Tutorial: Continuous Verification of Critical Software

Mike Dodds
Galois Inc.
miked@galois.com

Stephen Magill
Galois Inc.
stephen@galois.com

Aaron Tomb
Galois Inc.
atomb@galois.com

***Abstract*—This tutorial will describe how to integrate formal verification of cryptographic code into real-world software development. We will base the tutorial on our work with Amazon verifying critical portions of s2n, the open-source TLS implementation used in many Amazon services. This work shows it is now practical to integrate verification of critical software into developer workflows. Our aim with this tutorial is to enable others to apply this approach to their security-critical projects. For this reason, we will focus on the pragmatic aspects of integrating and maintaining a continuous verification system.**

I. Topic

This tutorial will describe how to integrate formal verification of cryptographic code into real-world software development. We will base the tutorial on our work with Amazon to verify critical portions of s2n, the open source TLS implementation used in numerous Amazon services. Our proofs guarantee that critical cryptographic algorithms in s2n are correctly implemented. Furthermore, at each change to the code, proofs are automatically checked and guarantees re-established with little to no interaction from developers. The result is continuous verification, ensuring that code remains correct throughout the lifetime of the software.

The verification approach we will describe makes use of an open source tool called the Software Analysis Workbench (SAW, available at https://saw.galois.com). This tool allows translation from concrete software implementations into mathematical models of software behavior, followed by the use of automated theorem provers to show that the extracted behavior satisfies user-provided properties for all possible inputs. The result is equivalent to performing exhaustive testing of the same properties, but whereas exhaustive testing for algorithms over large data spaces is intractable, the behavioral correctness proofs often complete in less than a few minutes.

More specifically, we have verified the s2n implementation of HMAC, DRBG, and the TLS handshake. For each of these, SAW proves the C implementation is memory safe and matches the specification given in the RFC standard. This rules out several important categories of attack – for one example, attacks against the TLS handshake state machine.

Our work with Amazon shows that verification of core pieces of commercial software is now practical and can be integrated into developer workflows. Our aim with this tutorial is to enable others to apply this approach to their security-critical projects. For this reason, we will focus on the pragmatic aspects of implementing and maintaining a continuous verification system.

II. Tutorial Format

Our tutorial will be broken into two parts:

- *Verifying cryptography using SAW* [90mins]
  - We will sketch the basics of formal verification: specification, logic, reasoning about programs.
  - We will introduce the types of algorithms SAW can verify, and describe how to set up scripts to orchestrate the verification of cryptographic code.
  - We will show examples of cryptographic algorithms as specified in RFCs and NIST documents, and compare them with more formal descriptions in our specification language, Cryptol.
  - We will include group and individual exercises based on modifying these example algorithms.
- *Building robust continuous verification systems* [90min]
  - We will describe the process of building robust, continually-checked proofs.
  - Using s2n examples, we will show how code changes can be automatically accommodated and point out which code changes require proof script updates.
  - We will show how to set this process up in continuous integration systems, using Travis CI as a concrete example.
  - We will include group and individual exercises that involve running SAW proofs on a test CI environment and updating proofs in response to code changes.

III. Target Audience

The audience for this tutorial is developers working on security-critical and/or cryptographic code. We suggest the following prerequisite knowledge:

- Experience in C / C++ / Java, or similar [*necessary*]
- Knowledge of security-focused engineering [*necessary*]
- Knowledge of cryptographic primitives [*useful*]
- Knowledge of mathematical logic [*useful*]
- Experience with continuous integration tools [*useful*]

IV. Learning outcomes

- An understanding of what types of software can be automatically verified.
- Experience using the basic features of SAW to perform verification.
- Knowledge of how to organize proofs to be robust to code changes.
- Knowledge of how to integrate SAW with Travis and other CI environments.

V. Related Material

- *From Testing to Proof Using Symbolic Execution*, Aaron Tomb, Galois Inc. Tutorial at StrangeLoop 2017.
  https://www.thestrangeloop.com/2017/from-testing-to-proof-using-symbolic-execution.html
- *Assuring Crypto Code with Automated Reasoning*, Aaron Tomb, Galois Inc. Talk at QCon 2017.
  https://www.infoq.com/presentations/cryptography-reasoning
- *Verifying s2n HMAC with SAW*, Joey Dodds, Galois Inc. Blog post, 2016.
  https://galois.com/blog/2016/09/verifying-s2n-hmac-with-saw/