MODELING AND VALIDATION OF SECURITY PROTOCOLS
By using The Object Oriented Framework and SPIN

David Kung, Wesam Okasha, Ashraf Ali, Sheila Nwokoro, Najeebuddin Mohammad
kung@cse.uta.edu

Technical Report CSE-2003-16
Table of Contents

Abstract ..............................................................................................................................3
Introduction ....................................................................................................................3
Related Work ................................................................................................................4
UML Background .......................................................................................................6
Promela Background .................................................................................................8
Approach .....................................................................................................................8
Validation .....................................................................................................................9
Conclusion & Future Work .........................................................................................19
References ..................................................................................................................19
Abstract

The Internet has revolutionized the world in every aspect of life. The most important one is of course the huge number of online businesses. Transactions worth billions of dollars are going on the Internet. This places an immediate need to develop protocols that are secure, secure in the sense that nobody, wishing to cause harm, can break into these protocols. Considering this need, a number of computer scientists and mathematicians have developed numerous security protocols. As obvious, before implementing these protocols on the Internet, these protocols are first verified and analyzed to find the flaws and many of these protocols are found to be flawed! A number of different techniques are available for the verification of security protocols. This paper presents a three-step method of modeling and verifying a security protocol using the Object Oriented Frameworks and SPIN. A case study on the Yahalom Security Protocol is also presented to outline the implementation of the proposed method.

Introduction

There is hardly any field of life that is not affected by the Internet. The Internet is used by people all over the world for many numbers of applications. One of the most important applications is those that involve money. These financial transactions should be protected from malicious attacks. This is not an easy task, for two reasons. 1) The Internet is an open environment that is available to everybody in this world. 2) Security protocols developed for secure transactions are hard to be proved as unflawed and secure. Moreover, a security protocols is essentially a few lines of a specification in a program and its analysis can be considered as similar to those of computer programs. In the case of a computer program, the user wants to avoid potential bugs while for a security protocol the intruder wants to exploit the bugs therefore it is very necessary for the security protocols to be bug free. The methods available for validating Security Protocols are mostly complex.

A Security Protocol is the conventions and rules by which different computers over a network communicate with each other and with the users of these computers. There are two main types of Security Protocols: Authentication Protocols and Authorization Protocols. Authentication Protocols have the responsibility of verifying the identity of a given principal while Authorization Protocol determines whether a principal or process has the right to execute a certain activity.

A number of researchers have worked on the issues related modeling, specification and verification of security protocols. However no one can claim that his or her method is flawless. Hoare discussed a method for analyzing Security Protocols based on process algebra called Communicating Sequential Processes (CSP) [Hoare 85]. Rosco extended this by developing a model checker for the same CSP [Roscoe 94]. A number of researchers have used the two methods and found them to be very successful but the drawback is that the
task of producing the CSP description of the system is very time consuming and does require a good knowledge of CSP. In addition, it has been founded by different studies that even the experts in CSP have made mistakes. In order to overcome these difficulties Gavin proposes a compiler CASPER for the analysis of security protocols but the CASPER does not cover all parts of the Security Protocols. Other methods used for verifying Security Protocols are Model Checkers and Theorem Provers. An interesting work in analyzing security protocols has been done by Alec by using a tool called CPAL-ES [Alec 01]. However the modeling of control within this tool is not robust.

In this paper, it is proposed to use Object Oriented (OO) Software Engineering and UML to specify, verify and analyze a security protocol. Object oriented software engineering has been used for the development of a huge pool of software projects and it has given birth to Unified Modeling Language (UML). The success and popularity of the UML and the Object Oriented paradigm has drawn the attention of many researchers to use the same principles for the verification of Security Protocols and they have shown promising results [Kung 02].

This paper fully agrees to this pool of researchers and also applies the principles of OO and UML to model and specify security protocols. The modeling is started with a class diagram of the security protocol and identifies classes like Principal, Initiator, Server, and Intruder etc. This class diagram is then translated into a sequence diagram and from the sequence diagram; a state machine is derived from it. The state machine is then translated into a promela program that is used to verify the specific security protocol.

Related Work

As mentioned, a number of researchers have proposed numerous methods and tools for the verification and analysis of Security Protocols. Hoare proposed the use of process algebra Communicating Sequential Processes (CSP) for analyzing Security Protocols and Lowe developed a model checker called Failures Divergence Refinement (FDR) to be used on CPS. They both modeled each agent as well as a general intruder in the protocol as CSP processes. Then, the resulting system is tested against specifications representing desired properties such as authentication and secrecy.

The FDR tests the protocol by searching the state space to investigate whether any insecure traces can occur and if it finds that the specification is not met, it shows that the system as being attacked. Although this method proved to be very successful in finding attacks on a large number of systems, the task of producing the CSP description of the system was found to be very time consuming and suitable for those proficient in CSP. In addition, an interesting observation was made when those proficient in CSP made a lot of mistakes. So it was highly desired to automate the process of generating the CSP description.

Lowe proposed a compiler called CASPER for the analysis of Security Protocols [Lowe 96]. Using this tool, the user specifies the protocol by giving an abstract notation that is then compiled by into CSP code by CASPER. The code
generated by CASPER is found to be suitable for model checking using. However it has been found that some classes of protocols cannot be analyzed by CASPER because those protocols don’t contain any linear sequence of messages. Neuman discusses the issues relating to such protocols [Neuman 93].

Burrows has discussed the use of belief logic for the analysis of Authentication Protocols [Burrows 93]. The belief logics formally express what an agent may infer from the messages it receives. The beliefs of different principals can be expressed in BAN logic and theses beliefs even change as the principal receives a message. These beliefs can change according to a set of inference rules. The original BAN logic allowed short proofs, which are expressed at a higher level of abstraction. This approach has identified weaknesses in some protocols but it has failed to identify serious weaknesses in other protocols. In order to solve these problems, a number of researchers have worked on belief logic, but still these approaches do not solve the problem of secrecy and even some of them are hard to put on a scientific footing. These researches including Boyd, has shown that BAN logic is not fully formal and this consequently permits approval of dangerous protocols [Boyd 93].

To Address these issues, Paulson has proposed a very interesting approach for proving the properties of a Security Protocols and that approach is “Indution” [Paulson 97]. Paulson admits that when a protocol is verified this way, it will require informal justifications, which involve arguments backwards, that certain events are impossible. The resulting proofs may be quite complicated but can be quickly generated by using the proof tool Isabelle/HOL. According to Paulson, this method is not restricted to finite state systems and the approach is not based on belief logics, instead this approach has fruitfully borrowed elements from both approaches. From the state exploration methods, it has borrowed the concrete notion of events such as “A sending X to B” and from the belief logics, this method has borrowed the idea of deriving guarantees from each protocol message. In this method, induction protocols are formalized as a set of all possible traces of events and can also model accidents and attacks. The properties are proved by induction on traces and these proofs are much too long to carry out on paper therefore a proof tool Isabelle is used. A complete Isabelle script can be executed in seconds. Proving Properties of Security Protocols using this way is oriented around proving guarantees but their absence can indicate possible attacks.

This approach has been applied to several key distribution protocols including Needham-Schroeder, Yahalom, and Otway Rees. It has been found useful for both symmetric key and public key protocols. This approach has drawbacks in the sense that it assumes that encryption is strong and no body can break it. Furthermore, it assumes that there is no confusion between items of different items e.g. between a key and nonce.
UML Background

Terry Quatranni in her famous book [Quatranni 00] has pointed out that for a successful project, three things are necessary; 1) Process 2) Tool and 3) Notation. The Notation used almost everywhere in Software Industry for OO development is UML. The UML is “unified” because of the following reasons [Rumbaugh 99]

1) The UML has combined commonly accepted concepts from many object oriented methods and has selected clear definition, notation and terminology for each concept.

2) The UML can be used at any stage of the Software lifecycle from requirements analysis to deployment. Furthermore it can be used for different types of Software process models e.g. iterative, incremental etc.

3) The UML can be used to model most application domains, which include large, complex, real time, distributed and computation intensive systems.

4) It can be used for systems implemented in various programming languages.

It should be pointed out that UML is not a programming language. It is a modeling language, and as Rumbaugh has noted in his book [Rumbaugh 99] it is a discrete modeling language that may not prove to be useful in modeling continuous systems such as those found in Physics. Now a question arises, if UML is a modeling language, then what does a model mean? The second question usually asked is why do we need to model. Gary Booch [Rambaugh 99] has answered these questions below.

(1) “A model is a simplification of a reality” A model can provide the blueprints of a system. A good model is supposed to include those elements that have broad effect and omit those elements that are not relevant to the given level of abstraction.

(2) “We build models so that we can better understand the system we are developing” and “we build models of complex systems because we cannot comprehend such a system in its entirety”

Building Blocks of UML

The UML consists of three types of building blocks [Rambaugh 99]

1) Things
2) Relationships
3) Diagrams

Things in the UML can be of four types 1) Structural things 2) Behavioral things 3) Grouping things 4) Annotational things. Structural things are the nouns of the UML diagram and Behavioral things are the dynamic parts of the UML models. Grouping things are the organizational...
parts of the UML model and Annotational things are the explanatory parts of
the UML models.

**Relationships** in the UML are of four kinds 1) Dependency 2) Association 3)
Generalization 4) Realization

**Diagrams** in the UML are of nine kinds.

1) **Class Diagram**: This diagram shows a set of classes, interfaces and
collaborations and their relationships

2) **Object Diagram**: This type of diagram shows a set of objects and their
relationships

3) **Use Case Diagram**: A use case diagram shows a set of use cases and
actors and their relationships

4) **Sequence Diagram and**

5) **Collaboration diagrams**: Both of these diagrams are interaction
diagrams. Interaction diagrams show an interaction that consists of a set
of objects and their relationships

6) **Statechart Diagram**: It shows a state machine that consists of states,
transitions, events and activities

7) **Activity Diagram**: It is a special type of state chart diagram that shows
the flow from activity to activity within a system

8) **Component Diagram**: It shows the organizations and dependencies
among a set of components.

9) **Deployment Diagram**: This type of diagram shows the configuration of
runtime processing nodes and the components that live on them.

One point should always be remembered as said by Booch [Raumbaugh 99]
“The only way to learn UML is by writing models in it”
Promela Background

Promela stands for Process or Protocol Meta Language. It is a modeling language that is used to describe a system. Concurrent processes can be created dynamically using this language. Promela programs mainly consist of processes, variables and message channels. Processes are considered global objects that specify behavior and therefore cannot be declared locally. They have message channels that are used for communication purposes. They can be either synchronous or asynchronous. These message channels can be declared either globally or locally within a process. The message channels and global variables define the environment in which the process will run. Promela is very similar to the C language in syntax but it does not have any pointers. The system state of a Promela consists of the global and local variables, the contents of channel buffers, and the location counter of the process instances. The advantage of a Promela system is that it can be exhaustively and efficiently be analyzed for correctness violations. Once a model system is specified in Promela, tools like Spin can be used to perform random and interactive simulations of the system’s execution. Deadlocks can be detected easily during the verification process.

Approach

The security protocol validation technique presented in this paper cossets of three steps:

- In the first step, the security protocol under consideration under go an OO analysis. During this analyzes process, all classes, objects, methods, attributes, actors, and relations will be identify. The output of this first step would be a class diagram and a sequence diagram(s) representing the security protocol.

- In the Second step, all the sequence diagram(s) produced by the first step are transformed into state chart diagram(s).

- In the third step, we produce a Promela program from the state chart diagrams generated in step two. The Promela code will be run under SPIN to validate the security protocol under consideration.
Validation

We shall use the yahalom protocol as an example to illustrate the modeling approach. The protocol is stated below.

1) A -> B : A, Na
2) B -> S : B, { msg2 (A, Na, Nb) }Kbs
3) S -> A : { msg3(B, Kab, Na, Nb) }Kas, { msg4a(A, Kab) }Kbs
4) A -> B : { msg4a(A, Kab) }Kbs, { msg4b(Nb) }Kab

Where A is the initiator, B is the responder and S will be the server that will create shared keys for communication between A and B. Stated below is the sequence of this protocol.

In line 1, A sends B a challenge which will be decrypted by B. B then creates a message that includes its own challenge to A and the original challenge sent by A. This of course will be encrypted by B’s public key and sent to the server as illustrated in line 2. In line 3, the server send A two message will be broken down. A will decrypt and get its original nonce which will convince it of B’s identity and also the challenge from B. A will then be able to encrypt this from the public shared key Kab, which will be decrypted from message 4.

In a security protocol, there are 5 main objects that will be considered: the server, the initiator, the responder, the intruder and the network. Each security protocol then has a principal, which is specific to the particular protocol.

The first step in modeling a security protocol is the class diagram. The class diagram is a UML diagram, which represents each object as a class or an interface and also shows the collaboration relationship between each object. This will help the software engineer to know how to use the classifier correctly and effectively. Shown below in Figure 1 is the class diagram for the yahalom protocol. The objects initiator, intruder, server and responder all inherit from the principal, which is specific to the yahalom protocol. The principal then uses the network in order to send or receive messages. The intruder comes in 3 forms of impersonations namely; the server, initiator and the responder. The main intruder class controls all these.
The second step in modeling the security protocol is to develop a sequence diagram. The sequence diagram is also a UML diagram used to show the interactions between the objects and their relationships. It is more like a set of actions and reactions between each object. Shown below is the sequence
diagram for yahalom protocol in figure 2. You can see the transactions between the initiator, server and the responder is to not only create messages, but to also send, receive, encrypt and decrypt them. These functions can only be done through the help of the network.

Figure 2. Sequence Diagram for the yahalom protocol

The third step in modeling these protocols is to convert the sequence diagram to state machines. These state machines will form a basis of translating the entire
protocol and verifying it with either a theorem prover or a model checker. By using standard rules and refining them developed the steps listed below.

From Sequence diagram to State Machine
1) Start by defining n different state machines from n objects in the sequence diagram
2) For each state machine, consider the following step
3) The first event (input or output) is the starting state.
4) Consider each object’s duration in the sequence diagram.
5) Every output event is a state. Label each state with the output event.
6) Every input event to the same object is translated into the following:
   a) A state labeled with the input event name.
   b) The transition from one state to another.
7) If there is no output from a state then consider an output from that state to an end state.

Figure 3 – 5 represents the state machines constructed for the initiator, the server and the responder from the steps listed above

Figure 3 State Machine of ‘Initiator’ from the Yahalom Protocol
Figure 4 State Machine of ‘Resonder’ from the Yahalom Protocol

Figure 5 State Machine of ‘Server’ from the Yahalom Protocol
The final step as mentioned previously was checking the validity of the protocol by means of a theorem prover or a model checker. In this example, we shall be using a model check called spin. The validity of the protocol is done by first translating the state machine to spin's programming language called promela. It should be noted that using model checkers on security protocols only show that errors have not been found and not guarantee the non-existence of them. These translations from state machine to promela is done by a sequence of steps constructed during the course of this experiment. The steps were constructed from a bottom up approach were promela source code was written pertaining to state machine of the yahalom protocol. Then steps were derived and abstracted to a high level that would be understood by a person of whose level of intelligence were assumed below.

*Good computer science background*

*Knows the basic syntax and semantics of promela*

Listed below are the steps.

**Steps for Translating from State Machine to Promela**

1. For communication protocols, we shall define a message of type mtype in promela
   
   ```
   mtype = {MESSAGE};
   ```

2. The network object defined in the communication protocol is translated as a global synchronous message channel in promela.
   
   ```
   chan `network_variable` = [0] of {mtype};
   ```

3. Each state machine is a separate process in promela. The name of the process is the same as the name of the state machine
   
   a) Start by defining each state machine as process;
   
   ```
   proctype `statemachine_name`() { }
   ```

4. There is an init process used to instantiate all processes created in a promela code
   
   a) Define an init process;
   
   ```
   init{
   }
   ```
   
   b) Instantiate each process in the init process
   
   ```
   Init{
   run `statemachine_name`();
   }
   ```

5. For each state in a state machine, it is translated as a label in promela, i.e. name of the state followed by a colon.
   
   a) Define every state in state machine as shown below
6. In a state machine, each transition from a state to another state is labeled with a method name. The implementation of this method can be defined in a macro (#define) in Promela and can be called after the label declaration. The method implementation can otherwise also be defined after the label declaration without the need of any macro.

7. If there is a transition from one state to another new state, then use the goto statement of Promela to jump from one label to another.

```proctype statemachine_name()
{
    state_name: /* method implementation */
    goto new_state_name;
}
```

8. If there is a transition from one state to two different states with the same transition label then the “if” selection statement of Promela should be used to implement the guard condition.

```proctype statemachine_name()
{
    state_name: if :: (some_guard_condition) -> stmt;
    :: else -> stmt;
    fi;
}
```

9. If the label of a transition is send(), or a label requiring sending messages on the network, then it will be translated as the standard send channel operation.

   a) Let us define msg as local variable of type mtype
   The standard send channel operation in Promela will be
   ```network_variable! Msg **```

10. If the label of a transition is receive(), or a label requiring receiving messages on the network, then it will be translated as the standard receive channel operation.
a) Let us define \textbf{msg} as local variable of type \texttt{mtype}

The standard receive channel operation in promela will be

\begin{verbatim}
  network_variable?Msg **
\end{verbatim}

11. The end state is translated as a final empty label statement

\begin{verbatim}
Final_state_name: skip;
\end{verbatim}

** same variable defined in step 2

\textit{From the step above, the following promela is derived which is shown below.}

/* Promela program for the Yahalom protocol */

\begin{verbatim}
#define false 0
#define true 1
#define nonce 1000
#define sharedkey 50

mtype={MESSAGE};
chan myNetwork=[0] of {mtype};

proctype Initiator()
{
  bool sent=false;
  mytype msg;
  byte temp=10;

  s1: msg=nonce;
      goto s2;

  s2: msg=msg+nonce; /* some manipulation to message */
      goto s3;

  s3: if
    :: (sent==false) ->
      sent=true;
      myNetwork!msg; goto s4;
    :: else ->
      myNetwork!msg;
      goto s7;
    fi;

  s4: myNetwork?msg;
\end{verbatim}
goto s5;
s5: msg=msg+temp;
    goto s6;

s6: msg=msg+temp;
    goto s2;

s7: skip;
}

proctype Responder()
{
    bool received=false;
    bool decrypt=false;
    byte msg;
    byte temp=10;

s1: myNetwork?msg;
    if
    :: (received==false) ->
        received=true;
        goto s2;
    :: else -> goto s6;
    fi;

s2: msg=nonce;
    goto s3;

s3: msg=msg+temp;
    goto s4;

s4: msg=msg+nonce;
    goto s5;

s5: myNetwork?msg;
    goto s1;

s6: if
    :: (decrypt==false) ->
        decrypt=true;
        goto s6;
    :: else -> goto s7;
    fi;
s7: skip;

} proctype Server()
{

    bool encrypt=false;
    byte msg;
    byte temp=10;

    s1: myNetwork?msg;
       goto s2;

    s2: msg=temp;
       goto s3;

    s3: msg= msg+sharedkey;
       goto s4;

    s4: if
        :: (encrypt==false) ->
            encrypt=true;
            goto s4;
        :: else -> goto s5;
        fi;

    s5: msg=msg+temp;
       goto s6;

    s6: myNetwork!msg;
       goto s7;

    s7: skip;

}

init
{

    atomic
    {
        run Initiator();
        run Responder();
        run Server();
    }

}
Conclusion & Future Work

As shown above, working on the yahalom protocol had given great insights as to how effective it is in verifying a security protocol by using the object oriented framework and spin. Not only do they give us a better understanding of how the protocol really works, but also gives us room for growth. It should be noted that during the on course of the validation, the intruder object was not introduced in the class, sequence nor the state machine. This is an entirely different research area which will be an open door in improving security protocols. By introducing an intruder dynamically in any security protocol and observing how the object may or may not break into the protocol, it will surely be a great indication on the validity of the protocol.

References


