Uses of Active and Passive Learning in Stateful Fuzzing

Cristian Daniele
cristian.daniele@ru.nl
Radboud University
Nijmegen, The Netherlands

Seyed Behnam Andarzian
seyedbehnam.andarzian@ru.nl
Radboud University
Nijmegen, The Netherlands

Erik Poll
erikpoll@cs.ru.nl
Radboud University
Nijmegen, The Netherlands

ABSTRACT

This paper explores the use of active and passive learning, i.e. active and passive techniques to infer state machine models of systems, for fuzzing. Fuzzing has become a very popular and successful technique to improve the robustness of software over the past decade, but stateful systems are still difficult to fuzz. Passive and active techniques can help in a variety of ways: to compare and benchmark different fuzzers, to discover differences between various implementations of the same protocol, and to improve fuzzers.

1 INTRODUCTION

Fuzzing (or fuzz testing) is a testing technique used in the late eighties to find vulnerabilities in UNIX utilities by sending malformed messages. Despite the technique being known for more than 30 years, only recently the software security community showed interest in stateful fuzzers, i.e. fuzzers specifically tailored to stateful systems.

Research into methods to infer state models refers back to the 1980s as well, notably with the L* algorithm [2] as a black-box technique to infer the state model of a system by so-called active learning, i.e. interacting with the system and observing its responses.

Passive learning is another approach to infer state models of systems. It does not require interaction with the system but just needs a set of collected traces [13].

These three techniques (stateful fuzzing, active learning and passive learning) can be combined in various ways to improve the security testing of stateful systems. Having a state model of the System Under Test (SUT) can be an important advantage for stateful fuzzing, and both active and passive learning can be used to supply such a model (see Section 2.1). Moreover, active and passive learning can be used to benchmark stateful fuzzers (see Section 2.2) or for differential testing (see Section 2.3).

1.1 Fuzzing and Stateful Fuzzing

Fuzzing is a testing technique used to find vulnerabilities in software [7, 15]. In the last decade, especially since the advent of AFL [14], the use of fuzzers have become very successful, revealing many security flaws.

To fuzz a SUT we feed it randomly generated, often malformed inputs to check if these trigger bugs. Stateful fuzzing – i.e. fuzzing a stateful system – is more challenging: instead of just sending a message we may need to send a sequence of messages (what we will call a trace) to get the SUT in the right state where a bug can be triggered [4]. Knowing the state model of the SUT can help as we can make sure we visit all the states, effectively fuzzing the SUT in each state.

1.2 Active Learning

Active learning infers the state model of a SUT by interacting with it [11]. The idea is that we gradually improves our understanding of the model by trying out sequences of inputs and checking if the observed output correspond with the expected behaviour, and if not, refining the state machine model for the behaviour.

Active learning can be very accurate but also very slow, as it involves exhaustively trying our all the possible traces up to a given length. The state space can explode because of the number of possible messages and the size of the state models. It can be regarded as a limited form of fuzzing [5], where we only mutate the order of a fixed set of messages but not these messages themselves. But a core difference between active learning and stateful fuzzing is that the former aims to infer the state model of the SUT, while the latter aims to trigger bugs.

Active learning has been proven extremely effective and broadly applicable thanks to its black-box nature.

1.3 Passive Learning

Passive learners are quite different from the stateful fuzzers and the active learner tools. While fuzzers and active learning tools *actively* send messages to the SUT, passive learners take a set of traces that have been collected beforehand as input to then infer a state machine [13]. During this process, also known as *grammatical inference*, there is no further interaction with the SUT. The set of traces used could be produced by using a stateful fuzzer.

Passive learning used in combination with a stateful fuzzer can gives a good approximation of the state model much faster than active learning. It could even produce a more detailed model than an active learner tool since the traces given to the passive learner may also contain malformed messages (and not just malformed traces of correctly formed messages). However, that depends heavily on the effectiveness of the (stateful) fuzzer.

2 COMBINATIONS

Stateful fuzzing, active learning and passive learning can be combined to achieve different goals.

2.1 Improving the effectiveness of stateful fuzzers

Some stateful fuzzers already use passive or active learning ([1, 3, 9]), but most of them do not. Since active and passive learning techniques work in a black-box fashion, they can be easily added to existing stateful fuzzers in order to improve their state-awareness.

2.2 Benchmarking of stateful fuzzers

Benchmarking fuzzers is challenging and benchmarking stateful fuzzers is even more so [6]. Knowing the state model can help here

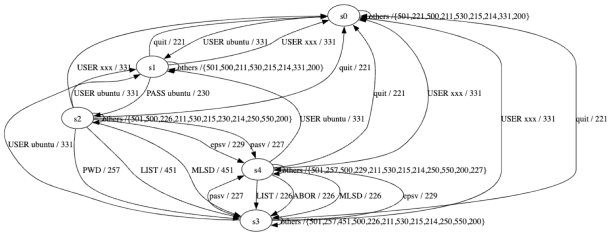

Figure 1: State model of LightFTP inferred by the active learning tool LearnLib[10]

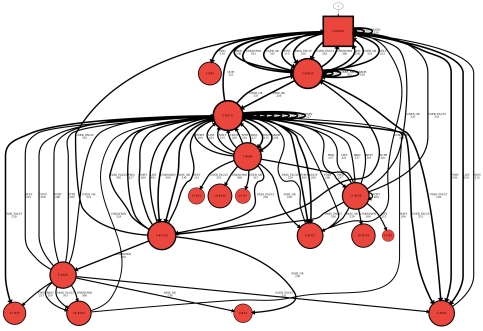

Figure 2: State model of LightFTP inferred by AFLNet and the passive learning tool FlexFringe[12]

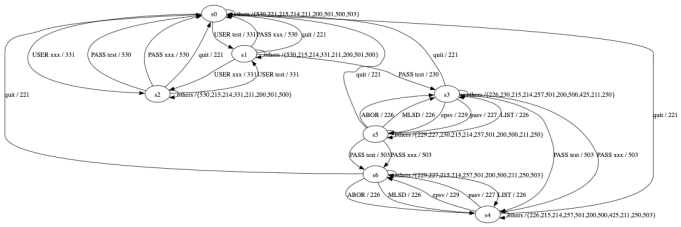

Figure 3: State model for ProFTPd inferred by LearnLib

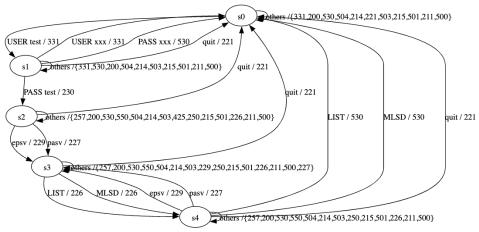

Figure 4: State model for PureFTPd inferred by LearnLib

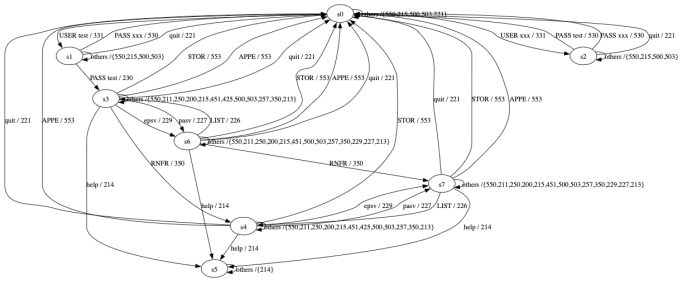

Figure 5: State model for bftpd inferred by LearnLib

as it allows us to count how many states are visited during the fuzzing and how heavily each state has been fuzzed. For example, if *fuzzer A* reaches 3 of the 8 states and *fuzzer B* 7 of the 8 states we can argue that *fuzzer B* is more effective than *fuzzer A* in exploring the state model. Both active and passive learning can help in providing the state model than we can then use to measure the coverage.

The state models inferred by active and passive learning tools can be different (e.g. see Fig. 1 and 2). As already mentioned in Section 1.3, the set of traces given as input to a passive learner may not only involve mutations in the order of the messages (which active learning would also explore) but may also involve mutations of individual messages (which active learners typically do not explore, but which most fuzzers will). Differences between state models inferred by active and passive learning may point to flaws in the SUT, more specifically in the program logic when it comes to handling malformed messages.

2.3 Differential testing

Comparing the state models of different implementations of the same protocol can also be valuable [8]. As shown in Fig. 1, 3, 4 and 5, different implementations of FTP have different state models. Such differences between these state models may point to flaws in the program logic or ambiguities in the specification.

3 CONCLUSIONS

This paper sheds light on the differences and similarities between active learning, passive learning and stateful fuzzing and ways these could be combined. In particular, we suggest how stateful fuzzers might benefit from models obtained by active or passive learning to improve their effectiveness, allowing benchmarking and do differential fuzzing.

ACKNOWLEDGMENTS

This research is funded by NWO as part of the INTERSECT project (NWA.1160.18.301).

REFERENCES

[1] Bernhard K Aichernig, Edi Muškardin, and Andrea Pferscher. 2021. Learning-based fuzzing of IoT message brokers. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 47–58.

[2] Dana Angluin. 1987. Learning regular sets from queries and counterexamples. *Information and computation* 75, 2 (1987), 87–106.

[3] Osbert Bastani, Rahul Sharma, Alex Aiken, and Percy Liang. 2017. Synthesizing program input grammars. *ACM SIGPLAN Notices* 52, 6 (2017), 95–110.

[4] Cristian Daniele, Seyed Behnam Andarzian, and Erik Poll. 2024. Fuzzers for stateful systems: Survey and Research Directions. *Comput. Surveys* 56, 9 (2024), 1–23.

[5] Joeri De Ruiter and Erik Poll. 2015. Protocol state fuzzing of TLS implementations. In *24th USENIX Security Symposium (USENIX Security 15)*. 193–206.

[6] Ahmad Hazimeh, Adrian Herrera, and Mathias Payer. 2020. Magma: A ground-truth fuzzing benchmark. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 3 (2020), 1–29.

[7] Jun Li, Bodong Zhao, and Chao Zhang. 2018. Fuzzing: a survey. *Cybersecurity* 1 (2018), 1–13.

[8] William M McKeeman. 1998. Differential testing for software. *Digital Technical Journal* 10, 1 (1998), 100–107.

[9] Van-Thuan Pham, Marcel Böhme, and Abhik Roychoudhury. 2020. AFLNET: a greybox fuzzer for network protocols. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 460–465.

[10] Harald Raffelt, Bernhard Steffen, and Therese Berg. 2005. Learnlib: A library for automata learning and experimentation. In *Proceedings of the 10th international workshop on Formal methods for industrial critical systems*. 62–71.

[11] Alaa Tharwat and Wolfram Schenck. 2023. A survey on active learning: state-of-the-art, practical challenges and research directions. *Mathematics* 11, 4 (2023), 820.

[12] Sicco Verwer and Christian A. Hammerschmidt. 2017. Flexfringe: a passive automaton learning package. In *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 638–642.

[13] Nan Yang, Kousar Aslam, Ramon Schiffelers, Leonard Lensink, Dennis Hendriks, Loek Cleophas, and Alexander Serebrenik. 2019. Improving model inference in industry by combining active and passive learning. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 253–263.

[14] Michal Zalewski. 2014. American Fuzzy Lop (afl). https://lcamtuf.coredump.cx/afl

[15] Xiaogang Zhu, Sheng Wen, Seyit Camtepe, and Yang Xiang. 2022. Fuzzing: a survey for roadmap. *ACM Computing Surveys (CSUR)* 54, 11s (2022), 1–36.