

Verifying Implementations of Security Protocols in C

Mihhail Aizatulin¹, **François Dupressoir**¹
Andrew Gordon², Jan Jürjens^{1,2}

¹The Open University

²Microsoft Research Cambridge

Journée des 4ème années
ENS de Cachan – Antenne de Ker Lann
January 23, 2009

The problem with security

- ▶ Lots of work exist on proving protocols secure.
- ▶ And lots of work exist to make sure that the corresponding properties are valid in the real world.
- ▶ But what about implementations?

“It seems people just aren’t very good at writing or reviewing security-sensitive code. It seems to me that we need better static analysis tools to catch this kind of obvious error.” - Ben Laurie

Static analysis: existing solutions

- ▶ Tools for security analysis of implementations:
 - FS2PV/FS2CV** Verifying F# implementations of protocols such as TLS [Bhargavan *et al.*, 2008]
 - F7** Refinement types for security in F# [Bhargavan *et al.*, 2006].
- ▶ Tools for analyzing C:
 - CCured/Deputy** Type-based memory safety for legacy code.
 - Cyclone** Strongly typed C dialect for memory safety.
 - Model checkers** Slam, Blast...
 - VCC** General C verifier. Similar to Spec#, but adapted to C in a concurrent context.

Analysing security protocols in C: existing solutions

Csur allows to write a new but secure implementation of a protocol, given a specification.

But we want to work on legacy implementations.

Aspier A model-checking approach [Chaki and Datta, 2008].

Finite model-checking, so ensures security up to a certain number of parties.

What we want to do

- ▶ Statically analyse security protocol implementations for security issues.
- ▶ On real programs, written in C (that is, OpenSSL, really).
- ▶ Possibly without reinventing the wheel.
- ▶ With as little human intervention as possible (annotations, code understanding).

Limits

At least in a first approach, we won't:

- ▶ Verify any cryptographic functions: we assume they are correct and provide what is required by the protocol.
- ▶ Check for memory safety: we assume this is done somewhere else (e.g. CCured, operating system...).

Possible Leads and Options

- ▶ Extract a “model” from C code? Work directly at the source level?
- ▶ Perform pure verification (express the security properties in the verification logic)?
- ▶ Use intermediate steps (verify C code against a model, then verified against the spec)?
- ▶ Types? Model-based? Contract-based?

Correspondences with VCC

We are here in an approach where we work at C level, trying to prove a Needham-Schroeder-Lowe client (Alice) compliant with its specification.

- ▶ Started by reproducing a ProVerif-like approach in VCC:
 - ▶ `assume(a1)`, `assert(a1)` allow to statically verify correspondences.
 - ▶ But using `assume` makes it hard to prove anything (`assume(false)`, anyone?)
 - ▶ And we need to figure out where to stick those commands. And prove we didn't forget any...

Static Monitoring with VCC

- ▶ The other idea is to write up a simple minimal version of the protocol, with symbolic crypto functions.
 - ▶ Being simple, it is easily verifiable by itself.
 - ▶ Using VCC, we just plug in monitoring calls everytime a protocol message is sent or received in the implementation.
 - ▶ But we still need to wrap function calls to call the abstract functions... Necessary information should exist in the documentation.

Any Questions?



Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Stephen Tse.

Verified interoperable implementations of security protocols. pages 139–152, 2006.



Karthikeyan Bhargavan, Cédric Fournet, Ricardo Corin, and Eugen Zălinescu.

Cryptographically verified implementations for TLS. Alexandria, VA, October 2008. ACM.



Sagar Chaki and Anupam Datta.

Automated verification of security protocol implementations. Technical Report CMU-CyLab-08-002, CyLab, Carnegie Mellon University, 2008.