Information Technology Journal 8 (7): 1065-1070, 2009
ISSN 1812-5638
© 2009 Asian Network for Scientific Information

Applying SMV for Security Protocol Verification

[1,2]Jia Mei, [1]Huaikou Miao and [1]Pan Liu
[1]School of Computer Engineering and Science, Shanghai University, Shanghai, China
[2]Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai, China

Abstract: With the rapid development of the internet, a lot of attentions have been paid to the reliability of the security protocols. Model checking can be used to obtain the assurance that a protocol can not be threatened by an intruder. In this study, on the basis of former researches, an approach is presented for using efficient and complete formal verification tool SMV to model and verify security protocol. By this approach, we can construct related model easily and verifying the property automatically. We illustrate the approach by taking Otway-Rees protocol as an example and discover an attack upon the protocol. Finally, the protocol is adapted to satisfy the security properties.

Key words: Model checking, security protocol, SMV

INTRODUCTION

Security protocols define the rules of exchanging messages between the principals in order to establish a secure communication channel between them. The environment of these communications is very hostile because no transmission channel can be considered safe.

Model checking can be used to obtain the assurance that a protocol can not be threatened by an intruder. Intuitively, model checking of a security protocol consists in checking whether a model of the protocol accepts an execution (or contains a reachable state) that is representing an attack on the protocol.

Past approaches to reasoning about security protocols, e.g., authentication protocols, have relied on either pencil-and-paper proof or machine-assisted proof through the use of interactive theorem provers, e.g., the Boyer-Moore Prover (Goldschlag, 1990), Gypsy (Good *et al*., 1975) and UNISEX (Kemmerer and Eckmann, 1985). Proofs of properties of these protocols relied on either specialized logics, e.g., Burrows-Abadi-Needham's logic of authentication, or an encoding of a specialized logic in the theorem prover's general-purpose logic. These proofs are tedious to do. If done by hand, they are prone to error and they can be applied only to small examples. If done by machine, they require tremendous user patience, since the machine usually insists on treating the critically creative and the boring bookkeeping steps of the proof all with equal importance; they often take hours to complete, even discounting human user time; and they are also prone to error since they still rely on human intervention and ingenuity.

The use of formal methods to analyze the correctness of security protocols became prevalent with the development of the BAN logic in 1989 (Burrows *et al.*, 1990). The BAN is a modal logic of belief for the specification and verification of security protocols and it is the most known and famous logic dedicated to security protocols. Since, then, plenty of derived logics have been advanced (Abadi and Tuttle, 1991; Li *et al*., 1990). But there are some limitations in BAN logic: security protocols have not traditionally been expressed in a completely formal manner and so it is inevitable that there must be some conversion of an informal description to a formal description if formal analysis is to take place. However, the BAN logic does not correspond very well to usual formal descriptions of security protocols.

Lowe (1996) used the FDR system, a model checker for CSP, to find a weakness in the Needham-Schroeder public-key protocol and to analyze a corrected version of the protocol, with small bounds on the parameters. But using CSP to describe security protocol is not easy and error-prone.

Since 1996, many researchers have presented how their methods can be used to specify and verify security protocols (Millen and Shmatikov, 2001; Long, 2005; Huai and Li, 2004). But their approaches lack support from related verification tools.

The SMV provides a mean of representing the system as a set of states and transitions. It is a complete formal verification tool. It allows for the specification of the required system properties in Computational Tree Logic (CTL) notation and automatically checks these properties against the state machine representation. Because of

Corresponding Author: Jia Mei, School of Computer Engineering and Science, Shanghai University, Shanghai, China

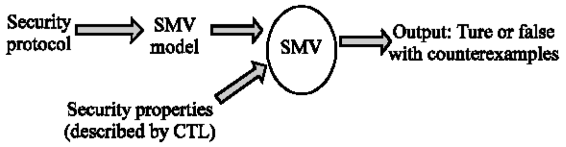
Security protocol
SMV model
SMV
Output: Ture or false with counterexamples
Security properties (described by CTL)

Fig. 1: Verification process of security properties by SMV

above characteristics, SMV is more suitable to describe security protocols and verify their properties than other verification tools.

In this study, we present an approach about modeling and verifying security protocols based on SMV (The verification process is illustrated in Fig. 1). Present approach introduces SMV into the security protocol verification and this approach realizes verifying security protocol automatically.

We use Otway-Rees protocol as an example to illustrate our approach. By using SMV, we discover that the system of the Otway-Rees Protocol does not satisfy the security properties. The SMV outputs a counter-example which is an attack. We analyze the counter-example and adapt the protocol to satisfy the security property.

BASIC ASSUMPTIONS FOR ANALYZING SECURITY PROTOCOLS

First, In order to concentrate on the security of the protocol itself as opposed to the security of the cryptosystem used, the vast majority of research in this area has made the following assumption:

- The decryption key must be known in order to extract the plaintext from the cryptograph
- There is enough redundancy in the cryptosystem that a cryptograph can only be generated using encryption with the appropriate key. This also implies that there are no encryption collisions. If two cryptographs are equal, they must have been generated from the same plaintext using the same key
- Principals are linked together with communication channels to exchange messages. We assume that these communication channels are insecure, that is an intruder can act passively or actively on the transferred information

The intruder can:

- Overhear and/or intercept any messages being passed in the system
- Decrypt messages that are encrypted with his public key so as to learn new nonces
- Introduce new messages into the system, using nonces he knows
- Replay any message he has seen (possibly changing plain-text parts), even if he does not understand the contents of the encrypted part

These assumptions are important because they allow us to abstract away the cryptosystem and analyze the protocols themselves.

MODELING THE SECURITY PROTOCOL FOR SMV VERIFICATION: CASE STUDY

Description of Otway-Rees protocol: As an illustrative example, we use Otway-Rees Symmetric-Key authentication protocol. The principals of this protocol involves two users A and B and a server S whose role is go pass a new session key, K_S, to A and B. Initially, S shares keys K_A and K_B with A and B, respectively. The steps in a successful run of the protocol are as follows. Here, the notation $\{X\}_K$ indicates the string X encrypted using the key K.

- **A sends to B:**

$$M, A, B, \{N_A, M, A, B\}K_A$$

- **B sends to S:**

$$M, A, B, \{N_A, M, A, B\}K_A, \{N_B, M, A, B\}K_B$$

- **S sends to B:**

$$M, \{N_A, K_S\}_{K_A}, \{N_B, K_S\}_{K_B}$$

- **B sends to A:**

$$M, \{N_A, K_S\}_{K_A}$$

The values N_A and N_B are random nonce values chosen by A and B to ensure that their replies from S are new messages and not old ones replayed. The value M is another random nonce chosen by A. It can be seen that A relies on B to relay the messages between A and S. The values A and B are identifiers for A and B. At the end of the protocol it is intended that A and B are both in possession of the shared key K_S and believes it is good for communication with the other. The protocol can also be depicted by state diagram (Fig. 2).

Modeling Otway-Rees protocol by SMV

Data structure for the messages of the protocol: In Otway-Rees protocol, which has four messages, the SMV

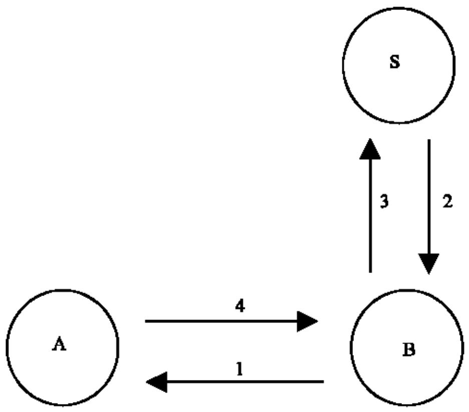

Fig. 2: State diagram of Otway-Rees protocol

characterization of the message format must have enough fields for all of the kinds of information that occur in any of the 4 intended messages. We use a module to define a data structure msg that contains the following fields:

```
MODULE msg
VAR
mtype: {msg1, msg2, msg3, msg4}; Type of message
source: {A, B, S, I}; Source of message
dest: {A, B, S, I}; Intended destination
key: {KA, KB, KS, KI} Encryption key
data [11]: {M, A, B, I, NA, NB, NI, none} Data segments of message
```

The finite state system: We define a finite state system which has different principals in each role. The system configuration is 1 initiator, 1 responder, 1 server and 1 intruder. Initiator A, server S and responder B are honest principal, but intruder I not, it can impersonate initiator, responder and server.

The sets of the system are as follows:

- The set of initiators is {A, I}
- The set of server S is {S, I}
- The set of responders is {B, I}
- The set of public keys is $\{K_A, K_B, K_S, K_I\}$
- The set of nonces is $\{N_A, N_B, N_I\}$

According to the definition of Otway-Rees Protocol, we can find that the protocol has 5 running modes as follows:

- A↔B
  A runs the protocol once with B
  B↔S
  B runs the protocol once with S
  Above mode is the normal running mode of the protocol
- A↔I
  A runs the protocol once with B
  I(A)↔B
  I impersonates A to establish a fake session with B
  I(B)↔S
  I impersonates B to establish a fake session with S
- A↔I
  This running mode involves two simultaneous runs of the protocol. A runs the protocol once with I and I runs the protocol once with B
- A↔I(B)
  A runs the protocol once with I who impersonates B
  I(A)↔B
  I impersonates A to establish a fake session with B
  I(B)↔S
  I impersonates A to establish a fake session with B
- A↔B
  A runs the protocol once with B
  B↔S(I)

B runs the protocol once with I who impersonates S We formulate each principal in the security protocol as a SMV module instance, which are honest principals and intruder:

```
MODULE main
VAR
A: initiator (I.outMA) Initiator A
I. outMA is a input formal parameter
B: responder (I.outMB) Responder B
I. outMB is a input formal parameter
S: server (I.outMS) Server S
I. outMS is a input formal parameter
I: intruder (A.outM, B.outM, S.outM); Intruder I
```

In the SMV program, the input of initiator A, server S and responder B comes only from intruder I and the output of A, B and S passes to I. Thus, we integrate the network with the intruder as one module in the sense that the intruder will control the network and each principal communicates with others via the intruder, i.e., the principals are not interacting directly with each others but indirectly through the intruder module (Fig. 3). In this way, the intruder can overhear, delete, or store each message and generate new messages by using his knowledge of overheard messages.

Figure 4 shows the state transition graphs of initiator A, responder B, Server S and Intruder I. the sets of each principal's states in the protocol are as follows:

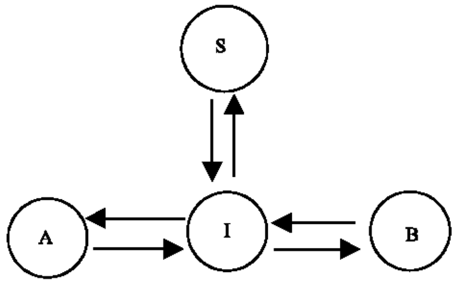

Fig. 3: Principals with intruder

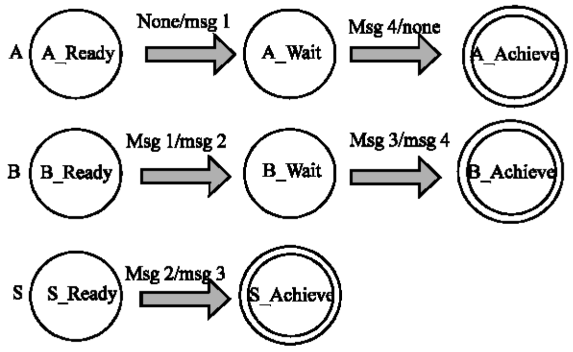

Fig. 4: State transition graphs of initiator A and responder B

- A's states ranges over {A_Ready, A_Wait, A_Achieve}
- B's states ranges over {B_Ready, B_Wait, B_ Achieve}
- S's states ranges over {S_Ready, S_Wait, S_ Achieve}
- I's states ranges over {I_Ready, I_Wait, I_Achieve}

There are 4 steps of runs of the protocol, corresponding to the three states in A's, B's and S's state transition diagram. A begins in A_Ready state and sends msgl to B in B_ Ready state, then requests msg4 from B in A_Wait state, ends in A_Achieved state. B begins in B_Ready state and requests msgl from A in A_Ready state, then sends msg2 to S, ends in B_Achieve state after receiving msg3 from S and sending msg 4 in B_Wait state. S begins in S_Ready state and requests msg 2 from B in B_Ready state, then B sends msg 2 to S, ends in S_Achieve state after receiving msg 2 from B and sending msg3 to B. We use B.rspf to denote the principal who sent the message to B and it is the same as I.rspf and S.rspf.

Intruder I can attack other honest principals by placing follow messages into the network:

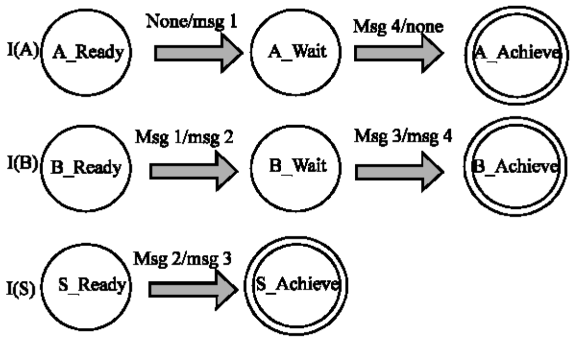

Fig. 5: State transition graphs of intruder I

- Introduce fake messages into the system by the intercepted random nonce values
- Replay intercepted encrypted messages

The state transition graphs of intruder I are more complex. Figure 5 shows the state transition graphs of intruder I which impersonates A, B and S.

Security properties: The security properties of the system of the Otway-Rees Protocol to be verified are the following CTL formula:

```
AG ((A. State = A_Ready) AF ((A. State =A_Achieve) and
(B.rspf = A) and (S.rspf = B) and (A. State =B_Achieve) and
( S.State=S_Achieve)))
```

The meaning of CTL formula is as follows:

- Initiator A runs the protocol once with B and begins in A_Ready state, then the system will eventually end in a state which A is in A_ Achieve state, B is in B_Achieve state and S is inS_Achieve. In the protocol, B believes that the initiator of the protocol is A and S believes that it receives the message from B

VERIFICATION RESULT

The protocol model described by SMV language and the CTL formula for security properties are put into SMV. We find that the Otway-Rees protocol does not satisfy the security properties after running SMV. The SMV outputs a counter-example which is an attack. In this attack, the intruder I impersonates B. After intercepting the message sent by A, I replaces plaintext B to I and then transmits the message which includes $\{N_I, M, A, B\}_{K_I}$ to S. S identifies two communicating parties according to the plaintext and uses K_A and K_I to decrypt the cryptograph. N_A in one decrypted part is

Table 1: Verifying process

Protocol for verification	Tool	Verification result	Adapted parts	Verification result of adapted protocol
Otway-Rees protocol	SMV	The counter-example: the intruder I impersonates B	How to determine the legality of B in msg 2	True: The adapted protocol satisfies the security protocol

consistent with K_A and N_I in the other part is consistent with K_I, so S considers I is legitimate and sends K_S shared by A and B to I. Finally, I gets K_S successfully. The attack process can be described formally as follows:

- **A sends to I(B):**

$$M, A, B, \{N_A, M, A, B\}_{K_A}$$

- **I(B) sends to S:**

$$M, A, I, M, A, I, \{N_A, M, A, B\}_{K_A}, \{N_I, M, A, B\}_{K_I}$$

- **S sends to I(B):**

$$M, \{N_A, K_S\}_{K_A}, \{N_I, K_S\}_{K_I}$$

- **I(B) sends to A:**

$$M, \{N_A, K_S\}_{K_A}$$

By analyzing the counter-example, we can adapt the protocol like this:

After receiving msg 2 which is sent by B, S should decrypt the cryptograph first and compare the identifiers in the plaintext to corresponding identifiers in the cryptograph. The legality of B is determined by the consistency of the comparative result. So, it can avoid above attack.

Other parts of the protocol haven't been changed. The verification result is shown that the adapted protocol is secure (Table 1).

CONCLUSION AND FUTURE WORK

This study, reports the results of a feasibility study on using SMV for multi-party security protocols. With verifying results of Otway-Rees Protocol, we believe we have achieved promising success that SMV can do as well as other tools. Further research is needed to characterize more formally what make a translation from high-level notation into SMV, such as Casper (Lowe, 1997) and CAPSL (Millen and Shmatikov, 2001). And as our experience specifying protocols in SMV grows, we will be able to use SMV to analyze more complex security protocols, including electronic commerce protocols.

With this case study, we find that model checking has the significant advantage over BAN logic in that much of the hard work is done automatically by computer. Perhaps in future we are able to develop a cryptographic protocol design and analysis integrated software so that a perfect security protocol can be designed in a matter of a few hours.

ACKNOWLEDGMENTS

This study is supported by National Natural Science Foundation of China (NSFC) under grant No. 60673115, National High-Technology Research and Development Program (863 Program) of China under grant No. 2007AA01Z144 and the National Grand Basic Research Program (973 Program) of China under grant No. 2007CB310800, the Research Program of Shanghai Education Committee under grant No. 07ZZ06,and Shanghai Leading Academic Discipline Project, Project Number: J50103.

REFERENCES

Abadi, M. and M.R. Tuttle, 1991. A semantics for a logic of authentication. Proceedings of the 10th Annual ACM Symposium On Principles of Distributed Computing, Montreal, Quebec, Canada, Aug. 19-21, ACM, New York, USA., pp: 201-216.

Burrows, M., M. Abadi and R. Needham, 1990. A logic of authentication. ACM. Trans. Comput. Syst., 8: 18-36.

Goldschlag, D.M., 1990. Mechanically verifying concurrent programs with the boyer-moore prove. IEEE Trans. Software Eng., 16: 1005-1023.

Good, D.I., R.L. London and W.W. Bledsoe, 1975. Interactive program verification system. Proceedings of International Conference on Reliab Software, April 21-23, IEEE, Los Angeles, CA., USA., New York, pp: 482-492.

Huai, J. and X. Li, 2004. Algebra model and security analysis for cryptographic protocols. Sci. China, Series F: Info. Sci., 47: 199-220.

Kemmerer, R.A. and S.T. Eckmann, 1985. Unisex: A unix-based symbolic executor for pascal. Software Practice Exp., 15: 439-458.

Li, G., R. Needham and R. Yahalom, 1990. Reasoning about belief in cryptographic protocols. Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy, May 7-9, IEEE, Oakland, CA., Piscataway, NJ., United States, pp: 234-248.

Long, B.W., 2005. Formal verification of a type flaw attack on a security protocol using object-Z. Proceedings of 4th International Conference of B and Z Users, April 13-15, Springer-Verlag, pp: 319-333.

Lowe, G., 1996. Breaking and fixing the needham-schroeder public-key protocol using FDR. Proceedings of the 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Mar. 27-29, Springer Verlag, Passau, Germany, pp: 147-166.

Lowe, G., 1997. Casper: A compiler for the analysis of security protocols. Proceedings of 10th IEEE Computer Security Foundations Workshop, June 10-12, IEEE, Los Alamitos, CA., United States, pp: 18-30.

Millen, J.K. and V. Shmatikov, 2001. Constraint solving for bounded-process cryptographic protocol analysis. Proceedings of 8th ACM Conference on Computer and Communications Security, Nov. 5-8, Association for Computing Machinery, Philadelphia, PA, United States, pp: 166-175.