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

December 3, 2024

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 **2**
- 1.1 Stateful Fuzzing . . . 2
- 1.2 Zephyr . . . 2
- 1.3 Contributions of this Paper . . . 2

2 Related Works **2**
- 2.1 Read . . . 2
- 2.2 ToDo . . . 3

3 Approach and Implementation **3**
- 3.1 NativeSim . . . 3
- 3.2 Layer 1 Mocking . . . 3
- 3.3 Feedback . . . 3
  - 3.3.1 Coverage . . . 3
  - 3.3.2 State . . . 3
- 3.4 LibAFL . . . 3
  - 3.4.1 Snapshotting . . . 3
  - 3.4.2 Overcommit . . . 3
  - 3.4.3 MappingMutators . . . 3

4 Evaluation and Results **3**
- 4.1 Consistency . . . 3
- 4.2 Performance . . . 3
  - 4.2.1 Overcommit . . . 3
- 4.3 Mutation Strategies . . . 3
  - 4.3.1 Naïve `havoc_mutations` . . . 3
  - 4.3.2 Fixed Header Checksums . . . 3
- 4.4 Feedback . . . 3
  - 4.4.1 Coverage . . . 3
- 4.5 States Visited . . . 3
  - 4.5.1 States Counted . . . 3
  - 4.5.2 State Combinations . . . 3

5 Discussion **3**
- 5.1 Contributions . . . 3
- 5.2 Future Work . . . 3

Bibliography **4**

1 Introduction

1.1 Stateful Fuzzing

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].

1.2 Zephyr

1.3 Contributions of this Paper

2 Related Works

2.1 Read

- *Stateful Greybox Fuzzing* [3]: "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." [3] Built on LibFuzzer

- *StateAFL: Greybox fuzzing for stateful network servers* [4]: 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* [5]: 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* [6]: Ijon [5], 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* [7]: 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* [8]: take combination of executed branches into consideration, reduce to manageable adaptive fitness function

- *FuzzFactory: domain-specific fuzzing with waypoints* [9]: framework to add custom feedbacks like number of basic blocks executed, amount of memory allocated, etc.

- *ParmeSan: Sanitizer-guided Greybox Fuzzing* [10]: Use sanitizers checks as fuzzing targets

- *Fuzzing with Data Dependency Information* [11]: Use execution of new data dependencies as feedback

- *StateFuzz: System Call-Based State-Aware Linux Driver Fuzzing* [12]: 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* [13]: 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 multi-party (≥ 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.
- *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.

2.2 ToDo

- *AFLNET: A Greybox Fuzzer for Network Protocols* [14]: FTP and RTSP
- *Autofuzz: Automated network protocol fuzzing framework* [15]
- *EPF: An Evolutionary, Protocol-Aware, and Coverage-Guided Network Fuzzing Framework* [16]
- *A model-based approach to security flaw detection of network protocol implementations* [17]
- *GANFuzz: a GAN-based industrial network protocol fuzzing framework* [18]
- *TCP-Fuzz: Detecting Memory and Semantic Bugs in TCP Stacks with Fuzzing* [19]
- *FitM: Binary-Only Coverage-Guided Fuzzing for Stateful Network Protocols* [20]

3 Approach and Implementation

3.1 NativeSim

3.2 Layer 1 Mocking

3.3 Feedback

3.3.1 Coverage

3.3.2 State

3.4 LibAFL

3.4.1 Snapshotting

3.4.2 Overcommit

3.4.3 MappingMutators

4 Evaluation and Results

4.1 Consistency

4.2 Performance

4.2.1 Overcommit

4.3 Mutation Strategies

4.3.1 Naïve `havoc_mutations`

4.3.2 Fixed Header Checksums

4.4 Feedback

4.4.1 Coverage

4.5 States Visited

4.5.1 States Counted

4.5.2 State Combinations

5 Discussion

5.1 Contributions

5.2 Future Work

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 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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.