Symbolic-Model-Guided Fuzzing of Cryptographic Protocols

City and country Nancy, France.

Team or project in the lab
Team PESTO at LORIA lab (Inria Nancy, CNRS and Université de Lorraine).

Name and email address of the advisor
Lucca Hirschi, `lucca.hirschi@inria.fr`

Name and mail of the head of the laboratory
Jean-Yves Marion, `jean-yves.marion@loria.fr`

Indemnisation
The internship is supported by the ANR Chair in AI *ASAP*.

Today's information society crucially relies on secure information exchanges achieved by *cryptographic protocols*. Those distributed programs that leverage cryptographic primitives (*e.g.,* encryption, digital signature) to achieve various security goals are critical to many aspects of our modern society: finance, business, communication, etc. Any flaw in these protocols can have dramatic consequences, amplified by their ubiquity and our dependence on them. Yet, critical and widely used protocols have been repeatedly found to be **flawed in their design or their implementation**. A prominent class of such flaws are **logical attacks**, *i.e.,* attacks that solely exploit *flawed protocol logic* such as *Man-in-the-Middle* (MiM), *replay*, or *downgrade* attacks, etc.

As such flaws are subtle and hard to catch, **formal methods** have been proposed to analyze protocol *design specifications* since the 80s. Symbolic verification is a first-class, extremely successful such method [1]. It offers a **mathematical model capturing logical attacks**, *i.e.,* the **symbolic model** also called *Dolev-Yao model*, as well as rigorous and mechanized methods to reason about protocols. However, a fundamental, inherent issue is that symbolic verification **operates on abstract specification models only**. Security proofs thereon are of no practical use when the programs that end-users deploy or run are insecure. Unfortunately, history shows that frequent implementation bugs actually introduce vulnerabilities that were nonexistent in the specification, notably ***implementation-level logical attacks***. This is particularly well illustrated by the long history of such attacks in the ubiquitous and critical TLS and WiFi protocols ([2, 3, 8, 6, 7] to only name a few).

Programmers or auditors interested in precluding logical attacks from protocol implementations are left with ***testing*** since security-oriented *program verification* is extremely expertise-demanding and does not really scale beyond primitives or minimal protocols. As opposed to formal verification, testing is unsound by design (*i.e.,* bugs may be missed) but provides a certain level of confidence by excluding all the potential flaws covered by the body of tests. Therefore, a good *coverage* of the tests is paramount. Narrowing down to security, the gold standard is ***fuzz testing*** [4, 5] due to its ability to automatically generate test cases that maximize the coverage typically thanks to *feedback-driven evolutionary algorithms* utilizing *mutations*. Today, fuzzing is paramount in the **industry software development practices**, *e.g.,* Google, Cisco, Microsoft use it at scale. The state-of-the-art fuzzing techniques are adequate to find safety vulnerabilities (sometimes with potential security implications) but are unfortunately **unable to find logical attacks** since they operate at a too low level (*e.g.,* random bit-flips on network packets, code-based coverage). Prior works have proposed fuzzers operating in some ad-hoc *state machine model* [3, 2] that is also **too weak to capture the class of logical attacks**; *e.g.,* message contents cannot be tampered with by the adversary while most logical attacks rely on this.

Symbolic verification captures logical attacks at the design level only while fuzzing is industry-ready and operates on implementations but is limited to low-level, safety-oriented flaws, which are often the low-hanging fruits. Therefore, **effective and usable techniques to preclude logical attacks on implementations** are desperately lacking.

Objectives. This internship objective is to develop a **symbolic-model-guided fuzzing** framework. that will enable **checking implementations for the absence of logical attacks** with TLS as a case study. The central idea is to consider symbolic traces (from a symbolic model) as the input space of the Program Under Test (PUT) that will be fuzzed and then executed on the PUT through *concretization* (symbolic terms are evaluated into bitstrings). Fortunately, we already have a preliminary, proof-of-concept design and implementation in Rust for TLS 1.2 and 1.3 that will serve for this internship as a solid basis and test-bed for exploring new directions.

Intern's tasks. First of all, the intern will get familiar with formal verification in the symbolic model, fuzzing, as well as with the existing Rust code base.

Next, depending on the intern's affinity for and knowledge of the different involved aspects of this project, we will be able to adapt the project goals and choose one among several research directions, such as:

- design domain-specific feedback metric to incentives the fuzzer to seek for new symbolic traces. The underlying fundamental question is: what is a good "symbolic feedback" that promotes semantically different symbolic traces?
- design new fuzzing mutations and benchmark them with our test-bed,
- design an efficient grammar-based fuzzing engine and evaluate it with our test-bed,
- define scoring metrics that can be effectively computed by symbolic verifiers and that can help the fuzzer promoting test cases that are close to attack traces.

There also more practical and yet interesting problems to work on: balancing the different fuzzing strategies with appropriate benchmarks and profiling, isolate TLS components that are worth fuzzing and expose them to the fuzzing harness, run long-term fuzzing campaigns on the Grid 5K facilities, etc. Should we find any vulnerability, we would follow standard and ethical responsible disclosure practices.

The precise direction this project will take shall be agreed upon with the intern at the beginning of the project.

Expected ability of the student. We expect mathematical maturity, basic knowledge in logic, basic theoretical computer science. Knowledge in security and cryptography is not mandatory but is definitely a plus. For the implementation, a good command of Rust is necessary.

If the candidate is interested, continuation towards a PhD, for which we already have funding. on related topics is possible.

References

[1] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. SoK: Computer-Aided Cryptography. In *Symposium on Security and Privacy (SP)*. IEEE, 2021.

[2] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A Messy State of the Union: Taming the Composite State Machines of TLS. In *Symposium on Security and Privacy (SP)*, pages 535–552. IEEE, May 2015.

[3] Joeri De Ruiter and Erik Poll. Protocol State Fuzzing of TLS Implementations. In *USENIX Security*, pages 193–206, 2015.

[4] Patrice Godefroid. Fuzzing: Hack, art, and science. *Communications of the ACM*, 63(2):70–76, January 2020.

[5] Valentin Jean Marie Manes, HyungSeok Han, Choongwoo Han, Sang Kil Cha, Manuel Egele, Edward J. Schwartz, and Maverick Woo. The Art, Science, and Engineering of Fuzzing: A Survey. *IEEE Transactions on Software Engineering (TSE)*, pages 1–1, 2019.

[6] Mathy Vanhoef. Fragment and forge: Breaking Wi-Fi through frame aggregation and fragmentation. In *Proceedings of the 30th USENIX Security Symposium*. USENIX Association, August 2021.

[7] Mathy Vanhoef and Frank Piessens. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. In *Conference on Computer and Communications Security (CCS)*, pages 1313–1328. ACM, October 2017.

[8] Mathy Vanhoef and Eyal Ronen. Dragonblood: Analyzing the Dragonfly handshake of WPA3 and EAP-pwd. In *IEEE Symposium on Security & Privacy (SP)*. IEEE, 2020.