

Improving the Software Architecture of Active Automata Learning Tools

Olivier Levillain

Context

Network protocols are pervasive in today's infrastructure. However, they are often described in a vague manner (specifications are written in natural language, and lack formal descriptions). This leads implementers to handle ambiguities (sometimes implicitly) and can cause security flaws such as authentication bypass.

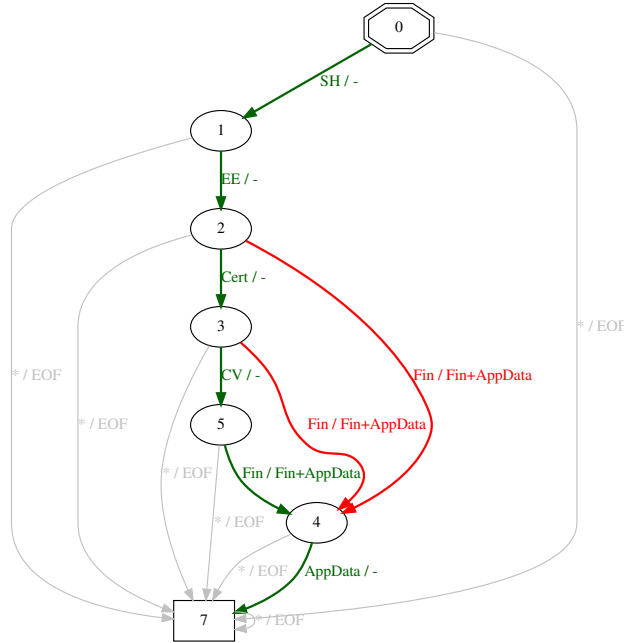


Figure 1: CVE-2020-24613

To improve the situation, one method is to study the internal state machines of the network stacks. We have been working on this front with Active Automata Learning (AAL) tools for several years, with applications to TLS [0], OPC-UA [1] and SSH [2].

One of the limitations of AAL is its slow speed. This performance issue partly comes from the combinatory explosion needed to confirm the state machine found by the algorithm, which is inherent to the approach.

However, for each interaction with the target (also called the SUL, for System Under Learning), we waste time collecting the feedback from the network stack, especially because of the delay we have

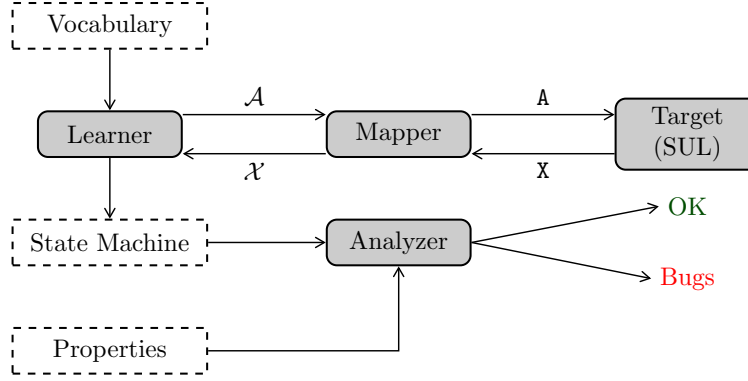


Figure 2: Active Automata Learning Classical Architecture

to wait to make sure we are not missing messages from the target (this corresponds to a timeout parameter).

To overcome this limitation, a previous project injected a library to capture the syscalls related to network operations to detect earlier that the system under learning is actually done talking.

Proposal

This project aims at improving the current prototype, which works with two inference tools (**pylstar** and **rlstar**) and two targets (OpenSSL and wolfSSH). One of the aspect is to reuse an existing library, Netfuzzlib [4], to rewrite the preloaded library from the previous project in a more generic manner.

To improve the situation, it would also be interesting to write some tools to analyze the syscall patterns used by a given TLS or SSH stack, using tools like **strace**. We envision the possibility to have a generic approach which is independant of the actual protocol implemented. One known difficulty to properly capture syscalls with a preloaded library, which will not be solved during the project is handling multi-process stacks (e.g. OpenSSH), but merely analyzing the syscalls might help verify the feasibility of the capture in such complex cases.

We imagine different steps for the beginning of this project:

- Read the Netfuzzlib paper [3] and familiarize with the instrumentation idea [4];
- Read the documentation and the code from the previous project (which was the work of Van Nam Pham);
- Rewrite the preloaded library with Netfuzzlib if this makes sense;

If time permits, two other directions could be pursued, as exploratory ideas:

- study libAFL [5, 6] and design a way to merge syscall capture and fuzzing in simple cases, to provide proper network stack fuzzing (this could be later combined with the actual knowledge of a state machine to provide stateful fuzzing);
- think about the issues we have when we switch from TCP-based protocols to UDP-based protocols. Indeed, UDP does not really offer any form of signaling; in particular, it is not always easy to know, from the syscalls, that a pseudo-connection (as in DTLS or QUIC) is actually closed.

Prerequisite

- Development Skills in C and in Rust
- Understanding of the system calls related to network communications
- Knowledge of several network protocols (optional, but useful)

Administrativa

Send an email to olivier.levillain@telecom-sudparis.eu if you are interested in this project.

Biblio

- [0] Aina Toky Rasoamanana, Olivier Levillain and Hervé Debar. *Towards a Systematic and Automatic Use of State Machine Inference to Uncover Security Flaws and Fingerprint TLS Stacks*. ESORICS 2022. <http://paperstreet.picty.org/yeye/2022/conf-esorics-RasoamananaLD22/>
- [1] Arthur Tran Van, Olivier Levillain and Hervé Debar. *Mealy Verifier: An Automated, Exhaustive, and Explainable Methodology for Analyzing State Machines in Protocol Implementations*. ARES 2024. <http://paperstreet.picty.org/yeye/2024/conf-ares-TranVanLD24/>
- [2] Aina Toky Rasoamanana. *Derivation and Analysis of Cryptographic Protocol Implementation*. PhD Thesis. École Doctorale IP Paris. <https://theses.fr/2023IPPAS005>
- [3] Jeroen Robben, Mathy Vanhoef, *Netfuzzlib: Adding First-Class Fuzzing Support to Network Protocol Implementations*. ESORICS 2024. <https://papers.mathyvanhoef.com/esorics2024.pdf>
- [4] Netfuzzlib GitHub Repository. <https://github.com/JeroenRobben/netfuzzlib>
- [5] Andrea Fioraldi, Dominik Christian Maier, Dongjia Zhang, Davide Balzarotti. *LibAFL: A Framework to Build Modular and Reusable Fuzzers*. ACM CCS 2022. https://www.s3.eurecom.fr/docs/ccs22_fioraldi.pdf
- [6] LibAFL GitHub Repository. <https://github.com/AFLplusplus/LibAFL>