A Semantic Paradigm for Component-Based Specification Integrating a Notion of Security Risk

Gyrd Brændeland[1,2,⋆] and Ketil Stølen[1,2]

[1] Department of Informatics, University of Oslo, Norway
[2] SINTEF, Norway
gyb@sintef.uio.no

Abstract. We propose a semantic paradigm for component-based specification supporting the documentation of security risk behaviour. By security risk, we mean behaviour that constitutes a risk with regard to ICT security aspects, such as confidentiality, integrity and availability. The purpose of this work is to investigate the nature of security risk in the setting of component-based system development. A better understanding of security risk at the level of components facilitates the prediction of risks related to introducing a new component into a system. The semantic paradigm provides a first step towards integrating security risk analysis into the system development process.

Keywords: component, formal specification, risk analysis, security.

1 Introduction

The flexibility of component-oriented software systems enabled by component technologies such as Sun's Enterprise Java Beans (EJB), Microsoft's .NET or the Open Source Gateway initiative (OSGi) gives rise to new types of security concerns. In particular the question of how a system owner can know whether to trust a new component to be deployed into a system. A solution to this problem requires integrating the process of security risk analysis in the early stages of component-based system development. The purpose of security risk analysis is to decide upon the necessary level of asset protection against security risks, such as a confidentiality or integrity breach. Unfortunately, the processes of system development and security risk analysis are often carried out independently with little mutual interaction. The result is expensive redesigns and unsatisfactory security solutions. To facilitate a tighter integration we need a better understanding of security risk at the level of components. But knowing the security risks of a single component is not enough, since two components can affect the risk level of each other. An example is the known buffer overflow vulnerability of previous versions of the media player Winamp, that may allow an unauthenticated attacker using a crafted file to execute arbitrary code on a vulnerable system. By default Internet Explorer opens affected files without prompting the user [20]. Hence, the probability of a successful attack is much higher if a user

⋆ Corresponding author.

T. Dimitrakos et al. (Eds.): FAST 2006, LNCS 4691, pp. 31–46, 2007.

utilises both Internet Explorer and Winamp, than only one of them. As this example illustrates we need a strategy for predicting system level risks that may be caused by introducing a new component. A better understanding of security risk at the level of components is a prerequisite for compositional security level estimation. Such understanding also provides the basis for trust management, because as argued by Jøsang and Presti [12] there is a close dependency between trust and risk. The contributions of this paper is a novel semantic paradigm for component-based specification explaining

- basic components with provided and required interfaces;
- the composition of components into composite components;
- unpredictability which is often required to characterise confidentiality properties (secure information flow);
- the notion of security risk as known from asset-oriented security risk analysis.

This paper is divided into ten sections. In Sections 2 and 3 we explain our notions of security risk analysis and component. In Section 4 we introduce the basic vocabulary of the semantic model. In Sections 5 to 7 we define the semantic paradigm for component-based specifications. In Section 8 we describe how security risk analysis concepts relate to the component model and how they are represented in the semantics. In Section 9 we attempt to place our work in relation to ongoing research within related areas and finally, in Section 10, we summarise our findings.

2 Asset-Oriented Security Risk Analysis

By security risk analysis we mean risk analysis applied to the domain of information and communication technology (ICT) security. For convenience we often use *security analysis* as a short term for *security risk analysis*. ICT security includes all aspects related to defining, achieving and maintaining confidentiality, integrity, availability, non-repudiation, accountability, authenticity and reliability of ICT [11].

Hogganvik and Stølen [7] have provided a conceptual model for security analysis based on a conceptual model originally developed in the CORAS project [3]. The CORAS risk management process is based on the the "Code of practise for information security management" (ISO/IEC 17799:2000) [9] and the Australian/New Zealand standard "Risk Management" (AS/NZS 4360:2004) [22].

With some adjustments the model is expressed as a class diagram in UML 2.0 [18], see Figure 1. The associations between the elements have cardinalities specifying the number of instances of one element that can be related to one instance of the other. The hollow diamond symbolises aggregation and the filled composition. Elements connected with an aggregation can also be part of other aggregations, while composite elements only exist within the specified composition.

We explain Figure 1 as follows: *Stakeholders* are those people and organisations who may affect, be affected by, or perceive themselves to be affected by, a decision or activity or risk [22]. The CORAS security analysis process is

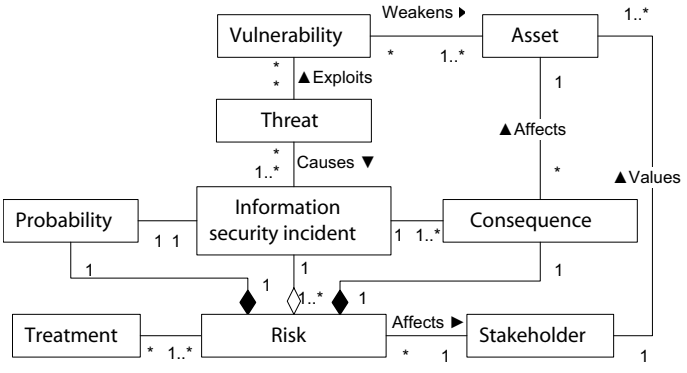

Fig. 1. CORAS conceptual model of security analysis terms

asset-oriented. An *asset* is something to which a stakeholder directly assigns value and, hence, for which the stakeholder requires protection [23][1]. CORAS links assets uniquely to their stakeholders. A *vulnerability* is a weakness of an asset or group of assets that can be exploited by one or more threats [11]. A *threat* is a potential cause of an incident that may result in harm to a system or organisation [11]. An *information security incident* refers to any unexpected or unwanted event that might cause a compromise of business activities or information security, such as malfunction of software or hardware and access violations [11]. A *risk* is the combination of the *probability* of an event and its *consequence* [10]. Conceptually, as illustrated in Figure 1, a risk consist of an information security incident, the probability of its happening and its consequence. *Probability* is the extent to which an *event* will occur [10]. *Consequence* is the outcome of an event expressed qualitatively or quantitatively, being a loss, injury or disadvantage. There may be a range of possible outcomes associated with an event [23]. This implies that an information security incident may lead to the reduction in value of several assets. Hence, an information security incident may be part of several risks. *Risk treatment* is the process of selection and implementation of measures to modify risks [10].

3 The Component Model

There exist various definitions of what a software component is. The classic definition by Szyperski [24] provides a basic notion of a component that is widely adopted in later definitions: "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

[1] The Australian handbook [23] uses the term *organisation* instead of the broader term stakeholder. For simplicity of the conceptual model we prefer the broader term stakeholder which includes organisation.

Lau and Wang [15] criticise Szyperski's definition for not relating the component concept to a component model. Lau and Wang emphasise the importance of a component model as a provider of an underlying semantic framework, defining:

- the *syntax* of components, i.e., how they are constructed and represented;
- the *semantics* of components, i.e. what components are meant to be;
- the *composition* of components, i.e. how they are composed or assembled.

Lau and Wang present a taxonomy of current component models, comparing their similarities and differences with regard to these three criteria. They compare the component models facilities for composition both in the design phase and the deployment phase. Our approach focuses on the specification of components. Hence, composition takes place in the design phase.

According to Cheesman and Daniels [2] the main motivation for using a component-oriented approach is to make dependencies explicit, in order to facilitate management of component systems and independent deployment of components. Since the client of a component is not necessarily the same as the deployer of the component, they distinguish between two types of contracts corresponding to these two roles: usage and realisation contracts. This distinction motivates the separation of specifications into interfaces and components. Our conceptual model of a component, shown in Figure 2, is inspired by the definitions given in [2].

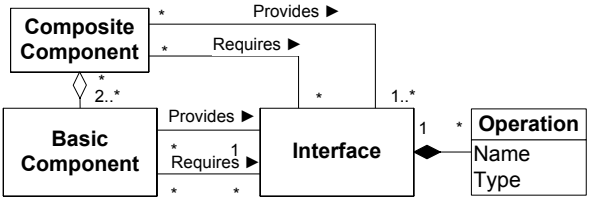

Fig. 2. Conceptual model of a component

We explain the conceptual component model as follows: An *interface* is a contract with a client, describing a set of behaviours provided by a component object. It defines a list of operations that the interface provides, their signatures and semantics. A *component* is a contract with the realiser. It describes provided interfaces and component dependencies in terms of required interfaces. By required interface we mean the calls the component needs to make, in order to implement the operations described in the provided interfaces. We distinguish between basic components and composite components. A basic component provides only one interface. We obtain components with more than one provided interface by the composition of basic components. Composite components can also be combined to obtain new composite components.

4 The Semantic Model of STAIRS

We build our semantic paradigm on top of the trace semantics of STAIRS [6,5]. STAIRS is an approach to the compositional development of UML 2.0 interactions. For a thorough account of the STAIRS semantics, see Haugen et al. [6,5].

The most common interaction diagram is the sequence diagram, which shows a set of messages arranged in time sequence [18]. A sequence diagram typically captures the behaviour of a single scenario. A sequence diagram describes one or more positive (i.e. valid) and/or negative (i.e. invalid) behaviours.

The sequence diagram in Figure 3 specifies a scenario in which the client lifeline sends the message displayAcc to the bank lifeline, which then sends the message check with argument *pin* to the environment. When the bank lifeline receives the message ok it sends the message acc to the client lifeline.

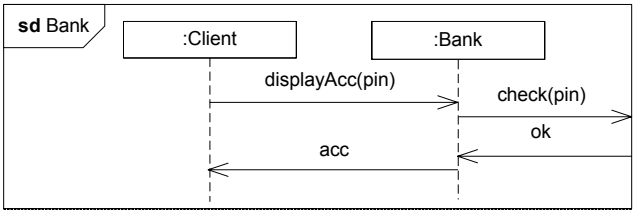

Fig. 3. Example interaction

Formally STAIRS uses denotational trace semantics in order to explain the meaning of a single interaction. A trace is a sequence of events, representing a system run. There are two kinds of events: sending and reception of a message, where a message is a triple (s, re, tr) consisting of a signal s, a transmitter lifeline tr and a receiver lifeline re. We let $\mathcal{E}$ denote the set of all events.

The set of traces described by a diagram like the one in Figure 3 are all positive sequences consisting of events such that the transmit event is ordered before the corresponding receive event, and events on the same lifeline are ordered from the top downwards. Shortening each message to the first letter of each signal, we thus get that Figure 3 specifies the trace $\langle !d, ?d, !c, ?o, !a, ?a \rangle$ where ! denotes transmission and ? reception of the message.

Formally we let $\mathcal{H}$ denote the set of all well-formed traces. A trace is well-formed if, for each message, the send event is ordered before the corresponding receive event. An *interaction obligation* (p_i, n_i) is a classification of all of the traces in $\mathcal{H}$ into three categories: the positive traces p_i, representing desired and acceptable behaviour, the negative traces n_i, representing undesired or unacceptable behaviour, and the inconclusive traces $\mathcal{H} \setminus (p_i \cup n_i)$. The inconclusive traces are a result of the incompleteness of interactions, representing traces that are not described as positive or negative by the current interaction.

The reason we operate with inconclusive traces is that sequence diagrams normally gives a partial description of a system behaviour. It is also possible to specify complete behaviour. Then every trace is either positive or negative.

5 Semantics of Basic Components

In this section we describe how basic components can be described semantically using STAIRS. Our semantic paradigm is independent of the concrete syntactic representation of specifications. In this paper we use sequence diagrams based on the semantic mapping defined in STAIRS, as they are simple to understand and well suited to exemplify parts of a component behaviour. We could have defined similar mappings for other specification languages.

A basic component has a unique identifier. In STAIRS this identifier is represented by a lifeline. As explained in Section 3 the provided interface of a basic component corresponds to the method calls it can receive and the required interface corresponds to the method calls the component needs to make to other component interfaces, in order to implement the operations described in the provided interface.

The denotation $[\![K]\!]$ of a basic component specification K in the STAIRS semantics is an interaction obligation (P_K, N_K) where P_K and N_K are the positive and negative traces over some set of component events E_K, respectively.

Example 1. The sequence diagram in Figure 4 specifies a scenario where a login lifeline receives the message login with arguments *id* and *pwd*. The login lifeline then sends the message authenticate to the environment. STAIRS uses the alt operator to describe that a system can include alternative behaviours. There are three alternatives: Firstly, when a user attempts to login she can either succeed or fail. If she fails there are two alternatives, of which only one is legal: When the login lifeline receives the reply fail it should reply with fail.

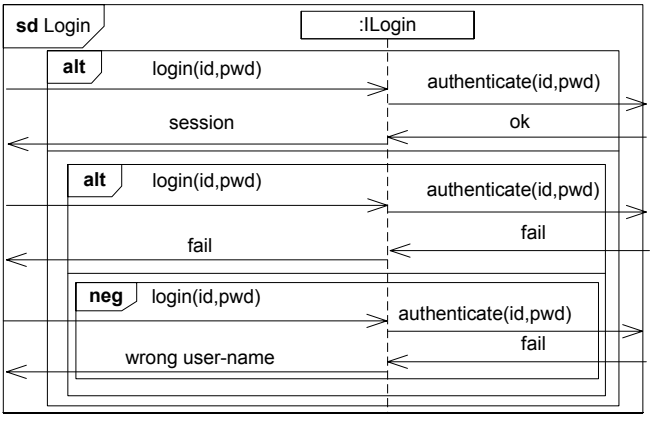

Fig. 4. Specifying component dependencies

We specify that the component should never return simply the message wrong user-name, by placing the event of returning this message within a neg construct in the sequence diagram. That is because we do not wish to reveal information

that can be useful for a potential impostor, if a login attempt fails. The sequence diagram in Figure 4 specifies an interaction obligation (P, N) where $P = \{\langle ?l, !a, ?o, !s\rangle, \langle ?l, !a, ?f, !f\rangle\}$ and $N = \{\langle ?l, !a, ?f, !w\rangle\}$ when shortening each message to the first letter of each signal. □

We define an interface and its denotation as an abstraction over a basic component. An interface describes the view of the user, who does not need to know how the login operation is implemented. We obtain the provided interface of a basic component by filtering away the interactions on the required interface. Hence, a provided interface corresponds to a basic component, if the component has no required interface.

6 Semantics of Composite Components

As described in Section 5 we distinguish between basic and composite components. A basic component provides only one interface. We obtain components with more than one interface by the composition of basic components. Composite components can also be combined to obtain new composite components.

In order to define composition we need the functions $\circledS$ for filtering of sequences, and $\circledT$ for filtering of pairs of sequences, defined by Haugen et al. [6,5]. The filtering function $\circledS$ is used to filter away elements. By $B \circledS a$ we denote the sequence obtained from the sequence a by removing all elements in a that are not in the set of elements B. For example, we have that

$$\{1, 3\} \circledS \langle 1, 1, 2, 1, 3, 2\rangle = \langle 1, 1, 1, 3\rangle$$

The filtering function $\circledT$ may be understood as a generalisation of $\circledS$. The function $\circledT$ filters pairs of sequences with respect to pairs of elements in the same way as $\circledS$ filters sequences with respect to elements. For any set of pairs of elements P and pair of sequences t, by $P \circledT t$ we denote the pair of sequences obtained from t by

- truncating the longest sequence in t at the length of the shortest sequence in t if the two sequences are of unequal length;
- for each $j \in [1, \ldots, k]$, where k is the length of the shortest sequence in t, selecting or deleting the two elements at index j in the two sequences, depending on whether the pair of these elements is in the set P.

For example, we have that

$$(1, f), (1, g) \circledT (\langle 1, 1, 2, 1, 2\rangle, \langle f, f, f, g, g\rangle) = (\langle 1, 1, 1\rangle, \langle f, f, g\rangle)$$

Parallel execution of trace sets is defined as:

$$s_1 \otimes s_2 \stackrel{\mathsf{def}}{=} \{h \in \mathcal{H} \mid \exists\, p \in \{1, 2\}^\infty : \pi_2((\{1\} \times \mathcal{E}) \circledT (p, h))) \in s_1 \wedge \pi_2((\{2\} \times \mathcal{E}) \circledT (p, h))) \in s_2\}$$

In this definition, we make use of an oracle, the infinite sequence p, to resolve the non-determinism in the interleaving. It determines the order in which events from traces in s_1 and s_2 are sequenced. π_2 is a projection operator returning the second element of a pair.

Given two components K_1 and K_2 with distinct component identifiers (lifelines). By $K_1 \otimes K_2$ we denote their composition. Semantically, composition is defined as follows:

$$[\![K_1 \otimes K_2]\!] = [\![K_1]\!] \otimes [\![K_2]\!]$$

where for all interaction obligations $(p_1, n_1), (p_2, n_2)$ we define

$$(p_1, n_1) \otimes (p_2, n_2) \stackrel{\mathsf{def}}{=} (p_1 \otimes p_2, (n_1 \otimes p_2) \cup (n_1 \otimes n_2) \cup (p_1 \otimes n_2))$$

Note how any trace involving a negative trace will remain negative in the resulting interaction obligation.

We also introduce a hiding operator δ that hides all behaviour of a component which is internal with regard to a set of lifelines L. Formally

$$[\![\delta L : K]\!] \stackrel{\mathsf{def}}{=} (\delta L : \pi_1.[\![K]\!], \delta L : \pi_2.[\![K]\!])$$

where for a set of traces H and a trace h

$$\delta L : H \stackrel{\mathsf{def}}{=} \{\delta L : h \mid h \in H\}$$
$$\delta L : h \stackrel{\mathsf{def}}{=} \{e \in \mathcal{E} \mid re.e \notin L \vee tr.e \notin L\} \circledS\, h$$

where the functions $tr.e$ and $re.e$ yields the transmitter and receiver of an event. Finally we define composition with hiding of local interaction as:

$$K_1 \oplus K_2 \stackrel{\mathsf{def}}{=} \delta(ll.[\![K_1]\!] \cup ll.[\![K_2]\!]) : K_1 \otimes K_2$$

where the function ll yields the set of lifelines of an interaction obligation.

7 Generalising the Semantics to Support Unpredictability

As explained by Zakinthinos and Lee [26], the purpose of a confidentiality property is to prevent low level users from being able to make deductions about the events of the high level users. A confidentiality property will often typically require nondeterministic behaviour (unpredictability) to achieve this. Unpredictability in the form of non-determinism is known to be problematic in relation to specifications because non-determinism is also often used to represent underspecification and when underspecification is refined away during system development we may easily also reduce the required unpredictability and thereby reduce security. For this reason, STAIRS (as explained carefully by Seehusen and Stølen [21]) distinguishes between mandatory and potential choice. Mandatory choice is used to capture unpredictability while potential choice captures underspecification. One of the main concerns in STAIRS is the ability to distinguish

between traces that an implementation *may* exhibit (e.g. due to underspecification), and traces that it *must* exhibit (e.g. due to unpredictability). Semantically, this distinction is captured by stating that the semantics of an interaction d is a *set* of interaction obligations $[\![d]\!] = \{(p_1, n_1), \ldots, (p_m, n_m)\}$. Intuitively, the traces allowed by an interaction obligation (i.e. its positive and inconclusive traces) represent potential alternatives, where being able to produce only one of these traces is sufficient for an implementation. On the other hand, the different interaction obligations represent mandatory alternatives, each obligation specifying traces where at least one must be possible for any correct implementation of the specification.

We adapt the definition of a basic component to allow mandatory behaviour alternatives as follows: The denotation $[\![K]\!]$ of a basic component is a set of interaction obligations over some set of events E_K. We also lift the definition of composition to handle unpredictability by point-wise composition of interaction obligations

$$[\![K_1 \otimes K_2]\!] \stackrel{\mathsf{def}}{=} \{o_1 \otimes o_2 \mid o_1 \in [\![K_1]\!] \wedge o_2 \in [\![K_2]\!]\}$$

The δ operator is overloaded to sets of interaction obligations:

$$[\![\delta L : K]\!] \stackrel{\mathsf{def}}{=} \{[\![\delta L : o]\!] \mid o \in [\![K]\!]\}$$

and composition with hiding is defined as before.

8 Relating Security Risk to the Semantic Paradigm

Having introduced the underlying semantic component paradigm and formalised unpredictability, the next step is to relate this paradigm to the main notions of security analysis and generalise the paradigm to the extent this is necessary. The purpose of extending the component model with security analysis concepts is to be able to specify security risks and document security analysis results of components. This facilitates integration of security analysis into the early stages of component-based system development. Security analysis documentation provides information about the risk level of the component with regard to its assets, i.e., the probability of behaviour leading to reduction of asset values. At this point we do not concern ourselves with *how* to obtain such security analysis results. We refer to [1] for an evaluation of an integrated process, applying the semantic paradigm. In the following we focus on how security analysis concepts can be understood in a component-setting and how they can be represented formally. In Sections 8.1– 8.4 we explain how the security analysis concepts of Figure 1 may be understood in a component setting. In Section 8.5 we formalise the required extensions of the semantic paradigm.

8.1 Representing Stakeholders and Threats

We represent *stakeholders* as lifelines, since the stakeholders of a component can be understood as entities interacting with it via its interfaces. We also represent

threats as lifelines. A threat can be external (e.g. hackers or viruses) or internal (e.g. system failures). An internal threat of a component is a sub-component, represented by a lifeline or a set of lifelines. An external threat may initiate a threat scenario by calling an operation of one of the component's external interfaces.

8.2 Representing Assets

For each of its stakeholders a component holds a (possibly empty) set of assets. An asset is a physical or conceptual entity of value for a stakeholder. There are different strategies we can choose for representing assets, their initial values and the change in asset values over time: Represent assets (1) as variables and add an operator for assignment; (2) as data using extensions to STAIRS introduced by Runde et al. [19] or (3) as lifelines indicating the change in asset value through the reception of special messages. We have chosen the latter because it keeps our semantics simpler (we do not have to add new concepts) and provides the same extent of expressive power as the other alternatives. Formally an asset is a triple (a, c, V) of an asset lifeline a, a basic component lifeline c and an initial value V. In a trace we represent the reduction of asset value by a special kind of message called reduce, which takes as argument the amount by which the asset value should be reduced. The value of an asset at a given point in time is computed by looking at its initial value and all occurrences of reduce, with the asset as receiver, up to that point in the trace. Events on the lifeline of an asset can only be receptions of reduce messages. The value of an asset can not go below zero.

8.3 Representing Vulnerabilities

As pointed out by Verdon and McGraw [25] vulnerabilities can be divided into two basic categories: flaws, which are design level problems, and bugs, which are implementation level problems. When conducting security analysis during the early stages of system development, the vulnerabilities that can be detected are of the former type. I.e., a vulnerability is a weakness in the component specification, allowing interactions that can be exploited by threats to cause harm to assets.

8.4 Representing Incidents and Risks

As explained in Section 2 an information security incident is an unexpected or unwanted event that might compromise information security. In a component setting we can represent security incidents in the same manner as we represent normal behaviour; by sets of traces. A risk is measured in terms of the probability and consequence of an information security incident. Hence, in order to represent risks we need to be able to represent the probability of a set of traces constituting an information security incident and its consequence.

Inspired by Refsdal et al. [17] in our semantic model we represent this set of traces by a so called *risk obligation*. A risk obligation is a generalisation of an interaction obligation. Formally a risk obligation is a triple (o, Q, A) of an interaction obligation o, a set of probabilities Q and a set of assets A. The probability of the risk is an element of Q. We operate with a set of probabilities instead of a single probability to allow the probability to range freely within an interval.

Example 2. Figure 5 illustrates how we can specify a risk in accordance with the extensions to the semantic paradigm, described above. As most dynamic web applications, the login component pass data on to a subsystem. This may be an SQL data base or a component interacting with a database. If the system is not protected against SQL injection an attacker can modify or add queries that are sent to a database by crafting input to the web application. The attack example is from Sverre H. Huseby's [8] book on web-server security.

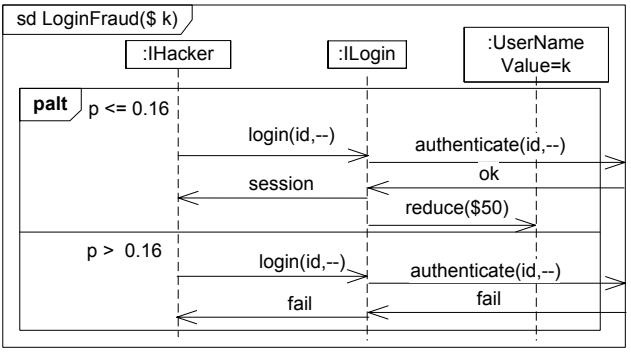

Fig. 5. Login without password using SQL injection

The sequence diagram in Figure 5 shows the interactions of a hacker using a modified query to attempt an SQL injection and the login lifeline receiving the query. Instead of a password the hacker writes a double hyphen (--). Unless the system is programmed to handle such metacharacters in a secure manner, this has the effect that the test for a matching password is inactivated allowing the hacker to login with only a user name. We have assigned the asset UserName to the basic component ILogin. As the sequence diagram illustrates an example run, we assume the initial asset value has been set elsewhere and parameterise the specification with the asset value k of type $. If the SQL attack is successful the asset value is reduced with $50.

We specify the risk as a probabilistic choice between the scenario where the attack is successful and the scenario where it fails. Probabilistic STAIRS [17] uses the palt construct to specify probabilistic alternatives, as illustrated in Figure 5.

In order to estimate the probability of a successful login using SQL injection, we must know both the probability of an attack (threat probability) and the

probability of the success of an attack (degree of vulnerability), given that an attack is attempted. We assume that an attack has been estimated to have a 0.2 probability. The probability that the attack will be successful is determined from looking at the system's existing vulnerabilities, such as lack of control mechanisms. In the example there is not specified any protection mechanisms against attempt at SQL injection. The probability of success given an attack is therefore estimated as high: 0.8. The alternative to the risk is that the modified query is rejected, and hence the asset value is not reduced. The consequence of the risk is the loss of $50 in asset value. We multiply the probability of an attack with the probability of its success to obtain the total probability of the risk. Hence, the probability of a successful false login is $0.2 * 0.8 = 0.16$. □

8.5 Generalising the Paradigm to Support Security Risk

Above we have outlined the relation between security risks as described in Figure 1 and our semantic paradigm. We now go on to adapt the semantic paradigm to capture this understanding formally.

In order to allow assignment of probabilities to trace sets, we represent basic components by sets of risk obligations instead of sets of interaction obligations. Moreover, contrary to earlier a basic component may now have more than one lifeline, namely the lifeline of the component itself and one additional lifeline for each of its assets. Hence, the denotation $[\![K]\!]$ of a basic component K is a set of risk obligations. Composition of components is defined point-wise as previously, i.e.:

$$[\![K_1 \otimes K_2]\!] \stackrel{\mathsf{def}}{=} \{r_1 \otimes r_2 \mid r_1 \in [\![K_1]\!] \wedge r_2 \in [\![K_2]\!]\}$$

Composition of risk obligations is defined as follows

$$(o_1, Q_1, A_1) \otimes (o_1, Q_1, A_2) \stackrel{\mathsf{def}}{=} (o_1 \otimes o_2, Q_1 * Q_2, A_1 \cup A_2)$$

where

$$Q_1 * Q_2 \stackrel{\mathsf{def}}{=} \{q_1 * q_2 \mid q_1 \in Q_1 \wedge q_2 \in Q_2\}$$

and $*$ is the multiplication operator. The use of the $\otimes$-operator requires that K_1 and K_2 are described independently as components. In STAIRS the $\otimes$ operator corresponds to parallel composition ($||$) (which is the same as $\gtrsim$ since K_1 and K_2 have disjoint lifelines). The scenario described in Figure 5 involves the palt construct, which imposes a global constraint on the interactions between the hacker and the login lifelines. Calculating the semantics of the overall scenario involves the use of several additional operators. See [6,5] for further details.

We also update the hiding operator δ to ensure that external assets are not hidden. An asset is external if it is associated with the interfaces of a basic component that has externally visible behaviour. We define the function A_{Ext} to yield the external assets with regard to a set of assets A, a set of basic component lifelines L and an interaction obligation o:

$$A_{\mathrm{Ext}}(A, L, o) \stackrel{\mathsf{def}}{=} \{a \in \mathcal{A} \mid \pi_2.a \in ll.\delta L : o\}$$

Given a component K and a set of basic component lifelines L, at the component level hiding is defined as the pointwise application of the hiding operator to each risk obligation:

$$[\![\ \delta L : K\ ]\!] \stackrel{\mathsf{def}}{=} \{\delta L : r \mid r \in [\![\ K\ ]\!]\}$$

where hiding at the level of risk obligation is defined as:

$$\delta L : (o, Q, A) \stackrel{\mathsf{def}}{=} (\delta(L \setminus A_{\mathrm{Ext}}(A, L, o)) : o, Q, A_{\mathrm{Ext}}(A, L, o))$$

Composition with hiding is defined as before.

9 Related Work

Fenton and Neil [4] addresses the problem of predicting risks related to introducing a new component into a system, by applying Bayesian networks to analyse failure probabilities of components. They combine quantitative and qualitative evidence concerning the reliability of a component and use Bayesian networks to calculate the overall failure probability. Although Fenton and Neil address the same problem as we do, the focus is different. At this point we do not concern ourselves with how the security analysis results are obtained. Rather than focusing on the process we look at how the results of security analysis can be represented at the component level to facilitate composition of security analysis results in a development process.

There are a number of proposals to integrate security requirements into the requirements specification, such as for example in SecureUML [16] and in UMLsec [13]. SecureUML is a method for modelling access control policies and their integration into model-driven software development. SecureUML is based on role-based access control and models security requirements for well-behaved applications in predictable environments. UMLsec is an extension to UML that enables the modelling of security-related features such as confidentiality and access control. These approaches have no particular focus on component-based specification. One approach that has a particular focus on component security is the security characterisation framework proposed by Khan and Han [14] to characterise and certify the security properties of components as a basis for deriving system-level risks during the deployment phase. These methods focus on specifying security properties of systems which is orthogonal to what we do. They include no notion of risk or probability. Rather than specifying security properties of systems, we focus on representing risks, i.e., we integrate the documentation of the probability that unwanted behaviour may occur into component specifications.

10 Conclusion

We have provided a semantic paradigm for component-based specifications explaining: basic components with provided and required interfaces; the composition of components into composite components and unpredictability which is

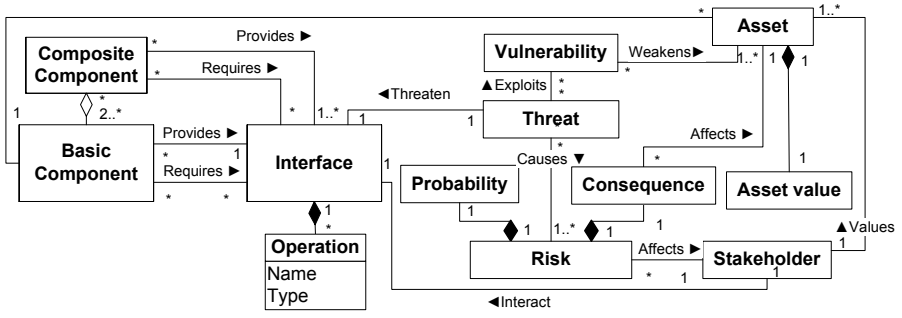

Fig. 6. Integrated conceptual model of a component risk specification

often required to characterise confidentiality properties. Furthermore we have extended the semantic paradigm with the notion of security risk as known from asset-oriented security analysis. Figure 6 summarises the relations between the conceptual component model and security assessment concepts: A component holds a set of assets that has value for its stakeholders. We limit the notion of a stakeholder to that of a component client or supplier interacting with it through its interfaces. We represent threats as lifelines that may interact with a component through its interface. There is a one-to-one association between an interface on the one hand and stakeholder and threat on the other, as a component interface can interact with one stakeholder or threat at a time. A vulnerability is represented implicitly as an interaction that may be exploited by a threat to cause harm to a components assets. Instead of representing the two concepts of information security incident and risk, we represent only the concept of a risk as a probabilistic interaction leading to the reduction of an asset value. In the extended component model we associate a threat directly with a risk, as someone or something that may initiate a risk.

The formal representation of security analysis results at the component-level allows us to specify security risks and document security analysis results of components. This is a step towards integration of security analysis into the system development process. Component-based security analysis can be conducted on the basis of requirement specification in parallel with conventional analysis. If new components are accompanied by security risk analysis, we do not need to carry out a security analysis from scratch each time a system is upgraded with new components, but can apply rules for composition to update the security risk analysis.

Acknowledgements

The research on which this paper reports has been funded by the Research Council of Norway via the two research projects COMA 160317 (Component-oriented model-based security analysis) and SECURIS (152839/220).

References

1. Brændeland, G., Stølen, K.: Using model-based security analysis in component-oriented system development. A case-based evaluation. In: Proceedings of the second Workshop on Quality of Protection (QoP'06) (to appear, 2006)
2. Cheesman, J., Daniels, J.: UML Components. A simple process for specifying component-based software. Component software series. Addison-Wesley, Reading (2001)
3. den Braber, F., Dimitrakos, T., Gran, B.A., Lund, M.S., Stølen, K., Aagedal, J.Ø.: UML and the Unified Process, chapter The CORAS methodology: model-based risk management using UML and UP, pp. 332–357. IRM Press (2003)
4. Fenton, N., Neil, M.: Combining evidence in risk analysis using bayesian networks. Agena White Paper W0704/01 (2004)
5. Haugen, Ø., Husa, K.E., Runde, R.K., Stølen, K.: Why timed sequence diagrams require three-event semantics. Technical Report 309, University of Oslo, Department of Informatics (2004)
6. Haugen, Ø., Stølen, K.: STAIRS – steps to analyze interactions with refinement semantics. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML 2003 LNCS, vol. 2863, pp. 388–402. Springer, Heidelberg (2003)
7. Hogganvik, I., Stølen, K.: On the comprehension of security risk scenarios. In: 13th International Workshop on Program Comprehension (IWPC 2005), pp. 115–124. IEEE Computer Society, Los Alamitos (2005)
8. Huseby, S.H.: Innocent code. A security wake-up call for web programmers. Wiley, Chichester (2004)
9. ISO/IEC.: Information technology – Code of practice for information security management. ISO/IEC 17799:2000
10. ISO/IEC.: Risk management – Vocabulary – Guidelines for use in standards, ISO/IEC Guide 73:2002 (2002)
11. ISO/IEC.: Information Technology – Security techniques – Management of information and communications technology security – Part 1: Concepts and models for information and communications technology security management, ISO/IEC 13335-1:2004 (2004)
12. Jøsang, A., Presti, S.L.: Analysing the relationship between risk and trust. In: Jensen, C., Poslad, S., Dimitrakos, T. (eds.) iTrust 2004. LNCS, vol. 2995, pp. 135–145. Springer, Heidelberg (2004)
13. Jürjens, J. (ed.): Secure systems develoment with UML. Springer, Heidelberg (2005)
14. Khan, K.M., Han, J.: A process framework for characterising security properties of component-based software systems. In: Australian Software Engineering Conference, pp. 358–367. IEEE Computer Society, Los Alamitos (2004)
15. Lau, K.-K., Wang, Z.: A taxonomy of software component models. In: Proc. 31st Euromicro Conference, pp. 88–95. IEEE Computer Society Press, Los Alamitos (2005)
16. Lodderstedt, T., Basin, D.A., Doser, J.: SecureUML: A UML-based modeling language for model-driven security. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002 - The Unified Modeling Language. Model Engineering, Concepts, and Tools. LNCS, vol. 2460, pp. 426–441. Springer, Heidelberg (2002)
17. Refsdal, A., Runde, R.K., Stølen, K.: Underspecification, inherent nondeterminism and probability in sequence diagrams. In: Gorrieri, R., Wehrheim, H. (eds.) FMOODS 2006. LNCS, vol. 4037, pp. 138–155. Springer, Heidelberg (2006)

18. Rumbaugh, J., Jacobsen, I., Booch, G.: The unified modeling language reference manual. Addison-Wesley, Reading (2005)
19. Runde, R.K., Haugen, Ø., Stølen, K.: Refining UML interactions with underspecification and nondeterminism. Nordic Journal of Computing (2005)
20. Winamp skin file arbitrary code execution vulnerability. Secunia Advisory: SA12381. Secunia (2006)
21. Seehusen, F., Stølen, K.: Information flow property preserving transformation of uml interaction diagrams. In: 11th ACM Symposium on Access Control Models and Technologies (SACMAT 2006), pp. 150–159. ACM, New York (2006)
22. Standards Australia: Standards New Zealand. Australian/New Zealand Standard. Risk Management, AS/NZS 4360:2004 (2004)
23. Standards Australia: Standards New Zealand. Information security risk management guidelines, HB 231:2004 (2004)
24. Szyperski, C., Pfister, C.: Workshop on component-oriented programming. In: Mülhauser, M. (ed.) Special Issues in Object-Oriented Programming – ECOOP'96 Workshop Reader, dpunkt Verlag, pp. 127–130 (1997)
25. Verdon, D., McGraw, G.: Risk analysis in software design. IEEE Security & Privacy 2(4), 79–84 (2004)
26. Zakinthinos, A., Lee, E.S.: A general theory of security properties. In: IEEE Symposium on Security and Privacy, pp. 94–102. IEEE Computer Society, Los Alamitos (1997)