Assessing Security to Compare Architecture Alternatives of Component-Based Systems

Axel Busch*, Misha Strittmatter*, Anne Koziolek*
*Karlsruhe Institute of Technology, Germany
{busch, strittmatter, koziolek}@kit.edu

***Abstract*—Modern software development is typically performed by composing a software system from building blocks. The component-based paradigm has many advantages. However, security quality attributes of the overall architecture often remain unspecified and therefore, these cannot be considered when comparing several architecture alternatives. In this paper, we propose an approach for assessing security of component-based software architectures. Our hierarchical model uses stochastic modeling techniques and includes several security related factors, such as attackers, his goals, the security attributes of a component, and the mutual security interferences between them. Applied on a component-based architecture, our approach yields its mean time to security failure, which assesses its degree of security. We extended the Palladio Component Model (PCM) by the necessary information to be able to use it as input for the security assessment. We use the PCM representation to show the applicability of our approach on an industry related example.**

Keywords*—*Security, Model, Assessment, Component, PCM

I. Introduction

The acceptance of a software service does not solely depend on its functionality, but quality considerations such as security become an increasingly important aspect and therefore are an enabler for many applications. To achieve high quality in software systems depends to a great extend on the software architecture. The component-based paradigm has been shown as a suitable approach to improve software quality [3]. Its capabilities for structuring and separation of concerns reduce the complexity of the overall system. This allows reasoning about the quality of the software system based on the attributes of its components. As a consequence, the estimation of quality aspects is focused on the building blocks of a software and therefore does not require to estimate quality attributes of the system as a whole.

The connectivity of software systems significantly increased in the last years. Since a growing amount of services are highly connective and often are data centric, an attack becomes more probable and also desirable for attackers and raises their expectable profit. In the period of 2010 - 2013, the number of incidents causing theft of personal information increased by more than 40 % [11]. Therefore, the design of secure computer systems is urgently required. But to be truthful, no computer system is totally secure. No matter how many time developers spend on securing their software, there will probably remain errors in the implementation or vulnerabilities that allow misusing the system or get the system under control. Nevertheless, it is easy to see that some software systems are more secure than others. Because of all that, we need a methodology to allow a systematic assessment of security attributes of software systems considering the component-based paradigm. A systematic comparison of several software architectures allows architectural trade-off decisions considering security, and finally helps to design and select high quality software.

To assess the security strength of a software system, we consider a set of security influencing factors. These factors interact and influence the process of possible system attacks. A typical attack involves a certain skilled attacker, possible starting and aiming points of an attack, the component design and the effort spent on security considerations, as well as a favorable deployment of these components on hardware, and finally the mutual security interference of these components.

In this paper, we propose an approach for the assessment of security attributes of component-based software systems with a main focus on distributed business information systems. Such a methodology allows a comparison of several software architectures regarding their security quality attributes. Comparing quality attributes of a software architecture enables automated analysis of multiple design alternatives. This is especially useful, when trade-off decisions over multiple quality attributes have to be made.

The deployment as well as the selection of the involved components significantly influence the overall system's security. For our security assessment approach, we define a hierarchical model that combines the aforementioned security influencing factors and considers the involved components as well as the deployment configuration.

In [13] the authors show a security model derived from the field of reliability, while in [9], [8] they regard a system as a whole for their security quantification approach or do not focus on component-based architectures [1]. Another approach [12] shows how to quantify security regarding costs. In [7] the authors show a model that considers modeling a certain attacker. Finally, in [15] the authors show the possibility of determining security properties of composed systems. We base our model on the following related approaches that already provide several aspects of our considerations: From [9], [8], we use the concept of modeling the security assessment approach using a Semi-Markov Process (SMP) and the considered metric. From [7], we use the concept of modeling a set of attackers.

However, all the aforementioned papers either do not address component-based architectures or they do not consider mutual security interferences or only focus on the attacker itself and do not consider the software architecture. None of these papers attempt to combine the aspects of component security, the attacker's skill, his goal, and the interference between components. Our approach combines parts from the aforementioned papers and provides an analytic model that results in a metric to compare different design alternatives in a component-based design process including all these aspects.

We apply our security assessment model to the Palladio Component Model (PCM). For that reason, we extend its metamodel for security annotations of components. Further, we include the attacker, his skill and goals and finally the security interference between components. To demonstrate our approach, we show an example which uses a component-based software architecture of an industry system [5].

The paper is organized as follows: Section II presents an overview of our security assessment approach and its submodels. In Section III, we provide a detailed high-level description of the submodels. Section IV shows the mathematical representation of the high-level description, combines the proposed submodels and shows three example analysis based on an industry system. Section V shows our PCM metamodel extensions. Finally, we discuss related work in Section VI and conclude in Section VII.

II. Security Assessment Approach

An overview of our security assessment approach is shown in Figure 1. For our model, we consider four influencing factors: the attacker, the attacker scenario, the security attributes of the system components, and the mutual security interference of the components. We represent these factors in an SMP model. The SMP provides suitable mechanisms, as states, transitions, and sojourn times, allowing us to model the different phases an attacker would have to pass, his success probabilities, as well as time effort. Using an SMP model, we calculate the *Mean Time To Security Failure* (MTTSF) to assess the system security. The MTTSF metric was proposed by Madan et al. in [9], [8], [6]. In our approach, we respect a certain set of system components, their attributes, and deployment configurations.

The *story* (Fig. 1: (1)) of a system attack comprises a set of entities. The *attacker*, who is motivated to *attack* a system, and the *system under attack*.

These entities are influenced by several *influencing factors* (Fig. 1: (2)). The *attacker's skill* reflects the attacker's general knowledge on system attacks. The attribute *attacker scenario* represents the actual purpose of the attack (e.g., achieve data access). *Component security* reflects the probability of a hidden vulnerability of a certain component, and the time of discovering one of these vulnerabilities. The *mutual security interference* considers a possible security interference between components that is potentially exploitable by an attacker. Finally, we use an *SMP model* for our mathematical representation of the aforementioned factors (Fig. 1: (3)).

The mathematical SMP model representation comprises four parts reflecting the four influencing factors mentioned above: The first part represents the attacker's skill. It is modeled by an exponential density (as suggested by [7]), and a *Mean Time of Attack* (MToA). The second part defines the starting or also called entry states, and the absorbing states of the Markov process. The third part defines the transition probabilities of the certain states and the effort spent on security for a certain component, which translates into the state sojourn times. The fourth part fuses states according to component security interferences. Finally, the model output is the *MTTSF* that represents the degree of security when comparing architecture alternatives on a certain hardware deployment configuration (Fig. 1: (4)) on an ordinal scale level.

III. Security Assessment Model

As mentioned in Section II, our security assessment model considers four influencing factors that affect an attack on the system and the system's overall security hence. These factors are encapsulated by four submodels that will be explained in the following from a high-level view. To introduce the model in a convenient manner, we will first introduce a running example for a better understandability of our model and its submodels.

A. Application Example

For the introduction and application of our model, we use a simplified version of the architecture of an industry system [5] as a running example. Figure 2 shows the architecture model in a mixed architectural and deployment view [1].

Our example realizes a software system that expects a certain input via the `Access` component. This input is computed by the `Parser` and forwarded by the `DA` component to `DB`. The system provides a service interface, the `Web` component, to get access to the `DB` component. Two of the components are accessible from outside namely the `Access` and `Web` components.

System topology.
Our example comprises three hardware containers. The first container runs a demilitarized zone (DMZ) server. Here, external data enters the internal network via the `Access` component. The second hardware part is the application server. On that, we deployed the `Web` component, the `Parser`, and the `DA` component. The third hardware component hosts a database server that runs the `DB` component. The DMZ server is physically connected to the application server, while the application server is connected to the database server. The DMZ and the database server are not physically connected. Different hardware contexts are typically connected by hardware links. We assume that connecting hardware containers is secure and the technology of connecting them has no additional effect on the security of a system. Additionally, we do not consider server replication for reliability purposes or performance improvements in the overall system.

Attacker.
The possible attackers are represented by a set of attacker models, one for each type of attacker. A system is typically attacked by less or more skilled attackers depending on the expectable profit of attacking a system and the value of its data.

Scenario.
We focus on the external provided interfaces. Thus, the example contains two possible entry points, the `Access` and the `Web` component. Both of them can be used by a user that interacts with the system in a designated manner, but also by an attacker when accessing the system in a harmful way. We define the goal of an attacker as getting access to one certain component or hardware container of the system. In our scenario we set the goal of the attack to be the `DB` component for data theft.

[1]Note that the architecture model was originally created for performance assessment and does not necessarily reflect the actual security attributes of this system. In fact, most security-related design decisions of this system are unknown to us. We only use the information published in [5] here and add our own assumptions where needed.

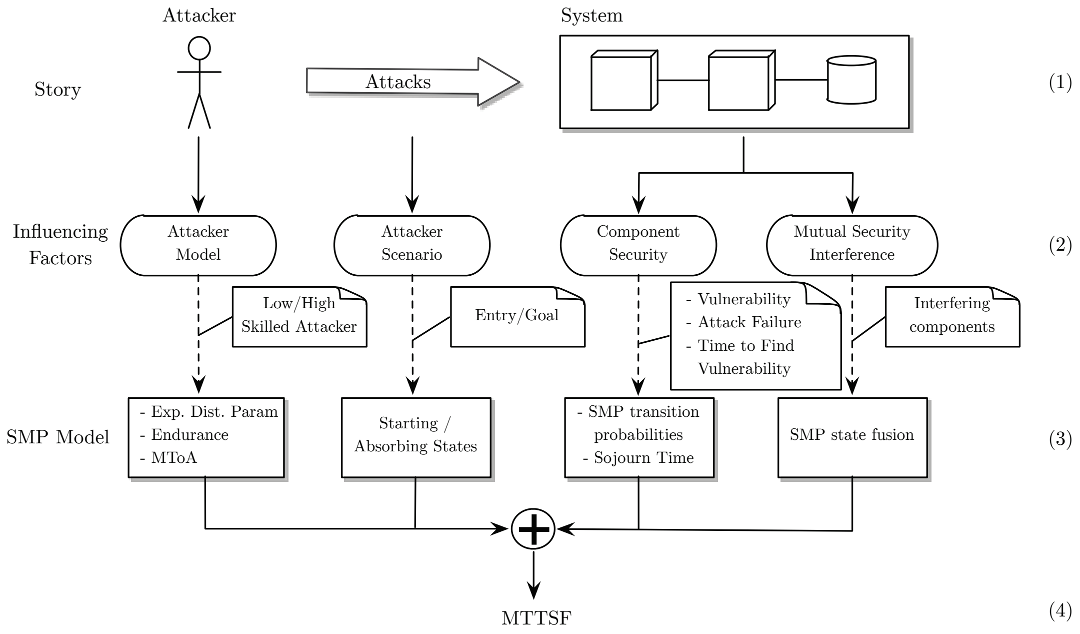

Fig. 1: Security Assessment Approach

Control flow.

The control flow depends on the attack scenario. Since we define the goal of our scenario to get access to the `DB` component, we can derive two different possible control flows:

1) The first control flow goes over the `Access` component on the DMZ server. The provided interface of the `Access` component is connected to the `Parser` on the application server, while the `Parser` itself is connected to the `DA` component. Finally, the `DA` component is connected to the `DB` component on the database server.
2) The second control flow starts with the `Web` component on the application server. The `Web` component accesses via the `DA` component to the `DB` component on the database server.

B. Attacker Model

Rationale.
The attacker model represents the behavior of a certain attacker. The Verizon data breach investigation report, cf. [14] reports that security breaches are mainly perpetrated by outside parties. In 2013, about 92 % of all security breaches were committed by this group. Therefore, we focus on external attackers in our attacker model.

An attacker has a certain success probability that depends on his skill. While attacking a system he passes through three phases. In the beginning he learns about the system, then he attacks the system using his standard attacking repertory. If this fails, he has to invent in new methods to attack a system successfully. Additionally, he has a certain endurance that represents the mean time he spends on attacking a system. The result of the model is the mean probability of a certain attacker successfully discovering a potential exploitable system vulnerability within a certain time.

Model.
Our attacker model comprises two parts: The phases of processing an attack and the attacker's skill.

The procedure of an attack is typically comprised of three phases: The learning phase, the standard attack phase and the innovative attack phase. Several of the following terms are taken from [7].

- *Learning phase:* The attacker learns about how to attack a system and the system itself to be able to perform an attack. In this stage, the number of successful attacks is comparatively low, due to missing experience with system attacks. A low skilled attacker would spend more time to educate himself compared to a higher skilled attacker.
- *Standard attack phase:* In the standard attack phase the attacker uses his knowledge and his repertory of attacking techniques to attack the system. In this phase major successful attacks are expected.
- *Innovative attack phase:* In the innovative attack phase the attacker's repertory ran out and he has to invent new methods to attack the system. Executing a successful breach, here, typically takes longer compared to the standard attack phase.
- *Skill:* The skill determines the probability of discovering a potential vulnerability in the system in a certain time.
- *Endurance:* The endurance is the mean time an attacker actually spends on searching for a system vulnerability.

Referring to [7], we use a modification of the cumulative distribution function of the exponential distribution that models the phases of an attack, the skill of a certain attacker, as well as his endurance:

$$\phi_{\lambda,\Delta}(x) = \begin{cases} 1 - exp(-\lambda \cdot (x - \Delta)), & 0 \leq x - \Delta \\ 0 & x - \Delta < 0, \end{cases} \quad (1)$$

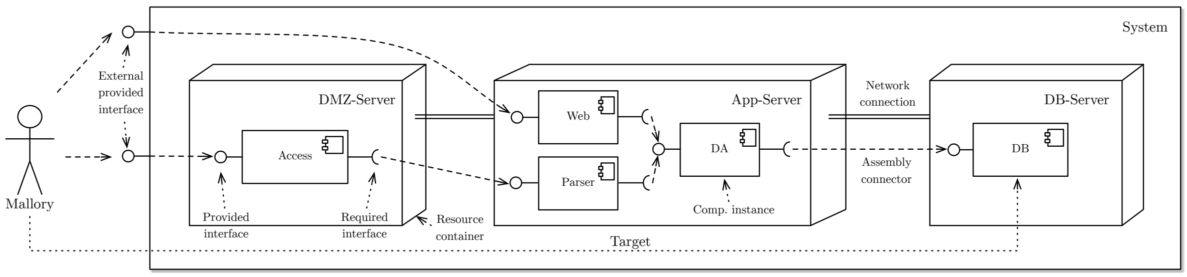

Fig. 2: Architecture overview of the running example (from [5])

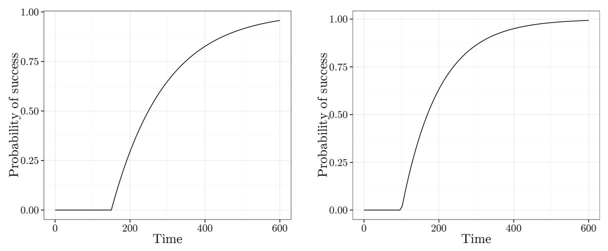

Fig. 3: Lower skilled (left) and higher skilled (right) attacker model parameterization

where λ is the rate parameter that determines the skill of the attacker in the standard and innovative attack phase and Δ is the parameter that represents the duration of the learning phase. When executing the attacker model, the *Mean Time of Attack* is applied as input x for the attacker model. This describes the mean time a certain skilled attacker spends on attacking a system.

Accordingly, our attacker model is parameterized by a pair of elements (λ, Δ).

For our attacker model, the skill of a certain attacker group is a random variable with known mean values. For all parameters of the models, we assume the values could be estimated by the system designer. Estimating the model parameters is not focus of this paper. We focus on introducing our model for security assessment.

Example.
Figure 3 shows two examples of parameterized attacker models. For our example we parameterize the attacker model in that way to represent a lower- and a higher skilled attacker. For the lower skilled attacker, we use $(\lambda = 0.007, \Delta = 150)$, while the higher skilled attacker uses $(\lambda = 0.01, \Delta = 100)$. Both parameters are examples, but could be extracted from log files. However, this step is not adressed by this paper.

C. Attacker Scenarios

Rationale.
An attacker typically attacks a system for a specific goal. The system comprises an entry point and the goal itself: the exit or goal point. The actions undertaken by the attacker depend on his goal. In our approach, we focus on component access to copy or modify data or to achieve higher system privileges. Another scenario could be to make the system unavailable. We do not consider the unavailability scenario, but it could be potentially covered by our model.

Model.
For our attacker scenario model, we use a Semi-Markov process. The attacker scenario entities are shown in Figure 4. It comprises the attacker's entry and goal points:

- *Entry Points:* The entry points, i.e. an interface, is the first entity under attack. Here, the attacker gets access to the system. Entry points are, for instance, open ports or a certain web interface that is accessible from outside. In the model, we consider a set of different entry points and thus, different ways an attacker could use to navigate (see (a) and (b) of Figure 4) through the system. The model allows to define a set of entry points.
- *Exit/Goal Point:* The goal point represents the state an attacker tries to reach and the component at which the attack is aimed. We focus on exactly one exit or goal point.

In our SMP model, we map the entry and exit/goal points on starting and absorbing states. Causing a *data breach* (cf. [9]) on the goal component means to reach the absorbing state in the SMP model.

Our approach focusses on attacks on the architectures external provided interfaces. Thus, the way through the system is given by the provided external interfaces and the goal point of the attacker. Attacking a specific component would force the attacker to move by a specific way through the system starting with one of the external interfaces. The specific way an attacker has to follow is dependent on the control flow of the architecture. An attacker cannot use any shortcuts to skip certain parts and can not change the control flow. Therefore, our model is designed such that an attacker has to access the system via defined interfaces. More precisely, he cannot access arbitrary components, but has to start his attack at a defined set of accessible interfaces.

Although we defined the goal to cause data breaches, the SMP model's flexibility allows to be adjusted to other goals. Since the core of our approach is a Semi-Markov process, the attacker scenario is modeled with the start and end condition of the Markov-chain. This could be tailored to arbitrary scenarios.

Example.
Figure 4 shows the control flow of our component-based architecture example. Since our example architecture allows access through two components, we define the entry states as the `Access` and `Web` components. Therefore, we use two states *Acc* and *Web* for their representation. We set our goal scenario to cause a data theft. An attacker would have to intrude the `DB` component to achieve access to the database. Therefore, we set the `DB` component to the goal point and introduce the *DB* state. `DB` is only accessible, if either control flow *Access-Parser-DA-DB* or *Web-DA-DB* is taken. We assume no specific knowledge of the attacker about the system architecture. Therefore, we assume equiprobability for both of the possible ways.

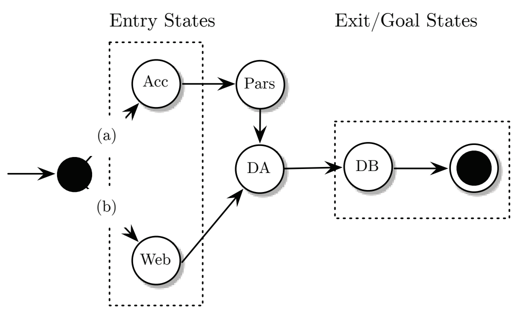

Fig. 4: Attacker Scenario: Two entry states and one goal state.

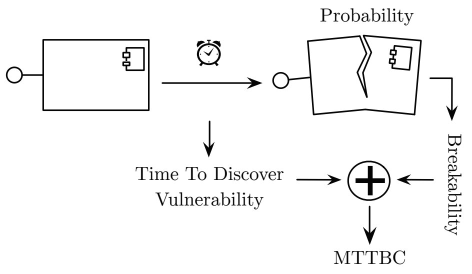

Fig. 5: Component Security Model

D. Component Security

Rationale.

This part describes the security of a certain software component. For our model, we use concepts from [9]. Depending on the effort a developer spent in securing a component, the probability of a hidden vulnerability in a component differs. Thus, a probability of hidden vulnerabilities models this effort spent by the component developers. Second, we introduce a time aspect: discovering a certain vulnerability takes time regardless of the probability of its presence. We decided to divide the probability and the time to discover a vulnerability, due to the following rationale: A development group spent time in testing and fixing a component. This influences the number of hidden vulnerabilities in the component in a positive manner. Nevertheless, if it would be straight forward to find one of the remaining vulnerabilities, the time to discover one vulnerability is comparatively short. Not including the time aspect would discriminate another component whose number of hidden vulnerabilities is higher, but the remaining are harder to discover.

Model.

Our component security model is shown in Figure 5. It is represented by a pair:

$$CompSec(c) = (TTDV, PoCoB), \tag{2}$$

while TTDV is the *Time To Discover Vulnerability* and PoCoB is the *Probability of Component Breakability*:

- *TTDV:* The *Time To Discover Vulnerability* reflects the mean time to be spend on discovering a potential issue of a certain component. This does not mean the issue is possible to be exploited in a convenient way. The metrics unit is *time units*.
- *PoCoB:* The *Probability of Component Breakability* describes the probability of the potential exploitable issue in the component is actually harmful. The metric represents the probability of the attacker can use this issue to break the component in a convenient manner to increase his permissions on the system.

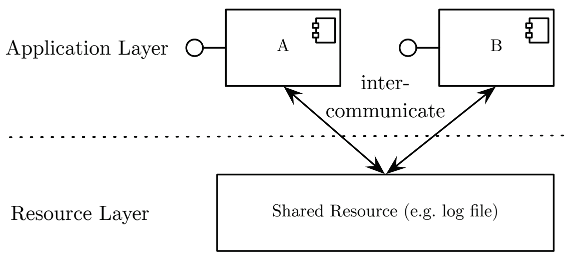

Fig. 6: Mutual Security Interference Illustration

The model's outcome is the *Mean Time To Break Component* (MTTBC) of a particular component c. It reflects the mean time to be spend on successfully breaking a component. The MTTBC is calculated as follows:

$$MTTBC_c = \frac{TTDV_c}{PoCoB_c} \tag{3}$$

An attacker would need a mean time, denoted by *TTDV* to uncover a vulnerability in the component. The probability of this vulnerability to be beneficial for breaking the system is denoted by *PoCoB*. Therefore, the mean time an attacker needs to uncover a vulnerability being harmful for the system is lengthened according to PoCoB.

Example.

Let us assume the `Access` component comprises the following attributes:

$$TTDV_{access} = 20\ time\ units \tag{4}$$

$$PoCoB_{access} = 0.25 \tag{5}$$

In other words an ideal attacker would need 20 time units to find an issue in the component that is potentially harmful. The probability of the issue discovered by the attacker is actually harmful is 25 %. According to Formula 3 the component's $MTTBC_{access} = 80$ time units. The result means that an ideal attacker would have to spend 80 time units on average to successfully break the `Access` component.

The input parameters could be derived from the developer's experience, development process, as well as platform and used technology, size of a component and its maturity level. However, determining concrete values is out of scope of this paper.

E. Mutual Security Interference

Rationale.

Mutual security interference describes the security impact of several components influencing each other. A scenario is shown in Figure 6. A mutual influence happens if, for instance, resources or permissions are shared across several components. A shared resource is, for instance, a common data access (e.g. read on and write from a common used file) or sharing information via memory sharing. Another reason for the mentioned interference is sharing permissions across several components. Two components running with shared user permissions would cause this interference. It could potentially lead to a compromisation of all components sharing the same permission if one component would be compromised.

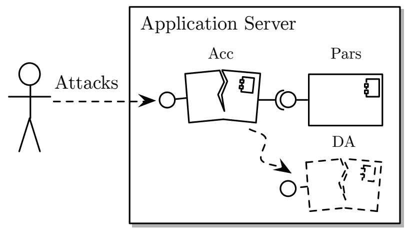

Fig. 7: Component `Access` is attacked. Due to the mutual security interference of `Access` and `DA`, `DA` is implicitly compromised, if the `Access` component is compromised successfully.

Model.
Mutual security interference creates additional paths the attacker can move through the system, in addition to the normal control flow. Figure 6 shows an example, in which component A intercommunicates with component B, while both use a shared resource, such as a log file. Thus, after compromising component A, attackers might be able to use the shared resource to compromise component B.

To include the mutual security interference in the model, we modify the concatenation and the transition probabilities of the SMP. This is done by the transformation from PCM to the SMP model.

Example.
We show an example in Figure 7. Let us assume that the `Access` component and the `DA` component run in the same memory space. Thus, if `Access` is compromised, it can as well be used to modify the data processed by `DA` and thus compromises `DA`. Here, all conditions for a mutual security interference between `Access` and `DA` are fulfilled.

IV. Security Modeling using SMP

We use a Semi-Markov process to represent our high-level hierarchical model mathematically. We have chosen the SMP representation according to [9]. The SMP process is highly suitable to represent our submodels with its model elements. The requirements to the mathematical representation is sketched in the following:

- An attacker is to be represented separately from the other model elements. Component security should not be mixed up with the attacker's properties. The mathematical model needs elements to represent his skill.
- An attacker is forced to access a system via defined interfaces and a system attack pursues a certain goal.
- In a component-based software architecture several software components are connected and interact together. They differ from each other in their security attributes.
- A certain service is realized by a particular combination of software components and corresponds to a particular control flow.
- Software components potentially influence each other in terms of security.

An SMP provides elements to represent a component-based software architecture and the aforementioned requirements. The states and transitions of the SMP's embedded discrete-time-markov chain (DTMC) allow to model an attacker and the

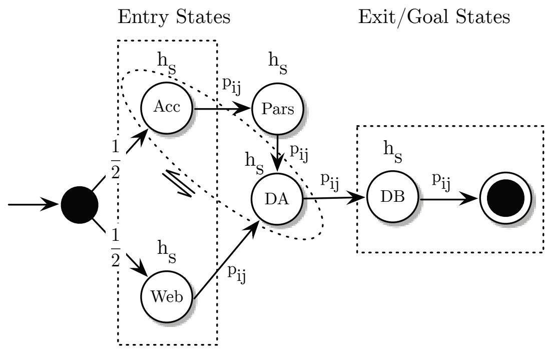

Fig. 8: SMP model representation

control flow of an application and thus, the attacker's movement in the system (see Sections III-B, III-C). The SMP's sojourn times represent the time aspect of the component security in each state (see Section III-D). An overview of the model elements is shown in Figure 8.

Base Model.
Let the underlying stochastic process be defined as $\{X(t) : t \geq 0, t \in \mathbb{N}_0^+\}$. Let us consider a system comprising k software components. When each component is represented by an individual state, then $S_k, k \in \mathbb{N}_0^+$ is defined as the set of all k states that represent the presence of an attacker in a certain component in the considered system plus the state for success Ω. The discrete state space of the stochastic model is represented by S. The transition probabilities p_{ij} are determined by the control flow of the system, as well as the attacker's scenario. The sojourn time of the DTMC is defined as h_k.

Component Security.
Our component security model *CompSec* of Section III-D is designed to be easily mapped to an SMP representation. We define

$$p_{ij} = PoCoB_i \tag{6}$$
$$h_i = TTDV_i, \tag{7}$$

while $i, j \in X_t$.

Composing component and attacker model.
Composing the component and the attacker model requires several additions to the previous introduced mapping of the component security model and the SMP. The attacker model influences two parts of the SMP: First, it influences the probability of the state transition probability p_{ij} of the embedded DTMC. In other words, it influences the probability of a successful attack (PoCoB) of a certain component according to the attacker's individual knowledge.

Second, it includes the mean time an attacker spends on attacking a certain component. We have

$$PoCoB_i^{\phi} = \phi_{\lambda,\Delta}(x) \cdot PoCoB_i \tag{8}$$
$$MTTBC_i^{\phi} = \frac{TTDV_i}{PoCoB_i^{\phi}}, \tag{9}$$

while $i \in I := \{0, \ldots, k-1\}$. $PoCoB^{\phi}$ is the adjusted $PoCoB$ that includes the attacker's skill and $MTTBC^{\phi}$ is the resulting $MTTBC$ containing the attacker model.

Attacker Scenario.
The attacker scenario defines the entry and exit/goal points of the attack and therefore the entry and the absorbing states of the embedded DTMC.

Let $S_e \subseteq S$ be the set of entry states, and $S_a \subseteq S$ be the absorbing state of the attacker scenario model.

In addition to the entry and exit states the attacker scenario defines the control flow of the software architecture. Let Θ be an $S \times S$ array.

$$\Theta := \begin{pmatrix} \theta_{00} & \theta_{01} & \dots & \theta_{0k} \\ \theta_{10} & \theta_{11} & \dots & \theta_{1k} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{k0} & \theta_{k1} & \dots & \theta_{kk} \end{pmatrix} = (\theta_{ij}), \tag{10}$$

while $(\theta_{ij}) \in \mathbb{N}_0^+$ depending on the transition possibilities from state i to state j (while $i, j \in X_t$) and number of possible visits depending of the control flow of the architecture. If a transition is possible it is $\theta_{ij} > 0$ and if not we have $\theta_{ij} = 0$. Using Θ, we calculate the number of relevant required services of a component:

$$\Xi = (\xi_i)_{i \in I} = \sum_{l \in \{j \in J | \theta_{ij} > 0\}} 1, \tag{11}$$

while $j \in J := \{0, \dots, k\}$.

Mean time to security failure.
Assembling our submodels to one overall model allows us to assess the overall security of the system. Including the attacker model representing the attacker's skill is optional. Thus, we start with combining the component security model with the control flow of the attacker scenario model:

$$MTTSF = \sum_{i-1} \frac{1}{\xi_i} \sum_j \frac{TTDV_i}{PoCoB_i} \cdot \theta_{ij} \tag{12}$$

$$= \sum_{i-1} \frac{1}{\xi_i} \sum_j MTTBC_i \cdot \theta_{ij} \tag{13}$$

while $i, j \in X_t$. Analogously, we include the attacker skill model to obtain our fully integrated formula to calculate the MTTSF metric:

$$MTTSF^{\phi} = \sum_{i-1} \frac{1}{\xi_i} \sum_j \frac{TTDV_i}{PoCoB_i \cdot \phi_{\lambda,\Delta}(x)} \cdot \theta_{ij} \tag{14}$$

$$= \sum_{i-1} \frac{1}{\xi_i} \sum_j MTTBC_i^{\phi} \cdot \theta_{ij} \tag{15}$$

The $MTTSF^{\phi}$ value now represents our model's outcome. The value can be used to compare two architecture alternatives. As the accurate estimation of the model's parameters is not yet well understood, we advice not to interpret this metric as absolute values. Instead, we use its outcome for comparing architectures.

A. Assembly Example

For our assembly example we use the configuration of Section III-A as a basis. We show three different configurations to apply our security assessment model. First, we show the example of our baseline system. Second, we show how the model behaves when a component degree of freedom is applied. Third, we show how to change the deployment configuration, i.e. removing components (such as the access component), changing the deployment configuration (such as removing hardware), and introducing mutual security interferences between components.

Reference scenario.
As a baseline we define our reference example that is related to the running example of Section III-A. Our example system contains 5 software components. Out of this, we define the system topology:

$$SWComp = (Access, Web, Pars, DA, DB)$$

Upon the system topology, we define security information to the topology elements. For example, assume that most components are well tested and have several security mechanisms in place. Thus, the developers set the PoCoB for an idealized attacker to be 20%. Further, the developers could decide to set the TTDV to 200, due to the architectural structure of the component. Two components, namely `Web` and `Pars`, are considered to be less mature and secure yet, so their PoCoB is set to higher values.

$$CompSec(SWComp) = ((200, 0.2), (250, 0.3), (125, 0.4), (150, 0.2), (300, 0.2))$$

We define the states for the SMP model:

$$S = \{Access, Web, Pars, DA, DB, \Omega\}$$

The system contains two entry interfaces and the set of exit/goal states:

$$S_e = \{Access, Web\}$$
$$S_a = \Omega$$

The control flow of the system is defined by an $S \times S$ array:

$$\Theta = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We use Formula 11 to calculate Ξ:

$$\Xi = \{1, 1, 1, 1, 1, 0\}$$

Finally, we need to define an attacker for our example. For this example, we use the previous defined higher skilled attacker from Section III-B:

$$att_{high} = (\lambda = 0.01, \Delta = 100) \Rightarrow \phi_{\lambda=0.01,\Delta=100}(200) \approx 0.632$$

Upon the previous defined submodels, we assemble them to achieve the architecture's mean time to security failure with the attacker model ($MTTSF_{\phi}$):

$$MTTSF_{\phi} \approx 5256.78 \ [time\ units]$$

Component variation scenario.
Our first scenario considers on replacing one or more components from the reference architecture with components from

the repository. The replaced components provide the same functionalities as the reference components, offer and provide the same interfaces, but their implementation differ and thus, their security attributes. We replace the *Access* component of the reference component with a better implementation and the *DA* component with a weaker implementation (e.g., due to performance aspects):

$$\begin{aligned} CompSec(Access') &= (250, 0.15) \\ CompSec(DA') &= (50, 0.4) \\ S' &= \{Access', Web, Pars, DA', DB, \Omega\} \end{aligned}$$

The $MTTSF'_\phi$ metric results as follows:

$$MTTSF'_\phi \approx 4795.37 \; [time \; units]$$

In this scenario, we replace the `Access` component, which is comparatively secure, with a slightly higher secure implementation of the component. At the same time, we exchange the comparatively secure `DA` component with an, in terms of security, low quality component. Intuitively, the overall security quality of the modified architecture should decrease. When modeling the reference architecture and the modified architecture with our security assessment approach, we observe a result in favor of the reference architecture, as expected.

Deployment variation scenario.
In our second scenario, we change the deployment of the reference architecture and introduce a mutual security interference between components. First, we remove the DMZ server and deploy the `Access` component on the application server. Second, we introduce a mutual security interference between the `Access` and the `Parser` component: As described in Section III-E, the mutual security interference influences the security attributes of the overall system, if the interfering components are deployed on the same hardware container. Thus, due to the mutual security interference of the `Access` and the `Parser` component, the `Parser` component becomes obsolete and we refine the state space S and the control flow Θ of our model:

$$S'' = \{Access, Web, DA, DB, \Omega\}$$

$$\Theta'' = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We use Θ'' to calculate Ξ'':

$$\Xi'' = \{1, 1, 1, 1, 0\}$$

Out of the redefined submodels, we calculate the outcome of our metric in this scenario:

$$MTTSF''_\phi \approx 4416.35 \; [time \; units]$$

Intuitively, a mutual security interference of components highly affects the overall security of the system. The result of this example confirms this suspicion. Comparing the result of the reference and the modified architecture observes a high impact on the system security when components interfere each other.

V. Component-based Security Modeling

Based on the security attributes of its constituent parts, our approach aims to determine an indicator to compare the assessed security of component-based architectures. We extend the PCM [2], a metamodel for component-based architectures, to allow an annotation of the security assessment model's information. Annotated PCM model instances can then be transformed into SMPs for the security analysis. We chose to extend the PCM, as it contains the necessary structural elements. Further, it allows us to integrate the assessment of security into the Palladio software engineering process. This also enables trade-off decisions between security and other properties, like performance, cost and reliability. However, our approach could be applied to related component-based architecture models.

A. Palladio Component Model

A PCM model comprises five submodels: repository, system assembly, resource environment, deployment and usage. Figure 2 shows the architecture of the example system. For illustration purposes, the figure shows a combination of several submodels and omits ones which are less important to our approach.

The *component repository submodel* contains interfaces and components. A component may require and provide several interfaces. Depending on the quality dimension which is to be modeled, there will be more information deposited within these components (e.g., a behavior abstraction in the case of performance and reliability). The repository itself is not shown in the figure.

The components from the repository are instantiated and connected within the *system assembly submodel*, which describes the component architecture of a system. In the figure, only component instances are shown. Their interfaces are represented by the ball-and-socket notation. Required interfaces can be connected to provided interfaces, by assembly connectors, which define the data and control flow. Additionally, in the system assembly submodel, the system interfaces are defined and connected to the associated interfaces of the component instances.

The *resource environment submodel* defines resource containers (e.g., servers, workstations) and the network topology. In the figure, the resource containers are represented by the cuboids and the network connections are represented by the lines between them. How component instances are deployed on the resource containers is defined by the *deployment submodel*. In the figure, the deployment model is shown implicitly by the placement of component instances in the resource containers.

The *usage submodel* specifies the amount of users, as well as their behavior.

B. PCM Security Extensions

Allowing to use PCM models as input for our analysis, we deposit new attributes within some of the submodels of the PCM. For our security analysis every PCM submodel except the usage model is relevant. Only the repository and the resource environment models have to be extended. We extend the components by two new attributes: the *time to discover vulnerability* and the *probability of component breakability*. When a component is instantiated into a system, all its instances inherit the values for these attributes. Additionally, to express mutual security interference a new relation is introduced between components. Technically, we decided not to alter the

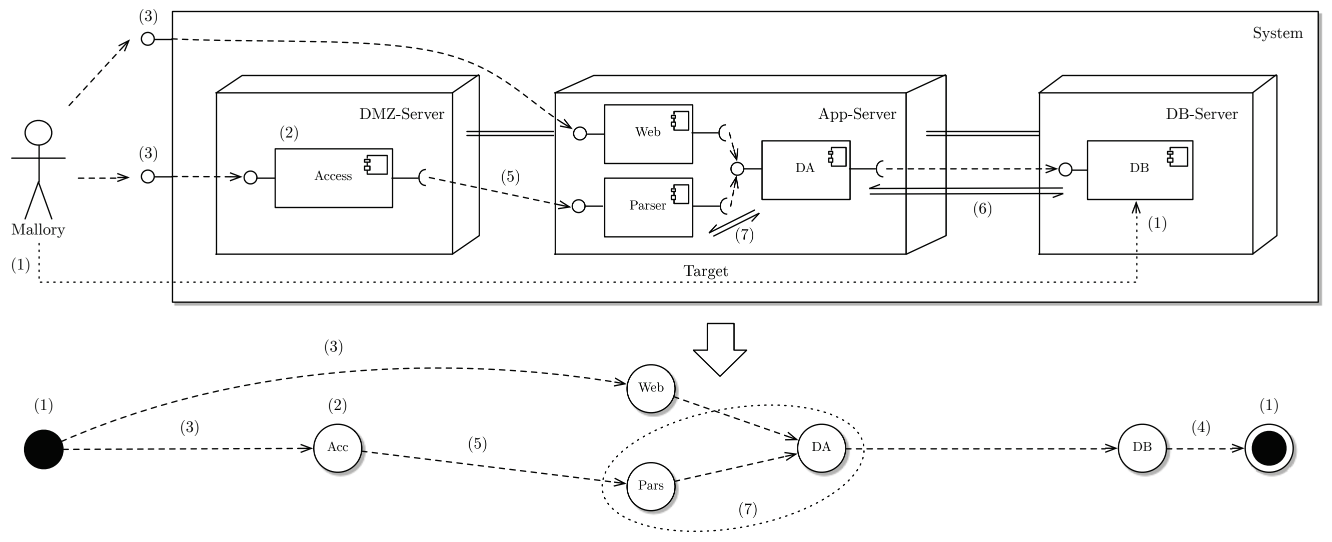

Fig. 9: Example transformation from architecture to analytical model

current metamodel by intrusively adding our attributes, but to create a metamodel extension. These extensions can either be realized by decoration or stereotyping.

In addition, we also need two new metamodels to support our approach: the attacker model and the attack scenario model. They are stored separately to enable arbitrary combinations of both, since an attacker is system independent, but an attack scenario is not. This improves the model's applicability for further extensions. The attacker model contains the attributes (λ, Δ) presented in III-B. Finally, the attack scenario model references the target component.

C. Transformation to Analytical Model

We first transform the security annotated architecture model into a Semi-Markov process, which then can be further analyzed. This transformation is specified in pseudocode in Algorithm 1 and illustrated in Fig. 9 by means of the running example. The resulting model contains one start and one goal state (1). For each component in the architecture, one state is created (2). The getComp helper function retrieves the component to which a system interface is connected by a delegation connector. Visiting costs are annotated onto these states, which are calculated according to Formula 3. Transition are created from the start state to states of components which are connected to the system provided interfaces (3). The target component gets a transition to the goal state (4). Further transitions are added for each assembly connector (5). The toTrans helper function creates a transition between the states of the components which are connected by the connector, according to the direction of the connector. Mutual security interferences between components of different resource containers are not relevant to be considered (6). The onSameContainer helper function returns true, if the two affected components are deployed on the same resource container. Within one resource container, a mutual security interference leads to a fusion of all affected components' states (7). The fuseStates helper function modifies the states and transition sets. One of the two states which is affected by m is marked for deletion. All transitions coming from or going to the marked state are rewired to the other state which is affected by m. Finally, the marked state is removed. There is no need to handle duplicate transitions, since a set does not contain any.

Algorithm 1 Transformation

```
Input:
$K \leftarrow$ Component instances
$g \leftarrow$ Goal components ($g \in K$)
$C \leftarrow$ Connectors
$I \leftarrow$ System interfaces
$M \leftarrow$ Mutual security interference
Output:
$C$                                                         // States
$T \subseteq C \times C$                                    // Transitions

Algorithm:
$C \leftarrow \{start, end\}$                               // (1)
$C \leftarrow C \cup K$                                     // (2)
for $i \in I$ do
    $T \leftarrow T \cup \{(start, getComp(i))\}$           // (3)
end for
$T \leftarrow T \cup \{(g, end)\}$                          // (4)
for $c \in C$ do
    $T \leftarrow T \cup \{toTrans(c)\}$                    // (5)
end for
for $m \in M$ do
    if $onSameContainer(m)$ then                            // (6)
        $(C, T) \leftarrow fuseStates(m, C, T)$             // (7)
    end if
end for
```

VI. Related Work

Sharma et al. showed in [13] a hierarchical model for including several quality attributes, e.g., security in component-based software architecture models. They employed Discrete Time Markov Chains (DTMCs) to model the software architecture of a system. For their security quantification they describe a model representing the probability of exposing a vulnerability of a component in a single execution and its effect on the system's insecurity. In difference to our approach, this model considers how often a certain component is accessed. Since we assume an attacker accesses a component as often as needed, we do not consider the number of attempts. Additionally, the referenced model is designed to consider the system as broken if at least one component is attacked successfully. Since a system is typically deployed on several machines a broken component does not mean to get access to all systems for instance the database system. In consequence, we designed our approach to consider the control flow of a system an attacker typically underlies.

Madan et al. developed in [9], [8] a model for security quantification of intrusion tolerant systems using a Semi-

Markov process. They define two different state-transition models that cover the scenario of a Denial-of-Service (DoS) and the goal of compromising the system. Both models describe how an intrusion tolerant system behaves under attack. These models determine the DTMC steady-state probabilities using state transition probabilities. Finally, they calculate a *Mean Time To Security Failure* that allows quantifying the security of the system as a whole. In contrast to this model, our approach enables to assess security of component-based architectures and is not limited to whole monolithic systems. Thus, we abstract the set of states to make it feasible to be used in component-based architecture models. Additionally, we included an attacker model to respect the skills of attackers.

Jonsson et al. used in [7] empirical data to develop an attacker model that represents the process of an attack. The approach comprises three phases: Learning, standard attack, and innovative attack phase. This behaviour is approximated using an exponential distribution function. In our work, we use a similar approach to reflect an attacker's behaviour. We use the exponential distribution to get a probability of a certain skilled attacker being successful to discover a vulnerability in a given time.

Several related papers consider quantifying security or security modeling: Wang et al. showed in [16] a framework for security measurements to approximate security aspects of networks using attack graphs. Dacier et al. used in [4] Markov-chains and attack scenarios and used for their model the time and effort of an attacker to intrude a system using a privilege graph. Depending on the system size this graph can easily become large sizing and would require modeling vulnerabilities at a low level. Schechter showed in [12] the cost to break metric to be effective to simulate the cost of an attack and to get an idea of how hard it is to break into a monolithic system from the cost aspect of view. McQueen et al. proposed in [10] a model that allows an estimation of the time to compromise one particular visible system component by a certain skilled attacker.

VII. Conclusions

We presented an approach enabling a comparison of component-based software architectures when considering security quality attributes. Therefore, we uniquely regard and combine several security depending aspects: We specify the attacker's skill, a specific attack goal, the security attributes of the system's components, and mutual security interferences between them, each, in its own submodel. We integrate these submodels and bring them together in one mathematical representation using a Semi-Markov process. Thus, we achieve one integrated model resulting in the *Mean Time To Security Failure*, a metric that allows to assess and compare the security attributes of component-based software architectures. We extend the PCM metamodel to enable security annotations according to our approach. Then, we show a transformation of the security annotated model to the Semi-Markov process. Finally, we demonstrate the approach using practical examples.

In the requirements analysis phase, the developers come often face to face with architectural trade-off decisions. Trade-off decisions for requirements such as performance and reliability are already well covered. However, security considerations can not be incorporated due to the lack of a methodology for a systematic security assessment of component-based architectures. This gap is addressed by our approach. The systematic assessment of security attributes supports architectural trade-off decisions considering security aspects. Thus, our methodology helps developers to assess the security attributes of several component-based architectures and use the results as input for a trade-off decision process. In this work, we showed three possible architecture alternatives and evaluated them using our methodology. In a development process, these results could potentially be used to support the developer to reduce the effort and costs of the decision making process as well as to improve the strength of the selected decision or in the first place enabling to consider security aspects when being concerned about security.

The modular structure of our model enables a set of possible further extensions: Special attack scenarios could potentially target more than one component at the same time. For example if two database systems that an attacker both would have to get under control. This case would necessitate to extend the model in order to allow the definition and consideration of a set of goal points. Another extension could include the resource containers and middleware to the model to consider the security attributes of hardware containers and the surrounding software stack.

VIII. Acknowledgment

This work was partially funded by the Helmholtz Association of German Research Centers. We especially thank to Kiana Rostami, Aleksandar Milenkoski, and Lucia Happe for their input.

References

[1] J. Almasizadeh and M. Azgomi. Intrusion process modeling for secutity quantification. ARES, 2009.

[2] S. Becker, H. Koziolek, and R. Reussner. The Palladio component model for model-driven performance prediction. *JSS*, 2009.

[3] I. Crnkovic. Component-based software engineering - new challenges in software development. *Software Focus*, 2001.

[4] M. Dacier, Y. Deswarte, and M. Kaaniche. Models and tools for quantitat. assessment of operat. security. IFIP Conference Procs., 2002.

[5] T. de Gooijer, A. Jansen, H. Koziolek, and A. Koziolek. An industrial case study of performance and cost design space explorat. ICPE, 2012.

[6] C. Griffin, B. Madan, and T. Trivedi. State space approach to sec. quant. COMPSAC, 2005.

[7] E. Jonsson and T. Olovsson. A quantitat. model of the security intrusion proc. based on attacker behavior. *IEEE Trans. Software Eng.*, 1997.

[8] B. B. Madan, K. Goseva-Popstojanova, K. Vaidyanathan, and K. S. Trivedi. Modeling and quantification of security attributes of software systems. DSN'02, 2002.

[9] B. B. Madan, K. Goševa-Popstojanova, K. Vaidyanathan, and K. S. Trivedi. A method for modeling and quantifying the security attributes of intrusion tolerant systems. *Perform. Eval.*, 2004.

[10] M. McQueen, W. Boyer, M. Flynn, and G. Beitel. Time-to-compromise model for cyber risk reduction est. Advances in Inform. Security. 2006.

[11] Open Research Foundation. OSF DataLossDB: Data Loss News, Statistics, and Research. http://datalossdb.org/.

[12] S. Schechter. Quantitatively differentiating system security. In *Workshop on Economics and Information Security*, 2002.

[13] V. S. Sharma and K. S. Trivedi. Quantifying software perf., rel. and sec.: An architecture-based approach. *Journal of Syst. and Softw.*, 2007.

[14] Verizon. Data Breach Investigations Report. 2013.

[15] M. Walter and C. Trinitis. Quantifying the security of composed systems. PPAM, 2006.

[16] L. Wang, A. Singhal, and S. Jajodia. Toward measuring network security using attack graphs. QoP, 2007.