Systematic Security Analysis for Service-Oriented Software Architectures

Yanguo (Michael) Liu, Issa Traore
Department of Electrical and Computer Engineering
University of Victoria, BC, Canada
Email: yliu@ece.uvic.ca, itraore@ece.uvic.ca

Abstract

Due to the dramatic increase in intrusive activities, architecture security analysis and design has emerged as an important aspect of the development of software services. It is a well-accepted fact in software engineering that security concerns like any other quality concerns should be dealt with in the early stages of software development. However, current software security risk analysis approaches still heavily rely on ad hoc techniques. These involve significant amount of subjective efforts creating greater potential for inaccuracies. In this paper, we propose a User System Interaction Effect (USIE) model that can be used systematically to derive and analyze security concerns from service-oriented software architectures. Many aspects of the model derivation and analysis can be automated, which limit the amount of user involvement, and thereby reduce the subjectivity underlying typical security risk analysis process. The model can be used as a foundation for systematic analysis of software services from different security perspectives.

1. Introduction

Recently, service-oriented modeling has emerged as an effective technique for specifying and designing software architectures. Service-Oriented Architecture (SOA) lays the foundation of a new distributed framework that facilitates exposure of software components as services. In reality, many security exploits in software systems are caused by malicious uses of publicly available services [1]. Accordingly, it is essential to guarantee a high level of security in the design and implementation of software services. Even though several approaches and notations have been proposed to model security concerns under the SOA development framework [2], a huge amount of subjective and manual effort and expertise are still required in the security assessment and evaluation of software services designs. Likewise the best modeling techniques cannot prevent security flaws from slipping through design models.

In this paper, we propose a User System Interaction Effect (USIE) model that can be used to derive systematically security concerns from service-oriented software architectures, and analyze these concerns in order to identify possible security design flaws. To illustrate the concepts and models introduced in this paper, we use as running example a Flower Shop (FS) application, which is a web-based application allowing customers to purchase flowers online.

The rest of the paper is organized as follows. Section 2 gives an overview of service-oriented analysis and design. Section 3 discusses the limitations of the current SOA framework in addressing security concerns, laying down the rationale for developing the USIE paradigm. Section 4 defines the scope of the USIE paradigm, presents its semantic foundation, and illustrates its modeling elements and notations. Security analysis based on USIE model may be conducted from multiple perspectives targeting various security properties or attributes. To illustrate this capability of the USIE model, section 5 presents as an example service (security) mechanism strength analysis using the USIE model. Finally, Section 6 makes some concluding remarks.

2. Service Oriented Architecture

The Service-Oriented Architecture (SOA) is an architecture in which functionalities of a software system are constructed and delivered as services to either end-user applications or other services [3]. Service-Oriented Analysis and Design (SOAD) is an emerging modeling paradigm that combines elements and practices adopted from object-oriented design,
component-based design, enterprise software architecting, and business process modeling [3].

Conceptually, the SOAD methodology involves three major abstractions: Business Processes, Services and Operations [4]. Business Processes abstraction is used to describe business functions or activities usually identified by domain experts or customers. A business process usually consists of a series of services that are executed in sequence according to a set of business rules. Services abstraction is used to describe the collection of services underlying each of the business processes. Typically, a software service under the service-oriented architecture is composed either of other services or by a collection of logically related operations that correspond to logical transactions among system components. Operation abstraction is used to describe the set of logical operations involved in the services. Basically, the development of Business Processes, Services and Operations follow a hierarchical pattern as shown in Figure 1, which illustrates partially the architecture of the FS sample application.

![Figure 1. The SOAD Hierarchy and Reference Model](image)

So far, a number of techniques and principles have been used to design and implement operations, services, and business processes within SOAD [3], [4], [5]. Traditional software development technologies such as Component Oriented Design (COD) and Object Oriented Design (OOD) are still important and applicable within the SOAD framework. In practice, business processes or functions are normally implemented or designed as composite services, which in turn are composed possibly of other composite services and atomic services. An atomic service usually consists of a collection of SOA operations that can be mapped into either a component interface or a specific object-oriented (OO) method with a structured interface. Design and realization of the components with their composition objects are still carried out using traditional COD and OOD techniques.

3. Rationale for USIE Paradigm

A great deal of knowledge regarding secure system development has been gathered in the past decades. However, as of today, security assessment of a software product at the design level is still conducted in an ad hoc way with a huge amount of subjectivity. From an engineering standpoint, subjective assessments are mostly unreliable and inaccurate and should be avoided whenever possible; instead, more systematic evaluation methods should be employed. In this context, developing techniques allowing systematic and automatic software security analysis of SOAD design artifacts is a necessity.

The UML [6] enhanced with distinct stereotypes and profiles, has become a key notation used to capture SOAD abstractions [5], [2]. Using UML, conceptual artifacts, such as business processes and system services, can be expressed in concrete and visual models, which are more understandable and easy to evolve and communicate between different stakeholders. However, security analyses of UML models, on which SOA designs are often based, involve some challenges. Specifically, two major challenges arise when using UML service specifications for systematic security analyses. Firstly, the standard UML notation lacks features for capturing satisfactorily security-related events. Based on the standard UML notation, it is hard to derive security concerns of software services such as the privileges granted by the service, the security
mechanisms involved and so on. Secondly, expressing instead of reasoning is the primary purpose of UML diagrams; therefore, it is rather difficult (for a computer) to conduct directly security analysis on UML notations.

![Figure 2. A Service Specification](image)

In order to address some of these challenges, various security-related features have been proposed to specify security intent within software design process [2, 7, 8]. For instance, many security design patterns have been proposed in the literature in order to capture best practices design mechanisms; a summary of currently available patterns can be found in [7]. In the context of SOAD, Johnston introduced a set of primitive modeling features allowing security intent to be explicitly captured when specifying software services [2]. These consist of dedicated stereotypes and profiles augmenting the standard UML notation. As an example, Figure 2 describes FS UpdateAccount service, which allows a user to update account information. The service execution is specified using a stereotyped UML sequence diagram. The service execution starts by the submission of UpdateAccount request by the service requestor, followed by the validation and processing of the request message. The specification is annotated using security stereotypes suggested by Johnston to capture underlying security intents [3]. For instance, in Figure 2, \texttt{<audit>>} indicates that the ControlUnit component will record a “Failed” event; a candidate profile for the \texttt{<audit>>} stereotype is also defined in [3].

Unfortunately, modeling security intent is not enough to address software security concerns. Even though software architects, by applying proper security modeling techniques, can effectively communicate security requirements, the actual design may still convey important security flaws. As an example, the service specification in Figure 2 relies on an “Audit” mechanism; a decision has to be made to either design an independent audit component or use the audit functionality provided by the operating system. Such design decisions could affect the overall security strength of the software service. In order to optimize the design of a software system from a security perspective, it is necessary to be able to evaluate the security levels of alternative designs for the same system. This need is also emphasized by the fact that many security attacks could be conducted successfully against service designs such as the one illustrated in Figure 2. For instance, based on this design, some of the attack scenarios that may be successful include (among other) dictionary and denial of service (dos) attacks. A dictionary attack may be conducted against the ValidateInput operation allowing an illegitimate user to steal the credentials of a legitimate one. A dos attack may simply bring down the server, preventing legitimate requestors from accessing the provided service.

In this context, there is an urgent need beyond modeling of security intent, to be able to isolate and analyze systematically such intent in order to identify and remove potential security design deficiencies. The goal of the USIE model is to address these deficiencies by isolating and capturing security intents and events, and expressing such information in a machine-processable form for automated security analysis of software services designs. More specifically, a USIE model captures in one hand the security dependencies underlying the hierarchical structure of software services, and on the other hand the traces of the security related events in user-system interactions as expressed by UML interaction diagrams. Such information can be collected automatically from UML models by defining or reusing adequate security profiles as shown in the example of Figure 2. The collected information can also be used to compute and analyze adequate security measures with the purpose of improving architecture design.

4. USIE Modeling

We describe in this section the semantic foundation and notation underlying the USIE model.

4.1 Abstract Service Model

Software services correspond to business functions derived from business processes analysis. Depending on the business process, the service may be composed of other services or consist of a collection of components that work collectively to achieve corresponding business function. Components are finer-grained than services, and also
map into business entities and corresponding behaviors or business rules in contrast with services, which as indicated earlier map into business functions. To deliver the business function it represents, a service may manage operations across a set of business entities. As such it creates and manages its own set of components.

According to Millen [9], a system can be defined as “a set of components configured to provide a set of user services”. As indicated earlier, each service owns and manages a set of components. These components deliver the business function represented by the service through various configurations. Likewise, according to Millen, a service can be defined as a set of alternative configurations, where each configuration consists of a collection of components connected in a specific way, each providing specific services referred to as the supporting services for the corresponding configuration.

In this context, Millen defines a system \( S \) as a set of service configurations with a partition \( \overline{S} \) on \( S \) representing the set of services delivered by the system. Each partition consists of a collection of configurations, and maps into a software service. Each configuration in its turn maps into a set of supporting services. The partitioning in configurations defines a hierarchical structure between services. In this hierarchy, terminal services, which are isolated from one another, are considered atomic services. An atomic service has only one configuration and has no supporting service.

As an example, Figure 3 depicts the service hierarchy for the Ordering Service of the FS application. This architecture can be refined further, but as suggested by Millen, the depth of the hierarchy is a design choice; it is up to the designer to decide the level of granularity of service specification [9].

In Figure 3, the squares represent services and ellipses represent service configurations. An arrow pointing from a square \( X \) to an ellipse \( Y \) indicates that \( Y \) is a configuration of \( X \). An arrow pointing from an ellipse \( X \) to a square \( Y \) indicates that \( Y \) is a supporting service of \( X \). For instance, the Ordering Service is based on a single configuration that is supported by two services: Account Service and Payment Service. An interesting service in this hierarchy is the Account Service, which has two alternative configurations where one configuration is defined for new users, and the other is defined for exiting users. More specifically, a new user needs to register by creating an account. After registration, he may start using the other functions offered by Account Service without going through authentication; so the support set for this configuration is \{register, update account, logout\}. An existing user, in contrast, does not need to register, but has to be authenticated. Hence, the support set for existing user configuration is \{login, logout, update account\}.

**Figure 3. Example: Service-oriented architecture for the Ordering Service of Flower Shop Application.**

### 4.2 USIE Model

A USIE model provides an abstraction of the security features associated with software services. A USIE model involves three different types of features: entities, links, and branches. A USIE entity is an abstraction for the main elements composing a service hierarchy. There are six types of USIE entities namely atomic service, composite service, configuration, (security) mechanism (component), resource (component), and null entity. A USIE link is an abstraction for the various kinds of relationships between entities, such as the dependency between services or the support relationship between configurations and services, or the links between an atomic service and the operations involved. A USIE branch is a junction between related links. A branch is labeled according to the semantic of the corresponding junction. For instance, the junction between dependency links may be labeled using “OR” or “AND” to express the underlying composition semantic. The junction between operation links may be labeled using one of \{“Sequential”, “Concurrent”, or “Exclusive”\} to express the underlying execution semantic.

A service-oriented software system can be represented by a collection of USIE graphs, each representing a corresponding software service or service configuration. A USIE graph \( U \) can be defined as a tuple \((N, E, \prec, L)\), where \( N \) represents a set of nodes corresponding to USIE entities,
$E \subseteq N \times N$ represents a set of directed edges corresponding to USIE links, $<$ is a partial order relation over $E$, and $L$ represents a set of labeled USIE branches characterizing the relationships between the edges in $E$. As explained in the abstract software model introduced in previous section, a software system can be represented through a hierarchical service structure, in which composite services are defined in terms of a set of alternative configurations. Configurations are defined in terms of their supporting services, and atomic services are defined in terms of the collection of operations involved. Accordingly, three different kinds of graphs, which capture this hierarchical structure, are defined in the USIE model, namely atomic service graph, configuration graph and composite service graph. The purpose of a USIE composite service graph is to describe the service hierarchy underlying a specific composite service. It is represented as a tree in which the nodes correspond to either configuration or service entities and the edges depict the supporting relationships between the services and configurations entities. A USIE configuration graph consists of USIE atomic service entities and service dependency relationships. A USIE atomic service graph is defined as a tree, in which the root node corresponds to a USIE atomic service entity representing the service itself, and leaf nodes correspond to USIE component entities (e.g., USIE security mechanism, USIE resource and USIE null entity). The directed edges in a USIE atomic service graph are USIE links representing the USIE operations associated with the USIE components. Each USIE operation link is labeled with the set of (security) privileges or rights associated with the operation. In practice, the USIE graphs of a service-oriented system can be either constructed directly by software architects or derived systematically from the software architecture description produced using an Architecture Description Language (ADL) such as the UML. We refer readers to [12] for a complete and detailed description of the USIE model and various examples.

4.3 Example

In this section, we illustrate USIE modeling through selected services from the FS application.

---

Figure 4. Annotated Service Specification

We illustrate USIE atomic service graph construction by considering as an example the UpdateAccount service illustrated by the sequence diagram of Figure 2. We start by annotating this sequence diagram using stereotypes corresponding to USIE modeling features. The annotated diagram is shown in Figure 4. For instance, in Figure 4, the component named Component_InputValidator is a security mechanism that implements user account information validation. As such it is annotated as a security mechanism using stereotype $<$<<SMComponent>>$. The GetAccount operation is annotated by a right stereotype $<$<<R>>$ simply because it is supposed to get some information from its target component. Figure 5 depicts the USIE atomic service graph derived from the sequence diagram shown in Figure 4. Specifically, the construction of the USIE model starts by creating the root node that is a USIE atomic service entity after the corresponding sequence diagram. Three USIE operations are involved in the atomic service execution, namely GetAccount, ValidateInput and SetAccount. Each of these operations is represented by a separate USIE operation linking the root node to corresponding target components. It is important to note that no explicit names are used to represent the operations in the USIE model (in contrast with a typical UML interaction diagram). From the service specification of Figure 4, we can notice that the three USIE operations are executed in sequence. As a result the corresponding USIE links can be related using a “Sequential” label. In particular, an “Exclusive” label is used to further characterize the SetAccount operation, by associating it with a dummy operation. This captures the fact that the execution of the SetAccount operation depends on whether or not the results of the ValidateInput operation is successful or not (as indicated in Figure 4). Notably, the ControlUnit component of Figure 2 is not represented in the USIE model of Figure 5.
since this component is not considered (here) to be security related.

Figure 5. USIE Atomic Service Graph for UpdateAccount Service

Figure 6. USIE Configuration Graph for Payment Service

To illustrate the construction of a USIE configuration graph, we consider as an example a configuration of the Payment Service in the FS application. Actually, the Payment Service is a composite service involving a single configuration supported by three atomic services, which include CheckoutCart, ProcessPayment, and Delivery services. The corresponding USIE configuration graph is illustrated by Figure 6. The graph consists of three atomic service nodes corresponding to the supporting services of the Payment Service. These atomic service nodes are connected by service dependency arcs. Specifically, the Delivery service depends on the ProcessPayment service, and the ProcessPayment service (in its turn) depends on the CheckoutCart service.

Each of the atomic service nodes is further refined by providing (in the same configuration graph) its corresponding USIE atomic service graph.

Figure 7. USIE Composite Service Graph for Ordering Service

As an example of composite service graph, Figure 7 depicts the composite service graph for the FS Ordering service. Note that Figure 7 is consistent with the service specification shown in Figure 3. In a USIE composite service graph, configuration nodes are defined in terms of their supporting services, which in their turn are defined in terms of their configurations, and so on and so forth. For instance, in Figure 7, the Ordering Service involves a single configuration, which is directly supported by Account Service and Payment Service. Account Service in its turn involves two different configurations, which in their turn are supported by other services.

5. Service Mechanism Strength Analysis Using USIE Model

Using USIE abstraction, a software system can be represented as a collection of services structured hierarchically where the top service is a composite service representing the application itself. Furthermore, each of the services and configurations
involved in the hierarchy can be described using a corresponding USIE graph that can be used as a basis for architecture-level security analysis of service-oriented software system. Such kind of analysis can help identifying potential security problems at the early stages of software development. Accordingly, software architects can adjust or revise system design to mitigate security risks associated with the operational life of the software system. We illustrate in this section, security analysis based on USIE model using the particular case of security mechanism strength. We refer the reader to [12] for several other UISE-based security techniques and examples.

5.1 Notion of Privilege-Mechanism Pair

In software applications, privileges carried by software services are normally restricted or protected through appropriate security mechanisms, and some privileges may deserve more restriction than others. A security mechanism can be, for instance, an encryption program, a password-checking program, or other kinds of security control or protection units. Generally, security mechanisms could be viewed as special kinds of modules that may involve one or more software components and provide particular security related services. In this framework, we are primarily interested in the security analysis of software services. We therefore focus on the strength of the security mechanisms involved in the protection of a particular software service. Specifically, the mechanism strength of a service is conveyed by the protection strength of the privileges associated with it. In other words, knowing how the privileges granted through a service are protected is the basis of the analysis of service mechanism strength.

Using USIE model, the set of privileges associated with a service and its (corresponding) relevant security mechanisms can be derived systematically. In this work, we use a notation named Privilege-Mechanism Pair (PMP) to derive and express such information. Specifically, we represent a PMP as a pair (Privilege, Mechanism Composition) where Privilege is represented as a triple (Subject, Object, Rights), in which Subject, Object, and Rights represent the subject, the target object, and the set of rights involved, and Mechanism Composition captures the list of security mechanisms protecting the privilege with their logical relationships.

5.2 Privilege-Mechanism Pair Derivation

In the USIE graphs, the mechanism composition information is implicitly illustrated by the labels associated with USIE branches. For instance, the labels for USIE operations (i.e., sequential, exclusive, and concurrent) indicate the execution order of the security mechanism components associated with corresponding operations. The labels for service dependencies (i.e., and, or) indicate the logical relationships between security mechanism components involved in the service executions. Accordingly, we can derive systematically the PMPs of a software service using corresponding USIE graphs. Algorithm 1 in the following defines the procedure to derive the set of PMPs from the USIE configuration graph of a software service.

Algorithm 1.

FUNCTION: AtomicService_Privileges_Derivation(a USIE AtomicService Graph G(s))
RETURNS a set of privileges Ps,
INPUTS: a USIE AtomicService Graph G(s) = (N N L E E L L)
where N = (n0 n1 ... nN), E = (e1 e2 ... eN), L = (l1 l2 ... lN);
INITIALIZATION: i = 0; P(i) = null;
DO P(i) = P(i) ∪ ((Service Requestor(s), i, Rights(s)));  
REPEAT: i = i + 1; 
IF (i ∈ N(operation))  
   DO P(i) = P(i) ∪ ((Service Requestor(s), Tag of (s), Rights(s))); 
END IF 
UNTIL: i = k
RETURN: P(k)

FUNCTION: Service_PMP_Derivation (a USIE Configuration Graph G(s))
RETURNS a set of Privilege-Mechanism Pair PMP(s),
INPUTS: a USIE Configuration Graph G(s) = (N N L E E L L)
where N = (n0 n1 ... nN), E = (e1 e2 ... eN), L = (l1 l2 ... lN);
INITIALIZATION: i = 0; P = null; p = null; mc = null; PMP(s) = null;
REPEAT: i = i + 1; 
IF (ni is AtomicService node)  
   DO P = AtomicService_Privileges_Derivation(G(ni));
END IF 
FOR EACH: p ∈ P  
IF (p has no dependencies)  
   DO mc = mechanism(ni);  
ELSE IF (p has "And" dependencies on service nodes n0 n1 ... nk)  
   DO mc = mechanism(ni) ∨ mechanism(n0) ∨ ... ∨ mechanism(nk);  
ELSE IF (p has "Or" dependencies on service nodes n0 n1 ... nk)  
   DO mc = mechanism(ni) ∨ mechanism(n0) ∨ ... ∨ mechanism(nk);  
END IF  
DO PMP(s) = PMPs ∪ ((p, mc));
END FOR EACH 
UNTIL: i = k
RETURN: PMPs
AtomicService_Privileges_Derivation, defines the steps for the derivation of PMPs from a USIE configuration graph. Specifically, the function Service_PMP_Derivation uses the USIE branches to guide the derivation of the security mechanism compositions for each of the privilege derived from the function AtomicService_Privileges_Derivation.

5.3 Mechanism Strength Metrics Definition

Based on the PMPs of software services, a variety of metrics can be developed to assess the security protection strength of software services. For instance, we can evaluate the protection strengths of service privileges with respect to the “Weakest Link” design principle presented in [10]. According to the “Weakest Link” principle, the security level of a system protected by a collection of security mechanisms is determined by the weakest protection mechanism. Based on this principle, we define in the following, Metric 1 as an example of Mechanism Strength metric related to the privileges underlying a software service.

**Metric 1. Privilege Mechanism Strength:** Given a software service \( s \) and the set of Privilege-Mechanism Pairs \( \text{PMP}(s) \), the Privilege Mechanism Strength of a privilege \( p_r \) in \( s \), denoted by \( \text{PMS}(p_r, s) \) is given by

\[
\text{PMS}(p_r, s) = \min_{(i \in \text{PMP}(s)) \& p_r \in \text{Privileges}(i)} \text{MechanismStrength}(i)
\]

Where \( \text{Privileges}(i) \) represents the set of privileges involved in the PMP \( i \) and \( \text{MechanismStrength}(i) \) represents the strength of the mechanism composition involved in the PMP \( i \).

To compare the strength of a composition of various security mechanisms, we need to normalize the individual measures involved. One way to achieve that is by assigning normalized weights to each of the security mechanisms involved and computing a global value based on the composition rules. The value of a mechanism weight is mechanism-specific, and could be provided by corresponding vendor or computed using specific evaluation techniques. For instance the rating technique proposed by Eloff for security mechanisms such as cryptographic primitives or password mechanisms could be used [11].

5.4 Example

Knowing how privileges are protected is critical in the design and operation of the FS application since its main purpose is to provide services online, an open environment where intrusions and hacking activities happen all the time. Software architects should be certain that appropriate security mechanisms are designed in FS services to protect each of the privileges associated with the delivery of sensitive information. To illustrate service mechanism strength analysis, we use the Payment Service of the FS application as an example.

As introduced in section 4.3, the Payment Service of the FS application is a composite service involving three atomic services; each provides privileges to access some system resources. The security mechanism involved in the execution of the Payment Service is Component_Cookiemanager. Suppose that the normalized weight for the security mechanism of Component_Cookiemanager is set to 0.5 (over a range of \([0,1]\)). We can compute systematically the protection strength for each of the privileges associated with the Payment Service using the security mechanism strength analysis technique introduced earlier. Specifically, the (systematic) analysis is conducted in three steps. Firstly, the USIE configuration graph of the Payment Service, the basis of the analysis, which is shown in Figure 6, has to be constructed. Secondly, the Privilege-mechanism pairs of the Payment Service, can be derived systematically from its configuration graph using Algorithm 1; Table illustrates the derived PMPs.

<table>
<thead>
<tr>
<th>No.</th>
<th>PMPs of the Payment Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((ServiceRequestor(Payment), CheckoutCart, {E}), null)</td>
</tr>
<tr>
<td>2</td>
<td>((ServiceRequestor(Payment), ProcessPayment, {E}, Component_CookieManager)</td>
</tr>
<tr>
<td>3</td>
<td>((ServiceRequestor(Payment), Delivery, {E}, Component_CookieManager)</td>
</tr>
<tr>
<td>4</td>
<td>((ServiceRequestor(Payment), Component_CookieManager, {E, R}, null)</td>
</tr>
<tr>
<td>5</td>
<td>((ServiceRequestor(Payment), Table_arrangementCart, {R}, Component_CookieManager)</td>
</tr>
<tr>
<td>6</td>
<td>((ServiceRequestor(Payment), Table_arrangements, {W'}, Component_CookieManager)</td>
</tr>
</tbody>
</table>
Table 2. PMS Values for the Payment Service Privileges

<table>
<thead>
<tr>
<th>Privileges of the Payment Service</th>
<th>PMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ServiceRequestor(Payment), CheckoutCart, (E))</td>
<td>0</td>
</tr>
<tr>
<td>(ServiceRequestor(Payment), ProcessPayment, (E))</td>
<td>0.5</td>
</tr>
<tr>
<td>(ServiceRequestor(Payment), Delivery, (E))</td>
<td>0.5</td>
</tr>
<tr>
<td>(ServiceRequestor(Payment), Component_CookieManager, (E))</td>
<td>0</td>
</tr>
<tr>
<td>(ServiceRequestor(Payment), Table_arrangementCart, (R))</td>
<td>0.5</td>
</tr>
<tr>
<td>(ServiceRequestor(Payment), Table_sessions, (W))</td>
<td>0.5</td>
</tr>
<tr>
<td>(ServiceRequestor(Payment), Table_flowers, (W))</td>
<td>0.5</td>
</tr>
<tr>
<td>(ServiceRequestor(Payment), Table_carts, (W))</td>
<td>0.5</td>
</tr>
<tr>
<td>(ServiceRequestor(Payment), Table_user, (W))</td>
<td>0.5</td>
</tr>
<tr>
<td>(ServiceRequestor(Payment), Table_arrangementCart, (W))</td>
<td>0.5</td>
</tr>
<tr>
<td>(ServiceRequestor(Payment), Table_cookie, (W))</td>
<td>0.5</td>
</tr>
<tr>
<td>(ServiceRequestor(Payment), Table_carts, (W))</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Thirdly, based on the derived PMPs, the protection strength of each of the privileges in the Payment Service can be computed using Metric 1. For instance, Table 2 presents the final protection strength analysis results for the privileges involved in the Payment Service.

Based on PMS measures, software architects can therefore identify unprotected privileges or determine whether the protections of specific privileges should be enhanced, and then adopt accordingly appropriate mitigating solutions. For instance, according to Table 2, privilege (ServiceRequestor(Payment), CheckoutCart, \(E\)) is unprotected in the payment service; therefore, the design of the payment service may need to be revised if software architects or security analysts determine the privileged needs protection.

6. Conclusion

In this work, we have proposed a User System Interaction Effect (USIE) model that can be used systematically to derive and analyze security concerns from service-oriented software architectures. This model can serve as foundation for systematic analysis of software service from different security perspectives.

No empirical results are provided in this paper to sustain our claims. Although such aspect is important, this is not the focus of this paper. The main purpose of this paper is to introduce the foundation and practical usage of the USIE model. In a concurrent effort, we have undertaken various empirical studies in order to validate and establish the usefulness of the proposed security analysis model. We refer interested readers to [13, 14] for instances of such kind of empirical studies.

References


