Modelling and conception of software in aspect to security

Seminar: Software Architecture
Universität Hamburg
Shahin Imanverdiyev & Senad Ličina

ABSTRACT

put Abstract here. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec nec efficitur enim. Mauris leo metus, fermentum dapibus lobortis eu, gravida at libero. Vestibulum eu sem eu sem finibus convallis nec at lectus. Donec finibus dui eget eros bibendum, sit amet pharetra odio rutrum. Etiam sodales suscipit mattis. Sed molestie sagittis rutrum. Vivamus non luctus velit, a venenatis felis. In non sem lacinia, vehicula nisi eget, elementum nibh. Curabitur faucibus est leo, vel placerat lacus dapibus eu. In ut enim a nisl facilisis viverra ut nec tellus.

1. INTRODUCTION

Modern society and economy increasingly depend on digital systems. As attacks against such systems can have devastating results, it is very important to secure such systems in a proper way. Nowadays the Internet is a heavily used and very important communication medium. An increasing number of devices are connected to the Internet, which is why it is important to transmit and store sensitive data securely.

The correct development of secure software is difficult. There have been numerous successful attacks abusing vulnerabilities of software systems in the past and it can be expected that people will try to detect and abuse such flaws in the future as well.

The traditional approach for security assurance has been "penetrate and patch", where security is assured by attempting to break into a running system and exploiting well-known vulnerabilities. Penetrate and patch happens too late in the development process and vulnerabilities will be available and possibly exploited until they are recognized fixed. This is why it is important to take security aspects into account in an early stage of the system development.

2. REQUIREMENTS

In this section some concepts are explained which are required to understand this paper. First the key concepts of security are discussed, ...

2.1 Security Principles

The CIA Triad is a well-known security model which is based on the three key principles of security: **confidentiality**, **integrity** and **availability**. It is used to identify problem areas in the security of software systems and requires that its principles are preserved for the system resources.

Confidentiality ensures that only authorized users are able to read private or confidential information. Access control mechanisms can be used to gain better confidentiality.

Integrity (in the field of computer security) prevents information from modification or deletion by unauthorized users and ensures that undesired changes can be undone.

Availability refers to the prevention of unauthorized denial of access to information or resources.

These key principles are often described as incomplete and thus extended by further principles like **authenticity**, **accountablility** or **non-repudiation**.

Authenticity is the assurance that information and resources are valid. The validation that all involved parties are truly who they claim to be is important for authenticity.

Accountablility refers to the property that every security crucial action can be traced back to the responsible

Non-repudiation assures that the existence of specific actions can not be denied.

2.2 Unified Modeling Language

> "The Unified Modeling Language (UML) is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system."
>
> – [BJR99, Chapter 1]

The Unified Modeling Language (UML) is the prevalent language for the specification of object orientated software systems. It describes architecture and behavior of software in a graphical manner using various diagram types.

Use case diagrams model a systems functionality as percieved by users (or actors). Use case diagrams consist of use cases and actors, they can be used to represent interactions between system and user. An example for a use case diagram is given with figure 1. The actors (customer, online banking & banking system) are depicted by stick-figures, the use cases (transfer funds, deposit funds & withdraw cash) are represented as ovals.

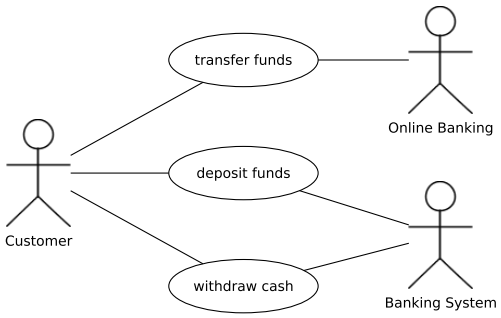

Fig. 1. Example of an use case diagram representing possible interactions between a customer and banking interfaces.

Class diagrams model static concepts within the implementation of an application. Classes are drawn as rectangles consisting of three sections for its name, its attributes and its operations. Relationships among classes are drawn as paths between them, whereby different arrow types have different meanings. Figure 2 shows a class diagram representing different implementations of the *AccountInterface* and an implementation for the *CustomerInterface*. The classes *CheckingAccount* and *SavingsAccount* are inheriting from the class *Account* which is implementing the *AccountInterface*.

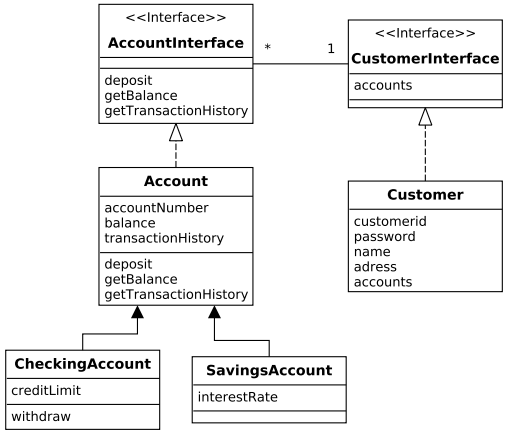

Fig. 2. A class diagram representing example classes for customer and accounts.

Statechart diagrams are visual representations of state machines. State machines model the sequences of states which an entity can go through. States are connected by transitions, which are allowed to have conditions. The initial state is depicted by a black dot, the final state as a black dot in a circle. Figure 3 shows the states an account object can go through.

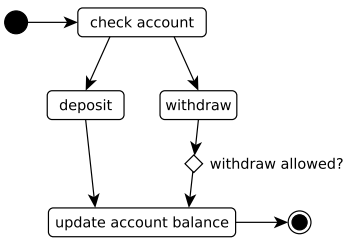

Fig. 3. A statechart diagram depicting the simplified processes cash withdrawal and deposition on an account.

Activity diagrams are a special case of statechart diagrams, used to model processes involving one or more components. Activity diagrams make use of synchronization bars to distinguish between sequential and concurrent groups of activities.

Sequence diagrams show object interactions by arranging messages in time sequence. The vertical lines are lifelines which are representing classifier roles. Messages are drawn as arrows between lifelines. An example sequence diagram for the scenatio where a customer is withdrawing money at an ATM is given with figure 4. Figure 4 shows an sequence diagram for the example scenario of money withdrawal at an ATM.

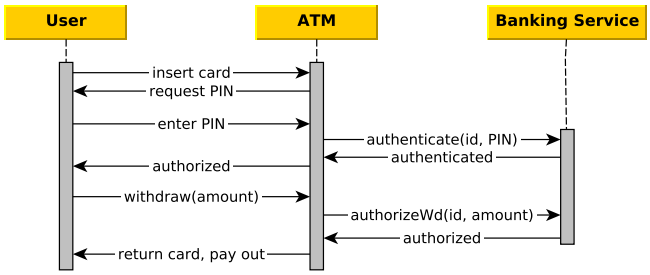

Fig. 4. Money withdrawal at an ATM as a sequence diagram.

Deployment diagrams show networks of conneted nodes, their component instances and objects. Nodes are drawn by boxes and may contain components, which are drawn as rectangles with two smaller rectancles on the left side. Communication links (on the physical level) are drawn as solid lines connecting nodes, whereas communication dependencies are drawn with broken arrows. Communication dependencies can define interfaces and contain class models. Figure 5 is a deployment diagram of an example setup for online banking.

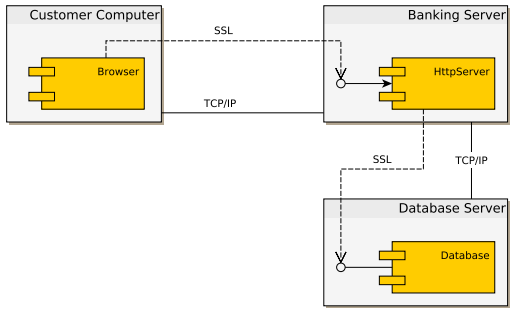

Fig. 5. Online banking infrastructure as a deployment diagram.

Subsystems can be seen as special packages, grouping model elements which are representing a portion of a system. They can be used to describe relations between the elements they contain. Subsystems may have interfaces and connections to other subsystems.

The Unified Modeling Language offers some *extensibility mechanisms*, an easy way to extend UML is by defining **stereotypes**, **tagged values** and **constraints**.

Stereotypes can be used to define new model elements based on already existing model elemets by extending them. The notation for the use of stereotypes is defined by the symbol for the base element with a keyword above the name of the element. The keyword of a stereotype is its name within matched guillemets. Stereotypes could for example be used in a class diagram to classify method behaviour such as the «constructor» stereotype.

Tagged values are defined as selector-value pairs which may be attached to an element in order to augment it with further information. The selector (or *tag*) is a string and whereas the value can be of various types, it is encoded as string. Tagged values are noted within curly brackets and an equal sign between tag and value, like *{ tag = value }*.

Constraints are another possibility to enrich an element with further information. They are defined as semantic conditions which have to be fulfilled by the elements they are attached to.

3. MODELING SECURE SOFTWARE WITH UML

3.1 UMLsec

Table II. UMLsec tags as defined in [Jür06, Fig. 4.2].

Tag	Stereotype	Type	Mult.	Description
start	fair exchange	state	*	start states
stop	fair exchange	state	*	stop states
adversary	fair exchange	adversary model	1	adversary type
action	provable	state	*	provable action
cert	provable	expression	*	certificate
adcersary	provable	adcersary model	*	adversary type
protected	rbac	state	*	protected resources
role	rbac	(actor, role)	*	assign role to actor
right	rbac	(role, right)	*	assign right to role
secrecy	critical	data	*	
integrity	critical	(variable, expression)	*	integrity of data
authenticity	critical	(data, origin)	*	authenticity of data
high	critical	message	*	high-level message
fresh	critical	data	*	fresh data
adversary	secure links	adversary model	1	adversary type
adversary	data secutiry	adversary model	1	adversary type
integrity	data secutiry	(variable, expression)	*	authenticity of data
guard	guarded	object name	1	guard object

4. CONCLUSION

Typical Conclusion.

5. REFERENCES

[BJR99] Grady Booch, Ivar Jacobson, James Rumbaugh: The Unified Modeling Language Reference Manual. Addison-Wesley, Reading (US-MA) 1999.

[JS11] Jan Jürjens, Holger Schmidt: UMLsec 4 UML2 - Adopting UMLsec to Support UML2. Technische Universität Dortmund, Dortmund, Germany 2011.

[Jür06] Jan Jürjens: Secure Systems Development with UML. Springer-Verlag, Berlin, Germany 2006.

[SHRB11] Matthias Straka, Stefan Hauswiesner, Matthias Rüther, Horst Bischof: Skeletal Graph Based Human Pose Estimation in Real-Time. Graz University of Technology, Austria 2011.

Table I. UMLsec stereotypes as defined in [Jür06, Fig. 4.1].

Stereotype	Base Class	Tags	Constraints	Description
fair exchange	subsystem	start, stop, adversary	after start eventually reach stop	enforce fair exchange
provable	subsystem	action, cert, adversary	action is non-deniable	non-repudiation requirement
rbac	subsystem	protected, role, right	only permitted activities executed	enforces role-based access control
Internet	link			Internet connection
encrypted	link			encrypted connection
LAN	link, node			LAN connection
wire	link			wire
smart card	node			smart card node
POS device	node			POS device
issuer node	node			issuer node
secrecy	dependency			assumes secrecy
integrity	dependency			assumes integrity
high	dependency			high sensitivity
critical	object, subsystem	integrity, authenticity, high, fresh		
secure links	subsystem	adversary	dependency security matched by links	enforces secure communication links
secure dependency	subsystem		«call», «send» respect data security	structural interaction data security
data security	subsystem	adversary, integrity, authenticity	freshness	
no down-flow	subsystem		prevents down-flow	information flow condition
no up-flow	subsystem		prevents up-flow	information flow condition
guarded access	subsystem		guarded objects accessed through guards	access control using guard objects
guarded	object	guard		guarded object