Fraunhofer Institut
Sichere Telekooperation

Development of formal models for secure e-services

Roland Rieke
SIT – Fraunhofer - Institute for Secure Telecooperation,
Rheinstr. 75, D-64295 Darmstadt, Germany
E-Mail: rieke@sit.fraunhofer.de

January 10, 2003

Abstract

A methodology for the development of formal models for e-services is presented. Verification of the correct behaviour when given expected input and check for security properties by adding selected attack patterns is shown. An example scenario of a typical e-service configuration is given and the dynamic behaviour of different variants is analysed. To improve security of a system providing a collection of e-services it is essential to make each e-service secure using a design and verification method based on formal methods and tools [1].

1 Introduction

The goal of the development of formal models of e-services is to achieve a systematic and verifiable improvement of security of the system providing the services.

Reliable security primitives already exist, but the security of complex applications essentially depends on the correct and consistent interplay of security

[1]This work was funded by the "Bundesministerium für Bildung und Forschung" in the context of the SKe project (http://www.ske-projekt.de/).

primitives and resource management.

To verify the correctness of a given e-service a formal model of its components and their interplay is usually analysed by computing its dynamic behaviour and automatically inspecting the generated state space for postulated safety and liveness properties.

To additionally prove some selected security properties one can add the formal specification of a potential attacker to the given model and check if the security properties hold.

In section 2 the used methods and tool are presented. The methodology for development and evaluation of formal security models for e-services is described.

In section 3 the selected portal scenario and analysed security properties are informally described and formally specified using the presented method and tool. After finding a violation of a security property an extended version of the portal scenario using an additional cryptographic protocol is analysed.

In section 4 some implemented features to support attack simulations on formal models are described.

Finally the results of this study are summarised and some perspectives for further development are presented.

2 Outline of methodology for the development of formal models for e-services

2.1 Methods and tool used

Modelling is based on asynchronous product automata (APA), a flexible operational specification concept for cooperating systems [10]. APA are supported by the SH verification tool developed at Fraunhofer SIT [9, 11]. The tool provides components for the complete cycle from formal specification to exhaustive validation.

An APA can be seen as a family of elementary automata. The set of all possible states of the whole APA is structured as a product set; each state is divided into state components. In the following the set of all possible states is called state set. The state sets of elementary automata consist of components of the state set of the APA. Different elementary automata are "glued" by shared components of their state sets. Elementary automata can "communicate" by changing shared state components [6].

A small example to illustrate how APA can be used to specify a system and how to explore the computed reachability graph with the SH verification tool is presented in the appendix.

2.2 Methodology for the development of formal models for e-services

In the context of modelling an appropriate abstraction level must be chosen, so that a verification of the relevant security measures is still possible. In refinements of the model, where a complete analysis is not possible for complexity reasons, interesting parts of the search space can still be explored by manual control using the simulation mode. Different models of possible attackers can be included in the specification and the combined model can be explored to find states where an attack succeeds.

The development of an executable formal model for an e-service application requires the steps shown in figure 1.

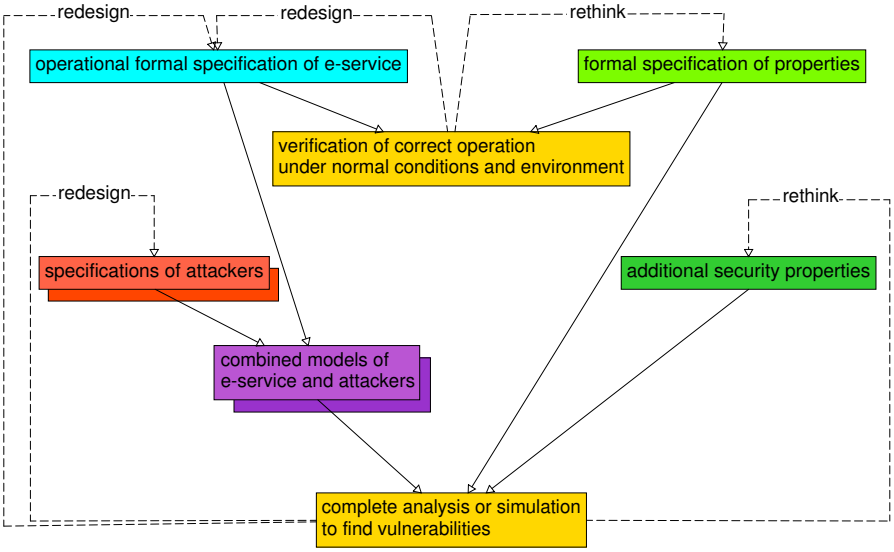

Figure 1: Development process for robust e-service models

Operational formal specification of the e-service

Derive an operational formal specification of the e-service from an informal specification of the required functionality (for example an UML model). Transfer the formal specification into the SH verification tool.
Note that specifications on different abstraction levels can be compared.

Formal specification of properties

System properties are explicitly given by break-conditions, temporal logic formulae or Büchi-automata. Temporal logic formulae can be checked on the abstract behaviour (under a simple homomorphism). A method for checking approximate satisfaction of properties fits exacly to the built-in simple homomorphism check [11].

Verification of correct operation under normal conditions and environment

- To find errors early in the analysis the check for given conditions during the computation of the reachability graph is implemented. Many safety properties (what happens is not wrong) can be checked this way.
- Computation of strongly connected components is very fast [8] and gives good insight into liveness behaviour (eventually something desired happens) of the model.
- Model checking can be used to search for particular states describing a violation of a security property.

Specifications of attackers

Adequate attackers [2] have to be specified here. But what kind of attacks should be considered here ?
A number of threat classes from the X.509 standard that computer networks face are detailed in [13]. These threat classes are: Identity interception, masquerade, replay, data interception, manipulation, repudiation, denial of service, misrouting and traffic analysis.

Many of these threats can be avoided in a given application scenario by choosing a decent cryptographic protocol for communication.

Aditional security properties

Security is not a single property of a protocol or an e-service. Depending on precisely what capabilities an attacker has, different properties for the system model have to be proven.

Combined models of e-service and attackers

The SH verification tool automatically "glues" together selected parts of e-service model and attacker. This is supported by the project management component of the tool.

Complete analysis or simulation to find vulnerabilities

If the attacker has too many alternatives the state space of the composition of the e-service specification and the attacker specification and their complex interplay may become too big to compute the complete behaviour. So we additionally need to be able to inspect selected parts of the state space. Simulation of typical paths of the reachability graph of the formal model under development is very useful and can be compared to the debugger used to develop software using standard programming

[2] Strong versus weak attacker:
A weak attacker might be able to start a replay attack, a very simple form of attack where some sequence of events or commands is observed, and then replayed in the attempt to trick the server to perform some action so that some vulnerability of the protocol can be exploited. A strong attacker might be able to manipulate the power line on the infrastructure or start some sort of denial of service attack to enforce a "reset" of an e-service. A very strong attacker might even be able to reboot the server with a completely different operating system.

languages. If attack patterns are already known simulation of those attacks in the extended formal model can clarify if the model resists this threat.

3 Development of a model for the portal scenario

Looking at some typical configurations that we modelled for usage in e-government applications we selected a typical example of an e-service implementation (called portal scenario) to develop a formal model using the above presented methodology.

An APA model of the portal scenario (see figure 2) except the firewall was implemented using the graphic editor of the SH verification tool. Domains of state components and functions used are defined in textual form in the preamble syntax of the tool.

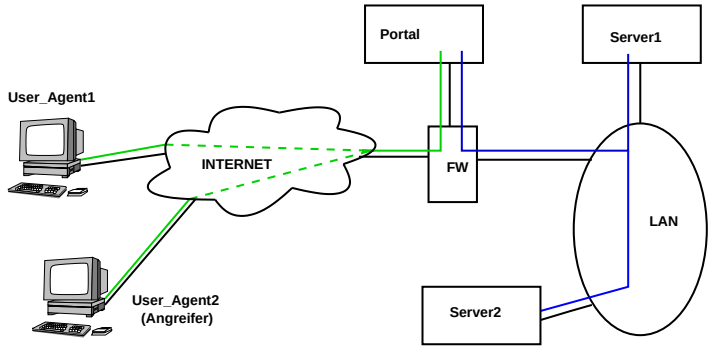

Figure 2: Portal scenario

3.1 Evaluation of portal scenario with attack simulation

A first version of portal scenario was entered into the SH verification tool together with the attacker model shown in figure 3. This is a weak attacker, it has the ability to log into the e-service as a normal client and tries to get some information sent to another client by just behaving like a normal client except it is reading everything it can get.

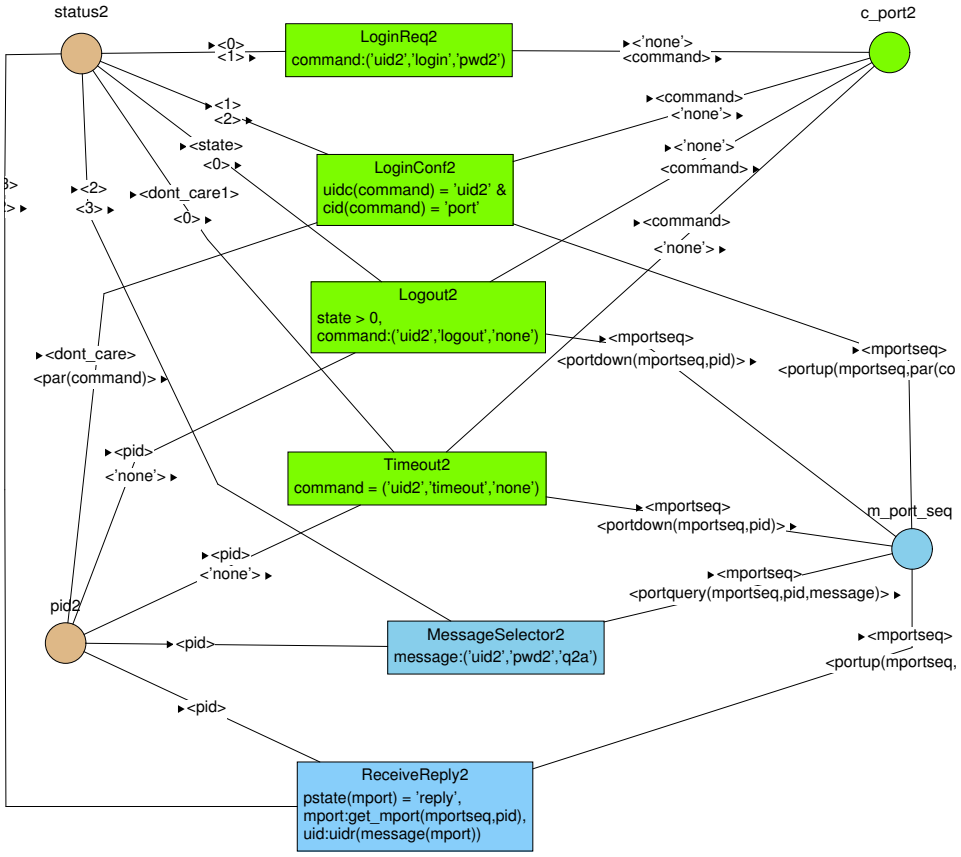

Figure 3: APA specification of attacker

Figure 4 shows the graphical interface to the simulation component of the tool during simulation of the portal model.

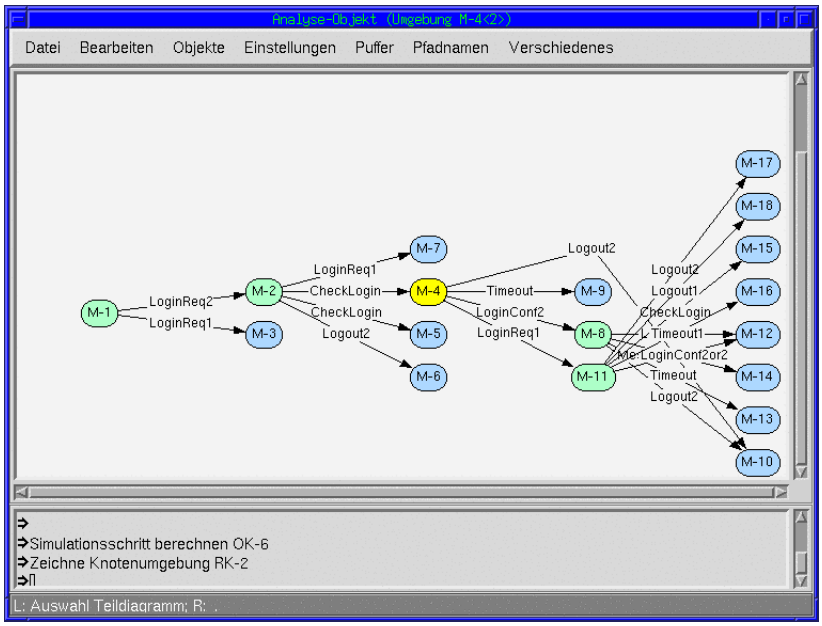

Figure 4: Graphical simulation using SH verification tool

After checking some properties about the correct behaviour without the attacker, the attacker was added and the following security property was specified:

No data from the server database produced for one agent must be delivered to another agent (the potential attacker).

Trying to verify this security property a sequence of 13 steps was found that breaks the property and constitutes an attack (see figure 5).

This threat can be classified as *misrouting*. This is possible because it is assumed that there is no end-to-end protocol between client and server but different protocols for client-portal and portal-server communication. Now some problems with local management of routing information make the attack shown here possible.

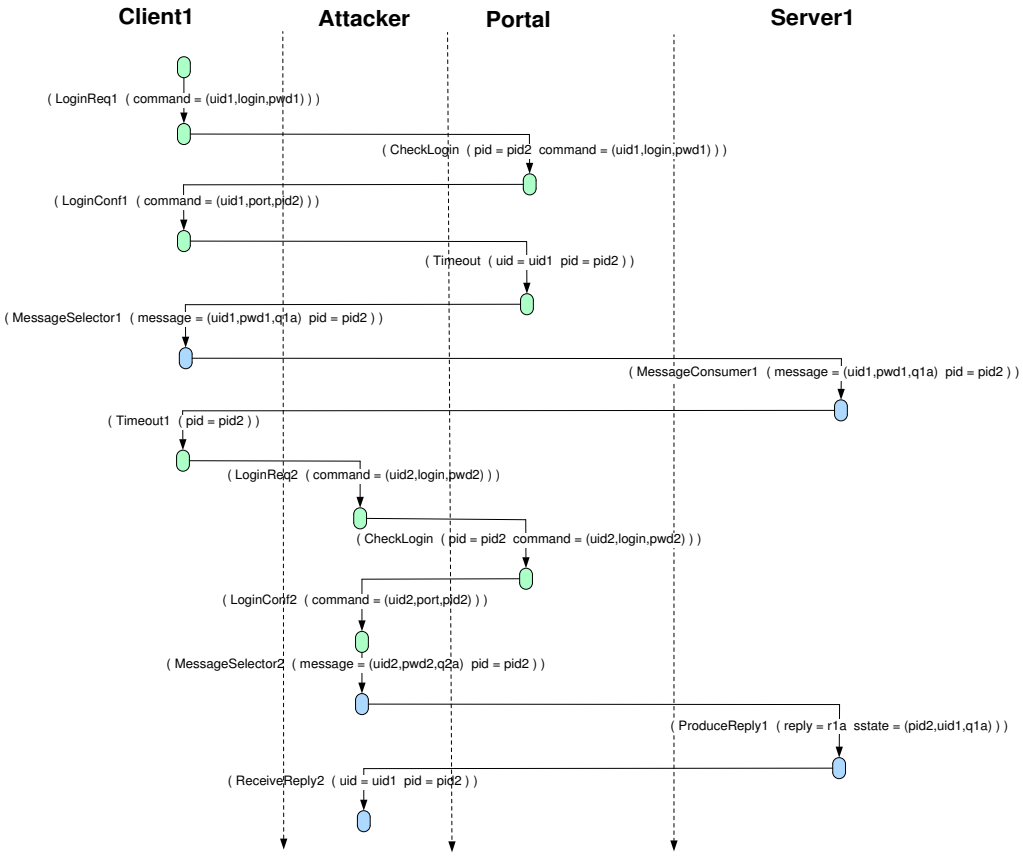

Figure 5: Steps of successful attack

The integrated algorithms for computation of minimal automata [4] in the SH verification tool can be used to compute the local behaviour of the agents, the portal and the server (see figure 6).

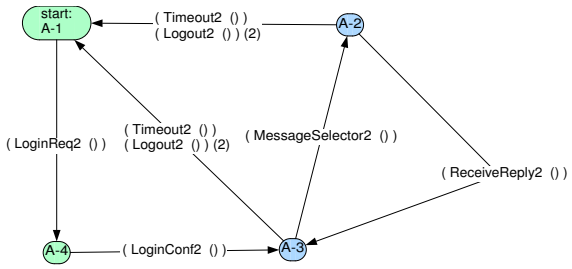

Figure 6: Local behaviour of attacker

On the basis of the computed graphs of the local behaviour of protocol agents it can be checked whether the APA model at this level of abstraction fits to the predefined behaviour of the protocol participants at another level given for example by an UML model.

3.2 Portal scenario with cryptographic protocol

The portal scenario was enhanced by adding a cryptographical protocol (abstracted) for client-portal communication and again possible attacks were searched. Figure 7 shows an overview of the components. A situation where two servers are computing answers for different queries, the portal has stored two different keys for client-portal communications and client2 (the attacker) has posted a query is shown.

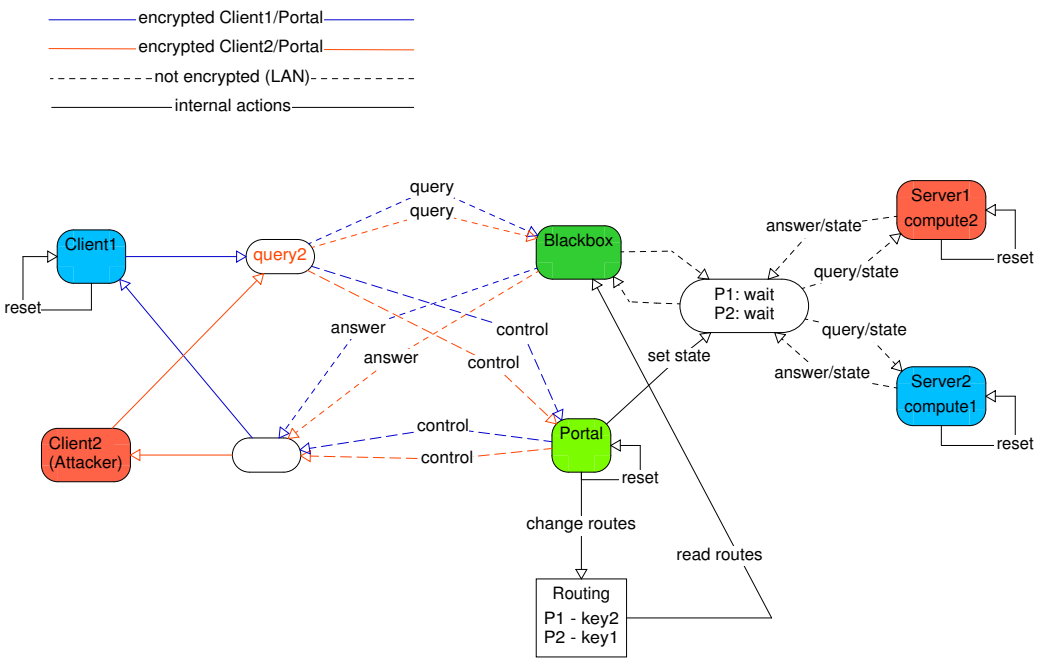

Figure 7: Portal scenario with cryptographic protocol

Figure 8 shows an APA specification of the enhanced portal component.

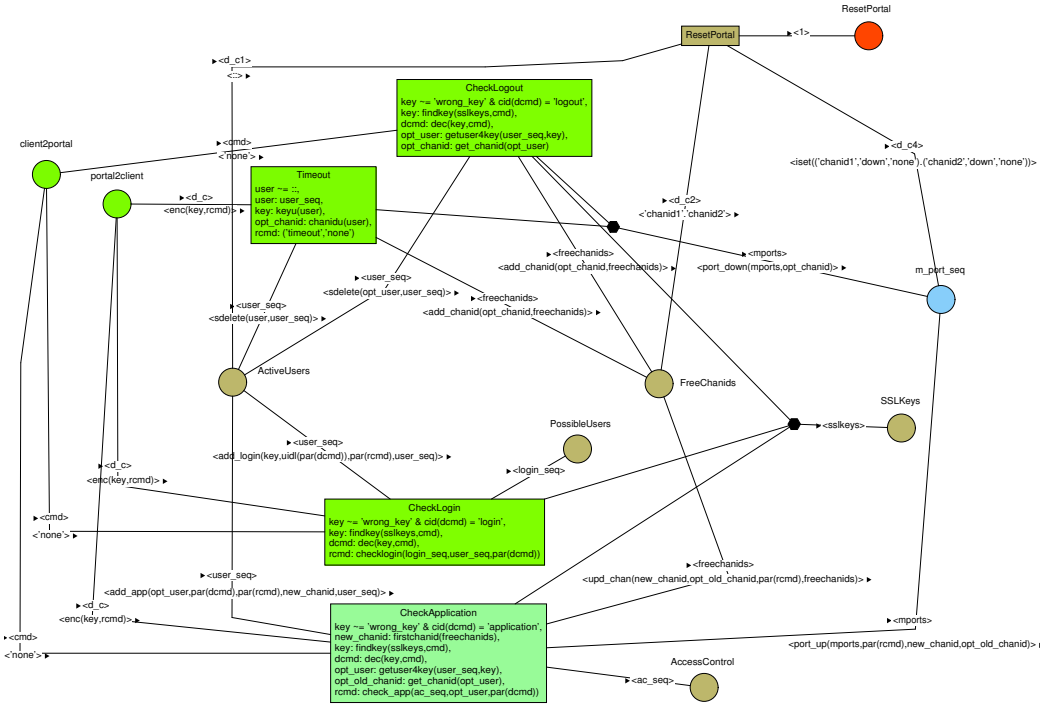

Figure 8: APA specifying the portal component

Communication channels protected by strong cryptography make poor targets. Attackers like to go after the programs at either end of a secure communications link because the end points are typically easier to compromise or try to utilise knowledge about faulty implementations of strong protocols or weak configuration at one side. If a strong protocol is used but some guidelines to use the protocol securely are not followed or some faulty implementation is used it is possible to break even a strong protocol like SSLv3 [12]. For example an attacker can try as a man-in-the-middle to downgrade a client/server pair to use a weaker version of the protocol or a weaker crypto suite and then exploit the known weaknesses.

In the enhanced portal scenario the attack described in section 3.1 was found again but in this case data delivered to the attacker are encrypted (with key of original receiver). So it seems that a weakened form of the given security property is sufficient:

No data from the server database produced for one agent must be delivered to another agent (the potential attacker) except encrypted data that the attacker cannot decrypt.

Nevertheless another more complex attack was found in the given model. It is a sequence of 21 steps including a reset of the portal and the cooperation of a second server.

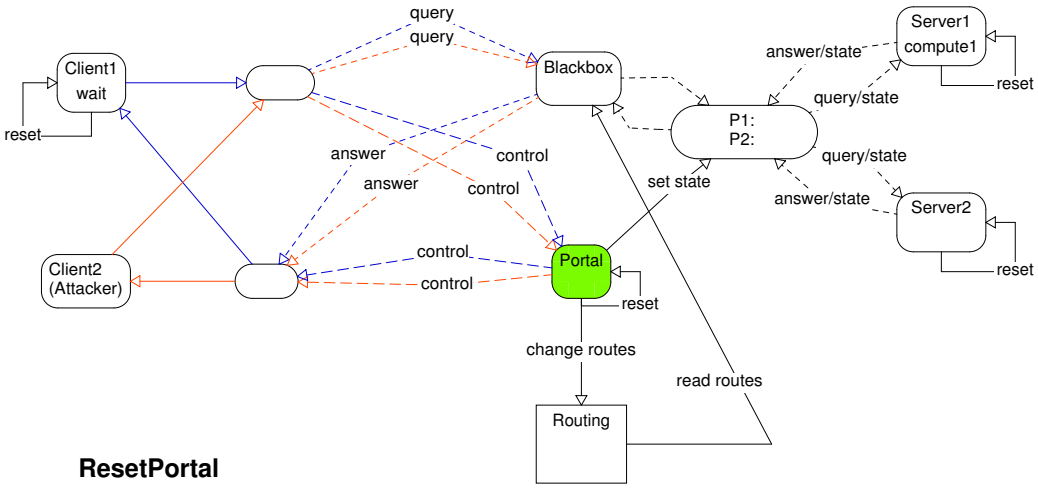

Figure 9: Attack sequence (step 10)

Figure 9 shows a situation after the reset of the portal component where client1 has posted a query and is waiting for an answer, the portal has an empty routing and crypto-key table and no state information and server1 is computing an answer for the previous query of client1.

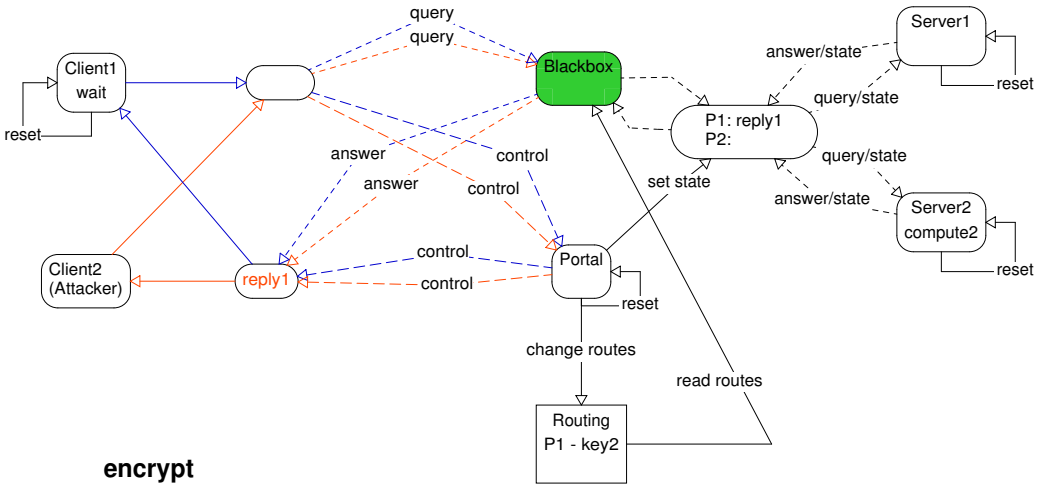

Figure 10: Attack sequence (step 21)

Figure 10 shows a follow-up situation after client2 has exchanged a key with the portal, logged in, posted a query that server2 is processing. Now server1 has produced a reply to the previous query (before the reset) of client1 and the portal assigns the wrong key from its routing table to this reply and directs the

blackbox to forward the encrypted reply to client2.

The vulnerability found here can also be seen as a race condition problem that leads in the end to the misrouting effect. "Race conditions are just the most security-relevant type of concurrency problem."[14]

4 Implemented features to support the development process of formal models

To implement support for the development process of formal models including debugging and attack simulations the following features and modifications have been implemented within the SH verification tool.

Visualisation of simulation paths on graphical presentation: Simple navigation through simulation paths by mouse-clicks is implemented.

Different views: APA can be viewed on different levels to hide unnecessary details.

Pattern based specification of components: Components of same type (for example several servers or clients with same functionality) can be specified once and instantiated many times [5]. A parser and compiler for pattern based specification have been implemented.

Invariants (break-conditions): To find errors early in the analysis, the check for given conditions during the computation of the reachability graph is implemented. So computation automatically stops if a state that matches a given condition (violation of invariant) is found.

Project management: A very flexible selection of variants of analysis scenarios was implemented. In a project tree components can be selected and deselected by mouse-click so it is easy for example to exchange libraries of symbolic crypto functions and analyse different versions and combinations of formal models.
Remark about composition of components:
It would be desirable to be able to verify different components of the e-service architecture separately and then combine the proofs to get less state space explosion during the verification. In the analysed examples however the functionality of the components is abstracted to a level where all modelled functionality influences the behaviour at the interface of the component, so it cannot (at least with our methods) be hidden somehow.

Split state components: Make it possible to insert an "intermediate layer" of attackers in the specification without changing the specification of the state components.

A feature found to be useful but not yet implemented is, to find out if attackers have a winning strategy against the other components of the modelled system. Alternating time temporal logic [3] would be useful for this purpose because it offers selective quantification over those paths that are possible outcomes of games, such as the game in which the system and the environment alternate moves. Currently we only find that for example there is a state where some invariant is broken, but not if attackers alone can enforce the whole system to reach that state.

Related work
The Murphi verification system [1] for example is a similar finite-state analysis tool but as far as we know it only implements fully automatic model checking and has no interactive graphical simulation mode as described in this work. For a comparison of an older version of the tool with other formal methods and tools see [7].

5 Conclusions

This study shows that even if the correct behaviour of an e-service is proven under assumptions about the interfaces to the environment and about reasonable input it is necessary to inspect the system behaviour and ask "what if" questions to check the behaviour of the model against given attack patterns or slightly changed assumptions about the environment.

Therefore a tool that can be used as sort of debugger on formal models is extremely helpful for the development process especially if the robustness of the model against given attacks is to be inspected and verified. If new attack methods are detected later it should be easy to check for vulnerabilities of the model by adding an appropriate module or intercepting some protocol. The SH verification tool with some additional features and modifications described in section 4 has been successfully applied for that purpose.

We have applied a similar approach to formal modelling and verification of security policy interaction issues in the MakoSi project [2] where the e-service modelled was an electronic whiteboard within a distributed collaborative engineering environment.

We are working to improve the current approach in the following ways:

More example scenarios will be analysed to find and classify common attack patterns that can be provided in standard libraries or example collections.

Features found to be useful during evaluation of new scenarios will be implemented within the SH verification tool. It would be nice for example to automatically find and check "similar" simulation paths when having changed some details of the specified system.

If follow-up projects support it, possible new methods for example to reduce the number of states that are explored when analysing the model or some theorem proving assistance will be implemented.

Acknowledgements

I am grateful to the members of our research group META for previous work on APA and fruitful discussions on the subject and especially to Jürgen Repp for implementing most of the simulation support in the SH verification tool.

References

[1] http://verify.stanford.edu/dill/murphi.html.

[2] http://www.makosi.de.

[3] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, Florida, October 1997.

[4] S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, New York, 1974.

[5] S. Gürgens, P. Ochsenschläger, and C. Rudolph. Authenticity and Provability - a Formal Framework. GMD Report 150, GMD – Forschungszentrum Informationstechnik GmbH, 2001.

[6] S. Gürgens, P. Ochsenschläger, and C. Rudolph. Authenticity and Provability - a Formal Framework. In *Infrastructure Security Conference 2002*, October 2002. Copyright: ©2002, Springer Verlag.

[7] P. Hartel, M. Butler, A. Currie, P. Henderson, M. Leuschel, A. Martin, A. Smith, U. Ultes-Nitsche, and B. Walters. Questions and answers about ten formal methods. In *Proc. 4th Int. Workshop on Formal Methods for Industrial Critical Systems*, volume II, pages 179–203, Pisa, Italy, July 1999. ERCIM, STAR/CNR.

[8] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, Mass., first edition, 1979.

[9] P. Ochsenschläger, J. Repp, and R. Rieke. The SH-Verification Tool. In *Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000)*, pages 18–22, Orlando, FL, USA, May 2000. AAAI Press.

[10] Peter Ochsenschläger, Jürgen Repp, and Roland Rieke. Abstraction and composition – a verification method for co-operating systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 12:447–459, June 2000.

[11] Peter Ochsenschläger, Jürgen Repp, Roland Rieke, and Ulrich Nitsche. The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems. *Formal Aspects of Computing, The International Journal of Formal Method*, 11:1–24, 1999.

[12] Eric Rescoria. *SSL and TLS: Designing and Building Secure Systems.* Addison-Wesley, Boston, 2001.

[13] Danny Smith. Selected Aspects of Computer Security in Open Systems. http://auscert.org.au/render.html?it=2255&cid=1920, The University of Queensland, 1993.

[14] John Viega and Gary McGraw. *Building Secure Software.* Addison-Wesley Professional Computer Series, Boston, 2002.

6 Appendix

A small example is used to illustrate how APA can be used to specify a system and how to explore the computed reachability graph with the SH verification tool. Let us assume we want to solve the following problem:

Given the puzzle in figure 11 construct an APA that computes all possible positions reachable by shifting numbered squares to the empty square from the initial state shown in figure 11 on the left and find out if the state on the right is reachable.

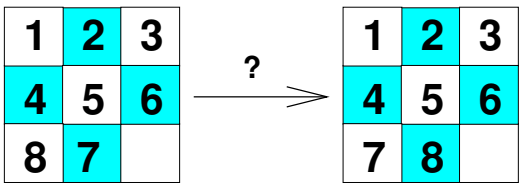

Figure 11: Is it possible to change positions of the *8* and *7*

The idea is to represent the actual state of the puzzle by 9 state components corresponding to the 9 locations in the puzzle and to model the shifting of a square by a state transition of an elementary automaton between each two positions.

Each elementary automaton has the form given in figure 12. This graphical representation shows an elementary automaton named A_1_2 with two neighbour state components $S1$ and $S2$. The circles represent state components and a box corresponds to one elementary automaton. The full specification of an APA includes the transition relations of the elementary automata and the initial state. A state transition of automaton A_1_2 may only change the content of directly connected state components $S1$ and $S2$ representing two neighbour positions in the 8-puzzle example.

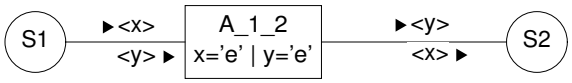

Figure 12: The elementary automaton A_1_2

In this example x is bound to the content of $S1$ and y to the content of $S2$. The inscription `x = 'e' | y = 'e'` in the box represents a restriction for the possible transitions of A_1_2. If one of the state components contains the value 'e' representing the empty square then the value of the other state component can be moved to this state component and vice versa.

The whole APA for this example is given in figure 13. Note that the squares with numbers are not part of the APA they just illustrate the initial state.

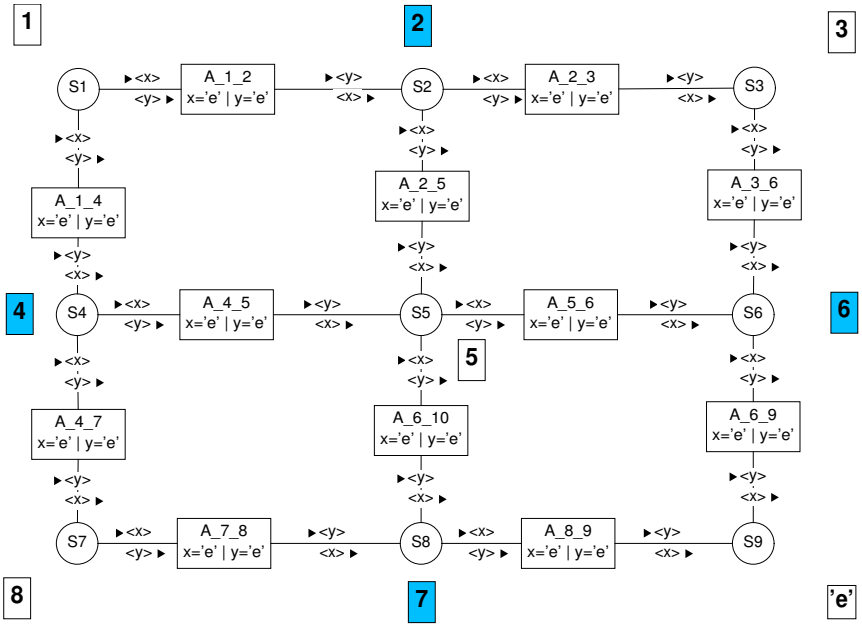

Figure 13: An APA with 12 elementary automata and 9 state components

Alternative behaviour is represented here by the asyncronicity of the possible transitions of the elementary automata that are neighbours to the empty square. For example in the initial position the automata labelled A_6_9 as well as A_8_9 can act. Both alternatives are evaluated by the tool. This situation can be inspected by starting a simulation and visualising the alternatives by drawing the node environment of the first node generated. Then on this drawn node simulation can be continued by selecting some other node in the direction to be inspected and compute and draw the environment of that node.

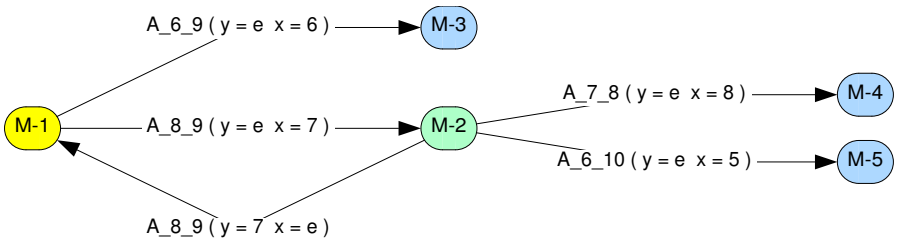

Figure 14: Simulation of 8-puzzle example

Note that there is a built-in check for equal states, in figure 14 the state following M-2 when shifting the square with content 7 back to the original position is identified with M-1.

One way to find out if the 8-puzzle has the property asked for in figure 11 is, to run a complete analysis [3] of the example and then inspect the generated reachability graph by search queries.

To find out if the state with changed positions of the *8* and *7* is reachable it is sufficient to evaluate the following query that describes the searched state:

```
(S1:<1>) & (S2:<2>) & (S3:<3>) &
(S4:<4>) & (S5:<5>) & (S6:<6>) &
(S7:<7>) & (S8:<8>);
```

This query will find no states matching, that is you can never reach such a state by using the given operations. q.e.d.

The example in figure 15 shows how the 8-puzzle problem can be modelled with the descriptional complexity shifted from the graphical structure - a complex graph of elementary automata - to a simple structure using only one elementary automaton but complex data structures and preamble functions. It furthermore illustrates how the choice-operator can be used instead of multiple asynchronous elementary automata to model alternative behaviour.

[3] The complete analysis of this example takes about 2.5 hours on a P700 using the Lisp-Version with compiler included. The reachability graph has 181440 different states. 483840 transitions are computed.

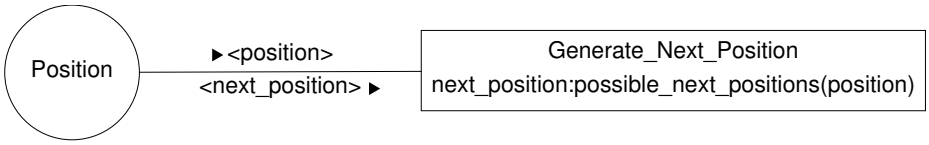

Figure 15: A different model of the 8-puzzle

Here one elementary automaton is used to compute the possible follow-up positions starting from the initial state. The whole puzzle is represented in a data structure using a 9-tuple named *Position*, each tuple element representing one square. A complex preamble function *possible_next_positions* is used to compute a sequence of the possible follow-up positions from a given position.

Alternative behaviour is represented here by the choice operator ":", a special syntactical form in the inscription of the elementary automaton. In this case *next_position* is set to one element chosen from the sequence generated by *possible_next_positions*(*position*). The computation of the reachability graph generates all possible choices at this point.

Note that this construct is often used in the modelling of agents in the protocol specification of e-services in the following sections. The agents have some internal state and from that all possible follow-up states are computed. The usage of the choice operator makes sure that all of them are explored.