# Post-Doctoral Position on the Implementation and the Analysis of Network Protocols

Olivier Levillain

## Context

In today's connected world, we heavily rely on network protocols to communicate with one another. To ensure the confidentiality and integrity of the exchanged data, we need to assess and improve the security of these protocols and their implementation. This observation obviously applies to security protocols, such as TLS and SSH, but it is also important to consider high-level, complex, application-level protocols, such as HTTP/2 and lower-level protocols such as DNS or BGP.

Security flaws are indeed pervasive in network protocols, at the specification level or due to implementation flaws: incomplete specification, memory corruption bugs, logical errors, state machine shortcuts, cryptographic attacks, etc.

Following the work done in the GASP project[1], this post-doc position aims at improving the existing results from the project on TLS and SSH [5, 4] and extending them.


Figure 1: Description of L⋆ Active Learning Approach

The methodology, shown in Figure 1 developed during the GASP project relies on the L⋆ algorithm, an active learning approach allowing us to infer the state machine of a network protocol implementation. Concretely, L⋆ requires a Learner, which actually runs the inference algorithm, and a Mapper, which translates abstract messages generated by the Learner into concrete messages for the SUT and translates the concrete reply from the SUT into abstract messages for L⋆. Figure 2 shows an example of the vulnerabilities that can be automatically uncovered.

Here are some research questions we would like to explore.

## Performance Improvement

The main bottleneck with active learning is the time spent waiting for timeouts to make sure the system under test (SUT) is actually done sending data. We already designed heuristics to reduce the waiting time in known situations (leading to a 25-fold performance gain).

To improve the performance further and make state machine inference more affordable, e.g. to include it in continuous integration pipelines, we could use fine-grain monitoring or instrumenting the SUT.

To this aim, the Learner (i.e. the component running the L⋆ algorithm) could monitor the syscalls emitted by the SUT to detect when it can stop waiting. Beyond this simple idea, we anticipate interesting challenges such as multi-threaded servers or hidden dependencies between processes such as a SSH server executing a command[2]

[1] GASP, a Generic Approach to Secure network Protocols, is a project funded by the French ANR (ANR-19-CE39-0001).

[2] This specific case is actually problematic for our heuristics, since executing a command does not result in any direct reply, but skipping the timeout wait might nevertheless lead to unexpected results (or even inference failure) since there can be interactions between the executed command and the next message to send.

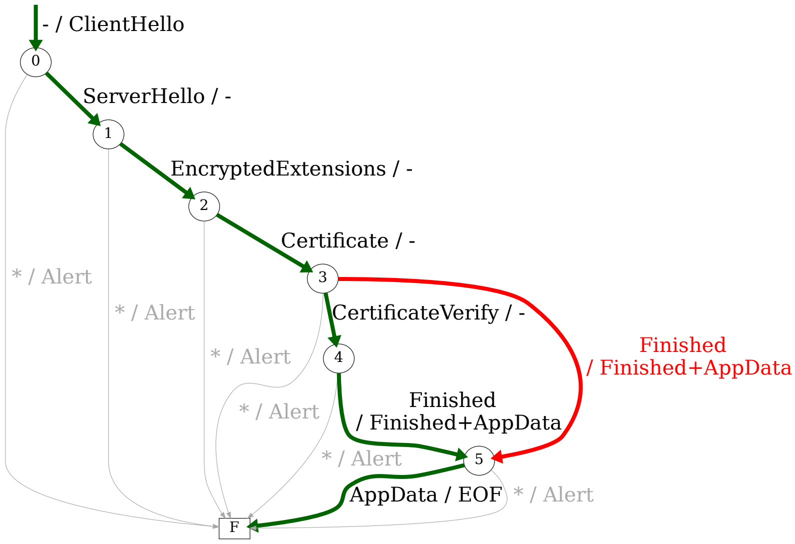


Figure 2: CVE-2020-24613, a security vulnerability affecting WolfSSL: a vulnerable client will accept a TLS connection with a server even if it does not send the `Certificate Verify` message, which authenticates the server. This authentication bypass can be automatically found using state machine inference. Each edge is labeled by `Sent/Received`, the messages sent to (resp. received by) the SUT by our tool to.

Another idea would be to link the SUT with a fake library overloading network functions such as `socket`, `recv` and `send`, to better control the actual interactions with the communication channel. This approach has already been described in the fuzzing community [2].

## Insider Insight for the State Machine States

Beyond merely inferring the state machine, instrumenting the SUT could allow us to characterize the states of the generated automaton, e.g. by identifying the invariants in the program variables specific to a given state.

Automatically extracting the state machine and the corresponding internal state of the underlying code would be very useful to understand what each state of the state machine corresponds to. This would give interesting insights to the developers trying to study a bug or a security flaw affecting their network stacks. Indeed, identifying a bug using $L^\star$ is just the start of the process leading to a verifiable fix.

The challenges to design and implement such an idea include multi-threading, randomization and other non-deterministic aspects of the implementation, as well as a possible combinatory explosion to pin-point the relevant variables to monitor.

## Stateful Fuzzing

An extension of our work would be to reuse our knowledge of the actual state machine of an implementation to improve code coverage during fuzzing sessions. Indeed, it is usually hard for fuzzers to reach the portions of the code which correspond to the execution of late steps in complex protocols.

Using our knowledge of the actual state machine, we should be able to fuzz the SUT in an arbitrary state, by first sending a message sequence to reach the expected state, and then inject the actual samples to test from this state. The idea resembles a recently developed approach called Dolev-Yao Fuzzing [1].

In addition to the previous idea, we could further improve the fuzzing process by reusing the instrumentation idea described earlier (the fake library handling network communications) and taking snapshots of the SUT in different states.

## Mapper Automatic Derivation

Developing a new mapper for each protocol we want to study can be a tedious task. To make this part of our methodology more efficient, we could write and use tools to automate (part of) the implementation of the mapper.

Several projects have been exploring the parser generation part of the process [3], which allow to alleviate the developper from most of the repetitive boilerplate tasks.

Moreover, the SSH mapper developped during the project follows a modular and flexible design, which could probably be abstracted and enriched with derivation tools, to propose a Domain-Specific Language to generate (part of) the mapper state machine.

## Practical Informations

The postdoc will be located either on the Évry or the Palaiseau campus of Télécom SudParis. It is a 12-month position.

The applicant must have a PhD thesis in cybersecurity or in a technical field relevant to the subject (network, systems).

Applications must contain a resume, a motivation letter, and a recommendation letter. They must be sent to `olivier.levillain@telecom-sudparis.eu`. If you have any questions, you can address them to the same address.

## References


[1] Max Ammann, Lucca Hirschi, and Steve Kremer. Dy fuzzing: Formal dolev-yao models meet cryptographic protocol fuzz testing. Cryptology ePrint Archive, Paper 2023/057, 2023. `https://eprint.iacr.org/2023/057`.

[2] Seyed Behnam Andarzian, Cristian Daniele, and Erik Poll. Green-Fuzz: Efficient Fuzzing for Network Protocol Implementations. In *16th International Symposium on Foundations & Practice of Security (FPS), Bordeaux, France*, December 2023.

[3] Olivier Levillain, Sébastien Naud, and Aina Toky Rasoamanana. Work-in-Progress: Towards a Platform to Compare Binary Parser Generators. In *35. IEEE Security and Privacy Workshops, SPW (LangSec) 2021, San Jose, CA, USA*, May 2021.

[4] Aina Toky Rasoamanana. *Derivation and Analysis of Cryptographic Protocol Implementation.* PhD thesis, École doctorale de l'Institut Polytechnique de Paris, June 2023.

[5] 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. In *27th European Symposium on Research in Computer Security, ESORICS 2022, Copenhagen, Denmark*, pages 637–657, September 2022.