Using the NuSMV Model Checker to verify the Kerberos Protocol

M. Panti L. Spalazzi S. Tacconi
Istituto di Informatica, University of Ancona,
Via Brecce Bianche, 60131 Ancona, Italy
{panti,spalazzi,tacconi}@inform.unian.it

Keywords : Collaborative Enterprise Security, Modeling and Simulation of Collaboration, Requirements Engineering in CE.

Abstract

The aim of this paper is to present a methodology for verifying cryptographic protocols by means of NuSMV, a symbolic model checker. We illustrate this approach by describing our analysis of the basic version of Kerberos, a widely used authentication protocol. The most innovative feature of our methodology is constituted by formal representation of security requirements. Indeed, we propose an extension of correspondence property, so far used only for authentication, to other requirements, as secrecy and integrity. This generalization leads to a unifying view of basic security requirements. The adequacy of our analysis is proved by finding an unpublished attack on Kerberos that may lead to serious consequences on the security of the systems that adopt it.

1 Introduction

In the last years, the diffusion of computer networks has received an enormous development. These systems provide to users a growing number of different services, each needy of peculiar security requirements. Authentication protocols are security mechanisms whereby each party involved in a transaction over a computer network can be assured of another's identity. Some of these protocols, in order to guarantee the privacy of subsequent communication, provide also key-exchange, namely the agreement of a secret session key between parties. In the field of collaborative enterprise, such a kind of protocols allows to guarantee security of operations. Various security protocols have been proposed and implemented in the real systems, but unfortunately many of them have been shown to have flaws, even a long time after they were published. In order to avoid the subtle misconceptions arising in the design of protocols, it has been devised several formal verification methods for analyzing them. Among the various methodologies proposed, model checking has been proved to be very useful for this purpose. Model checking is a general-purpose, automatic technique for verifying finite-state concurrent systems. INDEED, GIVEN THE MODEL OF THE SYSTEM TO ANALYZE, A MODEL CHECKER SIMULATE ALL ITS POSSIBLE BEHAVIORS IN ORDER TO VERIFY WHETHER A REQUIREMENT IS SATISFIED. In applying this approach to verification of cryptographic protocols, security requirements (properties) are expressed with a propositional temporal logic, and protocols are modelled as state-transition systems. In this context, model checkers' ability of generating the counter-examples for properties that are not true, allows the automatic generation of possible attacks on protocols, when the model of environment includes the presence of an attacker having the control of network where protocols run. The aim of this paper is to present a methodology for analyzing cryptographic protocols by means of NuSMV [5], a symbolic model checker. The most innovative feature of our methodology is constituted by formal representation of security requirements. Indeed, we propose an extension of correspondence property, so far used only for authentication, to other requirements, as secrecy and integrity. We illustrate the feasibility of our approach using the basic version of Kerberos protocol [8, 11, 13] as case-study. Kerberos is one of most broadly used authentication protocols but, as pointed out by some authors [3, 14], it suffers from some weaknesses. In our analy-

sis we have found an unpublished attack that exploits one of such weaknesses. This attack is possible since this protocol, under certain environmental conditions, is not as resistant to replay attacks as it should be.
The rest of paper is organised as follows. In section 2 we sketch a brief description of Kerberos. At this point, we describe how we have employed the model checking to verify the security of this protocol. At this purpose, we first describe the model of protocol in section 3, then we illustrate the formalization of security requirements in section 4. Finally, in section 5, we discuss the attack that we have found and its effectiveness.

2 Overview of the Kerberos protocol

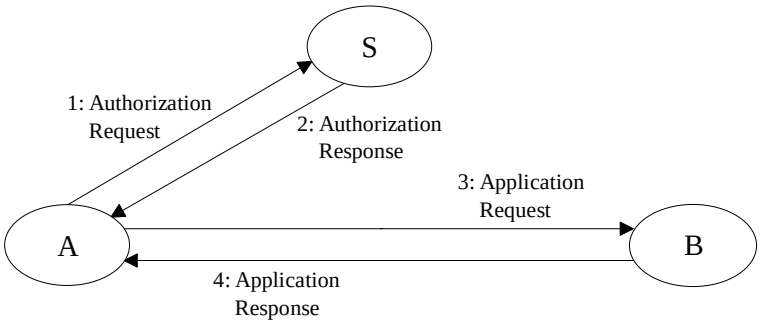
The Kerberos protocol was developed as part of Project Athena at MIT [11]. The aim of this protocol is to guarantee authentication and key-exchange between a client and a server. It is a variant of the Needham-Schroeder protocol [12], but it employs timestamps as nonces to remove flaws demonstrated in [6] and reduce the total number of required messages. Like the Needham-Schroeder protocol, the Kerberos protocol relies on symmetric-key cryptography. In literature [1, 4, 3, 8, 11, 13, 14] can be found several specifications for this protocol. In this paper, we adopt a formulation of the basic Kerberos authentication protocol similar to those described in [1, 4, 14]. The messages of this protocol, are showed in Figure 1 USING BOTH A DIAGRAMMATIC AND A FORMAL REPRESENTATION.
The protocol involves the principals A and B, the client (also said initiator) and the server (also said responder) respectively, and an authentication server S. This server is a trusted party that shares a key K_{as} with A, a key K_{bs} with B, and is devoted to generation of new session keys K_{ab}. The protocol makes use of the timestamps T_a and T_b, and the lifetime L. In step (1), the client A contacts the authentication server S in order to communicate its claimed identity and the name of the server B. In step (2), S sends to A two encrypted components. The first one contains the session key K_{ab} provided by S, a timestamp T_s specifying when the session key has been generated, the interval of validity for such a key, and the name of server B. The second component of message is called *ticket* and has similar information, but it can not be decrypted by the client A. In step (3), the client A forwards this ticket to server B, with a so called *authenticator*, a component encrypted with the new session key, having the name of client A, and its local time T_a. When the server B receives above message, extracts the session key from the ticket, and uses it to decrypt the authenticator. If the key used to encrypt authenticator matches with the key contained in the ticket, the server B can assume that the authenticator was generated by A. At this point, in order to authenticate the client A, the server B must also check the timestamp T_a to make sure that the authenticator is recent. If the result of verification is positive, the server B can deduce the identity of client A. Finally, in step (4), the server B demonstrates its identity to client A, sending it a message with the timestamp T_a extracted from authenticator, increased and encrypted with the session key K_{ab}.

3 Model of Protocol

Our methodology of analysis is based upon model checking, an automatic technique for verifying finite state concurrent systems. In the temporal logic model checking, the concurrent system is modelled as a state transition diagram, whereas properties are expressed in a propositional temporal logic. The NuSMV model checker, following the above approach, adopts a structured input language to describe the model of system and the temporal logic CTL [7] to express desired properties. THE MODEL OF PROTOCOL HAS A MODULAR STRUCTURE. EACH MODULE IS ASSOCIATED TO AN ENTITY OF THE SYSTEM AND DESCRIBES ITS BEHAVIOR. AS SHOWED IN THE OVERALL VIEW OF THE MODEL, DEPICTED IN FIGURE 2, THESE ENTITIES ARE THE HONEST PRINCIPALS A,B,S AND AND THE ATTACKER I.

3.1 Honest Principals

The model of Kerberos protocol is composed by several variables and processes. First of all, we have the communication channel, a set of variables which are shared among all the principals. In particular, we treat protocol messages flowing in the channel as records having a field for each message component. For each principal A,B and for the authentication server S, we have a set of processes. Each process of any principal corresponds to an instance of the principal involved in a particular execution of the protocol. Moreover, all the processes work asynchronously and concurrently. The concurrency is simulated by



(1) $A \rightarrow S : A, B$
(2) $S \rightarrow A : \{T_s, L, K_{ab}, B\}_{K_{as}}, \{T_s, L, K_{ab}, A\}_{K_{bs}}$
(3) $A \rightarrow B : \{T_s, L, K_{ab}, A\}_{K_{bs}}, \{A, T_a\}_{K_{ab}}$
(4) $B \rightarrow A : \{T_a + 1\}_{K_{ab}}$

Figure 1: The basic Kerberos authentication protocol

means of a non-deterministic interleaving of process running. Each process that constitutes a principal is a very simple automaton. *state* is the most important variable involved in these processes and it is initialized at *idle*. When the automaton of a principal sends a message h, the variable is set to $send_h$ and when it receives a message k, the variable is set to $receive_k$, when the protocol session ends successfully, the value is *authenticated*. Each principal has also a set of counters that are used to specify the security properties, as discussed in section 4. Such variables are the following:

- $X.begin\_init\_Y$: counts the number of protocol executions the principal X has begun as client (initiator) with principal Y.
- $X.end\_init\_Y$: counts the number of protocol executions the principal X has ended as client (initiator) with principal Y.
- $X.begin\_res\_Y$: counts the number of protocol executions the principal X has begun as server (responder) with principal Y.
- $X.end\_res\_Y$: counts the number of protocol executions the principal X has ended as server (responder) with principal Y.
- $X.send\_K_{ab}\_Y$: counts how many times the principal X sends the key K_{ab} to principal Y.
- $X.receive\_K_{ab}\_Y$: counts how many times the principal X receives the key K_{ab} from principal Y.

The value of each counter can range from 0 to the maximum number of admitted sessions. Notice that the counter $X.receive\_K_{ab}\_Y$ is incremented only if X is able to extract the key from the message. In the model of Kerberos, each meta-variable X and Y is instatiated with A and B.

3.2 Attacker

Modelling the attacker requires identification of relevant assumptions that allow to capture its possible behaviors. An attacker is usually assumed to be as powerful as possible, in order to verify the protocol in the possible worst situation where it can be executed. Therefore, we assume that an attacker has a complete control of the communication channel. In other words, it must be able:

- to *eavesdrop* each message sent over the channel;
- to *remove* a message sent to another principal;
- to *generate* new protocol messages;
- to *store* in the *memory* messages eavesdropped/deduced;
- to *send* messages to other principals.

Relevant messages that an attacker must be able to generate are those that can be built out of messages previously sent over the channel by honest principals. As a consequence, the model of attacker must be provided with a memory to store intercepted messages sent during the current or previous sessions of the

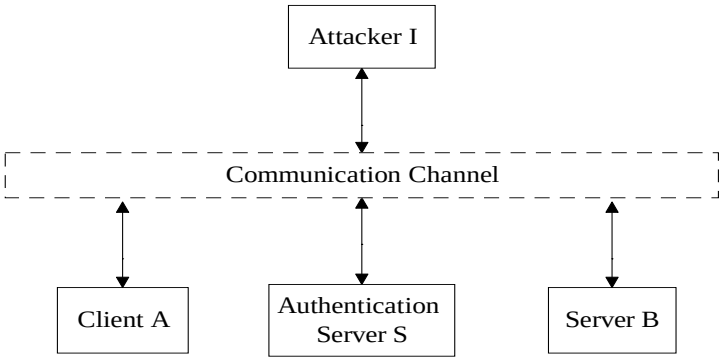


Figure 2: The overall view of the model

protocol. In the general case, an attacker must be able to build new messages starting from the readable components of the messages it has intercepted or from components belonging to its initial knowledge about protocol. The model of the attacker is made according to above assumption. The most important variables involved are the following:

- *state* ranges over the possible actions that the attacker can perform and whose value in each state represents the action it has chosen. According to attacker's abilities previously enumerated, *state* admits the following values: *eavesdrop*, *remove*, *generate*, *store*, *send*.

- *memory* represents the memory of the attacker and is an array whose elements range over the messages of the protocol. The attacker *store* messages in this array. The size of the array basically depends on the number of messages of the protocol and the maximum number of distinct sessions allowed in the model.

At any step, the attacker non deterministically chooses an action between all the possible actions, for this reason *state* can non deterministically take any of its values. The automaton of attacker performs the current action specified by the value of *state* and changes all the other local variables accordingly. Finally, in order to formalize security requirements, the model of attacker is also provided with counters similar to those of honest principals.

4 Security Requirements

There are several requirements that a security protocol needs to satisfy. In the analysis of Kerberos, we have focused our attention on: *authentication*, i.e. the parties convince each other of their identities, *secrecy*, i.e. nobody but the authorized parties must know a set of terms (typically, these terms are keys), and *integrity*, i.e. assurance that an information has not been altered by unauthorized entity. These properties express security goals that Kerberos aims to achieve. In order to formalize above requirements, we propose to adopt the so called *correspondence property*. This property has been introduced [15] for authentication. An attempt to generalize correspondence can be found in [10], but the authors still apply this property to authentication only. Now we explain how it can also encompass properties as secrecy and integrity.

4.1 Generalization of Correspondence Property

In the most general formulation, the *correspondence property* states that the relation between an *event* E and an *event* F must be a one to one mapping. Usually, we need to take into account multiple sessions of the same protocol in order to verify security. Indeed, many attacks occur only in the presence of multiple sessions. In such a situation, we may have several events of type E and F, and thus we need to count them. Therefore, the correspondence property can be reformulated as follows:

> If the event E occurs n times, then the event F must have occurred at least n times.

When we represent the generalized correspondence property in CTL, we use the counters (one for each event) introduced in the model. Therefore, the correspondence property can be formalized by means of

the following general schema:

$$AG(X.counter_F >= Y.counter_E) \quad (1)$$

$X.counter_E$ and $Y.counter_F$ denote counters of process corresponding to principal X and Y. These meta-variables must be instantiated to appropriate counters, depending on the property to formalize. AG is a CTL operator (it is an abbreviation for *Always Globally*). Informally speaking, $AG\ p$ means that the propositional formula p must be true in every possible state of the system.

4.2 Authentication

In the original formulation for authentication, the correspondence property states that when a principal finishes its part of the protocol, the other principal must have taken part in the protocol execution. In other words, if a principal X ends n protocol executions with Y as responder (initiator), then in the past the principal Y must have started at least n protocol executions with X as initiator (responder). The above property asserts that Y must prove its identity to X. When we have a mutual authentication protocol, there exists a similar property for representing X that must prove its identity to Y. A violation of this property means that a principal ends a session (i.e., it accepts the authentication of the other principal) that the other principal have not started, i.e., the attacker must have impersonated a principal. In order to formalize in CTL authentication property according on above schema, we use some of counters that we have introduced in the model. Thus, the authentication property is represented by following CTL formulas:

$$AG(A.begin\_init\_B >= B.end\_res\_A) \quad (2)$$

$$AG(A.begin\_res\_B >= B.end\_init\_A) \quad (3)$$

4.3 Secrecy & Integrity

The extension of correspondence property from authentication to other requirements is quite natural. For what concerns secrecy, we restrict our attention to session keys exchanged between the client and the server. For different kinds of components it is possible to proceed in similar way. Intuitively, secrecy means that an information can not be known by unauthorized entity. In other words, it is required that the attacker never receives this key. From another point of view, an attacker can receive only the keys intended for it. Therefore, an attacker can not obtain a key K_{ab} from a principal X more times than this key has been expressly sent to it. Since in the protocol the parties devoted to key sending are the authentication server S (in step (2)) and client A (in step (3)), we write following formulas:

$$AG(A.send\_K_{ab}\_I >= I.receive\_K_{ab}\_A) \quad (4)$$

$$AG(S.send\_K_{ab}\_I >= I.receive\_K_{ab}\_S) \quad (5)$$

Obviously, these formulas assume that the attacker can be a legitimate member of the system. Notice that when I is not a legitimate member of the system, nobody sends keys to I and thus $ag\_A.send\_K_{ab}\_I$ and $ag\_S.send\_K_{ab}\_I$ are always equal to 0. As a consequence, in this case the specification requires that $ag\_I.receive\_K_{ab}\_A$ and $ag\_I.receive\_K_{ab}\_S$ must be 0. Nevertheless, the above conditions are necessary but non sufficient. Indeed, an attacker, even if does not know the key K, can trick a principal X into thinking that it shares a key K with the principal Y, whereas in fact Y has not received this key. In order to include also this situation, above expression must be accompanied with the following:

$$AG(S.send\_K_{ab}\_A >= A.receive\_K_{ab}\_S) \quad (6)$$

$$AG(A.send\_K_{ab}\_B >= B.receive\_K_{ab}\_A) \quad (7)$$

Informally speaking, above expression means that if a principal X receives n times a key from Y, then in the past the principal Y must have sent at least n times this key. In Kerberos, secrecy and integrity of the session key guarantee also integrity of messages exchanged between parties after the protocol execution, since the combination of encryption and checksum provides also integrity for encrypted messages. Above formalization expresses also the integrity of the session key. Indeed, the correspondence property guarantee that a key is not altered, since if an alteration of a session key occurs on the network during its transit, the associated counter of its recipient does not increase and thus at least one of above properties does non hold.

5 The Attack on Kerberos

We have submitted to model checker $NuSMV$ the above model and requirements. From the verification arises that the expected security requirements do not hold. In order to justify this outcome, the model checker has generated a counter-example showing how

$(\alpha.1)\ A \rightarrow S\ :\ A, B$
$(\alpha.2)\ S \rightarrow A\ :\ \{T_s, L, K_{ab}, B\}_{K_{as}}, \{T_s, L, K_{ab}, A\}_{K_{bs}}$
$(\alpha.3)\ A \rightarrow I, B\ :\ \{T_s, L, K_{ab}, A\}_{K_{bs}}, \{A, T_a\}_{K_{ab}}$
$(\beta.3)\ I(A) \rightarrow B\ :\ \{T_s, L, K_{ab}, A\}_{K_{bs}}, \{A, T_a\}_{K_{ab}}$
$(\alpha.4)\ B \rightarrow A\ :\ \{T_a + 1\}_{K_{ab}}$
$(\beta.4)\ B \rightarrow I(A)\ :\ \{T_a + 1\}_{K_{ab}}$

Figure 3: The attack on Kerberos authentication protocol

an attacker can compromise an execution of the Kerberos protocol. The attack found, reported in Figure 3, is a counter-example of the properties (2), (7). In the attack trace, we denote the messages of different protocol sessions with Greek letters α and β. Moreover, when the attacker I eavesdrops and removes a message m in transit from X to Y, we write $X \rightarrow I(Y)\ :\ m$; when the message m is only eavesdropped, we write $X \rightarrow I, Y\ :\ m$; finally, when the attacker sends a message m to Y, impersonating X, we write $I(X) \rightarrow Y\ :\ m$. The first session proceeds normally until Step $(\alpha.3)$, when the attacker I eavesdrops and stores the message, in order to immediately replay it to server B, while session α is still running. Since the protocol requires that B is not contacted before the third step, the replayed message is interpreted by the server B as the begin of a new session β. For this reason B responds to both requests, sending the message responses of steps $(\alpha.4)$ and $(\beta.4)$. Alternatively, if the environment does not allows multiple executions, the attacker can replay the message $(\alpha.3)$ after the ending of session α, obtaining the same effect. As a consequence of this attack, the attacker is able to trick the server B into thinking that the client A has accomplished two protocol executions. This attack belongs to a family of attack that Lowe [9] denominates 'multiplicity attacks'. In general, this kind of attacks causes a principal Y to think that another principal X is attempting to set up two or more simultaneous sessions with it, when in fact X is trying to establish only one session. These attacks may lead to serious consequences since the attacker can eavesdrop messages sent in the genuine session and then replay them in the fake session. However, in this situation adversary can not decrypt these messages. In order to understand the real consequences of the above attack, let us suppose that the Kerberos protocol is used to protect an online commercial transaction between a customer A and a merchant B. In this scenario, after the regular purchase of good during the genuine session, an attacker that desires to damage A may cause the payment of good again in the fake session, by replaying the payment authorization message sent by A to B. The attack found is due to a weakness of the Kerberos protocol presented by Bellovin and Merritt in [3]. In particular, they assert that the use of authenticators with timestamps is very problematic. Indeed, this solution requires that the server B stores all live authenticators in order to detect an attempt of replay. Unfortunately, in many environments, as UNIX systems, servers typically act by forking a separate process to handle each received request. Since child processes do not share memory among them, there is not a simple way to inform one of these processes about the identifiers received by others. Possible solutions for inter-process communication may be very expensive. Therefore, this feature is rarely implemented and thus the presented replay attack can really occurs. Obviously, if we provide the model checker with a model of protocol where the control of freshness of third message is present, the above attack does not arise from verification.

6 Related Work

The literature about formal verification of cryptographic protocols is very wide. We restrict our brief survey on attempts addressed towards the Kerberos protocol. The first work aimed to formal analysis of Kerberos is [4]. It is probably the most influential paper in the field of formal verification of cryptographic protocols. The authors provide a logic (universally said *BAN logic*) to describe the beliefs of parties involved in a protocol. Another work about analysis of Kerberos is [2], where authors show how they have analyzed the protocol by means of the verification tool *Isabelle*. Both these works, based on *theorem proving*, are useful to evaluate the *trustness* of protocols, since they allow to detect design weaknesses. A different kind of approach, followed by us, is that one

based on *model checking*. By applying this technique in the analysis of cryptographic protocols, it is possible to directly find attacks as counter-examples of security properties that do not hold. An attempt to use this approach to verify Kerberos is [10]. In this work, it is used a model checker named *Murphy*. Our work differs from the above BOTH IN THE FOLLOWED METHODOLOGY AND IN THE OBTAINED RESULTS. FOR WHAT CONCERNS THE VERIFICATION METHODOLOGY, IN FORMALIZING THE SECURITY REQUIREMENTS TO VERIFY WE HAVE PROPOSED AN EXTENSION OF *correspondence property*, SO FAR USED ONLY FOR AUTHENTICATION, TO OTHER REQUIREMENTS, AS SECRECY AND INTEGRITY. WHEREBY THIS APPROACH, WE HAVE OBTAINED A REPRESENTATION OF SECURITY REQUIREMENTS MORE SIMPLE AND UNIFORM. FINALLY, FOR WHAT CONCERNS THE RESULTS, OUR ANALYSIS HAS DETECTED A DIFFERENT ATTACK ON KERBEROS, SO FAR UNPUBLISHED.

7 Conclusion

In this paper, we have illustrated an automatic verification of the basic version of the Kerberos protocol by employing the NuSMV model checker. In particular, we have shown how to extend the correspondence property in order to obtain a unifying formalization of all its security requirements (authentication, secrecy, and integrity). From our analysis it arises an unpublished attack that allows an attacker to impersonate the initiator (client) of the protocol. We have also explained what are the environmental conditions that allow the attack to occur.

References

[1] M. Abadi and R. Needham. Prudent Engineering Practice for Cryptographic Protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[2] G. Bella and L. Paulson. Using Isabelle to Prove Properties of the Kerberos Authentication System. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[3] S.M. Bellovin and M. Merritt. Limitations of the Kerberos Authentication System. *ACM Computer Communication Review*, 20(5):119–132, 1990.

[4] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactionson Computer Systems*, 8(1):18–36, 1990.

[5] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A New Symbolic Model Verifier. In *Proceedings of* 11^{th} *Conference on Computer-Aided Verification*, number 1633 in Lecture Notes in Computer Science, pages 495–499. Springer, 1999.

[6] D.E. Denning and G.M. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[7] A. Emerson. *Handbook of Teoretical Computer Science*, volume 2. Elsevier, 1990.

[8] J.T. Kohl and B.C. Neuman. The Kerberos Network Authentication Service. Internet RFC 1510, 1993.

[9] G. Lowe. A Family of Attacks upon Authentication Protocols. Technical report, University of Leicester, 1997.

[10] W. Marrero, E. Clarke, and S. Jha. Model Checking for Security Protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[11] S.P. Miller, B.C. Neumann, J.I. Schiller, and J.H. Saltzer. Kerberos Authentication and Authorization System. Project Athena Technical Plan, Section E.2.1 - MIT, 1987.

[12] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.

[13] B.C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, 32(9):33–38, 1994.

[14] B. Schneier. *Applied Cryptography : Protocols, Algorithms and Source Code in C*. John Wiley & Sons, 2^{nd} edition, 1996.

[15] T. Woo and S. Lam. A Semantic Model for Authentication Protocols. In *Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy*, volume 25, pages 178–194, 1993.