, Z. Zheng, H. Wang, and J. Luo, “Mta fuzzer: A low-repetition rate modbus tcp fuzzing method based on transformer and mutation target adaptation,” *Computers & Security*, vol. 144, p. 103 973, 2024, ISSN: 0167-4048. DOI: https : / / doi . org / 10 . 1016/j.cose.2024.103973. [Online]. Available: https : / / www . sciencedirect . com / science / article / pii / S0167404824002785.

[28] Q. Xiong, H. Liu, Y. Xu, *et al.*, “A vulnerability detecting method for modbus-tcp based on smart fuzzing mechanism,” in *2015 IEEE International Conference on Electro/Information Technology (EIT)*, 2015, pp. 404–409. DOI: 10.1109/EIT.2015.7293376.

[29] Y. Lai, H. Gao, and J. Liu, “Vulnerability mining method for the modbus tcp using an anti-sample fuzzer,” *Sensors*, vol. 20, no. 7, 2020, ISSN: 1424-8220. DOI: 10.3390/s20072040. [Online]. Available: https://www.mdpi.com/1424-8220/20/7/2040.

[30] F. Tacliad, “ENIP Fuzz: A scapy-based ethernet/ip fuzzer for security testing,” Ph.D. dissertation, Monterey, California: Naval Postgraduate School, https://hdl.handle.net/10945/56714, Sep. 2016.

[31] P. Zhao, S. Jiang, S. Liu, and L. Liu, “Fuzz testing of protocols based on protocol process state machines,” in *2024 4th International Conference on Electronic Information Engineering and Computer Science (EIECS)*, 2024, pp. 844–850. DOI: `10.1109/EIECS63941.2024.10800590`.

[32] X. Zhang, C. Zhang, X. Li, *et al.*, “A survey of protocol fuzzing,” *ACM Comput. Surv.*, vol. 57, no. 2, Oct. 2024, ISSN: 0360-0300. DOI: `10.1145/3696788`. [Online]. Available: `https://doi.org/10.1145/3696788`.