Formal Methods for Security

Erik Poll

Digital Security group

Radboud University Nijmegen

FMATS workshop, December 2011
Overview

• **Formal methods**
  • in general and for security

• **Case studies: formal methods for security protocols**
  • to illustrate the different ways in formal methods can be used
  • based on our group’s formal & informal investigations
Formal methods involve models of which properties (eg bridge won’t collapse) can be specified and verified (modulo modelling & abstraction errors) using some methodology/theory.

\[ F_1 + F_M = 2 \sin \beta \cdot F_2 \quad F_1 = L \cdot H \cdot \rho \quad \ldots \]

Starting point for all: specification

- which for a bridge is very simple & unchanged for ages
Formal Methods for Software Engineering??

specs
incl. functional requirements
security requirements

model??
properties??

product, ie code
From specs to code

code itself is also possible formal model!

candidate formal models?
Formal methods at different levels

• Formal methods for *programming languages*, eg
  • type system to rule out buffer overflows
  • static analysis to detect XSS vulnerabilities

• Formal methods for *abstract algorithms & protocols*, eg
  • prove that your shortest path algorithm is functionally correct
  • prove that HTTPS is secure

• Formal methods for *programs*, eg
  • prove that a program never throws a NullPointerException
  • prove that a program correctly implements HTTPS
security vs correctness

• A program is **correct** if it does what it should do
  • ie. *presence* of the *right* behaviour, under normal circumstances
• A program is **secure** if it is does not do what it should *not* do
  • ie. *absence* of *insecure* behaviour, under *any* circumstances
  • easy to overlook, and hard to check (eg by testing)
• A program also has to be correct for it to be secure?

**Good news:** some (generic) security requirements are independent of any detailed functional spec  (eg absence of integer overflows)

**Bad news:** security requirements may be hard to pin down  
(what does it mean for a system to be secure?)
Case studies:
formal methods for
(implementation of)
security protocols
Security protocols

• Why security protocols?
  • they are security-critical components in systems
    • eg HTTPS, EMV (Chip & PIN), electronic passports, ...
  • they are small but complex
  • they have clear security objectives

Note:
• forget about crypto, it’s the protocols that matter!
• we can study the abstract protocols, or their concrete implementations
Potential problems in security protocols

1. using insecure cryptographic primitives (eg. Oyster card)
2. using default keys (eg. lots of systems)
3. using an buggy protocol. Security protocols are tricky to get right!
4. using an buggy implementation. Software bugs can break
   a) correctness 
      Easy to detect, since the implementation won’t work
   b) security, by erroneously accepting or crashing on
      • incorrect (malformed) message or
      • incorrect order of messages.
      This is harder to detect, since the implementation will work
Some example formal models for security protocols

**Alice-Bob notation**

1. A → B: start session
2. B → A: ok
3. A → B: Nonce\(_A\)
4. B → A: encrypt\(_{KEY}(\text{Nonce}_{A})\)
5. A → B: ...
6. B → A: ...

Such (partial) models capture different aspects and hence can be used for different goals and in different ways (see next slides)
I. Security Protocol Analysis

• Given a formal description of the abstract security protocol, eg. in Alice-Bob notation, we can formally analyse some of its properties.
  • possible using tool support

Eg next talk by Joeri de Ruiter, and plenty of others.
II. Model based testing

- We automatically test if implementation conforms to the model
  - we feed randomly generated inputs to both model and code, and check if they behave the same
  - the model is used as test oracle
    - possibly also for generating tests & measuring test coverage
  - by aggressively testing many (all?) possible sequences we can test for security as well as correctness - “state-based” fuzzing
- Eg we have done this for the electronic passport. [W. Mostowski et al, FMICS 2009]
III. Program verification

• A more rigorous form of checking compliance of code & model: formal verification (with mathematical proof) that the code conforms to the model

• Eg for a Java implementation of SSH  [E.Poll and A.Schubert, WITS 2007]

A formal model can also be used, informally, by a human code reviewer

Erik Poll, FMATS 2011
III. Program verification

- Even without any formal model, we can use formal verification to verify that the code meets some security property.

**Problem:** what do we want to verify anyway?

- Automated program verifier: eg ESC/Java2
- Security property: eg in JML
- OK
III. Program specification: what to verify?

Typical easy properties to begin specifying:

   (i) important invariants      (ii) absence of runtime exceptions

plus the additional preconditions and invariants this requires.

public class ElectronicPurse extends javacard.framework.Applet {
    private int balance;  //@ invariant 0 <= balance;

    //@ requires buffer != null && 0 <= offset && offset+length <=
    buffer.length;

    public static void install (byte[] buffer, short offset, byte length) {
        ....
    }

Erik Poll, FMATS 2011
IV. Model extraction

- Automated learning techniques can be used (in combination with model-based testing) to infer an automaton for an implementation’s behaviour.

Automaton learned from a Dutch EMV bankcard
[Fides Aarts et al, ISoLA'10]
Conclusions

• **Central challenges**
  • Does code meet the specs?
  • Do specs & code not overlook or introduce security problems?

• **Formal models & methods can help in different ways**
  - Compare, by
    • Testing
    • Code review
    • Verification
  - Extract
  - Generate
  - Analyse

Erik Poll, FMATS 2011