UML-based Security Measures of Software Products

Yanguo(Michael) Liu, Issa Traore

Department of Electrical and Computer Engineering
University of Victoria, PO Box 3055 STN CSC
Victoria BC V8W 3P6, Canada
Email: yliu, itraore@ece.uvic.ca

1 Introduction

System security is informally defined as the ability that the system is able to resist accidental or deliberate attacks [Sommerville, 2001]. Software security has become more and more concerned in the development of software applications for the reason that our society becomes more and more reliant on software systems and software attacks are dramatically increasing in recent years. It is not only large organizations and governments that are susceptible to security attacks, security has also become a concern for the average citizen for the sake of the growth of Internet activities that have led to a dramatic increase in the number of network and computer system attacks. Citizens are more and more troubled by the security threats over computer networks, discouraging them to protect their confidential work in computers and transfer their personal information over the Internet.

A way to improve software quality involves using metrics to guide the development process. Metrics have been successfully developed for a broad range of external attributes including reliability, performance, and maintainability [Henderson-Sellers, 1996], [Chidamber et al., 1994], [Allen et al., 2001]. Notable exception remains in the field of software security, which is still immature [Nielsen, 2000]. In this regard, Vaughn questioned the feasibility of "measures and metrics for trusted information systems" [Vaughn, 2001]. According to him, metrics are possible in disciplines such as mechanical or civil engineering because they comply with the laws of physics, which can be used to validate the metrics. In contrast, the software engineering discipline is not compliant with the laws of physics and faces huge challenges in establishing correctness. Vaughn, however, suggests that effective security metrics can be defined by accepting some risk in how they are used and by validating them in the real world through empirical investigation and experimentation.

Fenton identifies three classes of entities that are of interest of software engineering measurement, which are respectively products, processes and resources [Fenton, 1991]. The last several years have seen an increasing interest in rigorous approaches for measurement definition and evaluation on software product. Most of the proposed approaches formalize some intuitive understanding of some internal attributes of software products under the form of a collection of axioms
[Weyuker, 1988], [Briand et al., 1996], [Poles et al., 2000], [Rossi et al., 2003], then the set of metrics developed for internal software attributes are usually considered to be predictions for some external attributes of software systems. The early-stage evaluation on the external security attribute of a software product is the main concern in this paper, according to the Information Technology Security Evaluation Criteria (ITSEC), the discipline of computer security is founded on three basic properties: confidentiality, integrity, and availability [ITSEC, 1991]. There are of course several other aspects such as authentication, access control, accountability, which can be related, in one way or other, to these basic properties. In this paper, we focus only on confidentiality and derive corresponding metrics by relating it to a high level UML behavioural specification, namely sequence diagrams [UML, 2003].

The rest of the paper is organized as follows. Section 2 describes software security theory underlying our framework, Section 3 briefly overviews the UML sequence diagram and describes how it can be used to develop a User-System Interaction Effect (USIE) Model – a new paradigm introduced in this paper. Section 4 shows how confidentiality can be measured using USIE models. Section 5 illustrates a case study showing the effectiveness of our proposed metrics in software security evaluation. Section 6 concludes this paper and discusses future work.

2 Foundation Theory of Software Security

So far there is no widely accepted theory on software security. The main reason behind that is the inherent difficulty of the notion of software security. Software security involves multiple dimensions, such as authentication, authorization, audit, confidentiality, integrity and so on [Shreys]. Some of these dimensions, for instance, authentication, authorization, and audit, can be specified as system functional requirements, thus they can be verified and tested as normal system functionalities. Others like confidentiality and integrity correspond to non-functional requirements in engineering secure software. Non-functional requirements are difficult to capture and analyze, and it is quite common that software systems being developed without serious handling of non-functional security requirements.

Generally, software developers view software systems from two aspects, which are respectively structure aspects and behaviour aspects [Sommerville, 2001]. The static software architecture is illustrated on the structure aspects, and the services that a software system provides are demonstrated on the behaviour aspects. Security is much more related with software behaviour aspects since most of attacks in reality have to more or less take advantages of the deficiency of the provided software services. Our understanding of software security foundation is consistent with the work of Jacob that defines software security in terms of user-system interactions [Jacob, 1992]. We think that it is safe to say that security issues arise because of the occurrence of some user interactions with the system. An interaction may affect system security either in isolation or by interfering with other interactions. We limit the focus of this paper to the latter kind of
interaction since software confidentiality problems often rise on the latter case. Famous examples of security attacks based on interference between interactions include denial of service and race conditions. Denial of service is due to the fact that a non-authorized user prevents some legitimate users from properly using the system: the interaction executed by the attacker prevents the interactions initiated by legitimate users from carrying forward. Race conditions attacks consist of actively exploiting classical race conditions to access shared resource in an inappropriate way.

Furthermore, in [Jacob, 1992], Jacob provides formal definitions of key security concepts such as confidentiality and integrity and illustrates the relationships between these concepts and system functionalities by introducing several basic theorems. We present in this paper the confidentiality-related definitions. According to Jacob, “confidentiality is about limiting how much one user can infer about another user’ interaction with the system by making an interaction with the system themselves [Jacob, 1992].” Hence, confidentiality is function of the interference occurring among interactions between different users and the system. So possible measure of confidentiality may consist of evaluating this interference. We denote by $\text{Conf}(I_A \rightarrow I_B)$ the confidentiality of $I_A$ with respect to $I_B$, where $I_A$ represents an interaction of a user A with the system, and $I_B$ an interaction of a user B with the system. In other words, $\text{Conf}(I_A \rightarrow I_B)$ measures how much B can discover about $I_A$ via $I_B$.

3 Behavioral Modeling

3.1 Overview of UML Sequence Diagram

UML provides several diagrams that can be used to describe dynamic behaviour. Examples of such diagrams include sequence, collaboration, statechart, and activity diagrams. In this paper, we focus only on one behavior diagram, namely the sequence diagram. Extension of the approach presented in the sequel to other behavioral diagrams is left for future work.

A UML sequence diagram is used to describe interactions between objects in terms of sequences of messages they may exchange as the interaction unfolds over time to effect the desired operation or result [UML, 2003]. More precisely, it contains a set of role entities and a set of partially ordered messages, each specifying a communication between two role entities. In order to illustrate the concepts introduced in this paper, we use as a running example a subset of the requirements of a medical information system for patient records keeping. Figure 1 shows a UML sequence diagram that demonstrates a “ReadRecord” interaction between a doctor and the software system for keeping record. We assume that the medical records are stored in their encrypted form as a second layer of defense. This is not unusual with highly sensitive documents (e.g., password files etc.). We will come back to the motivations for such solution in Section 5. From Figure 1, any useful read scenario involves obtaining a key from the key server and decrypting the record. The record is then re-encrypted before storing.
3.2 User-System Interaction Effect Model

It appears from the security definitions given in Section 2, that the understanding of user interactions, the data and role entities involved play an important role in the analysis of security events. Regular UML sequence diagrams, however, provides only the communication information for an interaction while the responses of role entities are not expressed. However, UML semantics allow stereotypes to be specified for each communication in the sequence diagram, which provides the facility to express the communication effects [UML, 2003]. In order to facilitate analysis of software security events, we introduce a new paradigm named User-System Interaction Effect (USIE) model that combines the necessary information provided by regular sequence diagrams and the complemented information of stereotypes. More specifically, a USIE model captures, in one hand a trace of the communication in a user-system interaction as defined by a sequence diagram, and on the other hand highlights the communication effects expressed by the particular communication stereotypes.

More specifically, a USIE model can be derived from a UML sequence diagram and the complemented stereotypes of all the communication involved in the sequence diagram. It consists of a graph in which there are two kinds of nodes named InteractionStart and RoleEntity. An InteractionStart node represents the starting point of an interaction, and is in principle named after the interaction name (i.e. the name of the UML sequence diagram). A USIE model involves a single InteractionStart node. In contrast, a USIE model can have multiple RoleEntity nodes. A RoleEntity node corresponds to a role entity contained in the sequence diagram; it is named after corresponding role entity name. The nodes information of a USIE model can be directly derived from the corresponding sequence diagram.
The edges in a USIE graph have four characteristics. First, all the edges are directed. A directed edge in a USIE model represents a single communication (sending an event or invoking a method) between two role entities; it is directed from its source role entity to its target role entity. Second, an edge may optionally have attributes associated with it. There are two kinds of attributes named ChangeState and ReturnInformation. An edge has ChangeState attribute if the communication modifies the state of its target role entity; it has ReturnInformation attribute if the communication returns information to its source role entity. An edge can have zero or more attributes. Third, the edges in a USIE model are ordered according to the communication order expressed in the UML sequence diagram. Fourth, each node has exactly one incoming edge. In our methodology, the edge attributes, ChangeState and ReturnInformation, are expressed using the communication stereotypes of the sequence diagram semantics. Therefore, the edges information of a USIE model can be derived from the communication expressions of the corresponding sequence diagram.

We represent a USIE model using a graphical notation whose features are described in Figure 2. Figure 3 shows a USIE model that is derived from the sequence diagram of Figure 1 with all communications stereotyped. The edges in Figure 3 are ordered by numbering them from 0 to 4.

![Graphical notation for USIE elements](image)

**Fig. 2.** Graphical notation for USIE elements

### 4 Metrics based on Sequence Diagrams

#### 4.1 Notation

We define some basic notations that will be used to define our security metrics. Let $U = (N, E, <)$ represents a USIE model, where $N$ is a set of nodes, $E$ is a set of edges, and $<$ is a partial order relation over $E$. Each node in the USIE model must be named, we denote a node of the model by $N_a$, where $a$ is a string
representing the node name. Since edges in the USIE model are ordered, we denote an edge of the model by $E_i$, where $i$ is a non-negative number representing the order of the edge. Given nodes $E_i$ and $E_j$, $E_i < E_j$ if and only if $i < j$; we say that $E_i$ comes before $E_j$. We use several additional notations that are listed in Table 1 with their interpretations.

**Table 1. Notations for a USIE model**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Interpretations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_a : T$</td>
<td>Node $N_a$ of type $T$ (e.g. Role, Entity, InteractionStart)</td>
</tr>
<tr>
<td>Source($E_i$)</td>
<td>Source node of edge $E_i$</td>
</tr>
<tr>
<td>Target($E_i$)</td>
<td>Target node of edge $E_i$</td>
</tr>
<tr>
<td>Attributes($E_i$)</td>
<td>Attribute set of edge $E_i$</td>
</tr>
<tr>
<td>Incoming($N_a$)</td>
<td>Incoming edge of Node $N_a$</td>
</tr>
<tr>
<td>$N_a \bowtie U$</td>
<td>Node $N_a$ of $U$</td>
</tr>
<tr>
<td>$E_i \bowtie U$</td>
<td>Edge $E_i$ of $U$</td>
</tr>
</tbody>
</table>

### 4.2 Confidentiality Metrics

In this section, we present a method to evaluate quantitatively the confidentiality $Conf(I_A \to I_B)$ of an interaction $I_A$ with respect to another Interaction $I_B$. Confidentiality is related to information sharing, information leakage decreases confidentiality [Jacob, 1992]. In our methodology, we measure confidentiality by
identifying the potential information leakages between interactions. Kemmerer proposed that finding a pair of “Read” and “Write” operations on a shared resource could identify an information leakage channel between two processes [Kemmerer, 1983]. More specifically, whenever a process may read a resource that another process can write, the former process may deduce some information about the latter process, which decreases the confidentiality of the latter process.

Under this setting, identification of information leakage channels between two user-system interactions using USIE models is straightforward. A “Read” operation may correspond to an edge with ReturnInformation attribute in a USIE model while a “Write” operation may correspond to an edge with ChangeState attribute in a USIE model. Based on this consideration, we define an Information Leakage Channel as follows:

**Definition 1.** An Information Leakage Channel from a USIE model \( U_1 = (N_1, E_1, <) \) to a USIE model \( U_2 = (N_2, E_2, <) \) via node \( N_a \), denoted \( ILC_{N_a}(U_1, U_2) \), exists if

(a) \( (N_a : RoleEntity) \land (N_a \in N_1 \cap N_2) \)
(b) \( \text{ChangeEvent} \in \text{Attributes(Incoming}(N_a \bullet U_1)) \)
(c) \( \text{ReturnInformation} \in \text{Attributes(Incoming}(N_a \bullet U_2)) \)

An information leakage channel exists between two USIE models \( U_1 = (N_1, E_1, <) \) and \( U_2 = (N_2, E_2, <) \), whenever they share a common RoleEntity node, which is targeted in one model by a ChangeState edge, and in the other model by a ReturnInformation edge. Information leakage channels can be categorized according to their importance. An information leakage channel between interactions is significant if the information leaked by this channel can be sent to the InteractionStart node, otherwise the channel is secondary. We give the following definition:

**Definition 2.** Let \( ILC_{N_a}(U_1, U_2) \) be an information leakage channel via node \( N_a \), then the channel is significant, denoted by \( SILC_{N_a}(U_1, U_2) \), if

\[
\forall E_i \bullet U, E_i \leq \text{Incoming}(N_a \bullet U_2) \Rightarrow \text{ReturnInformation} \in \text{Attributes}(E_i)
\]

Figure 4[???] shows an example of leakage channels from an interaction \( I_A \) to an interaction \( I_B \). Channels labeled “S” represent significant channels; channels labeled “S” represent secondary channels.

When we evaluate \( \text{Conf}(I_A \rightarrow I_B) \), we are primarily concerned by the significant information leakage channels. Based on this consideration, we propose for \( \text{Conf}(I_A \rightarrow I_B) \) the following definition:

\[
\text{Conf}(I_A \rightarrow I_B) = \frac{1}{\text{NOSILC}(I_A, I_B)}
\]

Where “NOSILC \((I_A, I_B)\)” represents the “Number Of Significant Information
Leakage Channels” from $I_A$ to $I_B$. Based on this formula, the confidentiality of $I_A$ with respect to $I_B$ is equal to 1 only if no significant information leakage channels exists from $I_A$ to $I_B$. The confidentiality decreases as the number of significant channels increases. For instance, in Figure 4, $Conf(I_A \rightarrow I_B) = 0.5$.

![Figure 4. Information leakage channels from $I_A$ to $I_B$](image)

5 Case Study on Race Condition

5.1 Designs

Race conditions are mostly known for robustness problems they create in concurrent programs [Robert et al., 1992]. Some race conditions are, however, sources of serious security bugs [Bishop et al., 1996]. A security attack based on race conditions exploits a window of vulnerability between events executions in order to force the system to behave in unanticipated ways. As an example, let us consider the record keeping system introduced in Section 3. Assuming that the medical records are stored in their encrypted form, an intruder who is able to break into the system using, for instance, a password-cracking tool would also have to decrypt the record before being able to do any kind of useful job with it. Let’s assume that an intruder can access the (file or database) server storing the records but not the keys server. Hence, the intruder is able to access the encrypted records but cannot decrypt them. The intruder, however, can still run an attack tool concurrently with the record-keeping tool that repeatedly tries to access the records. Figure 5 shows a sequence diagram describing such interaction from the Intruder's perspective. Figure 1 describes the interaction from the doctor’s perspective. If there is no concurrency control mechanism in place, two
possible schedules among others that may occur are the following:

1. $\{\text{decrypt()}, \text{view()}, \text{read()}, \text{encrypt}()\}$
2. $\{\text{decrypt()}, \text{read()}, \text{encrypt()}, \text{view}()\}$

Schedule (2) is acceptable but not schedule (1). In schedule (1), the attacker takes advantage of a window of vulnerability to access the record in cleartext. An alternative design, which is more secure, may introduce a concurrency control mechanism that will prevent unauthorized access to the cleartext file. Figure 6 and Figure 7 show the sequence diagrams describing the doctor and intruder’s perspectives under this scenario.

5.2 Security Measures

Actually standard concurrency control mechanisms are not always efficient to handle such kinds of problems because they are primarily geared towards consistency issues. For instance, security anomalies may occur in concurrent read, which is not considered anomalous by traditional locking mechanism. But this is beyond the scope of this paper; here we are interested primarily in measuring the security level of concurrent software design.

In this respect, we apply our basic metrics to both the non-secure and secure designs, and show that our metrics can capture the difference between them. For instance, based on the non-secure and secure designs presented in section 5.1, two sets of USIE models that are shown in Figures 8 and 9 can be derived. In Figure 8, two significant information leakage channels exist for the “Doctor-ReadRecord” interaction with respect to the “IntruderViewRecord” interaction. In Figure 9, since the secure design forces secure concurrency checking, the in-
Fig. 6. Intruder Read record under Concurrent Control

Fig. 7. Doctor Read record under Concurrent Control
truder” “View” action of the secure system cannot proceed further while a record is being accessed. Hence, no channels exist for the “DoctorReadRecord” interaction with respect to the “IntruderViewRecord” interaction.

Based on section 4, we may derive confidentiality security metrics for both the non-secure and the secure system designs. The metrics are shown in Table 2.

**Table 2. Security Metrics**

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Non-Secure System</th>
<th>Secure System</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Conf(I_{Doctor,Read,Record} \rightarrow I_{Intruder,View,Record})$</td>
<td>0.67</td>
<td>1</td>
</tr>
</tbody>
</table>

As we discussed in section 5.1, the secure system design improves system security by preventing unacceptable schedules such as $decrypt()$, $view()$, $read()$, $encrypt()$ from happening. Our security metrics reflect the security improvement by showing that the legal user’s (e.g., a doctor) interaction with the system preserves confidentiality with respect to an intruder’s interaction during concurrent executions.

6 Conclusions and Future Work

Software security has been more and more concerned in recent years as software attacks grow dramatically. It is normally difficult to measure software security
attributes, particularly in the early stage of software development. Current practice on software security assurance still requires high expertise. In this paper, we proposed a systematic approach for software security analysis in an architectural level. More specifically, we related system security properties (confidentiality) with UML-based system dynamic behavior specifications - UML sequence diagrams; and we developed a USIE model and a formal framework to perform the quantitative security analysis. Then we derived confidentiality metrics for quantitatively analyzing software security attributes during the earlier stage of software development.

Our objective in the future will be to combine the basic metrics suggested in order to define more complex metrics and associated security analysis framework. Future work also consists of extending the scope of our metric suite by including security properties such as availability, and UML structural models (e.g., class, component, and deployment diagrams). We will also conduct empirical theoretical as well as empirical validation of these metrics.

References


