Applying Runtime Verification to Group Key Establishment

Secure Communication in the Quantum Era (SPS G5448)

Computer Science Annual Workshop
Malta - November 2018

Christian Colombo
Authenticated group key establishment

Secure communication depends on establishing common secret key
Authenticated group key establishment

Secure communication depends on establishing **common secret key**

Proving the protocol correct is first step...

a) $A$ sends $B$ the message $(A, E_B(MA), B)$,
b) $B$ answers $A$ by sending $(B, E_A(MB), A)$. 
What can go wrong at runtime?

...but in practice is far from enough

(High level) Wrong protocol implementation
The protocol implementation might deviate from the verified (theoretical) design

Low level threats
Arithmetic overflows, undefined downcasts, and invalid pointer references

Hardware
Can hardware be trusted?
Passive / Active attacks

Something bad accidentally happens

Vs

Something bad actively sought
Unintended consequences

- Timing attacks
- Cache timing attacks
- Microarchitecture side-channel attack
- Power/EM/acoustic attacks
- Fault attacks
- Reaction attacks
- Data remanence attacks
- Attacks on random number generators
Timing attack

If (secret)
  Do something lengthy
Else
  Do something simple

An external observer can learn the secret by observing the duration of the execution. (or the power used or any other side effect)
What has been done?
High level logic

- Before any data is sent by the client, the server hash is verified to match the client's version.

- If the operation is of type “Send”, then the message receiver ID must be in the set of approved receiver IDs.
Low level considerations

General considerations for any code

- Arithmetic overflows
- Undefined downcasts
- Invalid pointer references
Mid-level

Applicable to any crypto protocol

Data flow monitoring

E.g. Ensuring not control is decided on secret data

(which affects the timing of the program)
Frama-C

Is a framework supporting all of these levels

Low-level is inbuilt

Mid-level is provided through library support

High-level is provided through specification languages
Other tools/frameworks?

Copilot and other tools focus more on high level properties

Work on hyperproperties

  I.e. properties on several runs of a program
What are the challenges?
Challenges for RV

Over and above the usual correctness and overheads concerns

The monitor can present an additional security vulnerability

>> As a piece of code

>> As a reaction triggering device
Soundness/Completeness of dynamic analysis

if $h = 1$ then $b := 1$ else skip;
if $b \neq 1$ then $l := 1$ else skip;
output$_L(l)$

Assume $h$ is a high security variable
When $h \neq 1$,
Monitor can’t mark $b$ as high
(without analysing the “if” statically)
Soundness/Completeness of dynamic analysis

(a) Flow-insensitive analysis
(b) Flow-sensitive analysis
(c) Flow-sensitive analysis, hybrid monitors
Other techniques?

For example Multi-execution approach:

Generate low security outputs with only low security inputs in the system

→ **Result:** No high security output may depend on low security input
Our plan
Project Kick-off Meeting

Step 1-A: Choice of security model
Step 1-B: Implementation security of cryptographic primitives

Step 2-A: Identify candidate protocol
Step 2-B: Identify protocol-level security mechanisms

Step 3-A: Protocol and parameter optimization
Step 3-B: Deploy implementation-level security mechanisms

Final Report & Workshop
Questions to be answered

To what extent existing tools/frameworks are immune to attacks themselves

Is there any effect of the quantum prospect on RV?

What mix of new/existing techniques + technologies to adopt