Modelling and Verification of Layered Security Protocols: A Bank Application

Johannes Grünbauer[1], Helia Hollmann[2], Jan Jürjens[1], and Guido Wimmel[1]

[1] Department of Computer Science, Munich University of Technology
Boltzmannstr. 3, D-85748 Garching, Germany
{gruenbau|juerjens|wimmel}@in.tum.de
[2] Secaron AG, Ludwigstrasse 55, D-85399 Hallbergmoos, Germany
hollmann@secaron.de

Abstract. Designing security-critical systems correctly is very difficult and there are many examples of weaknesses arising in practice. A particular challenge lies in the development of layered security protocols motivated by the need to combine existing or specifically designed protocols that each enforce a particular security requirement. Although appealing from a practical point of view, this approach raises the difficult question of the security properties guaranteed by the combined layered protocols, as opposed to each protocol in isolation. In this work, we apply a method for facilitating the development of trustworthy security-critical systems using the computer-aided systems engineering tool AutoFocus to the particular problem of layered security protocols. We explain our method at the example of a banking application which is currently under development by a major German bank and is about to be put to commercial use.

1 Introduction

Security aspects have become an increasingly important issue in developing distributed systems, especially in the electronic business sector. Because of the fact that failures of security mechanisms may cause very high potential damage (e.g., loss of money through fraud), the correctness of such systems is crucial.

Designing security critical systems correctly is difficult. Also, it is easy to misunderstand assumptions on the environment in which e.g. protocols are to be used and what their secure functioning may rely on. Security violations often occur at the boundaries between security mechanisms and the general system [11, 1].

Therefore, the consideration of security aspects has to be integrated into general systems development [20, 1] and also take into account aspects of security management [7]. Common modelling techniques used in industry, such as collaboration diagrams, state charts and message sequence charts (MSCs) have to be tailored for that purpose.

A particular challenge lies in the development of layered security protocols motivated by the need to combine existing or specifically designed protocols

that each enforce a particular security requirement. Although appealing from a practical point of view, this approach raises the difficult question of the security properties guaranteed by the combined layered protocols, as opposed to each protocol in isolation.

In this work, we apply a method for facilitating the development of trustworthy security-critical systems using the computer-aided systems engineering tool AutoFocus [14, 15] to the particular problem of layered security protocols. Cryptographic protocols are specified with state charts. Together with a suitable attacker model, they are examined for security weaknesses using model checking.

We explain our method at the example of a banking application which is currently under development by a major German bank and is about to be put to commercial use.

We specify cryptographic protocols using state transition diagrams (STDs, similar to UML state charts). Together with the modelled adversary, this system is checked for security weaknesses automatically using the model checker SMV connected to AutoFocus to verify the desired security properties of the protocol.

The approach has the benefits of combining intuitive graphical modelling, simulation and model checking in one user-friendly CASE-tool, and allows to represent possible attacks as MSCs. Since the AutoFocus tool builds on the formal development method Focus [4], our approach also supports formal proofs in this framework. The intruder model used is rather flexible, e.g. the adversary can switch between acting as one or another party, intercept only certain messages or learn certain keys etc. Also, the AutoFocus tool integrates several formal tools, which in [3] was identified as a major obstacle to widespread adoption of formal methods.

To put our work into context, we give some background information and related work. There has been extensive research in using formal methods to verify security protocols, following an abstract way to describe protocols in [8]. A few examples are [6, 22, 25]. [28] considers refinement of security-critical systems. Aspects of security engineering have been considered in [1, 27, 21, 10]. As an example for the treatment of security in the context of general systems engineering, [17, 18] presents work towards using the UML notation in security engineering. AutoFocus has been used for security e.g. in [31].

This paper is structured as follows. In Section 2 we introduce the notation of AutoFocus. In Section 3, we give an overview over the banking application under consideration, specify a critical part (a layered authentication protocol) and carry out a security analysis.

We end with a conclusion and indicate further planned work.

2 The Tool AutoFocus

For modelling and verification of the layered protocol, we use the tool AutoFocus [14, 29]. AutoFocus is a CASE tool for graphically specifying distributed systems. It is based on the formal method Focus [5], and its models

have a simple, formally defined semantics. AutoFocus offers standard, easy-to-use description techniques for an end-user who does not necessarily need to be a formal methods expert, as well as state-of-the-art techniques for validation and verification.

Systems are specified in AutoFocus using static and dynamic views, which are conceptually similar to those offered in UML-RT, a UML profile for component-based communicating systems. AutoFocus has been used and adapted to model security-critical systems in a number of case studies (see e.g. [31, 19]).

To specify systems, AutoFocus offers the following views:

- **System Structure Diagrams (SSDs)** are similar to data flow resp. collaboration diagrams and describe the structure and the interfaces of a system. In the SSD view, a system consists of a number of communicating components, which have input and output ports (denoted as empty and filled circles) to allow for receiving and sending messages of a particular data type. The ports can be connected via channels, making it possible for the components to exchange data. SSDs can be hierarchical, i.e. a component belonging to an SSD can have a substructure that is defined by an SSD itself. Besides, the components in an SSD can be associated with local variables.
- **Data Type Definitions (DTDs)** specify the data types used in the model, with the functional language Quest [26]. In addition to basic types as integer, user-defined hierarchic data types are offered that are very similar to those used in functional programming languages such as Haskell [30].
- **State Transition Diagrams (STDs)** represent extended finite automata and are used to describe the behaviour of a component in an SSD. The automata consist of a set of states (one of which is the initial state, marked with a black dot) and a set of transitions between the states, where each transition t is annotated with
  - $\mathsf{pre}(t)$, a boolean precondition (guard) on the inputs and local variables
  - input patterns $\mathsf{inp}(t) = \mathsf{inp}_1?\mathsf{pat}_1; \mathsf{inp}_2?\mathsf{pat}_2; \ldots$, specifying that values are to be read at the ports inp_i that should match the patterns pat_i (terms in the functional language that specify values of data types and can include variables). During the execution of t, variables in the patterns are bound to the matching values. For example, the pattern $\mathsf{inp}_1?\mathsf{DataForm}(\mathsf{Form}(x))$ matches if the value $\mathsf{DataForm}(\mathsf{Form}(\mathsf{Acknowledgement}))$ is received on port inp_1 and binds x in the preconditions, output expressions and postconditions to $\mathsf{Acknowledgement}$.
  - output expressions $\mathsf{outp}(t)$ of the form $\mathsf{out}_1!\mathsf{term}_1; \mathsf{out}_2!\mathsf{term}_2; \ldots$
  - postconditions $\mathsf{post}(t)$ of the form $\mathsf{lvar}_1 = \mathsf{term}_1; \mathsf{lvar}_2 = \mathsf{term}_2; \ldots$

  In the concrete syntax of the STDs, the annotation is written as $\mathsf{pre}(t) : \mathsf{inp}(t) : \mathsf{outp}(t) : \mathsf{post}(t)$. Leaving out components is interpreted as **true** for preconditions, and as an empty sequence in the other cases. A transition

is executable if the input patterns match the values at the input ports and the precondition is true. At each clock tick, one executable transition in each component fires, outputs the values specified by the output patterns and sets the local variables according to the postcondition. The values at the output ports can be read by the connected components in the next clock cycle.
- **Extended Event Traces (EETs)** finally make it possible to describe exemplary system runs, similar to MSCs [16].

The Quest extensions [29] to AutoFocus offer various connections of AutoFocus to programming languages and formal verification tools, such as Java code generation, model checking using SMV, and bounded model checking and test case generation.

Note that although we had to select a specific tool for the case study, the general concepts we present in this paper do not depend on the use of AutoFocus. Its main prerequisites are an executable, component-based description technique and verification support.

3 The Authentication Protocol

The bank system under consideration is an Internet-based application which can be used by clients to fill out and sign digital order forms. The main security requirements of this application are that the personal data in the forms must be kept confidential, and that orders can not be submitted in the name of others.

For this purpose, when the user logs in, first an authentication protocol is run and a confidential (i.e. encrypted) connection is established. The second part of the transaction (filling out and digitally signing the order form) is carried out over this connection.

The authentication protocol is based on an SSL connection which is established at first and provides a secure connection with regard to confidentiality and server authentication. The session key generated during the SSL handshake is used to encrypt the messages of the authentication protocol on the second layer. The protocol authenticates the client by making use of a card reader and a smart card to compute digital signatures on the client's side. The need for the layered protocol arose here because the SSL client authentication feature could not be used due to technical restrictions imposed by the architecture of the bank system (the web server did not support the forwarding of client certificates).

The complete protocol run is shown in Figure 1. After the ClientHello message a randomly generated number or "number used once" (nonce) is sent by the web server. The client signs this nonce with his own private key and sends it together with his certificate back to the web server. The certificate contains the client's identity, a global identification number which references the client's data on the backend and his public key. The web server checks the signature of the nonce and compares the received nonce with the one sent before. Furthermore a plausibility check of the global ID will be done and it will be saved for later purposes. The authentication is finished after the checks have been successful. The web server sends now the global ID and an empty form to the backend

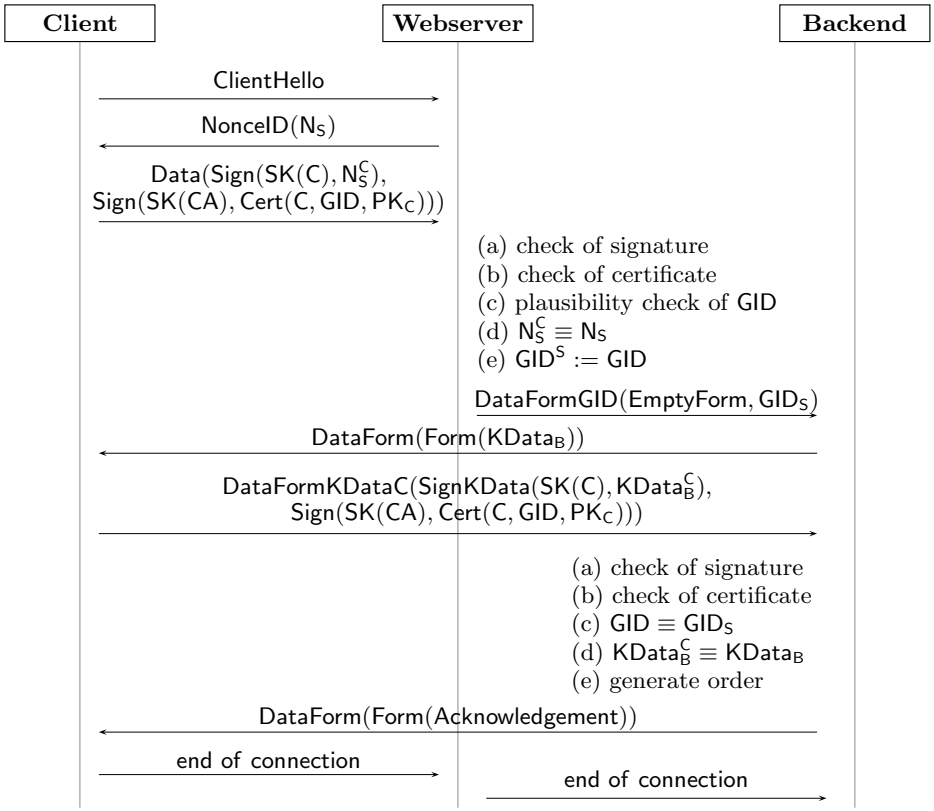


Fig. 1. Authentication Protocol

system, where it is filled with the client's data and sent back to the client. The global ID is also stored on the backend. The client signs his data with his private key, thus creating an electronic signature. The backend checks the signature of the received data object and the certificate. The received global ID and the signed data object are compared with the ones stored. On success a order is generated and an acknowledgement is sent to the client. The end of connection signal can be caused by a timeout or a logout event.

3.1 Modelling

An overview of the complete modelled system can be found in the SSD in Figure 2. There are four components which are connected via channels of type `TMessage`, which is a user-defined data type specified in the DTDs. Is contains the message formats for the protocol, such as `NonceID` and `DataForm`. The component *Client* represents the user's system with the smart card, the card reader and a web browser. The component *Webserver* is the company's interface to the

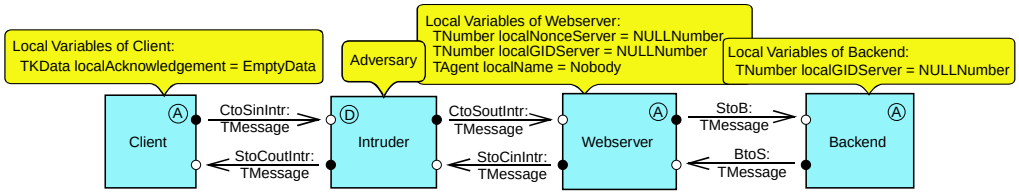


Fig. 2. SSD Main

Table 1. Local Variables of *Client*

Type	Name	Description
`TKData`	`localAcknowledgement`	stores the received acknowledgement

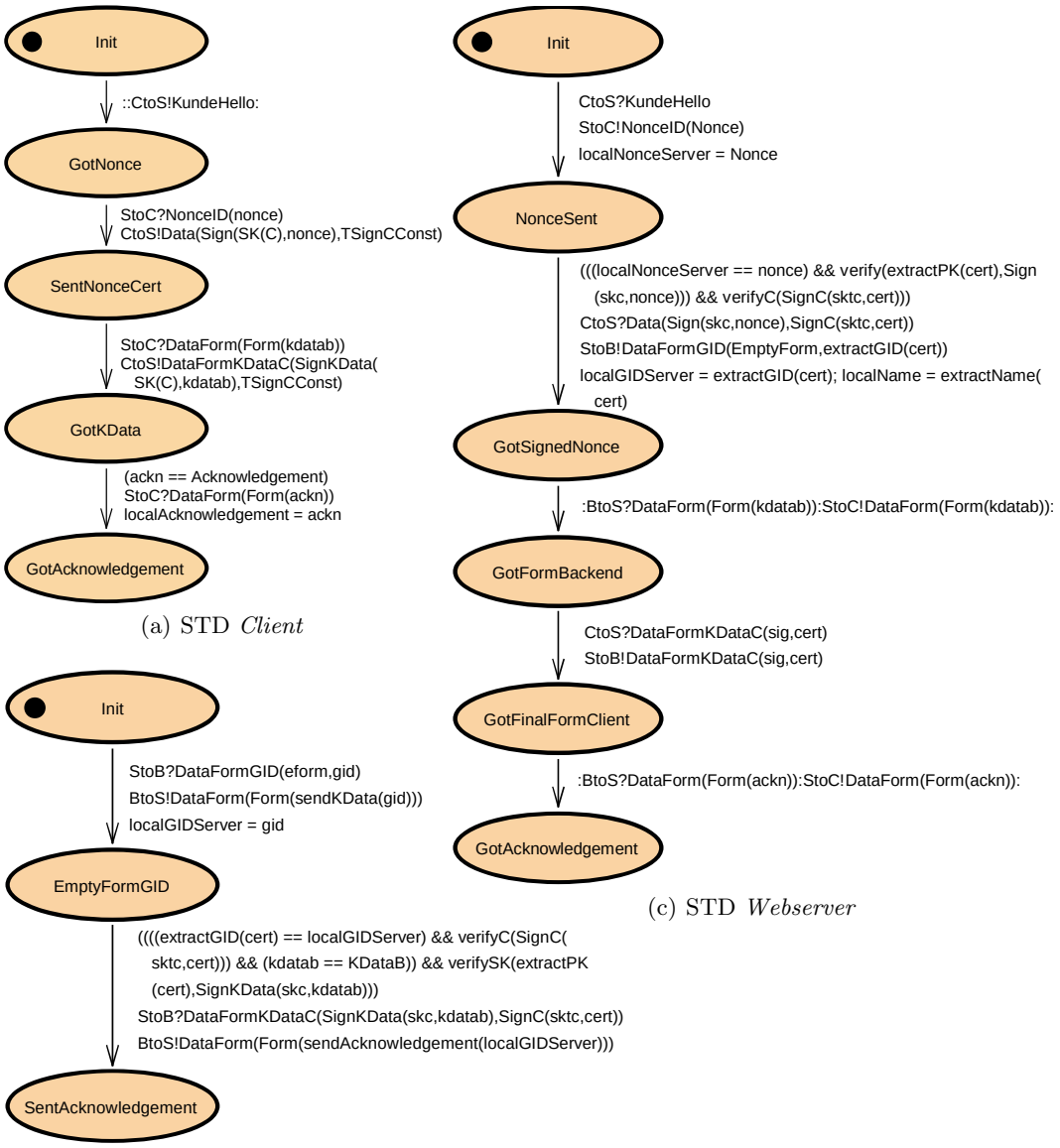
public network. Between these two components an *Intruder* is placed who tries to break the protocol run and get any client's data. Because we assume that the protocol is run over an SSL connection, either the client or the intruder can establish a connection with the web server. The fourth component is the *Backend*. It represents the host system of the bank and stores among other things the client's data. It is connected with the *Webserver* via the Intranet. Therefore we assume that no adversary is able read or manipulate data sent on these channels.

A description of the variables of the components *Client* and *Webserver* can be found in the Tables 1 and 2. For example, in `localGIDServer` of *Webserver* the global ID extracted from the client's certificate is stored. The values can be changed within the STDs which are associated to the components and describe their behaviour. In Figure 3 the STDs of the components *Client*, *Webserver* and *Backend* are shown.

The component *Intruder* is a black box view of the SSD in Figure 4(a) with the components *Overhear* and *FakeStore*. *Overhear* is the switching centre of *Intruder* and is used for forwarding the messages. If the client has established the SSL connection then the client will communicate with the web server. If the adversary starts the connection, the adversary will communicate with the web server. This behaviour is implied by the fact which component exchanged the

Table 2. Local Variables of *Webserver*

Type	Name	Description
`TNumber`	`localNonceServer`	stores the nonce sent by the web server.
`TNumber`	`localGIDServer`	stores the global ID from the client's certificate.
`TAgent`	`localName`	stores the name of the authenticated participant.



(a) STD *Client*

(b) STD *Backend*

(c) STD *Webserver*

Fig. 3. State Transition Diagrams

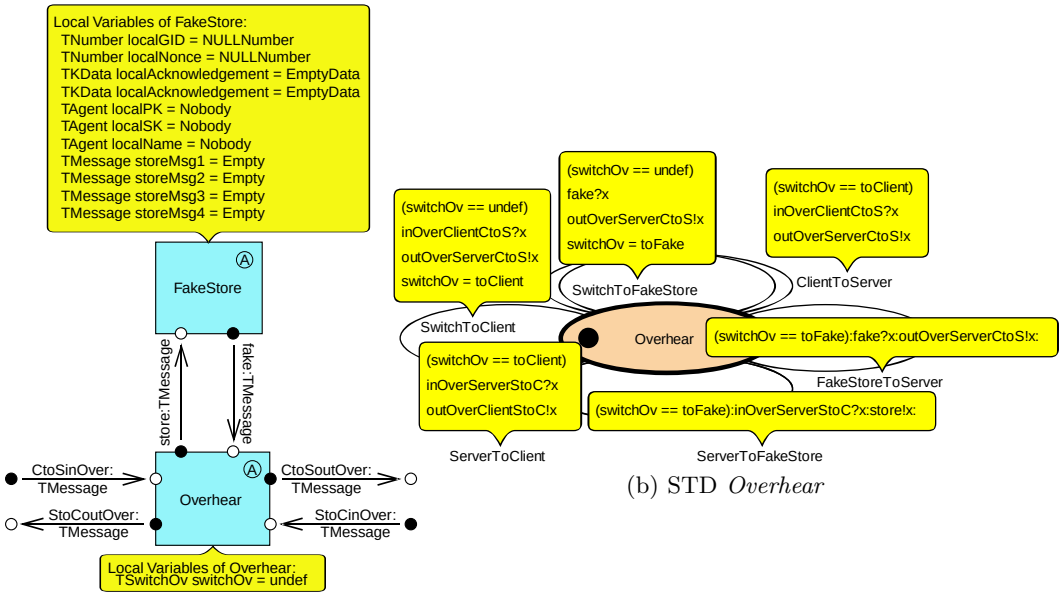


Fig. 4. Parts of the Intruder

session key with the web server within the SSL handshake. Figure 4(b) shows the STD of *Overhear*. It simulates the underlying SSL connection in the way described above and works as followed: At the start of the protocol run the internal variable `switchOv` is set to `undef`. If the client and the adversary start the protocol simultaneously, *Overhear* decides in a nondeterministic way which component is chosen for communication. Otherwise `switchOv` is set to `toClient` or to `toServer` depending on who sent the *ClientHello* message first.

3.2 Adversary

For modelling an adversary we take a *generic adversary* as a basis. A generic adversary has the following abilities or restrictions respectively.

- The adversary may know some data in advance and is assumed to know the design of the system excluding the private keys of the participants.
- The adversary can capture messages and delete them.
- The adversary can insert own messages into the protocol run. These can be arbitrary or protocol conform messages.
- The adversary cannot derive a private key from a public key. We assume the RSA encryption to be secure. In particular the adversary can only read encrypted messages with the corresponding key.
- It not possible to fake a signature. A message $\mathsf{Sign}(\mathsf{SK}(\mathsf{i}), \mathsf{x})$ can only be created with the private key of participant i. It can be verified with the corresponding public key.

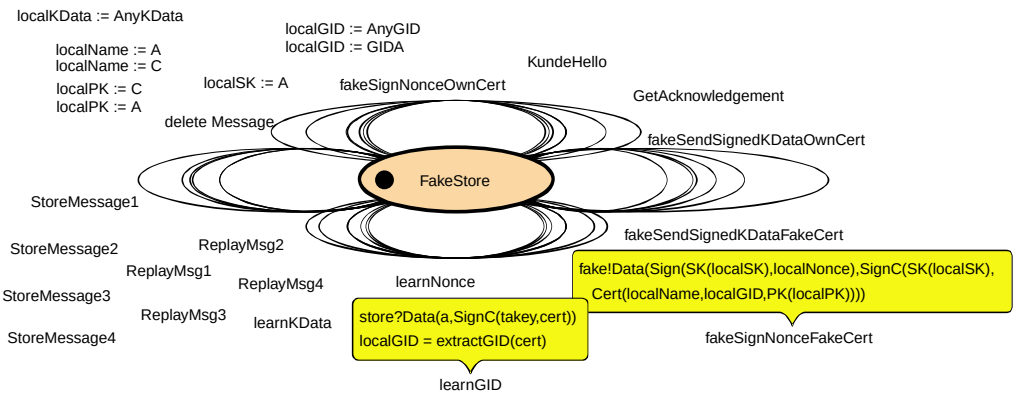


Fig. 5. STD *FakeStore*

– The adversary can split and recombine messages.

In general a generic adversary results in a model which is too complex for automatic verification. The computing time and space required by the model checker increases exponentially with the size of the model (given by its state space). Therefore we have to use a simplified adversary that is derived from the generic one in a justified way by limiting its behaviour and storage capacities. However, the modelled adversary should be strong enough so that possible attacks on the system with respect to the assumed threat scenario can be found. It is realized in the component *FakeStore* which has two purposes. Firstly, the adversary is able to authenticate as a regular client if he submits his valid certificate and can perform all operations of a normal client. Secondly, he will try to break the protocol run to get data of an arbitrary client and to perform operations such as generating fake messages.

The capabilities of the adversary are reflected by the transitions of the state transition diagram shown in Figure 5 and can be divided into several classes. The transitions *fakeSignNonceOwnCert* and *fakeSendSignedKDataOwnCert* enable the adversary to authenticate himself with his own certificate and to carry out transactions under his own identity. With the *StoreMessage* and *ReplayMsg* transitions it is possible to insert previous messages into the protocol run at a later time. The transitions with the prefix learn split the messages and extract their contents into local variables for later use. For example the transition learnGID takes the input pattern from port store and assigns the global ID from the certificate (with help from the function extractGID(cert)) to the local variable localGID. The transitions with the prefix fake use the values of the local variables to generate own messages. E.g. fakeSignNonceFakeCert sends a message with the values stored in the local variables. The unnamed transitions assign values to the local variables the adversary may know and thus is able to use. The values of the local variables are used in the faked messages the adversary sends. The

manual construction of all the possible transitions can be automated and the model is then less susceptible to errors.

Justification of the Specialized Adversary To show that the modelled adversary is adequate to the generic adversary we have to compare it to this one and make sure that he is strong enough to perform all assumed operations but is not too strong to perform operations beyond it. First of all we have to make sure that the adversary cannot read messages sent between *Client* and *Webserver* because of the SSL connection established before.

Furthermore it is neither possible for the adversary in the model to get the private key of the client nor to derive it from his public key since there is no corresponding transition. Therefore it is obviously not possible to sign data objects with anyone's private key. The adversary may know some data in advance, like the public keys of the participants and the structure of the protocol messages. With the assignment transitions in the model *FakeStore* assigns all possible values to the corresponding local variables. It is possible to delete, store and replay messages. Transitions with the prefix *learn* are used to store data in the local variables. Transitions with the prefix *fake* are taken for inserting own messages into the communication process by using the local variables. Furthermore *FakeStore* can only sign a data object successfully if using a certificate signed itself by a certification authority. On the basis of these possibilities and restrictions with regard to the generic adversary we call the modelled adversary adequate to the generic adversary.

3.3 Verification of the Protocol

In this subsection we describe how to verify the model with regard to its security properties. For this purpose AutoFocus generates in conjunction with the Quest extensions an input file for the symbolic model checker SMV which carries out the actual model checking process. The model checking algorithm of SMV [23] is based on representing the system behaviour using Binary Decision Diagrams (BDDs). Symbolic model checking is particularly efficient for highly nondeterministic systems, such as the attacker in our model. We specify the required security properties of the system with CTL, a temporal logic defining formulas over the paths of the tree given by the possible computations of the system. In this work we use the following CTL operators: $\mathsf{AG}(\mathsf{e})$ means that expression e is true over all paths in the tree. $\mathsf{E}(\mathsf{e}\ \mathsf{U}\ \mathsf{f})$ means that there exists at least one path in the tree where e is true until f gets true (see [9]). The security properties are translated to the SMV language as well, and during the model checking process, SMV checks if they are true with respect to the model. If SMV finds any flaw in the protocol, a message sequence chart is generated which helps to understand the way how the protocol can be attacked.

Authentication For the authentication of the client we demand that the adversary cannot authenticate under a wrong identity. For the explanation of a

correct client authentication we take a look at Figure 1. We call the authentication process correct, if the client sent the signed nonce back to the web server (step 3). That means, the client has to be in the state *SentNonceCert* and the web server in the state *GotSignedNonce*. Furthermore the web server has stored the value `C` in his local variable `localName` which is the identity extracted from the received certificate. The following temporal logic expression says that there does not exist any path ($\neg\mathbf{E}$) where *Webserver* is in state *GotSignedNonce* and stores the value `C` in his local variable `localName`, if *Client* has not been in state *SentNonceCert* before (the current control state of the Client or the Webserver is denoted with Client.@ respectively Webserver.@):

$$\neg(\mathbf{E}(\text{Client.@} \neq \textit{SentNonceCert}\ \mathbf{U} \\ (\text{Webserver.@} = \textit{GotSignedNonce} \wedge \text{Webserver.localName} = \texttt{C}))) \quad (1)$$

To verify this fact we have to add this as a property to the model and start the verification process. SMV checks the more than 700 000 reachable states in the model (out of approx. 10^{105} possible states in total) and after approximately 2 hours and 40 minutes[1] it comes to the result that the property is true under the assumptions.

Confidentiality of the Client's Data The second property is that the adversary cannot get the client's confidential data. We have to ensure that the global ID of the client which serves as a unique identifier of the client's data is hidden from the adversary. Furthermore the adversary must not get the data served by the *Backend* and sent back signed by the client. The temporal logic expression below says that in all paths the adversary does not get the global ID, the client's data and the acknowledgement:

$$\mathbf{AG}((\text{FakeStore.localGID} \neq \texttt{GID}) \wedge \\ (\text{FakeStore.localAcknowledgement} \neq \texttt{Acknowledgement}) \wedge \\ (\text{FakeStore.localKData} \neq \texttt{KDataB})) \quad (2)$$

After a computation time of approximately 2 hours and 50 minutes the model checker concludes that the property is true under the assumptions.

3.4 Justification of the Model

The authentication protocol is based on SSL which we assume to be secure based on other works (see [24, 12]). We show that there are no other attacks against the authentication protocol by the adversary.

The authentication protocol has to ensure that only authorized clients have access to their data and an adversary has no possibility to get data stored in

[1] Athlon XP 1800+, 256 MB RAM, Windows XP Professional Edition. A detailed investigation on the general relationship between the size of the model (number of components, channels and data type definitions) and the corresponding complexity would go beyond the scope of this work and will be subject of future research.

the backend system or to carry out transactions under a false identity. The confidentiality of the protocol data sent on the second layer is ensured if the adversary cannot decrypt the data objects. Assuming that the SSL protocol is secure we can say that the protocol on the second layer is secure, too.

An unintended service (see [13]) exists, if existing messages would help the adversary to break the protocol, e.g. the adversary could get the session key of the SSL protocol on the first layer by a simple request. With the session key it would be easy for the adversary to read the confidential data sent within the authentication protocol and thus to break it.

We weaken the possibilities of the adversary on the second layer to adjust it to the security properties of SSL. This is necessary to reduce the complexity of the model. It is realized in the component *Overhear* which relies on the security properties of SSL. If a connection between *Client* and *Webserver* has been established, communication is only possible between these two components. Based on SSL the adversary has no possibility to read or modify the data sent along the channels. Thus he is not able to send any requests to the SSL layer to get any information to break the protocol, e.g. the session key. This is also made clear in so far as both protocols own a disjoint set of messages and it is neither possible for the adversary to get the client's data nor to get secret information about the SSL connection. Thus an unintended service does not exist.

4 Conclusion and Further Work

This paper presented work regarding the formal analysis of layered security protocols using the computer-aided systems engineering tool AutoFocus. An example of a banking application which is currently under development by a major German bank was modelled and analyzed for security weaknesses using model checking.

It turned out that the used approach was adequate for its purpose. By abstraction from irrelevant detail, the protocol model was kept compact enough to allow verification with the used model-checker in a few hours. The particular challenges of layered security protocols were addressed in a suitable way by informally making sure that no security weaknesses could arise from the combination of the different protocols.

Overall, the approach, which has the benefits of combining intuitive graphical modelling, simulation and model checking in one user-friendly CASE-tool, seems to be well-suited for further industry-critical applications.

Future work includes automated generation of the attacker models and giving formal arguments for their justification. We intend to apply our analysis method in further case studies to other domains and protocols, e.g. in the automotive context.

References

1. R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems.* Wiley, 2001.

2. S. Anderson, S. Bologna, and M. Felici, editors. *SAFECOMP 2002 – The 21st International Conference on Computer Safety, Reliabiltiy and Security*, volume 2434 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2002.
3. R. Bloomfield, D. Craigen, F. Koob, M. Ullmann, and S.Wittmann. Formal methods diffusion: Past lessons and future prospects. In F. Koornneef and M. van der Meulen, editors, *Computer Safety, Reliability and Security 19th International Conference, SAFECOMP 2000*, volume 1943 of *Lecture Notes in Computer Science*, pages 211–226, Rotterdam, The Netherlands, Oct. 2000.
4. M. Broy, F. Dederich, C. Dendorfer, M. Fuchs, T. Gritzner, and R. Weber. The design of distributed systems - an introduction to FOCUS. Technical Report TUM-I9202, Technische Universität München, 1992.
5. M. Broy and K. Stolen, editors. *Specification and Development of Interactive Systems*. Springer, 2001.
6. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989.
7. P. Daniel. Security of critical systems - management issues. In *Critical Systems Conference 2001*, Birmingham, 23rd - 24th Oct. 2001. The Safety-Critical Systems Club and Software Reliability and Metrics Club.
8. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
9. E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. Elsevier Science Publishers, 1990.
10. R. Fredriksen, M. Kristiansen, B. Gran, K. Stolen, T. Opperud, and T. Dimitrakos. The CORAS framework for a model-based risk management process. In Anderson et al. [2], pages 94–105.
11. D. Gollmann. *Computer Security*. J. Wiley, 1999.
12. Johannes Grünbauer. Modellbasierte Sicherheitsanalyse einer Bankapplikation. Master's thesis, Technische Universität München, 2003.
13. Joshua D. Guttman. Security goals: Packet trajectories and strand spaces. *Lecture Notes in Computer Science*, 2171:197ff, 2001.
14. F. Huber, S. Molterer, A. Rausch, B. Schätz, M. Sihling, and O. Slotosch. Tool supported Specification and Simulation of Distributed Systems. In *International Symposium on Software Engineering for Parallel and Distributed Systems*, pages 155–164, 1998.
15. F. Huber, S. Molterer, B. Schätz, O. Slotosch, and A. Vilbig. Traffic Lights – An AutoFocus Case Study. In *1998 International Conference on Application of Concurrency to System Design*, pages 282–294. IEEE Computer Society, 1998.
16. ITU. ITU-TS Recommendation Z.120: Message Sequence Chart (MSC). ITU-TS, Geneva, 1996.
17. J. Jürjens. Critical Systems Development with UML. In *SAFECOMP 2002 – The 21st International Conference on Computer Safety, Reliabiltiy and Security*, Catania, Italy, Sept. 9–13 2002. EWICS TC7. Half-day tutorial.
18. J. Jürjens. *Secure Systems Development with UML*. Springer-Verlag, Berlin, 2003. To be published.
19. Jan Jürjens and Guido Wimmel. Security modelling for electronic commerce: The Common Electronic Purse Specifications. In *First IFIP conference on e-commerce, e-business, and e-government (I3E)*. Kluwer, 2001.
20. F. Koob, M. Ullmann, and S. Wittmann. The new topicality of using formal models of security policy within the security engineering process. In D. Hutter,

W. Stephan, P. Traverso, and M. Ullmann, editors, *Applied Formal Methods - FM-Trends 98, International Workshop on Current Trends in Applied Formal Method*, volume 1641 of *Lecture Notes in Computer Science*, pages 302–310, Boppard, Germany, Oct. 7-9, 1998 1999. Springer-Verlag, Berlin.
21. K. Lano, D. Clark, and K. Androutsopoulos. Safety and security analysis of object-oriented models. In Anderson et al. [2].
22. G. Lowe. Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR. In Margaria and Steffen, editors, *TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, Berlin, 1996.
23. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, 1993.
24. John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of ssl 3.0. In *Seventh USENIX Security Symposium*, pages 201–216, 1998.
25. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.
26. J. Philipps and O. Slotosch. The Quest for Correct Systems: Model Checking of Diagrams and Datatypes. In *Asia Pacific Software Engineering Conference 1999*, 1999.
27. T. Rottke, D. Hatebur, M. Heisel, and M. Heiner. A problem-oriented approach to common criteria certification. In Anderson et al. [2].
28. T. Santen, A. Pfitzmann, and M. Heisel. Specification and refinement of secure it-systems. In T. Muntean M. Butler, editor, *International Workshop on Refinement of Critical Systems: Methods, Tools and Experience (RCS'2002)*, 2002.
29. O. Slotosch. Quest: Overview over the Project. In D. Hutter, W. Stephan, P Traverso, and M. Ullmann, editors, *Applied Formal Methods - FM-Trends 98*, pages 346–350. Springer LNCS 1641, 1998.
30. S. Thompson. *Haskell: The Craft of Functional Programming*. Addison-Wesley Longman, 1999.
31. Guido Wimmel and Alexander Wißpeintner. Extended Description Techniques for Security Engineering. In *Trusted Information, the New Decade Challege. 16th International Conference on Information Security (IFIP/Sec)*, 2001.