M.Sc. thesis project(s):
Testing of Lightweight Security Protocols

**Background** Within the project aSSIsT (https://assist-project.github.io/), we are developing techniques for ensuring that implementations of security protocols (as well as other security-critical software) conform to their specifications, and are free from security vulnerability. In particular, we care about software for security of IoT devices. A motivation is that the number of Internet of Things (IoT) devices is projected to reach 11.6 billion by 2021, thus constituting half of all devices connected to the Internet.

A key building block in IoT software consist of implementations of security protocols. As an example, let us consider Datagram Transport Layer Security (DTLS) [3, 4], which is one of the primary protocols for securing IoT applications [5]. DTLS is based on TLS, a widely used security protocol responsible for securing communication over a reliable data transfer protocol. Unlike TLS, DTLS is meant to operate over UDP, an unreliable transfer protocol commonly used in IoT systems, but also increasingly being used in Voice over IP, tunneling technologies, and new Web protocols. Whereas significant effort has been invested into ensuring security of TLS implementations, those based on DTLS have so far received considerably less scrutiny.

Within aSSIsT and our research group, we have developed and are further developing techniques to ensure that implementations of DTLS and other security protocols conform to their specifications, and are free from security protocols. Two major techniques that we consider are state fuzzing and symbolic execution.

**State fuzzing** Implementations of network protocols must conform to their specifications in order to avoid security vulnerabilities and interoperability issues. Even seemingly innocent deviations from the standard specification may expose implementations to security attacks. Protocols that establish secure connections (e.g., SSH, TLS, DTLS, QUIC, etc.) must carefully manage the type and order of exchanged messages and cryptographic material, by maintaining a state machine which keeps track of how far the protocol has progressed. Any deviation from the order prescribed in the protocol’s specification may constitute anything between an inconsequential error to a serious vulnerability. Corresponding implementation flaws, often called state machine bugs, may be exploitable, e.g., to bypass authentication steps or establish insecure connections.

A technique for detecting state machine bugs is state fuzzing: It automatically infers state machine descriptions of protocol implementations using model learning. Model learning is an automated black-box technique which produces state machine models describing how an implementation handles message flows, by observing how the implementation responds to sequences of test inputs. The learned model is then analyzed to spot state machine bugs that result from flaws in the implementation’s control logic. We have applied state fuzzing to most existing implementations of DTLS, discovering numerous vulnerabilities and nonconformances [2]. For instance, we found a major vulnerability in the most widely used Java implementation of DTLS, whereby client authentication could be bypassed, which had to be fixed by the developing organization.

**Symbolic execution** Symbolic execution analyzes programs for which (some of) the input variables are designated as symbolic. It explores the code paths that are possible for some values of the symbolic input, and also provides input values that make executions follow specific code paths. Symbolic execution has been used to find errors and vulnerabilities
in a range of different types of software. In our work, we use it to test whether implementations of security protocols conform to their specifications. More specifically, we extract requirements concerning how a protocol implementation should process incoming packets and send outgoing packets. Requirements can concern any data field in such packets, e.g., concerning sequence numbers, length fields, offsets, etc. Implementation errors concerning such fields are not uncommon and can sometimes expose vulnerabilities: a famous example is the so-called heartbleed bug in TLS. We have applied this technique to the testing of DTLS (publication under submission).

Suggested M.Sc. Project(s) We plan to apply our techniques also to other lightweight security protocols. One protocol under consideration is EDHOC, which is designed to be more lightweight than DTLS and more suitable for resource-constrained environments. EDHOC is in the final stages of standardization [1], and infrastructure for testing EDHOC will be of high interest as new implementations are under development. EDHOC reuses several concepts and techniques from TLS and DTLS. An M.Sc. project would therefore investigate how to retarget the efforts we have performed for DTLS, concerning either state fuzzing or symbolic execution, and apply it to (a fragment of) EDHOC.

Application Send your application for M.Sc. project to Bengt Jonsson (bengt@it.uu.se)

Supervisors and Contact persons Bengt Jonsson (bengt@it.uu.se), Kostis Sagonas (kostis@it.uu.se), Paul Fiterau-Brostean (paul.fiterau_brostean@it.uu.se).

References


