
Lecture 9

Tools and Applications

10 novembre 2009

Table des matières

1	AVISPA	2
1.1	Tool description	2
1.2	Example	2
2	Scyther	4
2.1	Tool description	4
2.2	example	4
3	Proverif	5
3.1	Tool description	5
	3.1.1 Horn clauses	5
	3.1.2 Spi-calculus	5
3.2	example	6
4	Comparison	7
4.1	Algebraic properties	8
4.2	Conclusion	8

Introduction

In the previous lessons we have seen attacks on protocols (e.g. man in the middle attack) and some corrections we may have to counter those attacks. This lecture is about automated tools that detect some of the attacks possible given a protocol and that allows us to correct them. The tools presented in this lecture are the following :

1. AVISPA (/it Automated Validation of Internet Security Protocols and Applications)
2. Scyther¹
3. Proverif²

1 AVISPA

1.1 Tool description

The AVISPA project aims at developing a push-button, industrial-strength technology for the analysis of large-scale Internet security-sensitive protocols and applications. AVISPA can use 4 methods in order to check a given security protocol :

1. OFMC (*On the Fly Mode Checker*) which uses symbolic techniques.
2. CL-AtSe that uses simplification heuristics and redundancy elimination techniques.
3. SATMC (*SAT based Model Checker*) that uses SAT-solvers in order to find a proposition leading to a fail in the model.
4. TA4SP which build regular grammar in order to interpret and evaluate the intruder knowledge

There exists 2 different modes that can be used in AVISPA : Basic and Expert modes. Choosing the expert mode allows you to decide which of the tools you will use.

1.2 Example

As an example we will consider the Needham-Schröder protocol. Let's see ho to define it using AVISPA and the tool analysis. Let's begin to set Alice's role. This description will contain her name, her knowlegde and a set of rules (transitions).

```
role alice (A, B : agent,  
Ka, Kb : public_key,  
SND, RCV: channel (dy))  
played_by A def=  
local State : nat,
```

¹see web page : <http://people.inf.ethz.ch/cremersc/scyther/index.html>

²see project page : <http://www.proverif.ens.fr/>

```

Na, Nb : text
init State := 0
transition
0. State = 0 /\ RCV(start) =|>
State' := 2 /\ Na' := new() /\ SND({Na'.A}_Kb)
/\ secret(Na',na,{A,B})
2. State = 2 /\ RCV({Na.Nb'}_Ka) =|>
State' := 4 /\

```

Then the definition of Bob's role :

```

role bob(A, B : agent,
Ka, Kb : public_key,
SND, RCV : channel (dy))
played_by B def=
local State : nat,
Na, Nb : text
init State := 1
transition
1. State = 1 /\ RCV({Na'.A}_Kb) =|>
State' := 3 /\ Nb' := new() /\ SND({Na'.Nb'}_Ka)
/\ secret(Nb',nb,{A,B})
3. State = 3 /\ RCV({Nb}_Kb) =|>
end role
Definition of a session

```

Reading those 2 definitions leads us to the following (and well known) protocol :

$$A \rightarrow B : Na, A_Kb$$

$$B \rightarrow A : Na, Nb_Ka$$

$$A \rightarrow B : Nb_Kb$$

Then we have to define the context in which the protocol will be used : the session. This will define the agents and the secret. This will also define the basic knowledge of a potential intruder.

```

role session(A, B: agent, Ka, Kb: public_key) def=
local SA, RA, SB, RB: channel (dy)
composition
alice(A,B,Ka,Kb,SA,RA) /\ bob (A,B,Ka,Kb,SB,RB)
end role
role environment() def=
const a, b : agent,

```

```

ka, kb, ki : public_key,
na, nb, : protocol_id
intruder_knowledge = {a, b, ka, kb, ki, inv(ki)}
composition
session(a,b,ka,kb) /\ session(a,i,ka,ki)
/\ session(i,b,ki,kb)
end role
goal secrecy_of na, nb
end goal
environment()

```

2 Scyther

2.1 Tool description

Scyther is a tool for the automatic verification of security protocols. The verification algorithm is based on backward analysis. The claims represents the knowledge of both player (sender and receiver) and also the aim of an eventual intruder. Symetric and asymeric encryption, hash functions and key tables are swedgeorted in Scyther, too.

2.2 example

A protocol defining the role of both persons is given in input :

```

protocol ns3(I,R) {
role I {
const ni: Nonce;
var nr: Nonce;
send_1(I,R, {ni,I}pk(R) );
read_2(R,I, {ni,nr}pk(I) );
send_3(I,R, {nr}pk(R) );
claim_i1(I,Secret,ni);
claim_i2(I,Nisynch);
}
role R {
var ni: Nonce;
const nr: Nonce;
read_1(I,R, {ni,I}pk(R) );
send_2(R,I, {ni,nr}pk(I) );
read_3(I,R, {nr}pk(R) );
claim_r1(R,Secret,ni);
claim_r2(R,Nisynch);
} }

```

The claims that appear in both roles are the definition of the secret. The output of the algorithm is a set of graphs describing possible attacks from an intruder. For the intruder, the goal consists in discovering at least one of the secrets.

3 Proverif

3.1 Tool description

ProVerif is developed by Bruno Blanchet, researcher at Computer Science Laboratory of Ecole Normale Supérieure. This tool processes input files formatted as a sequence of Horn clauses or as a process in the applied pi-calculus (which will be translated into Horn clauses before being run).

Commands : `analyser -in horn toto.pv` (in case of using Horn clauses as input language) `analyser -in pi toto.pv` (for pi-calculus)

3.1.1 Horn clauses

In mathematical logic, a Horn clause is a clause (a disjunction of literals) with at most one positive literal. They are named after the logician Alfred Horn, who first pointed out the significance of such clauses in 1951.

Proverif uses definite Horn clauses, which have exactly one positive literal :

$$\neg p \vee \neg q \vee \dots \vee \neg t \vee u$$

It is more convenient to rewrite it in the form of an implication :

$$(p \wedge q \wedge \dots \wedge t) \rightarrow u$$

3.1.2 Spi-calculus

Spi-calculus is an extension of the original pi-calculus by Robin Milner, which has been developed by Martin Abadi and Cedric Fournet and is designed to express cryptographic protocols. The pi-calculus is a process calculus. It describes concurrent computations whose configuration may change during the computation. Pi-calculus processes use channels to communicate and can execute in parallel.

Example :

$$c < a > | cx.d < x >$$

The first process sends `a` on channel `c`, the second one inputs this message, puts it in variable `x` and sends `x` on channel `d`. In cryptography protocols, pi-calculus processes represent the participants that exchange the messages specified by the protocol. However, in

protocols, messages are not necessarily atomic values of the original pi-calculus. The names of the pi calculus are replaced by terms in the spi-calculus.

```

M,N ::= Terms
x, y, z Variable
a, b, c, k Name
f (M1, ...,Mn) Constructor application

P,Q ::= Processes
M < N > .P Output
M(x).P Input
let x = g(M1, ...,Mn) in P else Q Destructor application
if M = N then P else Q Conditional
0 Nil process
P|Q Parallel composition
!P Replication
(?a)P Restriction

```

As an example, here is the Denning Sacco protocol

```

Message 1: A ? B : { {k}_skA }_pkB
Message 2: B ? A{s}k , k fresh
(?skA)( ?skB) let pkA = pk(skA) in let pkB = pk(skB) in c<pkA> c<pkB>.
(A) !c(x_pkB).( ?k) c <pcrypt(sign(k, skA), x_pkB)>.
c(x).let s = sdecrypt(x, k) in 0
(B) !!c(y).lety' = pdecrypt(y, skB) in let k = checksign(y', pkA) in c <scrypt(s, k)>

```

3.2 example

Horn clauses are designed using a text format source. The following example describe the Needham Schroeder protocol :

```

(* Instantiation of the channel c *)
pred c/1 elimVar,decompData.
nounif c:x.

(* Declaration of functions by name/arity *)
fun pk/1.
fun encrypt/2.

(* To do: find the secret *)
query c:secret[].
Reduc

```

```

(*
Declaration of public knowledge:
- The channel
- Public keys
- How to encrypt/decrypt
- How to retrieve a public key
*)
c:c[];
c:pk(sA[]);
c:pk(sB[]);
c:x & c:encrypt(m,pk(x)) -> c:m;
c:x -> c:pk(x);
c:x & c:y -> c:encrypt(x,y);

(* Protocol specification*)

(* Alice's side *)
c:pk(x) -> c:encrypt((Na[pk(x)], pk(sA[])), pk(x));
c:pk(x) & c:encrypt((Na[pk(x)], y), pk(sA[])) -> c:encrypt((y,k[pk(x)]), pk(x));

(* Bob's side *)
c:encrypt((x,y), pk(sB[])) -> c:encrypt((x, Nb[x,y], pk(sB[])), y);
(* Adjunction of an artifact specifying what is the secret *)
c:encrypt((x,pk(sA[]))) & c:encrypt((Nb[x, pk(sA[])], z), pk(sB[]))
-> c:encrypt(secret[], pk(z)).

```

The analysis of this protocol outputs the following :

```
goal reachable : c :secret[]
```

This means the protocol is not secure and the attack is detailed in the following lines.

4 Comparison

The tools presented in this lectures are all protocol security checkers. But their properties are different and before using one of them we have to understand what are those differences. But those tools are needed in such a protocol analysis in order to establish quickly and precisely what are the flaws in the given protocol. The outputs given by the tool should be easily readable in order to make the analysis and the correction more efficient.

4.1 Algebraic properties

Let's remind what is OFMC (On the Fly Model Checker) :

Input :

- Transition System
- Goal (secret to discover)
- Algebraic properties

Output :

- SAFE (for a bounded number of sessions)
- Attack trace (if any)

The swedgeorted theories are the following :

- **Finite Theories F** The F-equivalence class of each term is finite
- **Cancellation Theories C** One side of the equation is a variable of the other side, or a constant.

OFMC swedgeorts a lot of features such as symemetric and asymmetric keys, cryptographic hash funtions, key-tables, user-defineable algebraic functions, ...

But OFMC may not be sufficient and some properties may not be taken in account. A protocol can be told as SAFE although there exists an attack.

Then there is two different properties we have to deal with :

1. Exclusive-OR
2. Diffie-Hellman

The tools deal with those aspects are the following :

- AVISPA
 - OFMC : uses symbolic techniques to explore states, driven by the goal to reach.
 - CL-Atse : uses constraint logic, redundancy elimination and heuristics to break the secret
- ProVerif

There exists two tools, XOR-ProVerif and DH-ProVerif that are included in Proverif which analyse the security of a given protocol using XOR and DH properties.

4.2 Conclusion

The tools given before usually return the same attacks with their differents modes. We have seen during the lecture that sometimes an algorithm does not finish while the other does, using the both increases the chances to find a flaw, if any.

- OFMC usually terminates faster than CL-AtSe but OFMC does not terminate while checking protocol with a large number of XORs whereas Cl-AtSe does.
- The number of XOR used in a protocol increases verification time.

- If there are not too many variables and constants, ProVerif is faster than AVISPA
- Considering the DH-properties, the protocols were analyzed quickly. This proves the polynomial complexity of DH-ProVerif and the fact that XOR is more complex than the equational theory

conclusion

We have seen during this session three different tools that may be used for protocol verification : AVISPA, Scyther and ProVerif. Then we've been introduced with the limits of such tools. Indeed, although one checker does not find any flaw, this does not mean there does not exist any flaw. But those tools are needed in order to point out efficiently flaws that could have taken much more time to find manually.