

Unbounded Verification, Falsification, and Characterization of Security Protocols by Pattern Refinement

Paper by : Cas J.F. Cremers

S.AIAwadi, H.Alnoon, S.AITamimi, O.BaniHashim & A.Rojat

November 27, 2009

Outlines

- 1 Introduction
- 2 Protocol Design
- 3 Pattern Notion
- 4 Algorithm
- 5 Comparison
- 6 Conclusion

Goals and Objectives

New verification algorithm for security protocols that allows:

- Unbounded verification
- Complete characterization
- Falsification

Introduction

- Semi-automatic tools available
- Bounded verification
 - Majority of algorithms deals with undecidability
 - Finite subset of behaviors are considered
 - Bounding is done on: message size, number of nonces, number of protocol sessions, or a combination
- Unbounded verification
 - Algorithm proposed is based on this type of verification
 - Prove that security protocols holds for all possible behaviors in presence of an intruder

Introduction

- Complete characterization
 - Finite set of representation of all protocol behavior
 - Present protocols to allow verification of protocols
- Falsification
 - The act of disproving a protocol
 - Always generates counterexamples

Protocol Design

- The most fundamental structure in the protocol design is the set of **terms**.

$$\textit{BasicTerm} ::= \textit{Const} | \textit{Nonce}_{ID} | \textit{Var}_{ID}$$

$$\textit{Term} ::= \textit{BasicTerm} | (\textit{Term}, \textit{Term}) | \{ \textit{Term} \}_{\textit{Term}}$$

- Includes basic terms and terms constructed using one of the two constructors:
- **Tupling** (t, t')
- **Encryption** $\{t\}_{t'}$, with t as the encrypted term and t' as the key.

Protocol Design (contd)

- Each variable has a type by definition.
- Type *Term* can be used for variables to capture type flaw attacks, or ciphertext forwarding.

$$\text{Type} : \text{Term} \rightarrow P(\text{Term})$$

- The function **Type** : $\text{Term} \rightarrow P(\text{Term})$ is used to define types for all the variables.
- **Substitution** $\sigma = [u_1, \dots, u_N / v_1, \dots, v_N]$

Protocol Design (contd)

- Protocols from the set **Protocols** define a finite set of roles.
- a **Role** is defined as a finite list of protocol events.

$$ProtEvent ::= send_{ID}(Term) | recv_{ID}(Term)$$

- **Thread** (strand, run or instance) is an execution of a protocol role.
- Identified by a unique identifier from the set of thread identifiers **ID**.
- a **Trace** (tr) is a set of events with an associated total order, denoted as:

$$tr = (E, \leq)$$

Protocol Design (contd)

- We assume a **Dolev-Yao** style intruder.
 - Full control over the network.
 - Every sent message is observed by him.
 - Every recieved message is supplied by him.
 - Can generate any number of any type of each variable.
 - Can impersonate the compromised agents with whom he conspires.

Protocol Design (contd)

- The set of **Events** that can occur during protocol executions consists of:
 - Protocol events
 - Events of the intruder.

$$\text{IntruderEvents} ::= \text{init} | \text{encr} | \text{decr} | \text{know}$$
$$\text{Event} ::= \text{ProtEvent} | \text{IntruderEvent} (\text{Term})$$

Protocol Design (contd)

- To capture the interaction between the events and the intruder knowledge,
- we define two functions:
 - Input function (in) : the terms that are required to be in the intruder knowledge to enable the event.
 - Output function (out) : the terms that are added to the intruder knowledge after an event.

Patterns and Traces

We introduce the notion of patterns in order to reason about infinite sets of traces of a protocol.

Patterns

- A tuple $pt = (E, \rightarrow)$
 - E is a set of events
 - \rightarrow relation of events from E
- Forms a Directed Acyclic Graph (DAG)
 - This generalizes a total order of events occurring in traces

Edges

Edges are represented with \rightarrow

- Unlabeled Edge ($e \rightarrow e'$)
 - Represent order of events within single role instance
- Labeled Edge ($e \xrightarrow{t} e'$)
 - Represents the earliest point a message known to intruder

Reflexive Transitive Closure (\rightarrow^*)

- Both Labeled and Unlabeled Variants

Pattern of a protocol

DAG (E, \rightarrow) is a pattern of a protocol iff

- 1 Binding donate message causality

$$e \xrightarrow{t} e' \Rightarrow t \in \text{out}(e) \wedge t \in \text{in}(e')$$

- 2 Terms are bound to the earliest possible event

$$\forall e, e', e'', t : (e \rightarrow^* e' \wedge e' \xrightarrow{t} e'' \wedge t \in \text{out}(e)) \Rightarrow e = e'$$

- 3 The pattern is consistent with the protocol Q

$$\forall e \in \text{ProtEvent} \cap E$$

Realizable Patterns

- A pattern pt can be considered as filter on the traces of Q
- For some patterns one can directly construct traces of the protocol that contain the pattern (Realizable Patterns)
- Realizable
 $((E, \rightarrow)) \equiv \forall e \in E : \forall t \in in(e) \text{ Var} : \exists e' \rightarrow^* e : t \in out(e')$
- Patterns not satisfying realizable predicate, are not clear if they represent traces of the protocol.

Pattern Refinement

- Refine patterns into realizable patterns
- If a pattern pt not realizable and $traces(Q, pt) \neq \emptyset$ there exists an event E requirements not satisfied
- Patterns can be refined by adding
 - Adding events
 - Adding edges
 - Performing well-typed substitutions
- Some patterns can never be refined.

InputOutput

Input

- Q a protocol
- pat a pattern

Output

- A set of realizable patterns

Initial "IF"

Basic case

```
if (pat is realizable) then  
    RETURN pat  
else  
    REFINEMENT(Q,pat)  
end if
```

REFINEMENT

STEP 1

Select a unbounded term

REFINEMENT

STEP 1

Select a unbounded term

STEP 2

perform all possible decryption in pat using *chain*
 $RES_1 \leftarrow \text{decryption_patt} \cup \text{REFINE}(Q, \text{pat})$

REFINEMENT

STEP 1

Select a unbounded term

STEP 2

perform all possible decryption in pat using *chain*
 $RES_1 \leftarrow \text{decryption_patt} \cup \text{REFINE}(Q, \text{pat})$

STEP 3

if (the selected term is labeled with an encryption) **then**
 add this encryption to pat
 $RES_2 \leftarrow \text{REFINE}(Q, \text{pat})$
end if

REFINEMENT (contd)

STEP 4

```
for All element of the completed pattern do  
  if (element is not an INIT) then  
    Add a new thread identifier NEW  
    for All agents of pat and NEW do  
      pat_tmp ← substitute previously unidentified elements in  
      pat  
      RES_3 ← RES_3 U REFINE(Q,pat,pat_tmp)  
    end for  
  end if  
end for
```

REFINEMENT (contd)

STEP 5

RETURN RES_1 U RES_2 U RES_3

Analysis of the algorithm

The algorithm terminates

Decreasing in STEP 1, STEP 2, STEP 3

Increasing honest agents events STEP 4

Verification By Pattern Refinement

- Confidentiality: pattern capture all traces with false property
- Characterization: algorithm is applied to the pattern to provide concise representation of possible protocol behaviors
- Authentication: check verification of aliveness, non-injective agreement.

Comparison

Tool	Point
Atehna Algorithm	<ul style="list-style-type: none"> • Has not been publicly released • only reported to be efficient • very close in description
ProVerif	<ul style="list-style-type: none"> • Allows for generation of counter examples
CPSA	<ul style="list-style-type: none"> • Strong connection to authentication tests • Not guaranteed to terminate
AVISPA	<ul style="list-style-type: none"> • Scyther outperformed the bounded tools OFMC, CL-ATSE, SATMC • Scyther and ProVerif outperformed TA4SP for unbounded verification

Conclusion

- Algorithm for unbounded verification falsification and characterization of security protocols.
- Provides state-of-the-art performance
- Ensure Termination whilst still proving meaningful verification result

Bibliography

- Cas Cremers Unbounded Verification, Falsification, and Characterization of Security Protocols by Pattern Renement In CCS 08 : Proceedings of the 15th ACM conference on Computer and communications security, Alexandria, Virginia, USA.
- The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols In Computer Aided Verification Proceedings of the 20th International Conference on Computer Aided Verification (CAV 2008), Princeton, USA, 2008.
- Scyther - Semantics and Verification of Security Protocols Thesis, University Press Eindhoven, 2006. ISBN 90-386-0804-7. - ISBN 978-90-386-0804-41

Questions

Thanks

Questions ?