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.

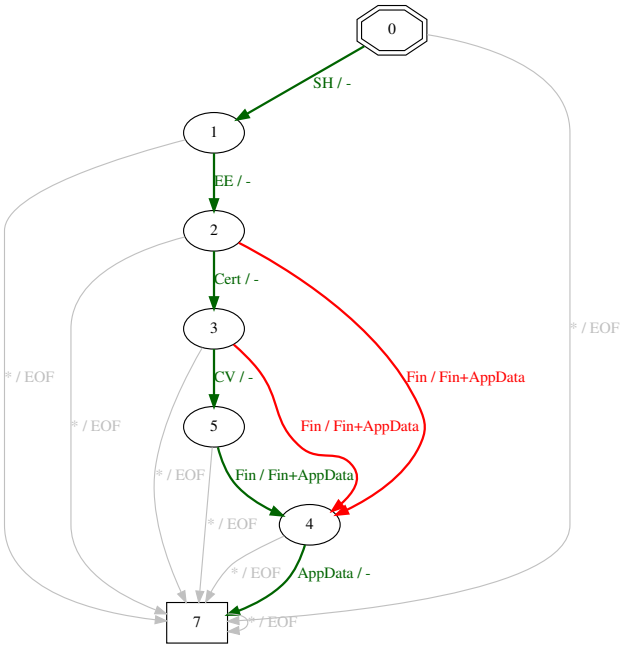

Figure 1: CVE-2020-24613

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].

Proposal

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.

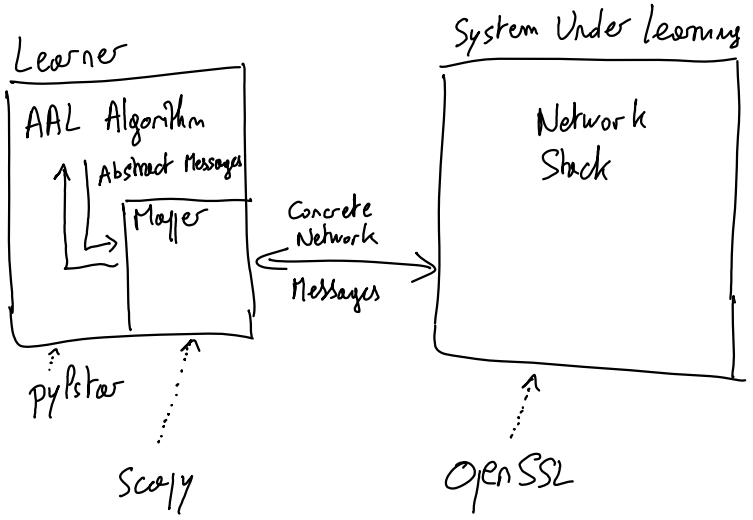

Figure 2: Active Automata Learning Classical Architecture

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, which can be classified in three categories:

- the delay we have to wait to make sure we are not missing messages from the target (this corresponds to a timeout parameter);
- the network communication (and the corresponding context switches which are required by the system;
- the initialiaztion part of the target (the code spent before the actual interaction).

To overcome this latter limitation, we envision a change in software architecture, inspired by similar techniques used in stateful fuzzing [3]. The approach consists in capturing network operations before syscalls are emitted, so we can skip the network communications, and the useless wait.

We imagine different steps for this project:

- Read the GreenFuzz paper [3] and familiarize with the instrumentation idea [4];
- Explore the GreenFuzz library to make it work on a simple example;
- Include a stub in the target to handle communications in a batch mode (removing the timeout);
- Embed the mapper in the target as a library (removing some network communications and context siwtches);
- Embed the whole learner in the target as a library (removing all network communications and context switches).

The different phases will probably have to be written in C in a first time, but switching to Rust at

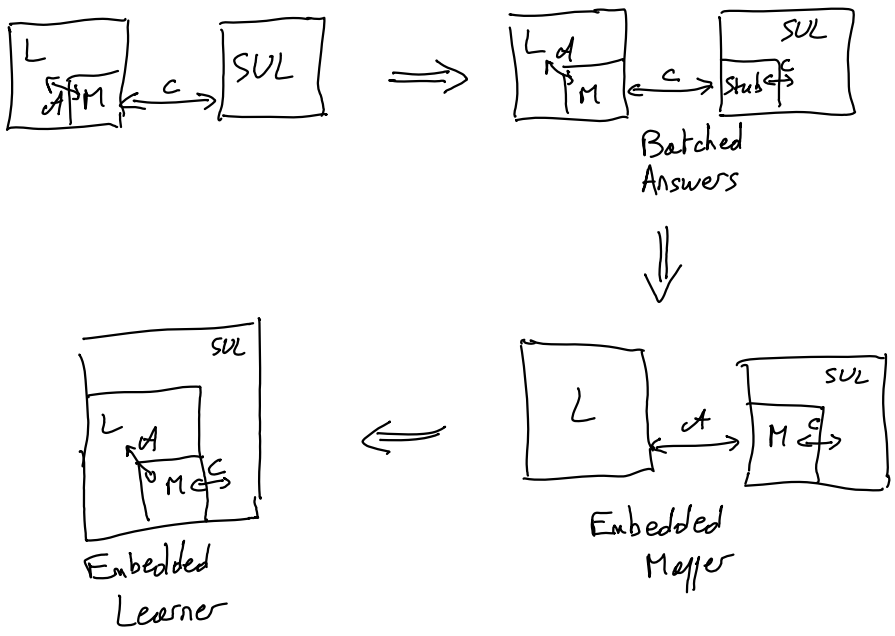

Figure 3: Active Automata Learning Classical Architecture

some point would be preferable.

Different protocols and implementations can be targeted to progress incrementally toward a fully functional architecture:

- MinIRC, a toy chat protocol used in a course;
- FTP, a simple text-based protocol
- TLS, a fully-fledged security protocol.

If time permits, we could use the new architecture to implement a fancy checkpoint-restart mechanism to skip entirely the initialization phase. This could rely on forking a process for each state reached during the learning phase.

Prerequisite

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

Administrativia

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] Seyed Benham Andarzian, Cristian Daniele, Erik Poll. *Green-Fuzz: Efficient Fuzzing for Network Protocol Implementations*. FPS 2023. https://www.cs.ru.nl/E.Poll/papers/greenfuzz.pdf

[4] GreeFuzz GitHub Repository. https://github.com/behnamandarz/GreenFuzz/