Formal Analysis of the Kerberos Authentication Protocol with PVS

Guodong Sun, Shenghui Su
College of Computer Science
Beijing University of Technology
Peking, China
sgd-150@163.com

***Abstract*—Formal methods are one of the most important technologies to analyze and design authentication protocols. Kerberos protocol is a famous identity authentication protocol and it is widely used in the network. Although some fruitful formal verifications of the Kerberos protocol have been implemented, there are still little tool-assisted formal analyses. In this paper some important authentication properties of the Kerberos authentication protocol are specified and verified with the help of PVS (Prototype Verification System). According to our analysis, the Kerberos authentication protocol satisfies the mutual authentication between the client and the server, and the client's identity can also be authenticated.**

Keywords-Formal methods; Kerberos; Authentication protocol; PVS

I. INTRODUCTION

Authentication protocols are intended to verify the identity of the communicating principals to one another. Formal methods [1] are proved to be one of the most important technologies to analyze and design the authentication protocols. PVS (Prototype Verification System) [2] is one of the most famous formal verification tools.

Kerberos protocol [3] is a three-party network identity authentication protocol which is used to an open network environment. Although some fruitful formal verifications of the Kerberos protocol have been implemented, there are still little tool-assisted formal analyses. In this paper, we show how the PVS system provides the necessary mechanized support to the analysis of authentication properties of Kerberos protocol.

This paper is organized as follows: in section 2, we briefly introduce the formal verification tool PVS; in section 3, the Kerberos authentication protocol is introduced; in section 4, some important authentication properties of Kerberos protocol are analyzed; finally, concluding remarks are made in Section 5.

II. THE INTRODUCTION OF PVS

PVS is a formal verification tool developed by Stanford Research Institute. It consists of a specification language integrated with support tools and a theorem prover. PVS tries to provide the mechanization needed to apply formal methods both rigorously and productively. The latest version of the system is version 4.2, and it can run on Sun4 workstation using the SunOS or personal computers of the Redhat Linux operating system.

The specification language of PVS [4] is built on higher-order logic. There is a fairly rich set of built-in types and type-constructors, as well as a rather powerful notion of subtype. Besides, theories can be linked by import and export lists. The theorem prover of PVS [5] is both interactive and highly mechanized, the user chooses each step that is to be applied and PVS performs it, displays the result, and then waits for the next command.

III. THE KERBEROS AUTHENTICATION PROTOCOL

In this paper we analyze the authentication property of the Kerberos protocol and the time property is not considered. So we study a reduced version of Kerberos protocol, which can be obtained from the Kerberos description in [6] by leaving out timestamping. The protocol can be described in six steps as follows:

1. $C \to KDC : C, TGS$
2. $KDC \to C : \{K_1\}_{K_C}, \{C, K_1\}_{K_{TGS}}$
3. $C \to TGS : \{C\}_{K_1}, \{C, K_1\}_{K_{TGS}}, S$
4. $TGS \to C : \{K_2\}_{K_1}, \{C, K_2\}_{K_S}$
5. $C \to S : \{C, N_C\}_{K_2}, \{C, K_2\}_{K_S}$
6. $S \to C : \{N_c\}_{K_2}$

Kerberos protocol provides a mutual authentication between C (the client) and S the server). Before C can get access to S, Kerberos checks twice the identity of the client C, first by KDC (Kerberos Distribution Center) and then by TGS (Ticket Granting Server). For authenticating itself to the server, the client needs a ticket issued by the TGS, and to get this ticket, another ticket issued by KDC is needed. The server S can also be authenticated to the client C by responding the right messages.

IV. THE ANALYSIS OF THE KERBEROS AUTHENTICATION PROTOCOL

A. *Model the Protocol*

1) Messages and Encryption

The first step of the analysis is to formalize the basic messages in the protocol. We use the abstract datatype [7] mechanism of PVS to define the message space. The message space is defined as follows:

```
Message: THEORY
  BEGIN
    Text: NONEMPTY_TYPE
    Identity：NONEMPTY_TYPE=nat
    nonce: NONEMPTY_TYPE=nat
   message: DATATYPE WITH SUBTYPES key, nonkey
    BEGIN
     Text(x_text: text)     : text?  : nonkey          % the plaintext
     Name(name: Identity) :Name? :nonkey               % user's name
     Nonce(x_nonce: nonce) :Nonce? :nonkey             % user's nonce
     Symkey(x_sym: Identity): Symkey? : key            % user's long-term symmetric key
     Seskey(x_ses, y_ses: Identity):Seskey? : key      % session key
     Conc(x_conc,y_conc:message):Conc?:nonkey          % concatenation of messages
     Encry(x_enc:key,y_msg:message):Encry?: nonkey  % encryption of messages
    END  message
   END Message
```

Since the Kerberos protocol uses symmetric key encryption, a message encrypted twice is just the original message. We use the following encryption function to perform the normalization of messages.

```
encrypt(k, m) : message =
  CASES m OF
Encry(x, y):
  CASES k OF
     Seskey(a, b): IF x=Seskey(a, b) THEN y ELSE Encry(k, m) ENDIF,
     Symkey(a): IF x=Symkey(a) THEN y ELSE Encry(k,m) ENDIF
    ENDCASES
ELSE Encry(k,m)
  ENDCASES
```

2) The Enemy Process

The enemy may deduce certain information from the messages it already knows, this deductive ability is given by the following message generation relation.

```
Gen(H)(m): INDUCTIVE bool =
    H(m)
OR (EXISTS m1, m2: Gen(H)(m1) AND Gen(H)(m2) AND m=conc(m1, m2))
OR (EXISTS m1: Gen(H)(conc(m1, m)))
OR (EXISTS m2: Gen(H)(conc(m, m2)))
OR (EXISTS m1, k: Gen(H)(m1) AND Gen(H)(k) AND m= encrypt(k, m1))
OR (EXISTS k: Gen(H)(k) AND Gen(H)(encrypt(k, m)))
```

The message generation relation |- can be defined as:

|- (H, m) : bool = Gen(H)(m)

The initial messages of a protocol are publicly known to the enemy. In our specification, the initial messages are given below:

```
INIT: set[message]= {m| Name?(m) OR INIT_Symkey(m) OR INIT_ Seskey(m)}
INIT_Symkey:set[message]={m|EXISTSi:i/=C AND i/= S AND i/= TGS AND m=Symkey(i)}
INIT_Seskey: set[message]={m|EXISTS i, j: (i/=C OR i/=TGS)AND(i/=C OR j/=S) AND m=Seskey(i,j)}
```

Finally, the enemy process can be described as the following process:

Process_enemy: process[event]=Enemy(INIT)

3) User Processes

a) The Client Process

```
Process_C: process[event]=
  Choice! x_Seskey, y_Seskey :
(trans(C, KDC, Conc(C, TGS))>>
(rec(C,KDC,Conc(Encrypt(Symkey(C),x_Seskey),Encrypt(Symkey(TGS),Conc(C,x_Seskey))))>>
(trans(C,TGS,Conc(Conc(Encrypt(x_Seskey,C),Encrypt(Symkey(TGS),Conc(C,x_Seskey))),S))>>
(rec(C,TGS,Conc(Encrypt(x_Seskey,y_Seskey),Encrypt(Symkey(S),Conc(C,y_Seskey))))>>
(trans(C,S,Conc(Encrypt(y_Seskey,Conc(C,N)),Encrypt(Symkey(S),Conc(C,y_Seskey))))>>(rec(C,S,Encrypt(y_Seskey, N)) >>Stop))))))
```

The client process engages in all the six steps of the protocol, first with the KDC, then the TGS, last the server S. The Choice! x_Seskey, y_Seskey means that the client prepares to receive an arbitrary session key from KDC in the second step of the protocol, and another arbitrary session key from TGS in the fourth step of the protocol.

b) The KDC Process

```
Process_KDC: process[event]=
  Choice! i: (rec(KDC,C(i),Conc(C(i),TGS)) >> (trans(KDC,C(i),Conc(Encrypt(Symkey(C(i)),Seskey(C(i),TGS)),Encrypt(Symkey(TGS), Conc(C(i), Seskey(C(i), TGS))))) >> Stop))
```

The process KDC non-deterministically receives a request for the TGS from some client C(i), then the KDC responds to this message by generating a session key for the conversion between C(i) and KDC.

```
Process_TGS:process[event]=
Choice!i,j:(rec(TGS,C(i),Conc(Conc(Encrypt(Seskey(C(i),TGS),C(i)),Encrypt(Symkey(TGS),Conc(C(i), Seskey(C(i), TGS)))),S(j)))>>(trans(TGS,C(i),Conc(Encrypt(Seskey(C(i), TGS),Seskey(C(i),S(j))),Encrypt(Symkey(S(j)),Conc(C(i), Seskey(C(i), S(j)))))) >> Stop))
```

The process TGS non-deterministically receives message consisting of a ticket, an authenticator and a request for some server S(j) from some client C(i). It responses this message by generating a new session key for the conversation between the client C(i) and the sever S(j). Also for a certain protocol run, C(i) and S(j) are certain.

c) The Sever Process

Process_S: process[event]=

Choice!i,Nx:(rec(S,C(i), Conc(Encrypt(Seskey(C(i),Conc(C(i),Nx))),Encrypt(Symkey (S),Conc(C(i),Seskey(C,S)))))>>(trans(S,C(i), Encrypt(Seskey(C(i),S), Nx)) >>Stop))

The server process non-deterministically receives message from some client C(i), and responses the client with the nonce encrypted with the session key in the first message.

B. Some Authentication Properties

1) The Client authenticate to the TGS

This authentication property is specified as the auth(T1, R1), in which T1 and R1 are the following sets of messages:

T1:set[event]={e|e=rec(TGS,C,Conc(Conc(Encry(Seskey(C,TGS),C),Encrypt(Symkey(TGS), Conc(C, Seskey(C, TGS)))), S))}

R1:set[event]={e|EXISTSi:e=trans(C,TGS,Conc(Conc(Encrypt(Seskey(C,TGS),C),Encrypt (Symky(TGS), Conc(C, Seskey(C, TGS)))), S(i)))}

A rank function [8] is a mapping from the protocol's message space to the set of integers. The rank function for this property is given below:

rank(m): RECURSIVE int=
CASES m OF
Name(z) : 1,
Nonce(z) : 1,
Symkey(z) : IF Symkey(z)=Symkey(C) OR Symkey(z)=Symkey(TGS) OR Symkey(z)=Symkey(S) THEN 0 ELSE 1 EDNIF,
Seskey(z1,z2) : IF Seskey(z1,z2)=Seskey(C, TGS) OR Seskey(z1, z2)=Seskey(C,S) THEN 0 ELSE 1 EDNIF,
Conc(z1, z2) : min(rank(z1), rank(z2)),
Encry(k, z) : rank_code(k, z, rank(z))
ENDCASES
MEASURE size(m)
rank_code(k, m, n) : int =
CASE k OF
Seskey(z1, z2): IF Seskey(z1, z2)=Seskey(C, S) AND m=Name(C) THEN 0 ELSIF Seskey(z1, z2)=Seskey(C, TGS) THEN n+1 ELSE n ENDIF,
Symkey(z): IF z=C OR z=TGS OR z=S THEN n+1 ELSE n ENDIF
ENDCASES

2) The Client authenticate to the TGS

The authentication property for the client c to the server S is given as auth(T2, R2), the messages T2 and R2 are given below.

T2:set[event]={e|e=rec(S,C,Conc(Encry(Seskey(C,S),Conc(C,N)),Encrypt(Symkey(S),Conc(C, Seskey(C, S)))))}

R2:set[event]={e|e=trans(C,S,Conc(Encry(Seskey(C,S),Conc(C,N)),Encrypt(Symkey(S),Conc(C, Seskey(C, S)))))}

This property means that if the server S receives a message concatenating, the client C's identity and a nonce encrypted by a session key, with a message encrypted with S's long-term key, and containing C's identity and this session key, it can be sure that that message originated from the client C.

3) A Failed Authentication

In the above authentication, we prove that when a server S receives a certain type of message, it can be sure that this message generated from one certain client C, but we cannot prove that this type of messages authenticate that C asked for a ticket for the certain server S from TGS. In this verification, the message T is the same as above, but the message R is the following message.

R3:set[event]={e|e=trans(C,TGS,Conc(Conc(Encry(Seskey(C,TGS),A),Encry(Symkey(TGS), Conc(A, Seskey(C, TGS)))), S))}

During our exercise of finding a proper rank function for this property, we find an attack to the protocol. This corresponds to the following attack:

$$3. C \to I(TGS): \{C\}_{K_1}, \{C,K_1\}_{K_{TGS}}, S$$

$$I(C) \to TGS: \{C\}_{K_1}, \{C,K_1\}_{K_{TGS}}, S''$$

$$4. TGS \to C: \{K_2\}_{K_1}, \{C,K_2\}_{K_{S''}}$$

$$5. C \to I(S): \{C,N_c\}_{K_2}, \{C,K_2\}_{K_{S''}}$$

$$I(C) \to S'': \{C,N_c\}_{K_2}, \{C,K_2\}_{K_{S''}}$$

After we fix the third step of the protocol by including the server name into the authenticator, this property is successfully proved by using the rank function defined in the above verification. The modification is given below.

$$3. C \to TGS: \{C,S\}_{K_1}, \{C,K_1\}_{K_{TGS}}$$

4) The Server authenticate to the Client

The messages T4 and R4 for this authentication property are given below:

T4: set[event]={e | e= rec(C, S, Encrypt(Seskey(C,S), N))}

R4: set[event]={e | e= trans(S, C, Encrypt(Seskey(C,S), N))}

This means that if the client C receives the message containing the previous nonce it generated encrypted with the session key between C and S, it can be sure this message originates from the server S.

The rank function for this property is given below:

rank(m): RECURSIVE int=
CASES m OF
Name(z) : 1,
Nonce(z) : IF Nonce(z)=Nc THEN 0 ELSE 1 ENDIF,
Symkey(z) : IF Symkey(z)=Symkey(C) OR Symkey(z)=Symkey(TGS) OR Symkey(z)=Symkey(S) THEN 0 ELSE 1 EDNIF,
Seskey(z1,z2) : IF Seskey(z1,z2)=Seskey(C, TGS) OR Seskey(z1, z2)=Seskey(C,S) THEN 0 ELSE 1 EDNIF,
Conc(z1, z2) : min(rank(z1), rank(z2)),
Encry(k, z) : rank_code(k, z, rank(z))
ENDCASES
MEASURE size(m)
rank_code(k, m, n) : int =
CASE k OF

Seskey(z1, z2): IF Seskey(z1, z2)=Seskey(C, S) AND m=Nc THEN 0 ELSIF Seskey(z1, z2)=Seskey(C, TGS) THEN n+1 ELSE n ENDIF,
Symkey(z): IF z=C OR z=TGS OR z=S THEN n+1 ELSE n ENDIF
ENDCASES

C. The Main Theorem for Authentication

The authentication property means the occurrence of a message set T guarantees the occurrence of the previous message set R. In order to prove the authentication property, we must verify the following main authentication theorem.

```
Authentication_by_rank: THEOREM
      Positive(rank, INIT)
    AND (FORALL  S, m :
Positive(rank, S) AND (S=> m) IMPLIES rank(m)>0)
       AND non_positive(rank, T)
       AND (FORALL i: user(i) # R |> RankUser(rank))
     IMPLIES
        Network(enemy(INIT), user) |> auth(T, R)
```

In order to verify the above theorem, we must firstly verify the following four lemmas, which are used as rewriting rules for the theorem.

- rank_init: LEMMA Positive(rho, INIT)
- non_positive_rankT: LEMMA non_positive(rho, T)
- validity_rho: LEMMA Positive(rank, S) AND (S=> m) IMPLIES rank(m)>0)
- user_rank: LEMMA FORALL i: user(i) # R |> RankUser(rank)

The lemmas rank_init and non_positive_rankT can be easily proved by the rank function. Lemma validity_rho relies on a property which following easily from a induction theorem for messages. The fourth lemma requires us to verity all user processes of the protocol satisfy this property.

D. Mechanical Verification

The PVS theorem prover is an automatic mechanical theorem prover which consists of a powerful proving command set. The mechanical verification of the three authentication properties of Kerberos follow the same routine. For example, the verification of the first authentication property is mainly to prove the following PVS lemmas:

- Rank_init: LEMMA Positive(rho, INIT)
- Non_positive_rankT1: LEMMA non_positive(rho, T1)
- Rank_enemy: LEMMA enemy(INIT) |> RankEnemy(rho)
- Rank_user_C: LEMMA Process_C # R1 |> RankUser(rank)
- Rank_user_KDC: LEMMA Process_KDC # R1 |> RankUser(rank)
- Rank user_TGS: LEMMA Process_TGS# R1|> RankUser(rank))
- Rank_user_S: LEMMA Process_S# R1 |> RankUser(rank))

The lemma Rank_init and Non_positive_rankT1 can be easily proved by the rank function defined for this authentication property. The proof of other lemmas consists mainly of PVS macro steps develop specifically for authentication protocols in [9], here is the proof of Rank_user_KDC:

```
(init-csp “Identity” “message”)
 (expand “Process_KDC”)
   (choice3)
   (prefix)
   (prefix)
   ((“1”(delete 2 3) (grind))
(“2”(prefix)))
```

After proving all the lemmas defined above, finally, as an immediate application of these lemmas, we can prove the expected property:

```
Authentication_by_rank:PROPOSITION
Network(enemy(INIT),protocol(C,KDC,TGS,S,Process_C,Process_KDC,Process_TGS, Process_S,))
 |> auth(T1, R1)
```

V. CONCLUSION

Authentication property is a vital property of cryptographic protocols. Because there are still little tool-assisted formal analyses of the Kerberos protocol, with the help of PVS, some authentication properties of a reduced version of Kerberos protocol is analyzed in this paper. Through the formal analysis, we can prove that the Kerberos protocol satisfies the mutual authentication between the client C and the Server S, and the client C can also be authenticated to the TGS. Besides, an attack is given during our analysis of another authentication property. We avoid this attack by fixing the third step of the protocol. Finally, we can conclude that PVS is powerful enough to be used in analysis and design of authentication protocols.

ACKNOWLEDGMENT

The author would like to thank the Professor Shenghui Su and other teachers and students for their important advice and suggestions.

REFERENCES

[1] R.M. Needham and M.D. Schroeder, “Using Encryption for Authentication in Large Networks of Computers”, Commun. ACM 21, 993-999(1978).

[2] J.Crow, S.Owre, J.Rushby, N.Shankar, and M.Srivas, “A tutorial Introduction to PVS,”available: http://www.csl.sri.com/papers/wift-tutorial/wift-tutorial.pdf, June 1995.

[3] J. Kohl, B. Neuman, and T. Ts’o, “The evolution of the Kerberos authentication service, pages 78-94”, IEEE Computer Society Press, 1994.

[4] S.Owre, N.Shankar, and J. M. Rushby. “The PVS Specification Language”, Computer Science Lab, SRI International, April 1993.

[5] N.Shankar, S.Owre, and J.M. Rushby, “The PVS Proof Checker: A reference Manual”, Computer Science Lab, SRI International, March 1993.

[6] J.Kohl and B.Neuman. The Kerberos network authentication service(version 5). Internet Request For Comment RFC-1510, September 1993.

[7] S. Owre and N. Shankar. Abstract Datatypes in PVS. Computer Science Laboratory, SRI International, Technical Report CSL-93-9R, December 1993, Substantially Revised June 1997.

[8] Steve Schnerder and Royal Holloway, Using CSP for protocol analysis: the Needham-Schroeder Public-Key Protocol, University of London, Technical Report, CSD-TR-96-14, Novemver 21, 1996.

[9] B.Dutertre, S.A. Schneider, Embedding CSP in PVS. An application to authentication protocol, in: TPHOL'97, LNCS, vol. 1275,Springer-Verlag,1997.