32

EXTENDED DESCRIPTION TECHNIQUES FOR SECURITY ENGINEERING*

Guido Wimmel and Alexander Wisspeintner
Institut für Informatik, Technische Universität München, D-80290 München, Germany
wimmel@ in.tum.de, wisspein @ in.tum.de

Abstract There is a strong demand for techniques to aid development and modelling of security critical systems. Based on general security evaluation criteria, we show how to extend the system structure diagrams of the CASE tool AUTO FOCUS (which are related to UML-RT collaboration diagrams) to allow modelling of security critical systems, in particular concerning components and channels. Both high-level and low-level models of systems are supported, and the notion of security patterns is introduced to provide generic solutions for security requirements. We explain our approach on the example of an electronic purse card system.

Keywords: Security Engineering, Graphical Description Techniques, Software Engineering, Requirements Engineering, Security Properties, Design Patterns, Security Patterns, Formal Methods, CASE, AutoFocus, UML-RT.

1. INTRODUCTION

In developing distributed systems—in particular applications that communicate over open networks like the Internet—security is an extremely important issue. Many customers are reluctant to take part in electronic business, confirmed by recent attacks on well-known portal sites or cases of credit card fraud via the Internet. To overcome their reluctance and make E-Commerce and mobile systems a success, these systems need to become considerably more trustworthy.

To solve this problem, on the one hand there are highly sophisticated collections of evaluation criteria that security critical systems have to meet, like the ITSEC security evaluation criteria (ITSEC, 1990) or their recent successor, the Common Criteria (CC) (Common Criteria, 1999). The Common Criteria

* This work was supported by the German Ministry of Economics within the FairPay project

describe security related functionality to be included into a system, like authentification, secrecy or auditing, and evaluation assurance levels (EALs) for its development. The strictest level is EAL7 (Common Criteria, 1999, part 3, p. 66), where a formal representation of the high-level design is required.

On the other hand, research has produced many formal methods to describe and verify properties of security critical systems, ranging from protocol modelling and verification (Burrows et al., 1989; Lowe, 1996; Paulson, 1998; Thayer et al., 1998) to models for access control, like the Bell-LaPadula model (Bell and LaPadula, 1973) or the notion of non-interference (Goguen and Meseguer, 1998).

Such formal methods however are rarely used in practice, because they require expert knowledge and are costly and time-consuming. Therefore, an integrated software development process for security critical systems is needed, supported by CASE tools and using graphical description techniques. This reduces cost, as security problems are discovered early in the development process when it is still inexpensive to deal with them, and proof of meeting evaluation criteria is a byproduct of software development. In addition, systems developed along a certain integrated "security engineering process" will be much more trustworthy

In this paper, we describe a first step towards using extended description techniques for security modelling. As a basis for our work, we use the AUTOFOCUS description techniques. The AUTOFOCUS (Huber et al., 1998b; Slotosch, 1998; Broy and Slotosch, 1999) system structure diagrams are related to UML-RT collaboration diagrams and describe a system as a set of communicating components. The corresponding CASE tool developed at Munich University of Technology supports user-friendly graphical system design and incorporates simulation, code and test case generation and formal verification of correctness. The main advantage of the use of AUTOFOCUS over a more general description technique as UML is its simplicity. Besides, there exists a clear semantics for the description techniques (for general UML description techniques, defining a formal semantics is still subject of ongoing research). As a start, in our work we focus on security properties of communication channels. We show how certain important security properties of communication channels, such as secrecy and authenticity, can be modelled at different abstraction levels of the system design and explain our ideas on the transition between these levels, using generic security patterns. We give definitions of the meanings of our extended description techniques, based on the AUTOFOCUS semantics. See also (Jürjens, 2001) for first work on integrating access control models into UML description techniques, and (Lotz, 2000) for formal definitions of security properties using the Focus method.

This paper is structured as follows. In Section 2, we give a short introduction to AUTOFOCUS. In Section 3, we present the extensions of AUTOFOCUS

system structure diagrams to model security properties of channels. The usage of these techniques is demonstrated in Section 4, with the help of an example model of an electronic purse system. We conclude in Section 5 with a summary and indicate further work.

2. AUTOFOCUS

AUTOFOCUS/Quest (Huber et al., 1998a; Slotosch, 1998; Philipps and Slotosch, 1999) is a CASE tool recently developed at Munich University of Technology with the goal to combine user-friendly graphical system design and support of simulation, code generation and formal verification of correctness.

AUTOFOCUS supports system specification in a hierarchical, view-oriented way, an approach that is well established and facilitates its use in an industrial environment. However, it is also based on the well-founded formal background Focus (Broy et al., 1992), and a fairly elementary underlying concept: communicating extended Mealy machines.

System specifications in AUTOFOCUS make use of the following views:

- **System Structure Diagrams (SSDs)** are similar to collaboration diagrams and describe structure and interfaces of a system. In the SSD view, the system consists of a number of communicating components, which have input and output ports to allow for sending and receiving 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)** define the data types used in the model, with the functional language Quest (Philipps and Slotosch, 1999). In addition to basic types as integer, user-defined hierarchical data types are offered that are very similar to those used in functional programming languages like Gofer (Jones, 1993) or Haskell (Thompson, 1999).

- **State Transition Diagrams (STDs)** represent extended finite automata and are used to describe the behaviour of a component in an SSD. Transitions consist of input patterns on the channels, preconditions, output patterns, and actions setting local variables when the transition is executed. As the main focus of this paper is extending SSDs, we will not describe STDs in detail at this place.

- **Extended Event Traces (EETs)** finally make it possible to describe exemplary system runs, similar to MSCs (ITU, 1996).

The Quest extensions (Slotosch, 1998) to AUTOFOCUS offer various connections of AUTOFOCUS to programming languages and formal verification tools, such as Java code generation, model checking using SMV, bounded model checking and test case generation (Wimmel et al., 2000).

3. EXTENDING SYSTEM STRUCTURE DIAGRAMS

The main difference between security critical systems and traditional systems is the consideration of attacks. A potential third party could try to overhear and manipulate security critical data. To decrease the risk of attacks to a minimum, special security functionalities are used. For example, encryption is a common principle to inhibit overhearing of the communication between two agents.

It is state of the art to use graphical description techniques for system specification. For the specification of security critical systems, we need special description techniques to deal with the particularities of those systems. We extend the AUTOFOCUS description techniques to fulfill the needs of security engineering. In this paper the extensions of the AUTOFOCUS SSDs, mentioned in Section 2, are described. The extensions of the structure diagrams allow the definition of security requirements. Furthermore it is possible to specify the usage of security functionality to fulfill the defined requirements.

We use special tags for the security extensions to the SSDs. These security tags are assigned to particular diagram parts and have a defined semantics. The following sections describe the different security extensions made to the SSDs.

3.1. SECURITY CRITICAL SYSTEM PARTS

To model and evaluate security aspects of distributed systems, it is always necessary to define its security critical parts. The identification of security critical parts should be done very early within the system development process. This task is typically part of the analysis phase. In the Common Criteria, the security critical parts of a system together form the Target Of Evaluation (TOE). The following definition will make our notion of security criticality more precise.

Definition 1 (Security Critical). By security critical parts of a distributed system we mean parts that deal with data or information that has to be protected against unauthorized operations (e.g. disclosure, manipulation, prevention of access etc.). In particular, security critical system parts are connected with security requirements such as secrecy, authentication or auditing, and according to required strictness of evaluation might be subject to formal modelling.

We want to make the distinction visible within the graphical system description. Therefore we annotate security critical system parts with the security tag <<critical>>. Both components and channels can be tagged. To mark non criti-

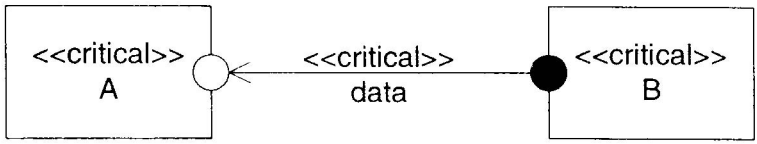


Figure 1 Security Critical System Parts

cal system parts, <<noncritical>> is used. A system part without a <<critical>> or <<noncritical>> tag is non critical by default. Figure 1 shows an SSD. It consists of two security critical components and one security critical channel.

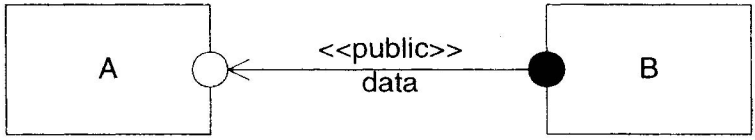
3.2. PUBLIC CHANNELS

The special aspect of security critical systems is the possibility of attacks. Hostile subjects can manipulate the system by overhearing and manipulating the communication between the subsystems. To distinguish private communication channels of a system from public channels, we use the two security tags <<private>> and <<public>> A <<private>> channel is a channel a hostile party has no access to. The hostile subject can neither overhear nor manipulate the communication that takes place via the channel. Vice versa a hostile party can overhear and manipulate all communications of a <<public>> channel. If neither <<private>> nor <<public>> are used, we assume by default that the communication channel is not publicly accessible.

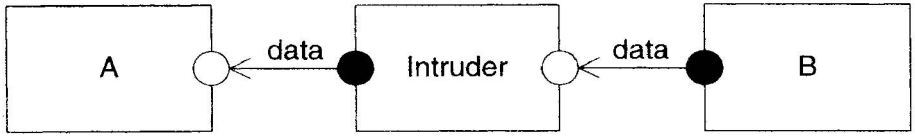
Definition 2 (Private Channel). A <<private>> channel has the same semantics as a normal channel within AUTOFOCUS, i.e. it is a dedicated connection from one component to another.

Definition 3 (Public Channel). The semantics of a <<public>> channel without secrecy and authenticity introduced in Section 3.6, is defined by the SSDs shown in Figure 2. Using a <<public>> channel (Figure 2(a)) is an abbreviation for having an intruder included in the model that has access to the channel (Figure 2(b)). The behaviour of the intruder is defined by the threat model—for example, the intruder usually can overhear, intercept or redirect messages. It is possible to model this behaviour in a flexible way using using AUTOFOCUS State Transition Diagrams (STDs) (Wimmel and Wisspeintner, 2000).

The identification of private and public channels should be done during the analysis phase of the system development process, right after the identification of the security critical parts. Every security critical channel must be analyzed with regard to accessibility—for the other channels, this is optional. The result of this analysis is documented by using the tags for private and public channels.



(a) ≪public≫ Channel



(b) Defined as an Intruder between the Two Agents

Figure 2 Semantics of a public Channel

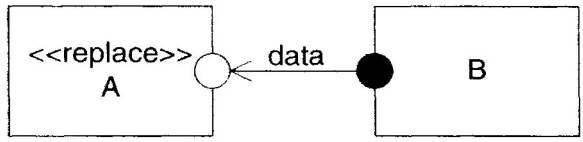
3.3. REPLACEABLE COMPONENTS

Conventional system structure diagrams always show a system in normal operation, e.g. an ordinary electronic purse card component with the specified functionality communicating with the point-of-sale component. In addition to manipulation of the communication link (man-in-the-middle attack), another attack scenario is imaginable: the attacker could try to communicate with the rest of the system *in place of* the purse card. In this case, there is no ordinary purse card in the system, but a faked one (that in particular usually does not contain private keys of ordinary cards, except if they leaked).

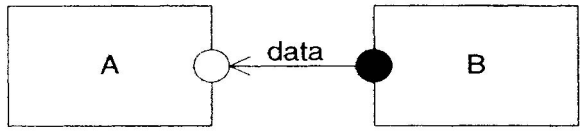
We mark components that can be replaced by faked ones by the attacker with the <<replace>> tag, and components that can not be replaced with the <<nonreplace>> tag. If neither <<replace>> nor <<nonreplace>> is used for a component, the component is non replaceable by default.

Definition 4 (Replaceable Component). Figure 3 shows the semantics of a <<replace>> component. Using a replaceable component (Figure 3(a)) is an abbreviation for specifying two different system scenarios. The first scenario describes the structure of the system with the specified component `A` (Figure 3(b)). In the second scenario (Figure 3(c)) the attacker exchanges the component by another component `A'`. `A'` has the same component signature like `A` but has an arbitrary behaviour that can be defined by the threat model.

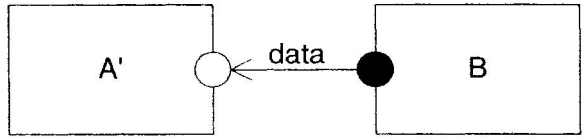
In the development process, replaceable components should be identified during the analysis phase together with the identification of private and public



(a) ≪replace≫ Component



(b) Scenario 1: Component not Exchanged



(c) Scenario 2: Component Exchanged

Figure 3 Semantics of a Replaceable Component

channels. It is only necessary to analyze security critical components with regard to replaceability.

3.4. ENCAPSULATED COMPONENTS

An encapsulated component is a component that only consists of not publicly accessible subcomponents. In this way an attacker has no possibility to manipulate or exchange the subsystems of this component. Furthermore, the communication within the component cannot be overheard. The security tag <<node>> is used to mark a component as an encapsulated one. The identification of encapsulated components is done together with the identification of private and public channels and replaceable components.

Definition 5 (Consistency Condition of <<node>>). A <<node>> component only consists of <<private>> channels and <<nonreplace>> components.

One example for an encapsulated component is an automated teller machine (ATM). An ATM is encapsulated in a way that unauthorized persons are not

able to manipulate system parts. Overhearing the internal communication is also not possible.

3.5. ACTOR COMPONENTS

Most systems interact with their system environments. It is often desired to illustrate the system environment in the graphical system design. Components that are not part of the system are called actors. We point out actors by using the tag <<actor>>. A typical example for an actor is a system user. The system user interacts with the system without being part of it.

Actor components can never be marked with the <<critical>> tag. An actor is not part of the system and therefore there is no need to analyze the actor itself with respect to security aspects. But an actor can interact with our system in a way that affects our security requirements. To visualize these critical interactions, channels between actors and the system components can be annotated with the <<critical>> tag.

3.6. SECRECY AND AUTHENTICITY

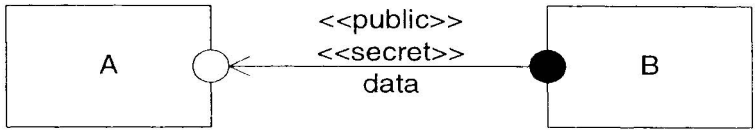
The most important security properties of communication channels—in addition to integrity, availability and non-repudiation—are authenticity and secrecy. For this purpose, we will introduce tags <<secret>> and <<auth>> for channels in the SSDs. The security properties of channels are identified in the high-level design phase, taking place after the activities of the analysis phase. It is only necessary to specify security properties for security critical, public channels.

There are many possible definitions for authenticity and secrecy in the security literature (see (Gollmann, 1996)). In the following, we give a definition based on our model.

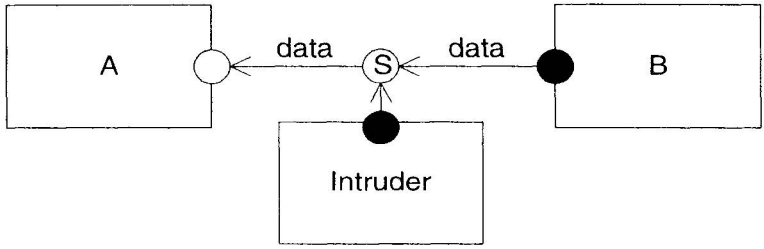
During the high-level design phase, we assume that the defined requirements of secrecy and authenticity are fulfilled automatically, if the corresponding tags appear on the channels. Consequently these requirements restrict the possibilities of an attacker. In the low-level design, the validity of these requirements has to be ensured by proper mechanisms.

Definition 6 (Secret Channel in High-Level Design). A message sent on a <<secret>> channel can only be read by its specified destination component. Therefore, we can assume in high-level design that a <<secret>> and <<public>> channel can not be read by the intruder. But the intruder could write something on it. Figure 4 shows the semantics of a secret and public channel.

Definition 7 (Authentic Channel in High-Level Design). A message received on an <<auth>> channel can only come from its specified source component. Therefore, an <<auth>> and <<public>> channel can not be written



(a) ≪secret≫ and ≪public≫ Channel



(b) Intruder can Write Data*

* The "S" circle represents a switch component that distributes all incoming data to all outgoing channels.

Figure 4 Semantics of a Secret and Public Channel

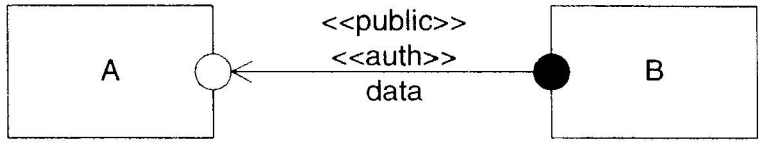
by the intruder. But the intruder could possibly read data from it. Figure 5 illustrates the semantics of an authentic channel.

There are some relations between our notion of security critical and secrecy and authenticity. A security critical channel references to data that should be protected against attackers (see Definition 1). Secrecy and authenticity are security properties. These security properties defines concrete requirements concerning the protection of data against attackers.

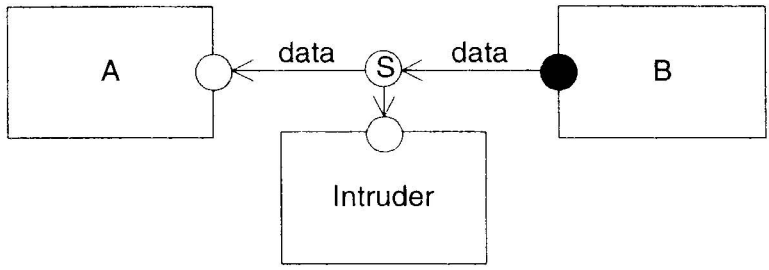
Definition 8 (Consistency Condition of <<secret>> and <<auth>>). If a channel is marked to be security critical and the communication is visible for an attacker, the data sent via the channel must be protected in a suitable manner. In this case, during the high-level design phase the protection of data must be ensured by a security property. A <<critical>> and <<public>> channel must be <<secret> or <<auth>> or both.

3.7. INTEGRITY

We assume that <<secret>> or <<auth>> channels also provide message integrity, i.e. a message received on an <<auth>> channel is guaranteed not to



(a) ≪auth≫ and ≪public≫ Channel



(b) Intruder can Read Data

Figure 5 Semantics of an Authentic and Public Channel

have been modified. In future, the integrity property could also be modelled separately.

3.8. CHANNEL REQUIREMENTS IN LOW-LEVEL DESIGN

In the low-level design phase, taking place after the high-level design phase, the system specification is refined. In this phase, security functionalities can be used to ensure the security properties and requirements.

We can define the usage of symmetric encryption, asymmetric encryption and corresponding signatures to realize the requirements of secrecy and authentification. The security tag <<sym>> marks a channel. It defines that the realization of the channel must use a symmetric encryption algorithm to ensure the secrecy requirements. Furthermore the <<asym>> tag is used to define the usage of an asymmetric encryption algorithm.

It is also possible to specify the encryption algorithm that should be used to guarantee secrecy. The security tag <<encalg <Name> [Parameters]>> can be used together with a channel to specify the usage of a specific encryption algorithm. We can specify additional parameters of the algorithm. For example it is possible to define the key length of the encryption keys. Authenticity can

be realized by using specific authentification protocols. The tag <<authprotocol <Name> [Parameters]>> defines a specific authentification protocol to be used.

By choosing a specific encryption algorithm for realizing secrecy, we perform a refinement step. Special encryption drivers are introduced to perform the encryption and decryption tasks. The data that is sent between the two encryption drivers is encrypted. To realize a specific authentification protocol, special protocol drivers are introduced. Furthermore, additional channels between the protocol drivers are needed to allow bidirectional communication.

After these refinements, the statements on access of the intruder to the channel in Definition 6 and Definition 7 change for <<public>> channels: the intruder now does have read and write access to the channel. The encryption mechanism on the channel must ensure that the intruder cannot manipulate the channel in an improper way, so that the security requirements stated by <<secret>> and <<auth>> are still fulfilled.

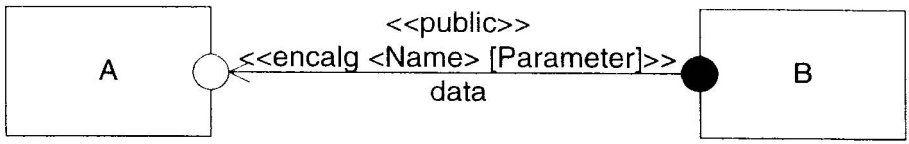
Definition 9 (Secret and Authentic Channels in Low-Level Design). In low-level design, <<public>> channels implementing secret or authentic communication have to be modelled by including appropriate security functionality.

A convenient way to do this is using *security patterns.* Security patterns are generic solutions for common security problems. Figure 6 shows such a pattern for the simple case of encryption (guaranteeing secrecy). The communicating components now include protocol drivers for encryption and decryption of the messages, so the original channel is replaced by an encrypted one. This encryption pattern could be extended by including protocol drivers for key agreement, a public key server or a whole public key infrastructure. Other security patterns, for instance providing auditing functionality, are possible.

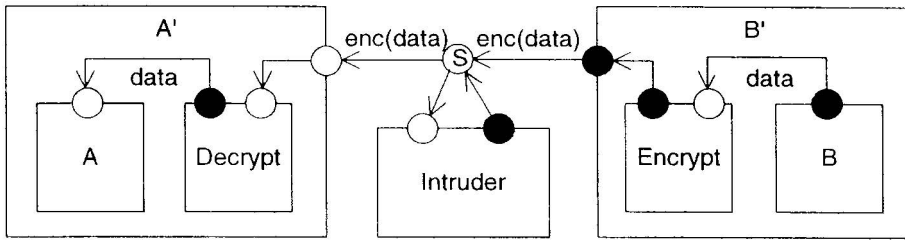
4. EXAMPLE—ELECTRONIC PURSE TRANSACTION SYSTEM

In the previous section, we have introduced extensions for SSDs to deal with security requirements. Now, we use the extended SSDs to model an example system from the area of E-Commerce, an electronic purse card system. The given specification conforms with the *Common Electronic Purse Specification* (CEPS), an international standard for an electronic purse environment (CEPSCO, 2000). Figure 7 shows the complete system structure of the electronic purse system consisting of several sub-components.

The system user possesses an electronic purse card. The card has some amount of money stored on it and the user can pay with the card at a dealer using a point-of-sale (POS) terminal. The electronic purse card is involved in several security requirements. For example, together with other components the electronic purse card must ensure that an attacker can not steal any money



(a) ≪public≫ Channel using a Specific Encryption Algorithm



(b) Intruder can Read and Write Encrypted Data

Figure 6 The Encryption Security Pattern

from the whole transaction system. Therefore the card is a security critical component (<<critical>>). Furthermore, the purse card should be realized by a single integrated chip. It is an encapsulated component (<<node>>) and consequently we assume an attacker has no possibility to overhear or manipulate the communication within the electronic purse card. An attacker could try to produce a faked card and he could replace the valid card by the faked one. The security tag <<replace>> of the component `ElectronicPurse` visualizes this possibility.

The electronic purse card can communicate with the other components over its interface. It offers ports to read the balance, decrease the balance and set the balance to a specific value. The `getBal` operation of the card is used by the `POS`, a card loading terminal at the bank of the card owner (`IssuerBank`) and by the `CardReader` component. The operation of reading the balance is not security critical because everybody who possesses an electronic purse is allowed to read the stored balance of the card. Consequently the channels `getBal` and `returnBal` do not have the <<critical>> tag. But an intruder could build a special adapter to overhear all communication between the electronic purse card and the other components. To express this possibility the communication channels `getBal`, `returnBal`, `decreaseBal` and `setBal` are marked with the <<public>> tag.

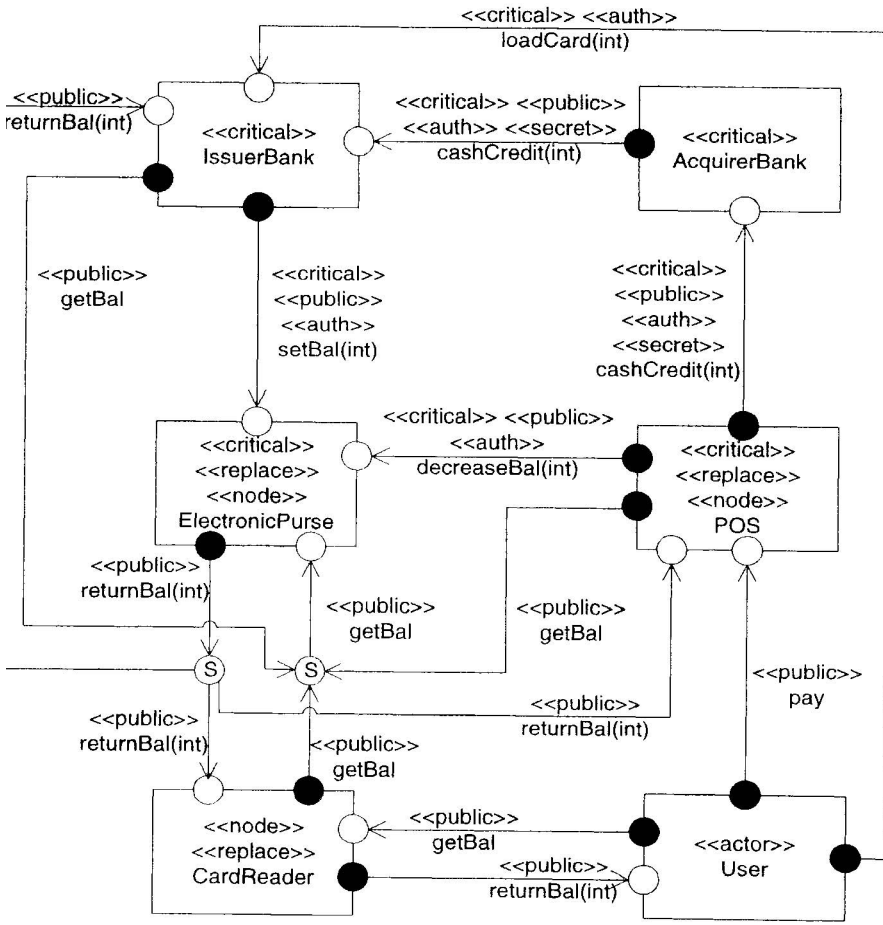


Figure 7 Electronic Purse Card System

If the user wants to load money onto his card, he must go to his bank (IssuerBank) and use a special card terminal. The channel setBal is used to transfer money from his bank account to the card. This channel is security critical, because it affects the security requirement about an attacker stealing money. We use the property of authentification (<<auth>>) to ensure that the card and the card terminal are valid.

The card reader is a simple device for the user to check the amount of money that is stored on the card. An attacker can exchange the card reader by another component (<<replace>>). On the other hand the card reader component is encapsulated (<<node>>). No security requirements are defined for this component and therefore the component is not security critical.

The POS component is a card terminal located at a dealer. The user can insert the electronic purse card into the terminal in order to pay for goods. The POS is directly involved in the money transaction process. Therefore this component is security critical. The POS is encapsulated to prevent manipulations within the unit. But a malicious dealer has the possibility to exchange the POS terminal by a faked unit.

The POS component can instruct the electronic purse card to decrease its balance by a given amount. The operation decreaseBal is part of the money transaction process and the communication to perform the operations is security critical. To comply with our security requirement that an intruder cannot steal any money, we must ensure that only a valid POS is able to decrease the amount of money on a money card. The POS must authenticate itself in a way that the card is sure to communicate with a valid POS. This fact is visualized in the SSD by the <<auth>> tag annotated to the decreaseBal channel.

The dealer can submit the earned money to his bank (AcquirerBank) using the cashCredit channel. This channel is security critical. The communication medium is a standard telephone line and the potential attacker has possibilities to overhear and manipulate the communications. This fact is expressed using the <<public>> tag on this channel. To ensure that an attacker can not overhear or manipulate the transferred data, the <<secret>> and <<auth>> tags are used.

The channel cashcredit between the two banks is used to transfer money from one bank to the other. The communication takes place via an open network (<<public>> channel). We must ensure secrecy and authenticity for this channel to protect the transactions data. Both bank components are security critical, but we do not see a risk that a potential attacker can act as a faked bank component. Thus the <<replace>> tag is not used for both banks.

Finally let us have a look at the User. The user is not part of our system. Therefore the <<actor>> tag is used. The user can initiate the paying process at a POS. The initiation of the paying process is not security critical (it just corresponds to inserting the card, whereas the amount of money to be withdrawn is negotiated outside of the system). The critical part of the paying process takes

place between the money card and the POS. Furthermore, the user can load the card with some amount of money. During this action, an amount of money is transferred from his bank account onto the card. This operation is security critical because an attacker could try to transfer money from a foreign account to his own electronic purse card. Thus we need some kind of authentification to perform this operation, e.g. the user must enter a PIN code before the money transaction is performed.

5. CONCLUSIONS AND FURTHER WORK

This work is only the beginning of an effort to extend graphical description techniques for distributed systems with security aspects to support methodical development of security critical systems. We used the CASE tool AUTOFOCUS, the description techniques of which are related to UML-RT, for its simplicity and clear semantics and the possibility to give our security extensions a straightforward and unambiguous meaning.

We showed how to extend AUTOFOCUS system structure diagrams by security tags, both for high-level and low-level design. The transition from high-level to low-level design is aided by the possibility to use security patterns. The description techniques were illustrated with the help of an example from the field of E-Commerce, an electronic purse card system.

We focused on the consideration of channels and system structure. In future, additional security properties such as integrity and availability are to be included. The specification of channels and components in low-level design needs to be detailed, using classifications as pointed out in (Eckert, 1998). Besides, it seems very promising to further examine security patterns providing generic architectures for specific security functionality and evaluate their use within the development process. The refinement of security requirements and security functionalities together with its influence on correctness verification is also part of our research activities.

Also, state transition diagrams (STDs) specifying the behaviour of a component can be extended in a similar way with security properties. For this purpose, it suggests itself to classify the data received and sent on the ports and to use models such as Bell-LaPadula or non-interference—similar as it is done in (Jürjens, 2001) for Statechart diagrams. When the behaviour of components is specified, formal proofs can be carried out (by hand or automatically via model checking) that the specified security properties are fulfilled.

EETs (extended event traces) can also be enriched by cryptographic primitives and security properties, and thus be used to specify and verify security functionality of a component. Examining software development of security critical systems with the help of AUTOFOCUS EETs (using protocols from the CEPS purse card system as a case study) is subject of ongoing work.

Acknowledgments

Helpful comments and encouragement from Oscar Slotosch are gratefully acknowledged.

References

Bell, D. E. and LaPadula, L. (1973). Secure computer systems: Mathematical foundations and model. Technical Report M74-244, The MITRE Corp., Bedford MA.

Broy, M., Dederich, F., Dendorfer, C., Fuchs, M., Gritzner, T., and Weber, R. (1992). The Design of Distributed Systems—An Introduction to FOCUS. Technical Report TUM-I9202, Technische Univerität München.

Broy, M. and Slotosch, O. (1999). Enriching the Software Development Process by Formal Methods. In *Current Trends in Applied Formal Methods 1998,* pages 44–61.

Burrows, M., Abadi, M., and Needham, R. (1989). A logic of authentication. *Proceedings of the Royal Society of London A,* 426:233–271.

CEPSCO (2000). Common electronic purse specifications: Business requirements. Version 7.0, available from http://www.cepsco.com.

Common Criteria (1999). Common criteria for information technology security evaluation version 2.1. Technical report, Communications Security Establishment (Canada), Service Central de la Sécurité des Systèmes d'Information (France), Bundesamt für Sicherheit in der Informationstechnik (Germany), Netherlands National Communications Security Agency, Communications-Electronics Security Group (United Kingdom), National Institute of Standards and Technology (United States), National Security Agency (United States).

Eckert, C. (1998). *Sichere, verteilte Systeme – Konzepte, Modelle und Systemarchitekturen.* professorial thesis, Technische Universität München.

Goguen, J. A. and Meseguer, J. (1998). Security Policy and Security Models. In *Proceedings of 1982 IEEE Symposium on Security and Privacy.*

Gollmann, D. (1996). What do We Mean by Entity Authentication? In *Proceedings of 1996 IEEE Symposium on Security and Privacy.*

Huber, F., Molterer, S., Rausch, A., Schätz, B., Sihling, M., and Slotosch, O. (1998a). Tool supported Specification and Simulation of Distributed Systems. In *International Symposium on Software Engineering for Parallel and Distributed Systems,* pages 155–164.

Huber, F., Molterer, S., Schätz, B., Slotosch, O., and Vilbig, A. (1998b). Traffic Lights—An AutoFocus Case Study. In *1998 International Conference on Application of Concurrency to System Design,* pages 282–294. IEEE Computer Society.

ITSEC (1990). ITSEC. Information Technology Security Evaluation Criteria—Harmonised Criteria of France, Germany, the Netherlands, the United Kingdom. Version 1.

ITU (1996). ITU-TS Recommendation Z. 120: Message Sequence Chart (MSC). ITU-TS, Geneva.

Jones, M. P. (August 1993). *An Introduction to Gofer.*

Jürjens, J. (2001). Towards Development of Secure Systems using UML. In *FASE '01: Fundamental Approaches to Software Engineering.* to appear.

Lotz, V. (2000). Formally Defining Security Properties with Relations on Streams. In Schneider, S. and Ryan, P., editors, *Electronic Notes in Theoretical Computer Science,* volume 32. Elsevier Science Publishers.

Lowe, G. (1996). Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR. In Margaria and Steffen, editors, *TACAS,* volume 1055 of *lncs,* pages 147–166. sv.

Paulson, L. C. (1998). The inductive approach to verifying cryptographic protocols. *Journal of Computer Security,* 6(1–2):85–128.

Philipps, J. and Slotosch, O. (1999). The Quest for Correct Systems: Model Checking of Diagramms and Datatypes. In *Asia Pacific Software Engineering Conference 1999.*

Slotosch, O. (1998). Quest: Overview over the Project. In Hutter, D., Stephan, W., Traverso, P., and Ullmann, M., editors, *Applied Formal Methods—FM-Trends 98,* pages 346–350. Springer LNCS 1641.

Thayer, F., Herzog, J. C., and Guttman, J. D. (1998). Strand Spaces: Why is a security protocol correct? In *Proceedings of 1998 IEEE Symposium on Security and Privacy.*

Thompson, S. (1999). *Haskell: The Craft of Functional Programming.* Addison-Wesley Longman.

Wimmel, G., Lötzbeyer, H., Pretschner, A., and Slotosch, O. (2000). Specification Based Test Sequence Generation with Propositional Logic. *Journal on Software Testing Verification and Reliability,* 10:229–248.

Wimmel, G. and Wisspeintner, A. (2000). The Needham-Schroeder Protocol—an AutoFocus Case Study. Internal report, Technische Universität München.