Case Study: Using PVS to Analyze Security Protocols

Kyle Taylor

Overview

- Using PVS:
  - Specify Communication
  - Specify Encryption
  - Specify Intruder
  - Specify Protocol
  - Analyze Results

What is PVS

- General
  - Prototype Verification System
  - Inductive Theorem Prover
  - Augments classical higher-order logic with an advanced type system
  - Implemented in Common Lisp
  - Based upon 20 years of experience
  - Emacs Interface
What is PVS Cont.

- Intended for the formalization of requirements and high level design
- Functional Specification Language
  - Haskell, Lisp, ML
- PVS consists of
  - Specification language
  - Predefined theories
  - Proof Checker

PVS Specification Language

- Type checking is undecidable because of constrained types
  - Generates proof obligations for the theorem prover
- Assumptions
- Definitions
- Axioms
- Conjectures
- Theorems
- Lemmas

PVS Proof Checker

- Provides a collection of powerful primitive inference procedures
  - Propositional and quantified rules
  - Induction
  - Rewriting
  - Decision procedures for linear arithmetic
- Requires Interactive proving
  - Contains automated proof strategies
PVS and Security

- PVS does not have built in security or communication primitives
- The primitives need to be defined when a specification is developed
- Luckily the definitions of the security primitives can be reused

Communication and Encryption

- Messages are defined as a function
  - \( \text{Said}(\text{Sender}: \mathbb{P}, \text{Receiver}: \mathbb{P}, \text{Fields}: \mathbb{F}) \)
- Fields are constructed by type coercion
  - \( \text{Agent}(<\text{Principle}: \mathbb{P}>) : \mathbb{F} \)
  - \( \text{Num}(\text{num}: \mathbb{nat}) : \mathbb{F} \)
  - \( \text{Pkey}(<\text{pkey}: \mathbb{K}>) : \mathbb{F} \)
  - \( \text{Con}(\text{field}: \mathbb{F}, \text{field}: \mathbb{F}) : \mathbb{F} \)
  - \( \text{Ped}(\text{pkey} : \mathbb{K}, \text{field}: \mathbb{F}) : \mathbb{F} \)

Communication and Encryption Cont

- Encryption
  - \( \text{Pub}(\text{principle}: \mathbb{P}) : \mathbb{K} \)
  - \( \text{Priv}(\text{principle}: \mathbb{P}) : \mathbb{K} \)
  - \( \text{Invkey}(\text{key}: \mathbb{K}) : \mathbb{K} \)
- Axiom
  - \( \text{Invkey} (\text{pub}(\text{A})) = \text{priv} (\text{A}) \)
Specifying Security

- Parts(S) – All sub fields of a set S
- Analz(S) – All sub fields of a set S that is accessible to an attacker
- Synth(S) – The set of fields that is constructible for S
- Secret() – Basic secrets
- Initial() – Initial attacker knowledge

Specifying a Protocol

- Each protocol step is a function
  - The function accepts a Message and a Trace
  - The function returns a bool
  - The function constrains the Message and the Trace

Specifying an Intruder

- Done with a function
  - The function accepts a Message, a Trace, and a set of initial knowledge
  - The function returns a bool
  - Constructs messages from a set of initial knowledge and the set of derivable knowledge present in a trace
  - Constructed message is from the intruder
ffgg Protocol

1. A $\rightarrow$ B: A
2. B $\rightarrow$ A: B, N$_1$, N$_2$
3. A $\rightarrow$ B: A, (N$_1$, N$_2$, M)$_{PKB}$ % (N$_1$, X, Y)$_{PKB}$
4. B $\rightarrow$ A: N$_1$, X, (X, Y, N$_1$)$_{PKB}$

ffgg Attack

- A $\rightarrow$ B: A
- (A) $\rightarrow$ B': A % masquerade
- B $\rightarrow$ A: N$_1$, N$_2$
- B' $\rightarrow$ A: N'$_1$, N'$_2$
- (B) $\rightarrow$ A: N$_1$, N'$_1$
- A $\rightarrow$ B: (N$_1$, N'$_1$, M)$_{PKB}$
- B $\rightarrow$ A: N$_1$, N'$_1$, (N'$_1$, M, N$_1$)$_{PKB}$
- (A) $\rightarrow$ B': (N'$_1$, M, N$_1$)$_{PKB}$
- B' $\rightarrow$ A: N'$_1$, M, (M, N$_1$, N'$_1$)$_{PKB}$

ffgg Specification

\[
\text{a1(E,H)}: \text{bool} = \exists (A, B) : \\
A \neq B \land E = \text{Said}(A, B, \text{Con}(\text{Num}(1), \text{Agent}(A)))
\]
ffgg Specification Cont

```plaintext
ffgg(Q: trace): RECURSIVE bool=
CASES Q OF
  cons(E, H): ffgg(H) AND
    (Fake(E, H, initial)
    OR a1(E, H)
    OR a2(E, H)
    OR a3(E, H)
  OR a4(E, H)),
null: TRUE
ENDCASES MEASURE length
```

ffgg Results

- PVS derived a case enumerating a trace of the attack that the user would have to prove

Discussion

- Lessons Learned
  - Learning PVS is not trivial
  - Specification and prover are not intuitive
  - Theorem proving has the ability to be very useful
  - Security is tricky
- Limitations
  - Amount of background required of the user
  - Amount of training time required
  - Ambiguity of specification
Future Work

- Develop a theorem prover that is specifically designed for Security protocols (Athena?)
- Provide a less powerful specification language to increase simplicity
- Provide better separation of modules

Conclusion

- 6 month estimate to learn PVS is very optimistic
- One on one training with a PVS guru is a must
- Limited use due to ambiguity of specifications

References