Verification of Authentication Protocols using SDL-method

Javier López, Juan J. Ortega, José M. Troya

Computer Science Department, E.T.S. Ingeniería Informática
University of Malaga, 29071 - Malaga - SPAIN
e-mail: {jlm, juanjose, troya}@lcc.uma.es

Abstract. Authentication between protocol agents is widely studied in the cryptographic protocol analysis area. It is essential in a virtual environment to rely on protocol parties' identity. In the academic literature there are many protocols that provide the authentication property. We present in this paper a new mechanism to verify authentication using SDL, general purpose specification language. We have defined a generic schema in SDL that allow us to specify a security system and check system behavior when a malicious agent (the intruder) is present. We have used the EKE authentication protocol to illustrate who the mechanism works.

Keywords: *Formal Techniques, SDL, Protocol Analysis, Authentication*

1. Introduction

A variety of cryptographic protocols are used to gain access to computer systems and to protect communications over Internet. Most of security systems have handshake or initialization protocols that are often used to exchange a secret in order to confirm the identity of participants. These are authentication protocols.

In recent years the cryptographic protocol analysis area [4,11] has seen an explosive growth, with numerous formalisms being developed. We can split them on three main categories: logic-based [1], model checking [14,16,18] and theorem proving [25]. Recently, there has been a tendency to try to combine these ones. Those specific purpose tools present one inconvenience; that is, protocol designers are not familiarized with this technology. Thus, we consider the application of cryptographic analysis results in the well-known protocol engineering area.

Our approach [21] uses a generic formal language and its associate verification methods and tools. We explain how SDL can be used to specify security protocols and cryptographic operations. At the same time show how security protocols can be modeled as safety properties and checked automatically by a model-based verification tool. In our method a simple and powerful intruder process is explicitly added to the specification, so that the verification of the authentication property guarantees the robustness of the handshake protocol against attacks of such malicious agent.

We have studied CASRUL [26], a system for automatic verification of cryptographic protocols. It translates a protocol given in common abstract syntax into a rewrite system. This rewrite system can then be preceded by a first order theorem prover for equational logic in the automatic detection of flaws. We have taken some of its characteristics for our methodology, specially in the intruder's behavior.

2. SDL Language

Specification and Description Language (SDL) [6,8,9] is a standard language for specifying and describing systems. It has been developed and standardized by ITU-T in the recommendation Z.100.

An SDL specification/design (a system) consists of a number of interconnected modules (blocks). A block can recursively be divided into more blocks forming a hierarchy of blocks. The channels define the communication paths through which the blocks communicate with each other or with the environment. Each channel usually contains an unbounded FIFO queue that contains the signals that are transported on it. One or more communicating processes describe the behavior of the leaf blocks, and extended finite state machines describe the processes.

In addition, SDL supports object-oriented design [7] by a type concept that allows specialization and inheritance to be used for most of the SDL concepts, like blocks, processes, data types, etc. The obvious advantage is the possibility to design compact systems and to reuse components, which in turn reduces the required effort to maintain a system. SDL has adopted the term *type*, which corresponds to the term *class* used in many of the object-oriented notations and programming languages.

Telelogic's Tau SDL Suite [10] provides an environment for developing SDL systems and implementations. The SDL Suite comprises a set of highly integrated tools that automate the transition from specification to real-time execution. With SDL's graphical language, SDL Suite describes, analyses, simulates, and supports generations of C/C++ applications. Thus SDL Suite simplifies testing and verifying the application by virtue of SDL language formal semantic that makes tool support in early phases possible. The engineer composes diagrams, supported by a formal, well-defined graphical syntax that defines the program functionality, eliminating the need to manually write whole sections of code. We use SDL Validator tool for verification purpose. Indeed, we are going to take advantage of its exploration algorithms and its check mechanism.

The SDL Validator [3] is based on state space exploration. State space exploration is based on automatic generation of researchable state for systems. Research state space means all possible states an application can find itself, and all possible ways it can be executed. A reachability graph is one way to conceptually view reachable state space, though one rarely computes it since it is too large for realistic applications.

The Validator has several ways to check SDL specification. These are essentially scenarios verification and observer process. In order to obtain scenarios specification we use the Message Sequence Chart (MSC) [5] language (Recommendation ITU-T Z.120). We can verify a MSC, checking if there is a possible execution path for the SDL system that satisfies the MSC, or checking MSC violation. Loading MSC and performing a state space exploration set up in a way suitable for verifying MSCs does this. The MSC verification algorithm is a bit state exploration that is adapted to suit the needs of MSC verification.

The more powerful way to check a SDL specification is the observer process mechanism. The basic idea is to use SDL processes (called *observer processes*) to describe the requirements that are to be tested and then include these processes in the SDL system. Typical application areas include feature interaction analysis and safety-critical systems.

By defining processes to be observer processes, the Validator will start to execute in a two-step fashion. First, the rest of the SDL system will execute one transition, and then all observer processes will execute one transition and check the new system state. The assert mechanism enables the observer processes to generate reports during state space exploration. These reports will show up in the list of generated reports in the Report Viewer.

3. Authentication Property

A security protocol [20] is a general template for a sequence of communications, using cryptographic techniques to meet one or more particular security related goals. The basic security services provided by security mechanisms (cryptographic algorithms and secure protocols) are: authentication, access control, data confidentiality, data integrity and non-repudiation[12,13].

In the sense of authentication [15] we include authentication of origin and entity authentication. Authentication of origin is taken to mean that we can be sure that a message that purports to be from a certain party was indeed originated by that party. A protocol maintains authentication of origin if it is always the case that Bob's node accepts a message, as being from Anne then it must indeed be the case that earlier Anne set exactly this message. In Entity authentication protocols, you can be a confident that the claimed identity of an agent with whom you are interacting is correct. You identify the origin of all message received (authentication of origin of all message). Notice that freshness of the message and time requirements need to be considerate in the type of protocols.

In security protocol analysis we have adopted several assumptions. These are that cryptography is secure, so no cryptanalysis techniques are considerate; cryptography is perfect; and all agents may freely and perfectly generate random number.

Actually, secrecy and authentication are the security properties more widely analyzed. Secrecy property [24] is quality to prevent the intruder from being able to derive the plaintext of messages passing between honest nodes. If there is any behavior of the system that reaches a state in which an item from the set $\prod$ shows up in plaintext from in the intruder set of knowledge this is deemed to represent a breach of secrecy property.

For an authentication protocol [24] to be correct, we usually require that a user Bob should not finish running the protocol believing that has been running with a user Alice unless Alice also believes that she has been running the protocol with Bob. Conditions such as this can be expressed as trace specifications, requiring that no event from a set $\prod$ has occurred unless another event from a set Ω has previously occurred. In such a specification, the set $\prod$ would contain a message or messages indicating that the responder has finished the protocol believing that Alice is the other party, and set Ω would contain a message or messages that would appear whenever Alice attempts to start the protocol with Bob. A trace of a process may be viewed as a record of the sequence of events it performs during an execution.

In order to analyze the behavior of the security protocol when an intruder agent exists in the environment we have to specify this special agent. The intruder controls transmission medium and can realize the kinds of attacks described below. The actions that can do to protocol messages are killing, sniffing, intercepting, re-routing, delaying, delivering, reordering, replaying, and faking. It is known as the Dolev-Yao's method [17].

4. Design and Analysis Mechanism using SDL

Our approach achieves the design and analysis of authentication protocols, in the same way that we can do it using other communication protocols. Firstly, we define the system requirements. These can be specified in natural language, or better in MSC language. Then, we translate the system requirements into an SDL system. Actually, we are doing it manually, but we are working to perform this step automatically.

The SDL system is composed by a package where data types are defined, and other package where one type process for each protocol agent, and a group of type process observer and process medium are specified.

In order to analyze security properties, we evaluate the SDL system behavior when different types of attacks are applied for medium processes. Observer processes check if certain situation is reached and in that case, produces an inform report. This report corresponds to a failure scenario.

The analyzer creates the medium and observer processes for any kind of vulnerability to be examined. We have implemented generic process medium and observers, but they must be extended depending on system environment.

The messages, which are sent by protocol agents, are constructed by a concatenation of elemental data types and cryptographic operations. These data types can be divided in agent identification, number (random, time-stamping, etc…), symmetric key and asymmetric key pair. Thus, the operations used are cipher, decipher, sign and hash function.

The package "analcryptlib" defines data type used by SDL specification. The SDL data types do not support recursive definition, so we make use of enumerated and *struct* data types. The elemental data types defined are: (a) agent Identification, it is an enumerated sort with all possible agents name; (b) number, it is a fresh and/or random value; Secret key, this represents symmetric key; (c) public key, this is composed by a key pair (private and public key); (d) encrypted message, that it is implemented with a *struct* sort composed by item message and item symmetric or asymmetric cipher; (e) signed message, it is defined as a structure sort with a message and the private key signer.

Freshness or temporary secrets are implemented appending an item that has the process instance values. The SDL sort PID allows doing it.

Furthermore, we define a type set of knowledge for each data type. Intruder utilizes these types to store message knowledge.

The generic model identifies each protocol agent with a process type SDL. All process types are stored in a package in order to be used in other specifications. An agent specification is absolutely independent of the rest of the system, so they are generated in separated modules. Furthermore, this specification permits concurrent instance so that we can evaluate this behavior in the analysis stage.

The generic state transition of process agent is triggered when it receives a message and it is correct. If is not correct, it will return to waiting message state. Then, the next message is composed to be sent to the receiver agent or it will stop if it is the final state of this protocol.

The process in SDL is a finite state machine, so it finishes when execute a stop statement or provide a deadlock if no signal arrives. Our model has to explore all possibilities; thus, we need to develop a mechanism to achieve that all signals sent must be processed.

Consequently, we have appended a state called "final" to notice about the end of the protocol execution, and a general transition composes on a common "save" statement and a continuous signal, less priority than input statement, that check if there are some signal waiting for dealing out. By means of this structure we transform a finite state machine in an infinite one, only for analyzing purposes.

At this point, we have specified a security protocol in the same way that we would have specified a communication protocol. Then, we can examine the classical liveness properties. It is important and we need that the specification is well formed, but it is not the main aim of a secure system

The intruder's behavior is split into two aspects, exploration algorithm and check mechanism. The exploration algorithm is done by medium process and observer processes perform the check mechanisms.

We consider two types of medium process model. The first, it is characterized by an exploration mechanism that search all possibilities. It begins examining all combinations of different initial knowledge for each agent. Afterwards, it proves with concurrent agents, at first use combinations of two concurrent sessions, and so on. Our algorithm finishes when an out of memory is produced or it detects that the significant intruder knowledge is not incremented. In the greatest part of the cases the problem completeness [2,22] is undecidable, so it is impossible to be completely sure about problem solution.

The second is developed with an intruder specialized in finding a determinate flaw. If we typify a kind of attack, we can evaluate the protocol trying to find a specific flaw. Perhaps this is not the best solution but it is very useful to protocol designer to be sure that almost for this kind of attack our protocol is not vulnerable. Furthermore, the majority of analysis tools that use model checking accept that this problem is undecidable, so we only get results about a definite vulnerability will not happen in the cases that we have examined.

The state transition of process medium is triggered when it receives any message. Then, it is stored in the intruder knowledge database, and an intruder decides which operations are going to do be done, and step to next state of routing. We have defined three different operations: eavesdrop, divert, and impersonate. Eavesdrop operation, meaning the intruder intercept the messages and rejects them. Divert operation, meaning the intruder intercepts the messages but they are not sent to the original receiver. Impersonate operation, meaning the intruder sends fake message to the original receiver. These belong to CASRUL specification.

The check mechanism is the observer process. Observer process is a special SDL type of process that is evaluated in each transition of the protocol specification. It gets at all variable of whole process instances, so we can check it automatically.

The security properties are proved using condition rules. Those rules check different situations where it is possible that exists protocol vulnerability. The elements that are checked are agent’s states, variable value, and sent signals.

Actually, we check secrecy and authentication properties. If we want to check secrecy properties, we examine if the intruder knowledge has o can be deduced a specific value that we consider secret. The authentication is examined checking that all the principals finish at the same time, when it is expected. Some authors call to this the correspondence or precedence flaw.

5. Example of protocol analysis

As an example to illustrate our proposal we are going to explain the specification and analysis of EKE protocol (Encrypted Key Exchange) [19]. EKE is a key exchange authentication protocol that resists dictionary attacks by giving passive attacker insufficient information to verify a guessed password. As stated, it performs key exchange as well, so both parties can encrypt their transmissions once authentication is established. In the most general form of EKE, the two communicating parties encrypt short-lived public keys with a symmetric cipher, using their shared secret password as a key. Since it was designed, EKE has been developed into a family of protocols, many of which are stronger than the original or add new desirable properties.

The basic EKE protocol is specified in the next message sequence:

```
1.  A → B:  { Ka }P

2.  B → A:  { { Re } Ka }P

3.  A → B:  { Na }Re

4.  B → A:  { Na,Nb }Re

5.  A → B:  { Nb }Re
```

Let "A" and "B" represent two agents, A is the initiator and B is the responder. "P" is a symmetric key shared by A and B. "Ka" is A's public key. "Re" is a fresh symmetric key generated by B. And "Na" and "Nb" are fresh and random number from A and B respectively.

Firstly, we produce the SDL specification of EKE protocol, similar to an ordinary communications protocol. Then, we create the medium and observer processes, in order to analyze the correspondence flaw defined in CASRUL analysis tool. This flaw is produced when we execute two sessions concurrently. During first session, "A" and "B" are instanced to "a" and "b" principals identification, respectively, while during the second session, "A" and "B" are instanced to "b" and "a" principals identification, respectively. The observer process checks if agents of sessions 1 and 2 reach a final state and if their corresponding parties do not reach it.

The package "analcryptlib" includes all messages definition for analysis purpose. The formats are defined as SDL struct sort. Those belong to generic choice sort called "TMESSAGE". At the same time, it is defined a generic struct sort called "TENCMESS" where security operations are applied. Those operators are "enc" to encipher, "denc" to decipher, "sing" to do a digital signature, and "hash" to apply a hash function.

Agents are defined as a process type included in SDL package called "agents". We specify agent initiator process type ("agentA"), and agent responder process type ("agentB"). Both processes type have states called "mess" plus a number of message. Each state has an input signal that is triggered when its related message is received. This message is checked before being accepted, and it stops if it is the final state or it composes the corresponding message, sends it, and steps to next state. Fresh data types are provided adding to their definition a process identification item (PID SDL sort). It is used to differ every concurrent session.

In order to examine all messages that receive every process agent, we have defined an asterisk state that saves all unprocessed signals, and in a lower priority level, it checks if there are any queued messages. If that is the case, it changes its state to that one related to that signal.

The Intruder model is divided in the explotarion algoritm and the check mechanism. These depend on analysis strategy that the analyzer must evaluate. There are a number of attacks types that actual researches have developed, which can be implemented in our analysis mechanism.

The process type called "CorresAttack" provides the exploration algorithm. This consists on a state that is triggered by any input messages and executes intruder's operation. Then, the divert operation is applied sending message form first session to second session, and reciprocally. This is called man-in-the-middle attack.

```
State checking
 if  ((GetState(A2)='final') AND (GetState(B1)='final'))
   AND(((GetState(B2)/='final') AND ((Getstate(A1)/='final'))))
 then
   REPORT "authentication error"
   STOP
 else
    if  (( GetState(A1)='final') AND (GetState(B2)='final'))
     AND (((GetState(B1)/='final')AND(( Getstate(A2)/='final'))))
    then
           REPORT "authentication error"
     STOP
    else
       state checking
```

Figure 1. Assert Condition to check authentication

In order to check the correspondence flaw we create the process type observer called "obvcorresattack". The main state has two possible conditions (figure 1). The first condition is true when agent A of session one and agent B of session two reach a "final" state, and the other agents do not reach it. The second condition checks the inverse situation.

When a check condition is true, a report action is executed, and it can stop searching or continue exploring for a new failure scenario. This report is provided in MSC language.

In order to explore the correspondence failure we have defined processes distribution that is shown in figure 2. The system is specified connecting through the instance of the process type "Corresattack" called "medattack", an instance of process type "agentA" and another of process type "agentB". Those instances are called "A" and "B" respectively.

This processes distribution enforce all messages, that are sent between A and B, to cross through the medium instance, where they are processed by intruder's operations. In that case, the intruder's operation is only to divert.

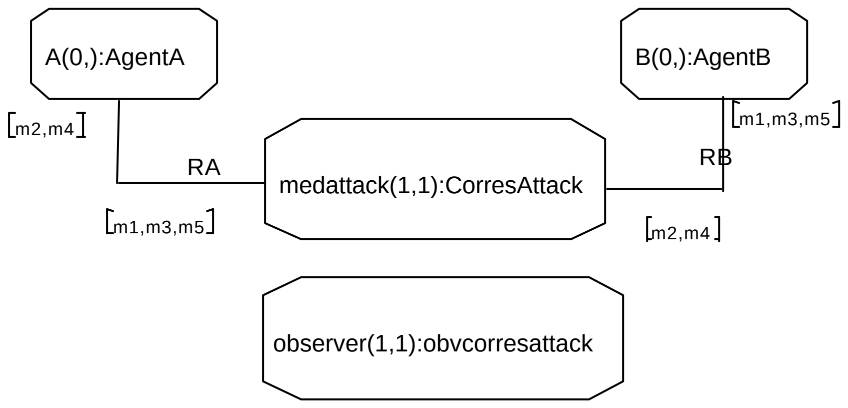

Figure 2. Processes Distribution for Analysing

Following we load the SDL specification in the SDL Validator tool. First we have to configure the Validator, in order to evaluate the system correctly. Then, we execute the exhaustive exploration option. It finishes quickly and generates the following report.

Figure 3 shows how two sessions start at the same time. When the intruder process intercepts the messages from a session and sends to the other session, then the agent A of first session and agent B of second session reach a "final" state. This means that the agent A of first session relies on what it is communicating with agent B of its session but, in fact, it is connected with agent B of second session.

Actually, this protocol failure is resolved adding agent name to messages. An instance of implementation is the Secure Remote Password (SRP)[27]. This is a strong password authentication system developed as *open-source* and commercial products.

```
First session instance
A1=b
B1=a
Second session instance
A2=a
B2=b

Messages Exchange
1. A1 → Intruder: { Kb }P
2. A2 → Intruder: { Ka }P
3. Intruder → B2: { Kb }P # redirect to B2 all messages
  # directed to B-agent
4. Intruder → B2: {Ka}P # redirect to B2 all messages to
                          # B agent. This is refuse
5. B2 → Intruder: { { Re }Kb }P
6. Intruder → A1: { { Re }Kb }P # redirect to A1 all messages
                               # directed to A-agent
7. A1 → Intruder: { Na }Re
8. Intruder → B2: { Na }Re
9. B2 → Intruder: { Na , Nb }Re
10. Intruder → A1: { Na, Nb} Re
11. A1 → Intruder: { Nb }Re
12. Intruder → B2: { Nb }Re
```

Figure 3. Message Exchange of Authentication Flaw

6. Conclusions and Future Works

We have presented a new mechanism to specify authentication protocols and their possible attacks. The authentication protocol is specified by a SDL system, and attacks are implemented as SDL processes that develop intruder behavior and check security properties. Protocol specification is independent of analysis procedure, so it can be used in others environments.

Our main goal has been to formally specify a security protocol in order to get an unambiguous representation and to check security properties, like authentication. Therefore, we have been able to analyze a security system and rely on its safety, or at least in the fact that it is not vulnerable for known attacks.

Actually, we are developing a translator to design the systems in extended MSC as requirement language that will generate automatically a SDL system. In order to check several types of attacks, we are gathering set of generic attacks. Furthermore, we are studying how to implement those attacks into a specific environment.

References

1. M. Burrows, M. Abadi, and R.Needham. A logic of authentication. In Proceedings of the Royal Society, Series A, 426(1871):233-271, 1989.
2. Gavin Lowe, “Towards a Completeness Result for Model Checking of Security Protocols”. In 11th IEEE Computer Security Foundations Workshop, pages 96-105. IEEE Computer Society, 1998.
3. D. Hogrefe. Validation of SDL systems. Computer Networks and ISDN Systems, 28 (12):1659–1667, June 1996.
4. Catherine Meadows. Formal verification of cryptographic protocols: A survey. In Advances in Cryptology - ASIACRYPT '94, number 917 in Lecture Notes in Computer Science, pages 133-150. Springer-Verlag, Berlin, 1995.
5. ITU-T, Geneva, Switzerland. Message Sequence Charts, 1999. ITU-T Recommendation Z.120.
6. ITU-T, Geneva, Switzerland. Specification and Description Language (SDL), 1999. ITU-T Recommendation Z.100.
7. Sarma J. Ellsberger, D. Hogrefe. SDL – Formal object-oriented language for communication systems. Prentice-Hall, 1997.
8. G. Holzmann. Design and Validation of Computer Protocols. Prentice-Hall, Englewood Cliffs, 1991.
9. Kenneth J. Turner, “Using Formal Description Techniques. An introduction to Estelle, LOTOS and SDL”.
10. Telelogic . “User´s Guide SDT” http://www.telelogic.com/.
11. Meadows. Open issues in formal methods for cryptographic protocol analysis. In Proceedings of DISCEX 2000,pages 237--250. IEEE Comp. Society Press, 2000.
12 S. Schneider. Formal analysis of a non-repudiation protocol. In Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW '98), pages 54-65, June 1998.
13. J. Zhou and D. Gollmann. Towards verification of non-repudiation protocols. In Proceedings of 1998 International Refinement Workshop and Formal Methods Pacific, pages 370-380, Canberra, Australia, Sept. 1998. Springer.
14. W. Marrero, E. Clarke, and S. Jha. Model checking for security protocols. DIMACS Workshop on Design and Formal Verification of Security Protocols, 1997.
15. T.Y.C. Woo and S.S. Lam, “A Semantic model for authentication protocols”, IEEE Symposium on Research in Security and Privacity, 1993.
16. J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murphi. In Proceedings of IEEE Symposium on Security and Privacy, pages 141-151. IEEE Computer Society Press, 1997.
17. D. Dolev and A. Yao. On the security of public key protocols. IEEE Transactions on Information Theory, IT-29:198{208, 1983. Also STAN-CS-81-854, May 1981, Stanford U.
18. R. Alur, T. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. Mocha: modularity in model checking. In A. Hu and M. Vardi, editors, CAV 98: Computer-aided Verification, Lecture Notes in Computer Science 1427, pages 521-525. Springer-Verlag, 1998.
19. S.M. Bellovin and M. Merrit. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In Proceedings of IEEE Symposium on Reseach in Security and Privacy, pages 72-84, 1992.
20. A.Menezes, P.C.van Oorschot, S.Vanstone, "Handbook of Applied Cryptography", CRC Press, (1997).
21. J. J. Ortega. Técnicas de Descripción Formal para el estudio de la Vulnerabilidad de Protocolos. V Reunión Española de Criptología y Seguridad, pages 201-213, Málaga 1998.
22. M. Rusinowich, M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete 14th IEEE Computer Security Foundations Workshop June 11-13, 2001 Cape Breton, Nova Scotia, Canada.
23. M. Rusinowich, F. Jacquemard, L. Vigneron. Compiling and Verifying Security Protocols Logic for Programming and Automated Reasoning, Reunion Island, November 2000. LNCS 1955.
24. P.Y.A. Ryan and Scheneider. The Medelling and Analysis of Security Protocols:the CSP Approach. Addison-Wesley, 2001, ISBN 0 201 67471 8.

25. G. Denker and J. Millen. Capsl intermediate language. In Formal Methods and Security Protocols,1999. FLOC '99 Workshop.
26. M. Rusinowich. CASRUL http://www.loria.fr/equipes/protheo/SOFTWARES/CASRUL/.
27. The Stanford SRP Authentication Project. RFC 2945