J. Lopez, J. J. Ortega, and J. M. Troya, "Security Protocols Analysis: A SDL-based Approach", *Computer Standards & Interfaces*, vol. 27, pp. 489-499, 2005.
NICS Lab. Publications: https://www.nics.uma.es/publications

ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computer Standards & Interfaces 27 (2005) 489–499

COMPUTER STANDARDS & INTERFACES

www.elsevier.com/locate/csi

Security protocols analysis: A SDL-based approach

Javier Lopez*, Juan J. Ortega, Jose M. Troya

Computer Science Department, E.T.S. Ingeniería Informática, University of Malaga, Spain

Available online 1 February 2005

Abstract

Organizations need to develop formally analyzed systems in order to achieve well-known formal method benefits. In order to study the security of communication systems, we have developed a methodology for the application of the formal analysis techniques, commonly used in communication protocols, to the analysis of cryptographic protocols. In particular, we have extended the design and analysis phases with security properties. Our proposal uses a specification notation based on one of the most used standard requirement languages, HMSC/MSC, which can be automatically translated into a generic SDL specification. The SDL system obtained can then be used for the analysis of the addressed security properties, by using an observer process schema. Besides our main goal to provide a notation for describing the formal specification of security systems, our proposal also brings additional benefits, such as the study of the possible attacks to the system, and the possibility of re-using the specifications produced to describe and analyze more complex systems.

Keywords: Security protocols; Formal analysis; MSC; SDL

1. Introduction

It is widely accepted that critical systems have to be analyzed formally in order to achieve well-known formal method benefits. These methods characterize the behavior of a system in a precise way and can verify its formal specification. In particular, the design and analysis of security systems can greatly benefit from the use of formal methods, due to the evident critical nature of such systems.

The European Telecommunications Standards Institute (ETSI) is an independent, non-profit organization, whose mission is to produce telecommunications standards for today and for the future. ETSI is using formal techniques in most of its leading technical areas: for example GSM, TETRA, DECT, N-ISDN, B-ISDN, V5 protocols, Network Architecture and Management, INAP, Hiperlan/2, and 3GPP. In many cases where SDL has been used, the standards have included extensive MSCs. ETSI does not prescribe the use of any particular technique, but the use of standardized languages such as UML, MSC, SDL, ASN.1, and TTCN (for testing) is strongly recommended. Of course, security communication protocols are also considered to be standardized.

On the other hand, the cryptographic protocol analysis research area has experienced an explosive

* Corresponding author.
E-mail address: jlm@lcc.uma.es (J. Lopez).

0920-5489/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.csi.2005.01.007

growth, with numerous formalisms being developed. We can divide this research into three main categories: logic-based [3], model checking [2,12], and theorem proving [13]. Although all three approaches have shown their applicability to simple problems, they are still difficult to apply in real, more complex environments such as distributed systems over the Internet. Moreover, we believe that the results obtained for the analysis of cryptographic protocols do not have a direct application in the design of secure communication systems. Probably, one of the major reasons is the lack of a strong relationship between the analysis tools for security systems and the formal methods techniques commonly used in the specification and analysis of communication protocols. Trying to bridge this gap is one of the major contributions of our work.

We have developed a methodology for the specification of secure systems that, additionally, also allows us to check that they are not vulnerable against both well-known and original attacks. Our approach uses a requirement language, *Security Requirements Specification Language* (SRSL), to describe security protocols, which can then be automatically translated into SDL [6], a widely used formal notation specifically well suited for the analysis of protocols. In addition, we have developed some verification procedures and tools for checking a set of security properties, such as confidentiality, authentication, and non-repudiation of origin. In our approach we use a simple but powerful intruder process, which is explicitly added to the specification of the system so that the verification of the security properties guarantees the robustness of the protocol against attacks of such an intruder. This is known as the Dolev–Yao's method [5].

Since SRSL is an extension of HMSC/MSC [7], available editors for *Message Sequence Chart* (MSC), *High-Level MSC* (HMSC), and the *Specification and Description Language* (SDL) can be used for writing SRSL specifications, as well as standard code-generators and SDL validation tools. In particular, we have built our translators and analysis tools using Telelogic's Tau SDL Suite.

The structure of the rest of the paper is as follows. After this introduction, Section 2 defines the security concepts and mechanisms used throughout the paper. Then, Section 3 provides an overview of our proposal. The SRSL language is presented in Section 4, while Section 5 discusses how the SRSL descriptions can be automatically translated into SDL and how the SDL specifications produced can be analyzed for proving security properties. Finally, Section 6 draws some conclusions and outlines some future work.

2. Specification of security properties

A security protocol is a general template describing a sequence of communications, which makes use of cryptographic techniques to meet one or more particular security-related goals. In our context, we will not distinguish between cryptographic and security protocols, considering both to be equivalent. The international organization ITU-T has defined the Recommendation Series X.800 [9,10] in order to specify the basic security services. Among these, the ones provided by the basic security services (cryptographic algorithms and secure protocols) are authentication, access control, data confidentiality, data integrity, and non-repudiation.

The notion of *authentication* includes both authentication of origin and entity authentication. *Authentication of origin* can be defined as the certainty that a message that is claimed to proceed from a certain party was actually originated from it. *Access control* ensures that only authorized principals can gain access to protected resources. Usually, the identity of the principal must be established, hence entity authentication is also required here. *Confidentiality* may be defined as the prevention of unauthorized disclosure of information. In communication protocols, this means that nobody who has access to the exchanged messages can deduce the secret information being transmitted. *Data integrity* means that data cannot be corrupted, or at least that corruption will not remain undetected. Accepting a corrupted message is considered as a violation of integrity, and therefore the protocol must be regarded as flawed. *Non-repudiation* provides evidence to the parties involved in a communication that certain steps of the protocol have occurred. This property appears to be very similar to authentication, but in this case the participants are given capabilities to fake messages, up to the usual cryptographic constraints. Non-repudiation uses signature mecha-

nisms and a trusted notary. We will distinguish two types of non-repudiation services: non-repudiation of origin (NRO) and non-repudiation of receipt (NRR). NRO is intended to protect the originator's false denial of having originated the message. On the other hand, NRR is intended to prevent the recipient's false denial of having received the message.

All previous services are commonly enforced using cryptographic protocols or similar mechanisms. It is worth noting that, in order to specify a security system, it is not necessary to know how the system is going to be analyzed, but it is essential to identify the security services required.

Now, considering the system from the attacker's perspective, additional security protocol vulnerabilities can be defined [16]: (a) *man-in-the-middle*, where the intruder is able to masquerade a protocol participant; (b) *reflection*, where an agent emits messages and studies the system's answers; (c) *oracle*, where the intruder tricks a honest agent by inadvertently revealing some information (notice that such an attack may involve the intruder exploiting steps from different runs of the protocol, or even involve steps from an entirely different protocol); (d) *replay*, in which the intruder monitors a (possible partial) run of the protocol and, at some later time, replays one or more of the protocol's messages; (e) *interleave*, where the intruder contrives for two or more runs of the protocol to overlap; (f) *failures of forward secrecy*, in which the compromised information is allowed to propagate into the future; and (g) *algebraic attack*, where it is possible for intruders to exploit algebraic identities to undermine the security of the protocol. Please note that these kinds of attacks depend on the environment of the system (network, users, etc...), and therefore not all of them are always achievable in a given context. However, we will study all potential situations, trying to cover all of them in all cases.

In order to keep clear the focus of the paper, the following assumptions have been made. First, we suppose that cryptography is perfect, so no cryptanalysis techniques are used. Second, all agents may freely and perfectly generate random numbers. And finally, there are no external interactions with other protocols.

3. Methodology overview

Our approach (depicted in Fig. 1) performs the design and analysis of security protocols in the same way the design and analysis of a traditional communication protocol is accomplished, though including the security aspects.

In first place, we need to gather the functional and security requirements of the system in any (usually informal) way. These informal specifications, together with the behavior about the types of possible attacks (if available), are the sort of information that can be described using our SRSL language.

SRSL is an extension of HMSC/MSC, augmented with textual tags. We make use of the HMSC/MSC text area to include these tags, which are used to identify the security characteristics of the data being transmitted, the intruder's possible activities, and the security analysis goals. In case the attacker's behavior is not explicitly provided, we automatically generate a generic process that tries to examine all possible attacks.

For drawing the graphical SRSL specifications, any standard MSC and HMSC editor can be used. In our case, we have used Telelogic's TAU, which also allows the automatic translation of the graphical MSC diagrams into their corresponding textual form. A translator program is then used to obtain the SDL system from the SRSL descriptions (this program has been written in C, using plain LEX and YACC tools). The SDL system produced is composed of: (1) a package with the data types of the system for the analysis; (2) a package with one process type for each protocol agent; and (3) a collection of process types ("observer" and "medium") for the analysis strategy.

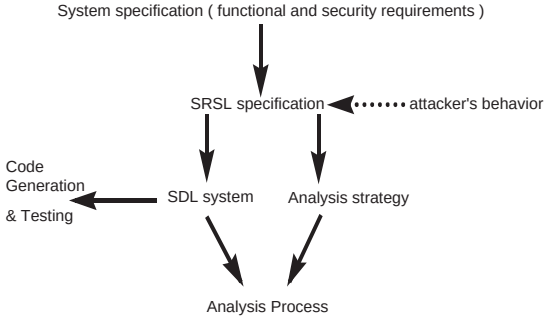

Fig. 1. Overview of our approach.

In order to analyze the security properties, we evaluate the behavior of the SDL system under different types of attacks (as specified by the medium processes defined in the analysis strategy). The observer process provided by the TAU Validator tool is used for these checks. Thus, we can check whether a specific state is reached, or whether a particular data is ever stored into the intruder's database knowledge.

We also make use of the *assert* mechanism, which enables observer processes to generate reports during the state space exploration. These reports are maintained by the Report Viewer, and can be examined to identify security flaws.

Currently, confidentiality and authentication are the security properties usually analyzed. By analyzing confidentiality we prevent the intruder from being able to derive the plaintext of messages passing between honest nodes. Our analysis consists of checking if the secret item can be deduced from the protocol messages and the intruder's database knowledge.

An authentication protocol is considered to be correct if a user Bob does not finish the protocol believing that it has been running with a user Alice unless Alice also believes that she has been running the protocol with Bob. Our analysis consists of looking for a reachable state where Bob has finished correctly and Alice will never reach her final state.

We also analyze non-repudiation of origin. For that we define the evidence of origin and who produces it (the origin). Our analysis consists of checking that the evidence is digitally signed by the origin agent, and that it cannot be created by any other agent.

In addition, the SDL system generated from the SRSL specifications can be used to automatically generate C or C++ code, which can interact with existing applications. In order to generate this code we need to replace the data types package with a corresponding package that defines the data types in ASN.1 or C. This prototype can also be used for testing, which is part of our future work.

4. The SRSL language

The main aim of SRSL is to define a high-level language for the specification of cryptographic protocols and secure systems. As pre-requisites for this language we need to request it to be modular to achieve reusability, to be easy to learn, and to incorporate security concepts.

As a natural base for SRSL we have considered the requirements language most widely used in the telecommunications arena, the MSC, and its extension HMSC. With MSC we can specify elementary scenarios, and compose them to define more complex protocols with HMSC. The version we have considered is the previous to the MSC 2000 release [8], but we believe that it is very useful for some features of this release.

SRSL is divided into two main parts. The first one contains the definition of the protocol elements ("*DEFINITION*") and the security analysis strategy. The second part describes the message exchange flow. The protocols' elements definitions are based on the HPSL language defined in AVIS project [1].

The first part is textual. The syntax of its main elements is shown in Fig. 2. These elements can be grouped into different categories, and are listed below (language keywords are written in *italics*):

1. Entities: *AGENT*, principal identification;
2. Message: *DATA*, message text; *RANDOM*, number created for freshness, also called nonce; *Timestamp*, actual time; *SEQUENCE*, counter.
3. Keys: *PUBLIC_KEY*, cryptographic public-key, formed by a pair of public and private keys; *SYMMETRIC_KEY*, used for symmetric encryption.

The "*KNOWLEDGE*" section contains the information needed to describe the initial knowledge of each party of the protocol.

The "security service" section is split into the "intruder strategy" section and the "security property" section. The first one defines a possible attack scenario. The second one describes which security property we try to achieve with this protocol. We have used three different security statements: *AUTHENTICATED(A,B)*, stating that B is certain of the identity of A; *CONFf(X)*, stating that the data X cannot be deduced (also called confidentiality); and *NRO(A,X)*, or non-repudiation of origin, which states that the data X (the evidence) must have been originated in A. These statements have a formal description [9,10] which is used to analyze them.

```
Security_information ::= Definition_section Security_service_section

Definition_section ::= DEFINITION Var_definition  Knowledge_section

        Var_definition ::= varlist : AGENT ;
                  | varlist : DATA ;
                  | varlist : RANDOM ;
                  | varlist : SEQUENCE ;
                  | varlist : PUBLIC_KEY ;
                  | varlist : SYMMETRIC_KEY ;

        Knowledge_section ::= KNOWLEDGE <listagent_id> : <varlist> ;

Security_service_section ::= [Intruder_strategy] [security_property]

        Intruder_strategy ::= SESSION_INSTANCE '[' <var> = <value> ']' ;
                        | INTRUDER_KNOWLEDGE initial_knowledge_list ;
        | INTRUDER intruder_behaviour_list ;

        initial_knowledge_list ::= intruder_knowledge_list , intruder_knowledge
                           | intruder_knowledge
        intruder_knowledge ::= <var> = <value>

        intruder_behaviour_list ::= intruder_behaviour_list , intruder_behaviour
                        | intruder_behaviour

        intruder_behaviour ::=   REDIRECT
                               |  IMPERSONATE
                               |  EAVESDROP

        security_property ::= SECURITY_SERVICE security_service_list ;

        security_service_list ::= security_service_list security_service_item
                         | security_service_item

        security_service_item ::=  | AUTHENTICATED ( <agentID> <agentID> );
                                   | CONF ( <var> ) ;
                                   | NRO ( <agentID> <var> ) ;
```

Fig. 2. SRSL security section syntax.

The message exchange flow is described using the standard MSC and HMSC facilities. MSC references are used to achieve reusability. We have specified a set of standard protocols in SRSL, that can be easily re-used in different contexts, and combined together to describe more complex protocols using their MSC references.

Some cryptographic operations can be applied to messages: Concatenate (“,”) for data composition; Cipher (“{<plaintext>}”<key>) to encrypt data; Decipher (“*decrypt*(<cipher_data>,<key>)”) to extract the plaintext; Hash (“<hash-function>(<data>)”), the result of a one way algorithm; and Sign (“[<plaintext>] <Public_Private_key>”), for getting a hash encrypted message with the signer’s private key. Further cryptographic functions can be defined if required.

In addition, the MSC expressions constructed using the inline MSC operators *alt*, *par*, *loop*, *opt*, and *exc* can also be used.

The keyword *alt* denotes alternative executions of several MSCs. Only one of the alternatives is applicable in an instantiation of the actual sequence. The *par* operator denotes the parallel execution of several MSCs. All events within the MSCs involved are executed, with the sole restriction that the event order within each MSC must be preserved. An MSC reference with a *loop* construct is used for iterations and can have several forms. The most general construct, *loop*<*n,m*>, where *n* and *m* are natural numbers, denotes iteration at least *n* and at most *m* times. The *opt* construct denotes an unary operator. It is interpreted in the same way as an *alt* operation where the second operand is an empty MSC. An MSC reference where the text starts with *exc* followed by the name of an MSC indicates that the MSC can be aborted at the position of the MSC reference symbol, and continue with the referenced MSC.

In order to illustrate our approach we will specify here a typical secure Web access to a bank portal via Internet. Fig. 3 shows the SRSL specification of the system in SRSL, using two agents: “User_Browser” and “Bank_Portal”. The “Bank_Portal” agent has a secure web service via the HTTPS protocol, which provides authentication of server. This is represented

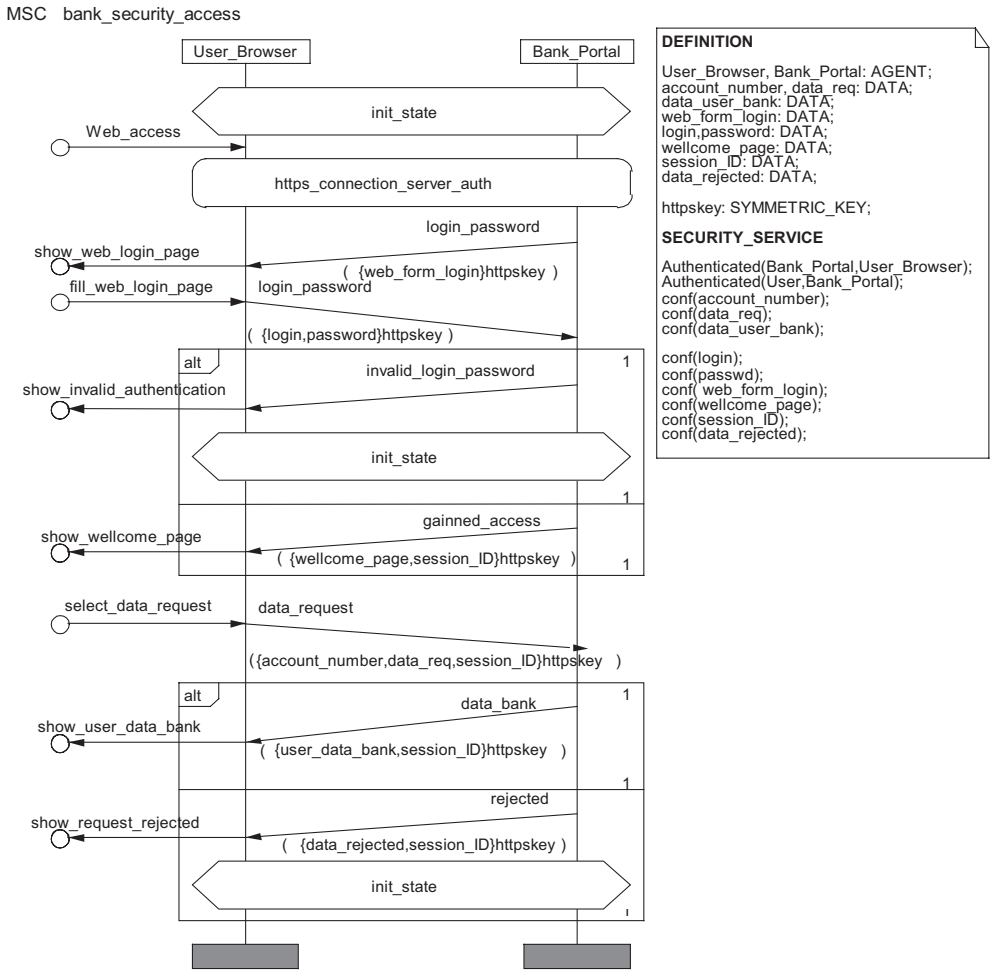

Fig. 3. SRSL security scenario of user's Web access to a bank.

by a MSC reference called "https_connection_server_auth" that implements the server authentication and the key exchange protocol defined in HTTPS. This MSC reference is defined in a package of standard protocols. The results of this scenario are authentication of server and a session key called "httpsskey". The first security requirement "*Authenticated (Bank_Portal, User_Browser)*" is achieved with this protocol.

The second requirement means that the user must authenticate itself to the bank's portal. This is accomplished by a mechanism that asks for the user's identification (login) and password, and subsequently validates it. The exception MSC reference called "invalid_login_password" is active if the login-password authentication process of the "Bank_Portal" fails. Notice that all messages are encrypted by the HTTPS protocol session key.

The rest of security requirements mean that the data transmitted is confidential. This goal is accomplished by making use of the session key established during the HTTPS connection.

Of course, other alternative security mechanisms could have been considered for specifying this system, which also met the five original requirements. The important point to note here is that we have chosen a form of specification that does not bind the developer to any particular security mechanisms, thus achieving separation of concerns and modularity. This is accomplished by allowing the security requirements to be defined at a higher level

of abstraction, and independently from the system's functional requirements.

In the case of a system that is already implemented, i.e., a legacy system, and that we want to analyze or document, we can describe instead the security mechanisms that have been implemented.

We have specified most popular security protocols, such as SSL/TLS, Kerberos, IPSEC, and authentications protocols defined in the SPORE website [14]. These specifications are used in a complex system, and we can compare different security protocols.

5. Security analysis

We use SDL for the security analysis. Firstly, we need to build an SDL system from the SRSL specifications. We have developed a program that automates this process. The program is written in the C language, and uses LEX and YACC standard tools. The input file is a protocol specification written in SRSL, and the program produces a valid SDL system. The generated SDL system is composed of three packages, and contains several processes.

The SDL package that defines the system data types and their operators is called "datacryptlib". It also contains elementary security data types, and the message format definition used in the protocol. This information is used by the rest of the system.

Another SDL package defines a process type for each principal. They are implemented in a standalone fashion so they can be reused in different situations. Fig. 4 shows two process types, which reference agents *A* and *B*, respectively.

The last package is about the observer processes. They implement the assert mechanism used in the validation process, and depend on the medium process (called "medattack" in Fig. 4) and on the security services that will be evaluated.

The SDL system is named after the protocol it defines, and consists of a single SDL block, which is composed of the process structure for analysis ("A", "B", "medattack"), an observer process ("observer"), and several medium process types ("explore", "redirect", and "router"). The process structure for the analysis consists of a medium process that controls all transmissions among agent processes. This control implements the attacker's procedure.

Note that medium process types have to be created inside this block because they implement the intruder's behavior, and therefore they may create process agent instances. The TAU tool we use requires all process instances to be defined within the same block. The following describes in detail all the system parts,

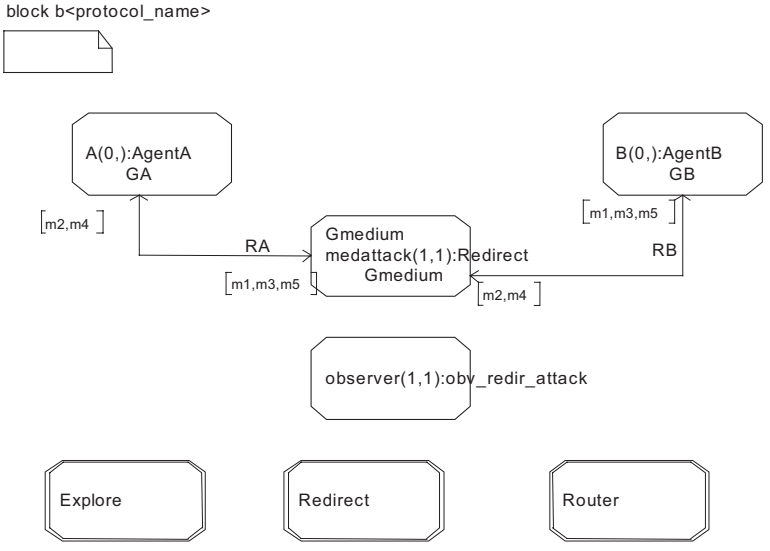

Fig. 4. SDL system structure.

and how the SRSL specifications are mapped onto the SDL description of the system.

5.1. Data types package

An SDL package contains the data types and the cryptographic functions used in the SRSL specification of the system. We may consider an SDL package for performing the analysis, and other package (written in ASN.1) for code generation. All cryptographic data and operators are standardized using ASN.1 notation, following PKCS standards.

Since the SDL data types do not support recursive definitions, we make use of enumerated and structured data types. The elemental data types defined in Section 4 are then mapped to enumerated SDL *struct* sorts.

The messages, which are sent by protocol agents, are constructed by concatenation of elemental data types and cryptographic operations. We define a *struct* sort for each message and set of elemental data types. The cryptographic functions are then applied to a set of elemental data types called "TENCMESS".

Freshness or temporary secrets are implemented by adding an item that references the process instance values. In particular, we use the SDL sort *PID* for this purpose.

Furthermore, we define a "set of knowledge" type for each data type. The analysis methods use these types to store message knowledge in order to prove the specified security properties.

5.2. Agents Package

The generic model identifies each protocol agent with an SDL process type. All process types are stored in a package called "agentlib" so they can be used in other specifications. An agent specification is totally independent from the rest of the system, so they are generated in separate modules. In addition, the specification allows concurrent instances, so we can evaluate this behavior in the analysis phase.

The generic state transition of an agent process is triggered when it receives a correct message (i.e., a message accepted by the agent). Then, either the next message is composed to be sent to the receiver agent, or the process stops if the protocol's final state is reached for this process. If the message is not correct, the process returns to the state where it is waiting for messages.

The MSC expressions used in SRSL are mapped into SDL as follows: an *alt* expression produces several signal trigger states; a *loop* expression makes all next transitions return to the initial section state; an *opt* expression is implemented by a *continue* signal; and finally, an *exec* expression is translated into an asterisk state.

An SDL process is a finite state machine, and therefore it ends when it executes a stop statement, or provides a deadlock if no signal arrives. Our model has to explore all possibilities. Hence, we need to develop a mechanism to ensure that all signals sent must be processed. Consequently, we have added a state called "final" to indicate the end of the protocol execution, and a general transition composed of a common "save" statement and a continuous signal, with less priority than the input statement, that checks whether there are signals still waiting to be processed. By means of this structure we are transforming a finite state machine into an infinite one, just for analysis purposes.

At this point, if we instance the medium process with a "Router" process type, we can specify a security protocol in the same way as we might specify a traditional communication protocol and, therefore, we can analyze some of the liveness properties of the system in a traditional way. In the next subsection, we will explain how the security properties can be checked.

5.3. Model medium–observer processes

In our approach, the intruder's behavior is divided into two main aspects, the exploration algorithm and the check mechanism. The first one is provided by a medium process, while an observer process performs the check mechanisms.

We can consider two kinds of medium processes. The first one is characterized by an exploration mechanism that tries to explore all possibilities. It starts by examining all combinations of the different initial knowledge of each agent. Afterwards, it checks the concurrent agents' execution, by first trying combinations of two concurrent sessions, and so on. Our algorithm finishes when an "out of memory" is detected, or when it detects that the significant

intruder knowledge is not incremented. In general, the completeness problem [15] is undecidable. Thus, the fact that the algorithms terminate without having found a flaw is not a proof that it has not flaws.

The second kind of medium process uses an intruder process specialized in finding a specific flaw. If we are able to characterize a particular kind of attack, we can then evaluate the protocol trying to find such a specific flaw. Perhaps this is not the best solution in general (the only result we get is that a specific vulnerability does not occur in the cases we have examined), but it is very useful for a protocol designer that wants to be sure that the protocol is not vulnerable with respect to that kind of attack.

The state transition of the medium process is triggered when it receives any message. After reception, the message is stored into the intruder's knowledge database. The intruder then decides which operation performs next, and proceeds to the next routing state. We have used three different operations defined in AVIS project: *eavesdrop*, *redirect*, and *impersonate*. In an *eavesdrop* operation, the intruder intercepts the message but does not send it to any agent. A *redirect* operation means that the intruder intercepts the message but does not forward it to the original receiver. In an *impersonate* operation, the intruder sends a faked message to the original receiver.

The mechanisms used to prove security properties are the SDL Validator condition rules. These rules check different situations where protocol vulnerability is possible. The observer process carries out the checking mechanism. This is a special SDL process type that is evaluated in each transition of the protocol specification. It has access to all variables and states of all process instances, so we can test it automatically. In order to implement it we have to create an SDL *struct* sort with a "v" operator for each structured type that we want to evaluate. Fig. 5 shows an example of an observer process. We can see the condition rules for checking the authentication property, and the report result. The report contains a security failure MSC scenario.

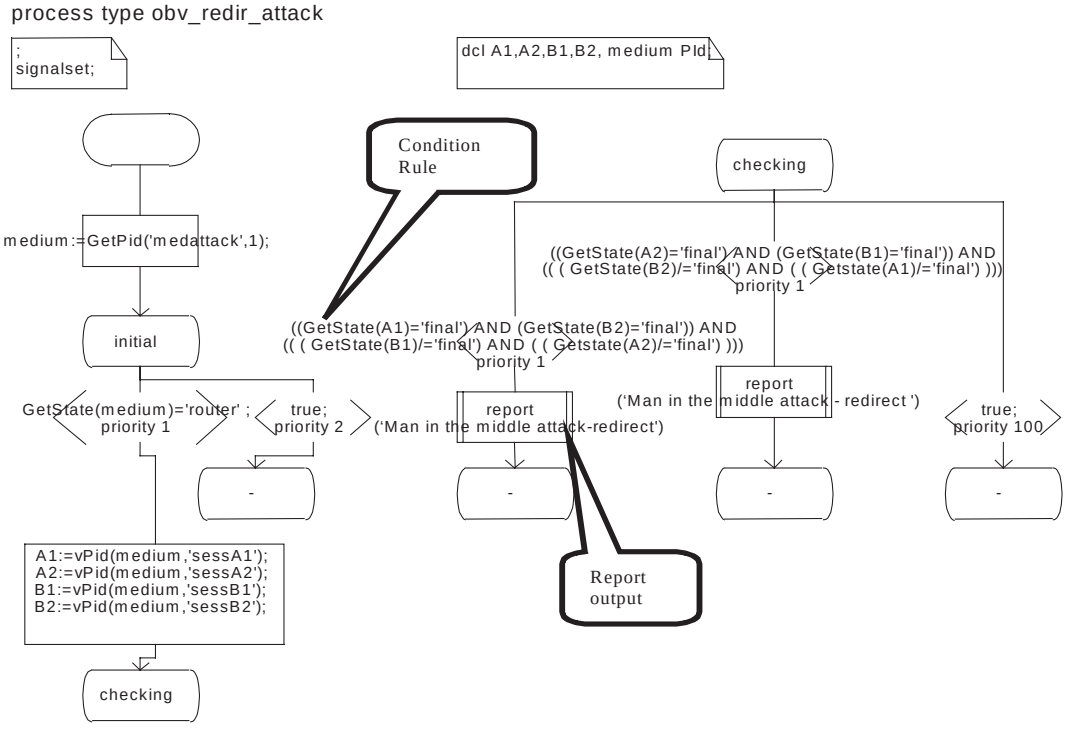

Fig. 5. Observer process "redirect" attack.

Currently, confidentiality, authentication, and non-repudiation of origin can be checked. For checking confidentiality, we examine whether a specific value (that we consider secret) can be deduced from the intruder's knowledge. Authentication is analyzed by checking that all the principal processes finish at the expected protocol step. Some authors [16] call this the correspondence (or precedence) property. Finally, non-repudiation of origin analysis consists of checking that, in the intruder's database knowledge, the evidence is signed digitally by the origin agent, and that it cannot be generated without this signature.

In order to validate our proposal we have carried out the analysis of some of the most classic cryptographic protocols, such as the Needham–Schroeder symmetric key, SSL, etc. In [11] we described the results of the analysis for the authentication protocol Encrypted Key Exchange (EKE) [4]. This protocol was specified in SRSL, using a well-known "man in the middle" attack evaluating two executions running in parallel sessions. The attack followed the *redirect* intruder's behavior. The resulting scenario produced by SDL Validator tool, describes a situation where only one of the two agents in each session has finished (agent A of the first session, and agent B of second one), but not the other one.

6. Conclusions and future work

We have presented a new analysis method for analysing and evaluating security protocols and their possible attacks. Security protocols are specified in SRSL, which can then be translated into a working SDL system. Attacks are implemented by SDL processes that specify the intruder's behavior and observer processes that check safety properties. One of the benefits of our approach is that protocol specifications are described independently from the analysis procedures, so they can be re-used in other environments as well. Furthermore, standard languages well known in telecommunication area are used, so training time is short.

Several kinds of security attacks can be analyzed using our approach. It is essential to study how they can be produced in a real environment. We examine the result scenario provided in an analysis procedure, and redesign the security protocol if necessary.

Currently, we are extending SRSL so more complex protocols can be specified, and other properties analyze. We are studying the use of MSC-2000 features as well as the new release of UML for specification purposes. Furthermore, we are developing a framework to implement protocol attacks in the Internet environment for testing.

Acknowledgements

The work described in this paper has been supported by the European Commission through the IST Programme under the research project IST-2001-32446 (CASENET), and by the Spanish Ministry of Science and Technology under the research project TIC-2003-8184-C02-01 (PRIVILEGE).

References

[1] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, L. Vigneron, The AVISS security protocol analysis tool, in: E. Brinksma, K.G. Larsen (Eds.), Proceedings of CAV'02, LNCS 2404, Springer-Verlag, 2002, pp. 349–353.
[2] R. Alur, T. Henzinger, F. Mang, S. Qadeer, S. Rajamani, S. Tasiran, Mocha: modularity in model checking, CAV 98: computer-aided verification, LNCS 1427, Springer-Verlag, 1998, pp. 521–525.
[3] M. Burrows, M. Abadi, R. Needham, A logic of authentication, Proceedings of the Royal Society, Series A 426 (1871) (1989) 233–271.
[4] S.M. Bellovin, M. Merrit, Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks, Proceedings of IEEE symposium on research in security and privacy, 1992, pp. 72–84.
[5] D. Dolev, A. Yao, On the security of public key protocols, IEEE Transactions on Information Theory IT-29 (1983) 198–208.
[6] ITU-T Recommendation Z.100 (11/99), Specification and description language (SDL), Geneva, 1999.
[7] ITU-T, Recommendation Z.120, Message sequence charts (MSC), Geneva, (1996).
[8] ITU-T, Recommendation Z.120 (11/99), Message sequence charts (MSC), Geneva, (1999).
[9] CCITT Recommendation X.800, Security architecture for open systems interconnection for CCITT applications, (1991).
[10] ITU-T Recommendation X.810 (ISO/IEC 10181-1), Information technology–open systems interconnection–security frameworks for open systems—overview, (1995).
[11] J. Lopez, J.J. Ortega, J.M. Troya, Protocol engineering applied to formal analysis of security systems, Infrasec'02, LNCS 2437, Bristol, UK, October 2002.

[12] W. Marrero, E. Clarke, S. Jha, Model checking for security protocols, DIMACS workshop on design and formal verification of security protocols, 1997.
[13] L. Paulson, The inductive approach to verifying cryptographic protocols, Journal of Computer Security 6 (1998).
[14] SPORE. Security protocol open repository. http://www.lsv.ens-cachan.fr/spore/.
[15] M. Rusinowich, M. Turuani, Protocol insecurity with finite number of sessions is NP-complete, 14th IEEE computer security foundations workshop June 11–13, 2001.
[16] P. Ryan, Scheneider, The Modelling and Analysis of Security Protocols: The CSP Approach, Addison-Wesley, 2001.

Javier Lopez received his MS and PhD in Computer Science in 1992 and 2000, from the University of Malaga, respectively. From 1991 to 1994, he worked as a System Analyst and in 1994 he joined the Computer Science Department at the University of Malaga as an Assistant Professor, where he currently is an Associate Professor. His research activities are mainly focused on information and network security, leading some national and international research projects in those areas. Prof. Lopez is the Spanish representative of the IFIP TC-11 (Security and Protection in Information Systems) Working Group.

Juan J. Ortega received his BSc and MSc degrees in Computer Engineering from the University of Malaga (Spain) in 1990 and 1995, respectively. Since 1997, he has been with the Computer Science Department, University of Malaga, where he is a part time Assistant Professor. His current research interests include formal methods applied to the evaluation of the security of distributed systems and computer networks.

Jose M. Troya received the MSc (1975) and PhD (1980) degrees in physics from the Universidad Complutense de Madrid. From 1980 to 1988, he was an associate professor at that university and, in 1988, became a full professor at the University of Malaga, where he leads the Software Engineering Group. His research interests include parallel programming, distributed systems, and software architectures. He is very involved in several national and international research projects, has written articles in the most relevant computer conferences and journals, supervised numerous PhD thesis, and organized several workshops and international conferences.