ISSN 0832-7203

An Automatic Validation Model for Security Mechanisms

Par : Fathya Zemmouri
Gilbert Babin
Peter Kropf

Cahier du GReSI no 04-03

Copyright © 2004. HEC Montréal.
Tous droits réservés pour tous pays. Toute traduction et toute reproduction sous quelque forme que ce soit est interdite.
HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal, Québec, H3T 2A7 Canada.
Les textes publiés dans la série des Cahiers du GReSI n'engagent que la responsabilit de leurs auteurs.

An Automatic Validation Model for Security Mechanisms

Fathya Zemmouri
Département d'informatique et de recherche opérationnelle
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal, Québec, Canada H3C 3J7
zemmourf@iro.umontreal.ca

Gilbert Babin
Service de l'enseignement des technologies de l'information
HEC Montréal
3000, chemin de la Côte-Ste-Catherine
Montréal, Québec, Canada H3T 2A7
tél: (514) 340-6291
fax: (514) 340-6132
Gilbert.Babin@hec.ca

Peter Kropf
Département d'informatique et de recherche opérationnelle
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal, Québec, Canada H3C 3J7
tél: (514) 343-2446
fax: (514) 343-5834
kropf@iro.umontreal.ca

Copyright © 2004. HEC Montréal.
Tous droits réservés pour tous pays. Toute traduction et toute reproduction sous quelque forme que ce soit est interdite.
HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal, Québec, H3T 2A7 Canada.
Les textes publiés dans la série des Cahiers du GReSI n'engagent que la responsabilit de leurs auteurs.

Abstract

In this paper, we propose a model to automatically validate the composition of multiple security mechanisms in the context of transactions between multiple participants. The goal of the validation is to demonstrate that the different security mechanisms preserve conformity to their specifications when used in combination. The underlying principle of the proposed approach is illustrated with a case study in the context of e-commerce. Specifically, we show the combined use of a digital signature, which provides data integrity and non-repudiation, a digital certificate, which guaranties authentification of participants, and a symmetric cryptographic algorithm, which ensures confidentiality.

Résumé

Cet article présente un modèle de validation automatique de la composition d'un ensemble de mécanismes de sécurité, utilisés pour sécuriser des transactions d'affaires entre plusieurs participants. L'objectif de cette validation est de démontrer que chaque mécanisme reste conforme aux spéficications, et ce, malgré le fait qu'il soit utilisé en conjonction avec d'autres mécanismes. Nous montrons comment l'approche de validation peut être utilisée à l'aide d'un exemple de transaction dans le contexte du commerce électronique. En particulier, l'approche servira à valider la composition d'une signature numérique, qui permet de vérifier l'intégrité des données et garantit la non-répudiation, d'un certificat numérique, qui garantit l'authentification des participants, et d'un algorithme de cryptographie à clé symmétrique, qui assure la confidentialité.

Mots clés

Sécurité, Validation de protocoles, Commerce électronique.

Copyright © 2004. HEC Montréal.

1 Introduction

The execution of secure electronic transactions between a number of participants requires a set of appropriate security mechanisms. There exist several methodologies to derive a solution that applies to the various trust problems, for example the SecAdvise advisor [5, 6]. The model proposed by SecAdvise, adapted from [4], defines a space of trust problems divided into subspaces of independent problems. The goal of SecAdvise is to provide an environment which integrates security mechanisms and which dynamically provides a trust solution that satisfies the security constraints required by the parts wishing to carry out a transaction without risk. The trust solution is a composition of trust units (a security mechanism or a security infrastructure) which reduces the risks under the current context/transaction to secure. However, before proposing a combination of units as a solution to reduce a whole set of risks, the following three steps should be considered:

1. validation of each unit belonging to the solution by validating its specification in a formal way, thereby contributing to the automation of the validation of the whole solution;
2. validation of the composition of units, which claims to cover a set of risks in a given environment. With this validation, one should prove that the composition is free from any intrinsic fault and that the security properties are checked;
3. detection of intrusion by simulating a validation model with an intruder.

In order to be able to carry out these steps, the following issues must be considered:

- *One has to prove that a trust unit claiming to cover a risk, respects its engagement.*
  Even by making the assumption that only the holder of the key can obtain the encrypted text (the assumption of perfect coding), attacks by usurpation of identity abound [1]. The aspects to be checked are of logical type. For example, the *Needham-Schroeder* public key protocol [2] was employed for more than fifteen years before it was attacked.
- *given two trust units* u_1[1] *and* u_2 *covering respectively two disjoined risks* r_1 *and* r_2*, does* $u_1 \cup u_2$ *really cover* $r_1 \cup r_2$ *?*
  A key question is whether or not the introduction of another mechanism affects the effectiveness of the first. The problem arises because $u_1 \cup u_2$ is in fact a new protocol and the development of a new protocol should be validated to correct any fault in the rules of procedures.

2 Case study

To illustrate the concepts introduced in the previous section, in the following section, we discuss a particular environment. Our case study consists of the validation of the union of four security mechanisms to cover a set of security risks. We suppose that two participants, a supplier and a customer, want to perform an electronic transaction and that they want to authenticate each other.

[1] We use the notation defined in [5] and refined in [6] to identify elements of the security problem and solution.

Copyright © 2004. HEC Montréal.

They wish to verify that the contents of the data exchanged remain confidential and unchanged, and that it is possible to prove non-repudiation on both sides. We use the PROMELA language for the formal specification and SPIN as the tool to perform automatic testing.

There are four security risks to cover in this context/transaction, namely:

- r_1: eavesdropping risk,
- r_2: tampering risk,
- r_3: message forgery risk,
- r_4: transaction refutation risk.

We assume that the following trust units were selected by a method that carries out all necessary calculations (transaction cost, maximum security risk reduction):

- u_1: the SSL protocol to cover $r_1 \cup r_2 \cup r_3$,
- u_2: a digital signature to cover r_4.

In fact, u_1 (SSL) is a composition of three atomic security mechanisms, cryptography to cover r_1, a digital certificate to cover r_2 and a hash function to cover r_3. We therefore have $u_1 = \{u'_1, u'_2, u'_3\}$.

The two transaction participants are referred to as client and server. The context/transaction is noted c. We have:

- $r_1, r_2, r_3, r_4 \in R$, where R is the set of security risks,
- $u_1 \in \mathcal{P}(\mathbf{U})$, $u_2 \in \mathbf{U}$, where $\mathbf{U}$ is the set of trust units (security mechanisms),
- r_1, r_2, $r_3 \in R_{u_1}$ and $r_4 \in R_{u_2}$, where R_{u_i} is the set of security risks associated to the trust unit u_i,
- $R_c = \{r_1, r_2, r_3, r_4\}$,
- $P_c = \{client, server\}$, the set of participants in the context/transaction c,
- to simplify the model of validation, we assume that the client and the server trust the same certification authorities.

We assume, without proving it, that $\tilde{u}_c = u_1 \cup u_2$, where $\tilde{u}_c$ denotes the trust solution selected to secure the context/transaction c, because our goal is not to demonstrate that u_1 and u_2 compose the right trust solution in context c.

Copyright © 2004. HEC Montréal.

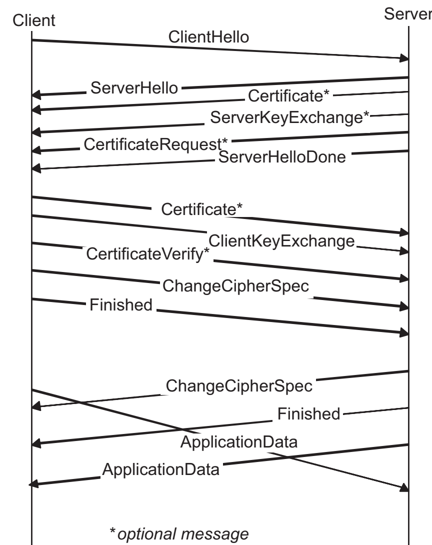

Figure 1: Messages exchanged when SSL opens a new session.

3 SSL validation model

As stated earlier, SSL covers risks r_1, r_2 and r_3 by providing the following three security services:

connexion confidentiality by using symmetric ciphering. SSL can negociate symmetric algorithms like DES, RC4, etc.;

authentification and identification of transaction participants. SSL uses public key cryptography. Different algorithms can be used: RSA, DSS, etc;

Data integrity. SSL adds a hashcode, called a *MAC* to the data exchanged to verify its integrity. Hash functions such as SHA or MD5 are used to compute this *MAC*.

SSL splits up the message to be transmitted in data blocks. It then compresses them, applies a *MAC*, and transmits the results to the other communication point. Messages received by the other pair are then deciphered, checked, decompressed, and assembled.

3.1 SSL messages

Figure 1 illustrates the messages exchanged between a client and a server when SSL opens a new session. The messages exchanged, according to their chronological order are:

HelloRequest Notification that the client should begin the negotiation process anew by sending a client hello message;

ClientHello Contains:

- the version of the SSL protocol by which the client wishes to communicate during this session,

- a client-generated random structure (*client_random*),
- the ID of the session the client wishes to use for this connection (*session_ID*),
- a list of the cryptographic options supported by the client, sorted with the client's first preference first (*cipher_suites*),
- a list of the compression methods supported by the client, sorted by client preference (*compression_methods*).

ServerHello Contains:

- the version of the SSL protocol supported by the server,
- a random number (*server_random*),
- the identifier of session (*session_ID*),
- the single cipher suite selected by the server from *ClientHello.cipher_suites*.
- the single compression algorithm selected by the server from *ClientHello.compression_methods*.

Certificate Contains the server's or client's (if the server claims it and the client has one) digital certificate. It must be a X.509 certificate version 3.0 which contains the server's public key contained in the *ServerHello* message. A X.509 certificate has the following form:

$$\begin{aligned} Certificate =& Entity_Name + CA_Name + Entity_PK + \\ & CA_SK\{H[Entity_Name + CA_Name + Entity_PK]\} \end{aligned}$$

where:

$Entity_Name$ is the server's or client's name,

CA_Name is certification authority's name,

$Entity_PK$ is an entity's public key, this is the key to be certified,

$CA_SK\{x\}$ is the authenticator of x, computed by ciphering x with certification authority's private (secret) key,

$H[y]$ hashcode of y;

ServerKeyExchange Sent by the server only if it has no digital certificate, or has only a certificate for digital signature;

CertificateRequest Sent by the server to request a certificate from the client;

ServerHelloDone Indicates that the server has finished sending *ServerHello* and subsequent messages;

ClientKeyExchange Contains the *PreMasterSecret*, encrypted with the server's public key;

CertificateVerify Used to provide the server with explicit verification of a client certificate. This message contains the following value:
$Hash\{MasterSecret + Hash(handshake_messages + MasterSecret)\}$
where *Hash* is either the MD5 or SHA algorithm; *handshake_messages* refers to all handshake messages starting from *ClientHello* up to but not including this message;

Finished Always sent immediately after a *ChangeCipherSpec* message to verify that the key exchange and authentication processes were successful. The *Finished* message is the first message protected with the just-negotiated algorithms, keys, and secrets. The *Finished* message contains *handshake_messages* which include all handshake messages starting at *ClientHello* up to, but not including, this *Finished* message.

Recipients of *Finished* messages must verify that the contents are correct and not altered by an intruder, by comparing its content with messages received so far. The handshake concludes with the server sending the *ChangeCipherSpec* and *Finished* messages.

3.2 SSL formal specification with PROMELA

A formal model of SSL's cryptographic protocol must include two participants (*client* and *server*) who exchange messages in conformance with the protocol rules. The aim of such a model is to help to isolate security faults in the protocol itself, if they exist, and not in the cryptographic system used by SSL. Therefore, we assume that the cryptographic system used by SSL is perfect, its modeling remaining abstract. In other words, we suppose that:

- the only way to decipher a message is to have the key with which it was ciphered,
- an encoded message cannot reveal the key with which it was ciphered, and
- the ciphered message has enough redundancy so that the decoding algorithm can detect if the message was ciphered with the right key,

It is thus possible that the model may not reveal other vulnerabilities caused by the cryptographic system used by SSL. In our model, the client and the server are represented by PROMELA processes, communicating via shared channels.

3.2.1 Variables definition

A variable can be any identifier, random key, nonce or data used by the protocol. The set of variables was obtained after a thorough analysis of the SSL specification. We have identified the following variables (defined using PROMELA):

```
mtype = {
    client, server, intruder, hello_request,
    client_hello, server_hello, client_certificate,
    server_certificate, certificate_request,
    server_hello_done, pre_master_secret,
```

handshake_messages_hash, finished_hash,
handshake_message_master_secret

}

3.2.2 Channel definition

Channels are the communication mechanism between PROMELA processes. We defined a channel type for each message format used by SSL. SSL messages have the following formats:

- *HelloRequest*{*server*},
- *ClientHello*{*client, client_hello*},
- *ServerHello*{*server, server_hello*},
- *Certificate*{*client, client_certificate,*{*client_certificate*}*SK*{*client*}},
- *Certificate*{*server, server_certificate,*{*server_certificate*}*SK*{*server*}},
- *CertificateRequest*{*server, certificate_request*},
- *ServerHelloDone*{*server, server_hello_done*},
- *ClientKeyExchange*{{*pre_master_secret*}*PK*{*server*}},
- *CertificateVerify*{*client,* {{*handshake_messages*}*HASH*}*SK*{*client*}},
- *Finished* {*client,* {*handshake_messages, client, masterSecret*}*HASH*},
- *Finished* {*server,* {*handshake_messages, server, masterSecret*}*HASH*}.

In order to validate the protocol, we assume that every message transmitted by *client* or *server* is intercepted by *intruder*, then retransmitted to the appropriate destination (i.e., the intruder can listen on every communication channel). To achieve this, every message sent includes the identity of the participant concerned with the transmission, the other participant always being *intruder*. Thus we need the following channels:

chan HelloRequest **= [**0**] of** {**mtype**};
chan ClientHello **= [**0**] of** {**mtype**, **mtype**};
chan ServerHello **= [**0**] of** {**mtype**, **mtype**};
chan Certificate **= [**0**] of** {**mtype**, **mtype**, **mtype**, **mtype**};
chan CertificateRequest **= [**0**] of** {**mtype**, **mtype**};
chan ServerHelloDone **= [**0**] of** {**mtype**, **mtype**};
chan ClientKeyExchange **= [**0**] of** {**mtype**, **mtype**};
chan CertificateVerify **= [**0**] of** {**mtype**, **mtype**};
chan Finished **= [**0**] of** {**mtype**, **mtype**};
chan NoCertificate **= [**0**] of** {**mtype**};
chan ChangeCipherSpec **= [**0**] of** {**mtype**}.

Copyright © 2004. HEC Montréal.

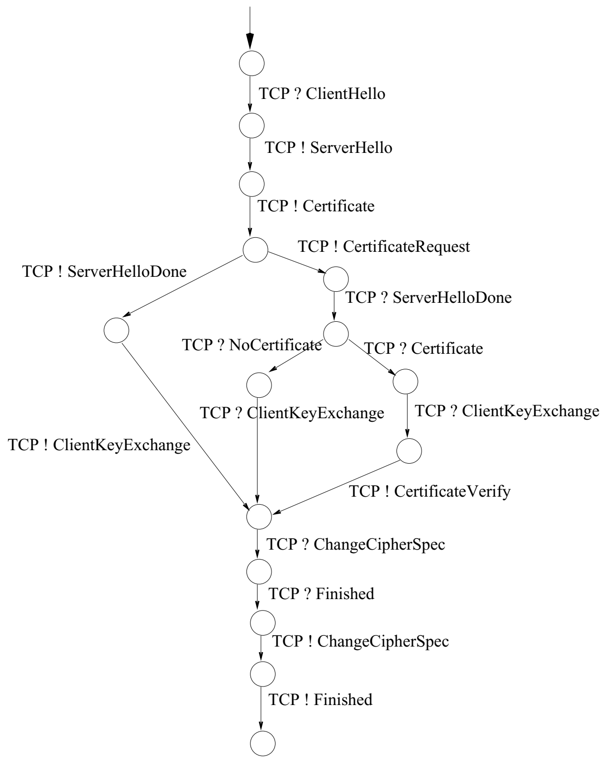

Figure 2: Server EFSM of the composition SSL/difital signature.

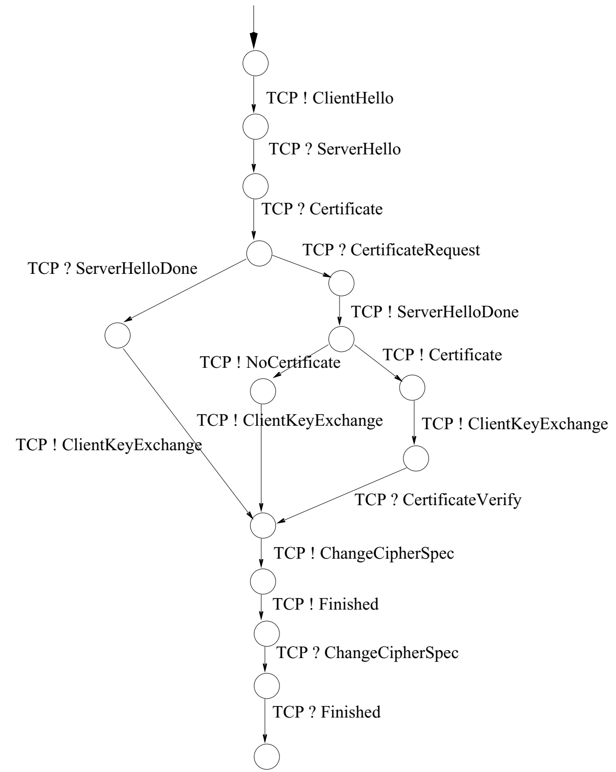

Figure 3: Client EFSM of the composition SSL/difital signature.

These channels are declared as global variables so that all processes may access them.

For example, if the *server* process sends the message *PK(client)*{*message*}, the following instruction is used:

c2pk **!** server, message, client

where *server* is the sender, *message* and *client* are the message body (*client* represents the public key *PK(client)*). The message reception by a particpant is similarly expressed by:

c2pk **? eval**(server), x1, **eval**(client).

This instruction will only accept as valid, messages with receiver *server* (this test is used to make sure that the message is indeed addressed to *server*) and with *client* as the value of the third field.

3.2.3 Server and client processes

Processes representing SSL participants must be parameterizable with data that change from one session to another or from one instance to another. Their definition must also include conditions representing protocol governing rules, which must be checked or validated.

These processes are illustrated in Figure 2 (server process) and 3 (client process), using extended finite state machines (EFSMs). The PROMELA code was generated from these EFSMs.

3.3 Defining Correctness Claims

3.3.1 Authentification checking

We express the fact that *server* is correctly authenticated by *client* with the following predicate: *client* commits to a session with *server* only if *server* accepts to share a protocol instance with *client*.

A similar predicate expresses the reciprocal property, that *client* was correctly authenticated by *server*.

Each of these predicates may be represented using a number of PROMELA global boolean variables, which becomes *true* during protocol execution. These are:

bit PclientInstance = 0;
bit PclientCommit = 0;
bit PserverInstance = 0;
bit PserverCommit = 0;

PclientInstance is *true* if, and only if, *client* participates in a session of the protocol with *server*. PserverInstance is *true* if, and only if, *server* participates in a session of the protocol with *client*. PclientCommit is *true* if, and only if, *client* commits to a session with *server*. PserverCommit is *true* if, and only if, *server* commits to a session with *client*.

Hence, *server* authentification by *client* can then be checked by making sure that *PserverInstance* becomes *true* before *PclientCommit*. In a similar way, the authentification of *client* by *server* can be checked by making sure that *PclientInstance* becomes *true* before *PserverCommit*. We used the precedence property, expressed as follows, to check the authentification of *client* to *server* and the *server* to *client*:

$$(G((G\neg PclientCommit) \vee (\neg PclientCommit U PserverInstance))) \wedge (G((G\neg PserverCommit) \vee (\neg PserverCommit U PclientInstance)))$$

3.3.2 Secrecy checking

The confidentiality criteria can only be validated with the presence of an *intruder* process. This is done by checking that the message exchanged between *client* and *server* does not become part of the *intruder* knowledge. This criteria must be checked at all states of the state space.

In the *intruder* process, we declare a boolean *message* which is initialized to *false*. This variable becomes *true* if the *intruder* manages to intercept the contents of the message exchanged between the two parties involved in the communication. Consequently, to verify confidentiality of the data exchanged in any state belonging to the state space, we simply use the following LTL formula:

$$G\neg message$$

3.3.3 Integrity checking

Integrity is checked by comparing the message sent by *client* (or *server*) with that received by *server* (or by *client*).

3.3.4 Validation results

The properties to validate exposed so far, that is authentification, integrity, confidentiality and non-repudiation, are positive in all the state space.

Tests automatically carried out by SPIN are also positive: never-claim, assertion violations, cycle checks, and invalid endstates, which express the correctness claims of the protocol, apart from the security services it provides.

4 The intruder process

The aim of the validation steps presented so far was to analyze the composition {SSL, digital signature} in the absence of an intruder. This validation is necessary for any trust unit composition, presented as a solution to a context/transaction.

Indeed, each composition of trust units constitutes in essence a new protocol. Therefore, we need to demonstrate that this new protocol is free from any intrinsic fault. However, validation is not complete until we introduce the *intruder* model. To assure reliability, the generation of the *intruder* process must be automatic. In other words, starting from the specification of a set of trust units, we should be able to automatically generate the behavior of an illegitimate agent, represented by an *intruder* process.

4.1 Intruder knowledge

The intruder must interact with the legitimate participants of a context/transaction, according to the ability the validation model provides. It must also be able to behave like a normal user of an open network.

At any point in time, the intruder's behavior depends on knowledge acquired until that point. Before each transaction execution, we assume that the intruder knows a set of data, such as the intruder's identity, his public key, the identity of other participants, their public key, and some secret keys which he already shared with other participants.

Every time he intercepts a message, the intruder can increase his knowledge. Indeed, if the intercepted message is ciphered by a key known to the intruder, he can decipher it and take note of its contents. On the other hand, if he is able to decipher the message, he will memorize the whole ciphered message. This step makes it possible to build a powerful intruder who may extract the maximum of information from the intercepted messages.

In addition to messages intercepted and information extracted, the intruder can forge other messages. This capability can be represented by adding to the intruder's knowledge data generated from messages intercepted in combination with data already acquired. However, in order to restrict the space of messages which can be generated by the intruder, we can exclude messages that could not be accepted by legitimate participants in the context/transaction to validate.

Our goal is to automate the intruder's behavior generation, starting from the trust unit's formal specification. By analyzing the method used in [3] to manually generate the intruder's behavior in order to detect a security attack on the *Needham-Shroeder* protocol, we were able to construct an EFSM for the intruder, starting from the trust units' EFSMs. The construction of the EFSM is described in the next section.

Copyright © 2004. HEC Montréal.

The initial set of input of the intruder's EFSM is obtained by the following two analysis steps. We first must determine the possible set of values that each variable can take for each trust unit process. This operation can be realized by a data flow analysis. Variables not tested in a process can take any value, whereas variables which are tested in conditional expressions may only take values which have been tested.

Second, we carry out a static analysis to restrict the domain of the intruder's knowledge to the minimum needed. In the first place, we need to determine the initial knowledge set. For example, in the case of a transactional protocol, this set is limited to:

- the set of all participants' public keys,
- the intruder's private key,
- the set of participants' digital certificates, and
- some generic data.

This set can see its contents increase when the intruder intercepts messages. The messages the intruder can intercept during a protocol execution determine the contents it can add to his knowledge base. If the intruder intercepts the message {*m_client, client*}*PK(intruder)*, it can add the element *m_client* to his knowledge.

To avoid redundancy of elements learned, the intruder saves the element learned in its most elementary form. For example, if the message *m_client, clientPK(intruder)* is intercepted, the intruder process saves *m_client*, and not the whole message, since it can be built from *m_client* and the identifier *client*, which already belong to the intruder's knowledge. In other words, the intruder process saves the message in its most complex form only if it cannot decipher it.

Since the set of messages the intruder can intercept in an execution protocol instance is finite, the set of messages the intruder can add to his knowledge base is also finite. Thus the intruder's EFSM is a finite nondeterministic automata, finite because it will have a finite Input set and non-deterministic because there is a multitude of decisions the intruder can make when intercepting a new knowledge element.

Let $RECEIVED$ be the set of all possible messages the intruder can intercept, and for each message $m \in RECEIVED$, let $LEARNED_m$ be the elements the intruder can learn from m.

The intruder's knowledge may be further restricted by excluding messages which can never be used by the intruder to generate valid messages (messages accepted by legitimate participants). Therefore, the set of elements useful to the intruder (named $UTIL$) can be derived from valid messages the intruder could send to the other partcipants. For each message of this set, we have to determine elementary data the intruder could use to build such message. Let $NeedKnowledge_m$ be the set of elements the intruder can use to build a useful message $m \in UTIL$.

Thus, the set of elements the intruder can know and use to attack an instance of protocol execution (named $varUtil$) is defined as follow:

$$varUtil = \bigcup_{m \in RECEIVED} LEARNED_m \bigcap_{m' \in UTIL} NeedKnowledge_{m'}$$

Copyright © 2004. HEC Montréal.

4.2 Formal intruder modelisation by an EFSM

Let $efsm_1 = (Q, q_0, M, A, T)$ and $efsm_2 = (Q', q'_0, M', A', T')$ be two EFSMs that represent processes of two participants of a context/transaction c. The two participants communicate by message queues belonging to M and M'. Therefore, we suppose that $M = M'$. We derive from $efsm_1$ and $efsm_2$ the following EFSM representing the intruder's behavior:

Let $efsm^i = (Q^i, q^i_0, M^i, A^i, T^i)$ be the intruder EFSM.

M^i The set of message queues the intruder can access. It must be the same as those of the two participants. Since we suppose the communication network to be open, $M^i = M = M'$.

A^i The set of variable names that the intruder can manipulate. As the intruder's goal is to intercept secret elements exchanged between participants, A^i is composed of two subsets: A_1 and A_2.

A_1 Variables initially known to the intruder.

A_2 Variables not yet known by the intruder and needed by the intruder to complete his knowledge base. We have thus $A_2 = (A \cup A') \cap A_1$.

We generate the intruder's EFSM in order to validate security properties of a composite trust solution using our approach (see Sect. 3). We suppose that all A^i variables are of boolean type. They take the value *true* if the intruder intercepts them and *false* if the intruder does not yet know their value. We initialize variables in A_1 to *true* and variables in A_2 to *false*.

Q^i There are five possible intruder states $q^i_0, q^i_1, q^i_2, q^i_3, q^i_4$.

q_0 The initial state.

q_1 The intruder state after he initializes all A^i variables.

q_2 The intruder state when a new element is added to the intruder knowledge.

q_3 The intruder state when a read operation is executable on a channel belonging to M.

q_4 The intruder state when a message intercepted belongs to $varUtil$.

T^i Transition function defined by:

$T^i(q^i_0, a) = q^i_1$. $efsm^i$ moves from state q^i_0 to q^i_1 by performing an initialization action a.

$T^i(q^i_1, a) = q^i_1$. $efsm^i$ stays in q^i_1 by performing a write action a on a message queue of M. This is equivalent to an intrusion attack by identity usurpation or a message deterioration.

$T^i(q^i_1, a) = q^i_2$. $efsm^i$ moves from q^i_1 to q^i_2 by performing a testing action a.

$T^i(q^i_2, a) = q^i_1$. $efsm^i$ moves from q^i_2 to q^i_1 by performing a writing action on a message queue.

$T^i(q^i_1, a) = q^i_3$. $efsm^i$ moves from q^i_1 to q^i_3 by performing a reading action on a message queue of M. This is equivalent to a passive listener intruder.

$T^i(q^i_3, a) = q^i_4$. $efsm^i$ moves from q^i_3 to q^i_4 by performing a testing action. The test is executable if the variable belongs to $varUtil$.

Copyright © 2004. HEC Montréal.

$T^i(q_4^i, a) = q_1^i$. $efsm^i$ moves from q_4^i to q_1^i by performing a variable assignement action a, the variable must belong to A^i.

5 The SecAdvise validation methodology

The SecAdvise approach selects trust units (i.e., security mechanisms) as a solution to a specific trust problem. The validation model we presented in the previous two sections has been developed as one of the criteria for solution selection. Indeed, any solution selection tool must make sure that the composition of trust units selected covers the whole risk set. In this paper, we showed how the PROMELA formal specification language and the SPIN validation tool can be used to perform such analysis. In order to generalize our approach to any trust solution, we introduce the following notation:

$\mathbf{EFSM}$ the set of extended finite state machines representing the trust units in the set $\mathbf{U}$. This set establishes a link between a trust unit $u \in \mathbf{U}$ and its EFSM representation. It formalizes the trust unit's specifications and makes the transition from a formal specification to a model validation automatic.

$efsm_u$ $\in \mathbf{EFSM}$ is an EFSM representing trust unit u. Thus, $efsm_u \in \mathbf{EFSM}$ and $efsm_U = \bigcup_{u \in U} efsm_u$ such that $U \in \mathcal{P}(\mathbf{U})$;

$\mathbf{D}$ the set of predicates associated to the set of risks $\mathbf{R}$. For each risk, we associate a set of predicates which will be used in verification. These predicates can be LTL formulae or may use other notations. Using these predicates, we can verify if a security risk is properly covered by one or more trust units. These verification predicates are independent of any trust unit that can cover the risk they describe. Thus, every trust unit that covers a given risk will be validated using the same predicates.

D_r $\in \mathcal{P}(\mathbf{D})$ the set of predicates associated to risk r;

$D_{\mathbf{R}_c}$ $\in \mathcal{P}(\mathbf{D})$ the set of predicates associated to all risks in the set $\mathbf{R}_c$ (the set of risks to be covered in a context/transaction c);

$\mathbf{I}$ $\in \mathcal{P}(\mathbf{EFSM})$ the set of EFSMs describing an intruder's behavior;

i_U $\in \mathbf{I}$ the EFSM associated with the intruder's behavior in order to validate the set of trust units U. This EFSM is automatically generated as described in the previous section of this paper.

Given these modifications to the SecAdvise theoretical model, we can summarize the two verification steps proposed herein as follows:

1. the verification of every predicate in $D_{\mathbf{R}_c}$ in every state of the automata $efsm_{\tilde{U}_c}$ and
2. the verification of every predicate in $D_{\mathbf{R}_c}$ in every state of the automata $i_{\tilde{U}_c}$ generated from $efsm_{\tilde{U}_c}$.

The first validation step tries to verify whether a security risk $r \in R_c$ is covered or not by the trust solution suggested by SecAdvise, that is, set $\tilde{U}_c$, whereas the second validation step makes sure that all risks in R_c are covered by $\tilde{U}_c$ in the case where an intruder is present.

Copyright © 2004. HEC Montréal.

6 Conclusions

We have formally described the automatic generation of an intruder's behavior with the goal of validating a given trust solution. The behavior of such an intruder must take into account the structure of processes representing the behavior of legitimate participants. Moreover, we must also automatically generate the whole set of the intruder's initial assumptions. This set must then be expanded accordingly, as the intruder intercepts and decodes messages exchanged between the different legitimate participants, with the help of macros updating knowledge variables.

References

[1] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0. A continually updated library of protocols analysed in the literature, available at www.cs.york.ac.uk/ jac, November 1997.

[2] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. In *Information Processing letters*, volume 56(3), pages 131–133, 1996.

[3] P. Maggi and R. Sisto. Using Spin to Verify Security Properties of Cryptographic Protocols. In *LNCS*. Springer Verlag, 2002.

[4] S. Robles, S. Poslad, J. Borrell, and J. Bigham. A Practical Trust Model for Agent-Oriented Electronic Business Applications. In *Proc. of the 4th Int'l Conf. on Electronic Commerce Research (ICECR-4)*, volume 2, pages 397–406, Dallas, Texas, USA, November 2001.

[5] R. Saliba, G. Babin, and P. Kropf. SecAdvise : A Security Mechanism Advisor. In *Distributed Communities on the Web (DCW 2002)*, LNCS 2468, pages 35–40, Sydney, Australia, April 2002. Springer Verlag.

[6] F. Zemmouri. Un modèle de validation automatique de mécanismes de sécurisation des communisations. Master's thesis, University of Montreal, 2003.

Copyright © 2004. HEC Montréal.