A Generic Model for Assessing Multilevel Security-Critical Object-Oriented Programs

Bandar M. Alshammari
Department of Information Technology
School of Computer and Information Sciences
Aljouf University, Saudi Arabia

***Abstract*—The most promising approach for developing secure systems is the one which allows software developers to assess and compare the relative security of their programs based on their designs. Thereby, software metrics provide an easy approach for evaluating the security of certain object-oriented designs. They can also measure the impact on security that caused by modifications to existing programs. However, most studies in this area focus on a binary classification of data, either is classified or unclassified. In fact, there are other models with other classifications of data, for instance, the common model used by Defense departments that classifies data into four security levels. However, these various classifications have received little attention in terms of measuring their effect. This paper introduces a model for measuring information flow of security-critical data within a certain object-oriented program with multilevel classification of its security-critical data. It defines a set of object-oriented security metrics which are capable of assessing the security of a given program's design from the point of view of potential information flow. These metrics can be used to compare the security of programs or assess the effect of program modifications on security. Specifically, this paper proposes a generic model that consists of several security metrics to measure the relative security of object-oriented designs with respect to design quality properties of accessibility, cohesion, coupling, and design size.**

***Keywords*—Multilevel Security Models; Object-Orientation; Security Metrics; Security Matrix; Unified Model Language**

I. Introduction

Recently, security has become one of the most crucial aspects of systems' development due to the increasing number of risks and breaches which systems are facing. Therefore, several studies have suggested different approaches for reducing these risks and preventing the existence of vulnerabilities. One example defines a number of safe coding practices which a programmer needs to follow in order to have a more secure product [1] [2]. Another approach relies on previously defined security vulnerabilities. This approach statically analyses programs' codes to check if the they contain any of the defined vulnerabilities [3] [4] [5] [6].

Another approach which can be easily applied is to define security metrics that can quantify the security level of certain programs. [7]. Security metrics can be an effective tool in helping software programers to identify level of risks in a given program. They also provide programmers with guidance of how to raise awareness within the organisation. Security metrics that are based on the designs of programs can provide systems' developers with an early guidance for discovering vulnerabilities and guiding them on following certain secure corrective steps.

Most of the work on security metrics focus on classifying data into two levels, either security-critical or not. However, this is not the case in real systems. Real security-critical systems have multiple level of classifications for their sensitive data. Therefore, this paper defines a generic model that takes into consideration this aspect. It studies the impact on security of four of the most common software design properties, which are used in order to enhance the software quality. These properties consist of Data Encapsulation, Cohesion, Coupling, and Design Size [8]. In this work, a number of security metrics for object-oriented designs are defined with respect to those quality design properties. These metrics are capable of quantifying the security level of certain programs with regard to the potential flow of security-critical information based on the security design principles of "reducing the size of the attack surface" [9] [10] [11] and " least privilege" [12] [2]. Such metrics will also assist software developers in assessing the impact on security of any modifications occurred to their programs.

The remainder of this paper is organised as follows. Section 2 shows the related work and how this paper defines a model that is distinct from previous work. Section 3 shows the research methodology that is used to define this model. Section 4 explains the assumptions which need to be considered when applying this model. Section 5 studies the relevant security design principles that this model considers. Section 6 illustrates the model defined by this paper and the defined security design metrics. Section 7 illustrates a case study of how to apply this model to a real system based on its design, and it also shows the results of the security metrics. Finally, Section 8 concludes the paper and explains future extensions of this work.

II. Related Work

Many researchers have proposed different approaches which help in developing more secure programs. One of these approaches defines a list of principles that can be used as guidance for developing secure systems [12] [2]. Other approaches have developed general coding principles that aim to discover code vulnerabilities [1] [13]. However, these approaches cannot discover security risks at an early stage and are not capable of quantifying security of given programs.

The approach which sounds promising with this regard is to define security metrics at an early stage of development. Some of the work in this area has proposed metrics for measuring software security of object-oriented programs such as the work of [14] [15] [16] [17] [18] [19] [20]. The main objective of these metrics is to assess programs' security at different stages

of its development life cycle in order to give a prediction of the existing vulnerabilities in the program.

Multilevel security, on the other hand, is a crucial aspect to information systems. In fact, many admit that it is a must for any system to have various levels of security. In other words, different security-critical components of any system have to be classified into different levels of criticality in terms of information security. There exists several security models for classifying information with regard to their security sensitivity. For instance, the US model defines four level of security classifications (Top Secret, Secret, Confidential, and unclassified)[1]. Another example is the British which uses a model of six levels of security classifications (Top Secret, Secret, Confidential, Restricted, Protected, and unclassified)[2].

Unfortunately, most of studies on software security metrics focus on defining metrics by classifying system's components to classified and not classified ones. Furthermore, there exists some studies that study multilevel security such as the work of Kotenko and Doynikova [21] which defines security metrics for various levels of the system. However, this work doesn't consider the information flow of various security values of security-critical data. Instead, it focuses on dividing different types of metrics for different parts of the system [21].

None of existing projects have developed metrics for program security based on its design artifacts that takes into consideration the different security level of the system's components. This paper defines a generic model that is applicable to any security-critical object-oriented program in order to assess its security with respect to the potential flow of security-critical data.

III. Research Methodology

The approach of Ourston and Monney's [22] states that any project's implementation should be conducted in two steps. The first one is the analytical part which is about defining the theory of the project. The second one is called the empirical part which is conducted to prove the analytical part. This paper follows this approach, and hence it defines the first part that is called *what to measure* and the second part which is called *how to measure*.

A. What to measure

This paper aims to define a model that is capable of assessing the security of object-oriented programs with multilevel of data security classifications. It concentrates on specific object-oriented design properties which have the most effect on security of programs. The proposed metrics need to measure the security of programs by identifying the potential information flow of security-critical data.

B. How to measure

This paper defines a number of security metrics for object-oriented programs with multilevel security classifications of their components. The developed metrics have to adhere to certain security design principles for developing secure programs. These metrics can be applied to any object-oriented program as long as its class diagram is provided. Such metrics will assist software developers in assessing the security level of their programs from an early stage of development based on the design artifacts of the programs.

Table I. Model Terminology

Name	Description
Security-Critical Attribute	An attribute which holds confidential data.
Security-Critical Method	A function which accesses or interacts with security-critical attributes.
Security-Critical Class	A class which has at least one security-critical attribute or one security-critical method.

IV. Model Assumptions

The model defined in this paper aims to introduce a set of security metrics for programs with multilevel classifications of data secrecy based on their designs. These metrics measure the potential flow of security-critical data of a given object-oriented design. Each metric is defined in relation with a specific security design principle that needs to adhere to in order to achieve a secure program. The model focuses on defining such metrics with respect to four object-oriented properties (i.e., data encapsulation, cohesion, coupling, and design size) [8]. These metrics are a comparative measurement which means that results of these metrics can be used to compare the relative security of various object-oriented programs with regard to these four design properties.

The metrics have been designed so that their results are within the range 0 to 1. As the value of the metric decreases, the more secure a program is. Similarly, as the value of the metric increases, the less secure a program is. This means that lowers values of these metrics, the more they adhere to their related security design principles. And higher values mean that they less adhere to their relevant security design principles.

One approach which can be used to apply security metrics based on the program's design is developed by Alshammari et al. [19] [20]. This approach depends on accurately providing annotated classes of security-critical data for a given object-oriented program using UMLsec [23] and SPARKS [24] annotations. In this approach, UMLsec annotations are used to annotate attributes, methods, and classes with *"'«secrecy»"'* if they contain or interact with security-critical data. SPARKS annotations, moreover, are used to show the interactions of methods with security-critical attributes, methods, and classes.

Instead, this model develops a new approach which helps in applying security design metrics defined here. This approach relies on the program's developers to provide an accurately designed two dimensional matrix. This matrix shows the security level of all interactions with security-critical classes that are caused by methods or classes in a given design.

To illustrate this approach, a class is extracted from a certain program called Student Information System is shown in Figure 1 and the matrix related to this class is shown in Figure 2. Suppose the security model used in this context is $\{0,1,2,3\}$; with 0 being not a security-critical attribute, and 3 is the most security-critical attribute. In this class,

[1] http://www.state.gov/m/ds/clearances/c10977.htm#5

[2] https://www.gov.uk/government/publications/government-security-classifications

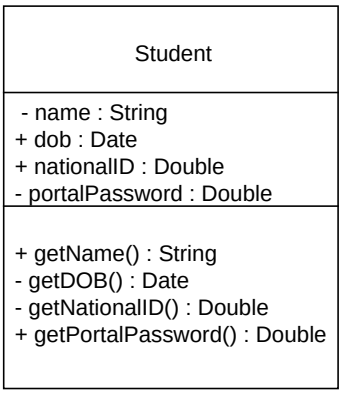

Figure 1. Student UML Class Diagram

		Attributes			
		- name	+ dob	+ nationalID	- portalPassword
Methods	+ getName()	0	0	0	0
	- getDOB()	0	1	0	0
	- getNationalID()	0	0	2	0
	+ getPortalPassword()	0	0	0	3

Figure 2. Student Class Security Matrix

suppose that `name` is not a security-critical attribute (hence, its security value is 0), and is accessed by method `getName`. `DOB` is supposed to be a security-critical attribute with security value 1, and is only accessed by method `getDOB`. Also suppose that `nationalID` is another security-critical attribute with security value 2, and is only accessed by method `getNationalID`. Suppose that `portalPassword` is also a security-critical attribute with security value 3, and is only accessed by method `getPortalPassword`. Then, the security matrix associated with the class diagram from Figure 1can be seen in Figure 2.

V. Relevant Security Design Principles

This section illustrates those security design principles that are relevant to the security design metrics defined by this model. In this paper, two design principles have been chosen to be studied when developing this model (i.e., Least Privilege and Reduce Attack surface). These were chosen based on previous work [19] which has shown that these two principles can have the most effect on developing secure systems, and hence they need to be intensively considered when defining security metrics. Therefore, these metrics are constructed with respect to these two principles in order to measure the security of designs from the perspective of potential information flow.

A. Least Privilege

This principle is described as allowing programs and users to complete a certain job with the least privileges [12] [2]. Its main advantages consist of minimizing interactions between privileged components in a given system, and hence minimizing loss in case of a successful attack [12] [2].

B. Reduce Attack surface

This principle also aims to make programs more secure by decreasing the number of components that can be reached from outside the system [9] [25]. There are several approaches for reducing such components. A common approach is described by Howard [9] suggest reducing the amount of running code by turning off any unnecessary features of the system. Another approach is to minimise the number of entry points in the system that can be accessed by untrusted users [9].

VI. Multilevel Security Assessment Model

Due to the importance of having multiple levels of data security for any systems, this paper defines a model that is capable of defining a set of security metrics that meet this purpose. This model considers four of the most common software design properties to define these metrics. these properties include the data encapsulation, cohesion, coupling, and design size of systems [8]. These metrics depend on the security levels defined by system's security analysts. Furthermore, the model develops a security matrix that can be used easily to extract the required information of these security design metrics.

These security design metrics are defined to be generic, which means that they can be applied to any program regardless of the type of security model it uses. For the purpose of illustration, let's address the set of values of the security model used in this context as $V = \{0, \ldots, v\}$, where an attribute labelled as 0 means it is not security-critical and an attribute labelled as v (the maximum number in the model) means it is the most security-critical attribute in that program. (let the magnitude operator $|S|$ returns the size of a given set S.) These metrics are defined as follows.

A. Accessibility of Security-Critical Attributes Metric (ASCAM)

The design property of data encapsulation in object-oriented designs is concerned with having restrictions on data accessibility inside a given program [8]. In terms of security, it has been shown that this property has a major effect on program's security including the work of Maruyama et al. [13] and Alshammari et al. [19]. These studies have shown that creating security-critical attributes less accessible from outside their classes make programs more secure. This will eventually satisfy the specifications of the security principle of "*reducing the attack surface size*".

This metric aims to measure the proportion of security-critical attributes which are accessible from outside their class in a given design for an object-oriented program with respect to their security levels. Therefore, this metric is defined as; "*The ratio of the number of criticality of non-private security-critical attributes in a given design to the total number of criticality of security-critical attributes in that design*".

Consider the set of attributes in a design D as A_i, $i \in \{1, \ldots, a\}$, set of security-critical attributes in D as SCA_j, $j \in \{1, \ldots, sca\}$ such that $SCA \subseteq A$, and set of non-private

(accessible) security-critical attributes in D as $ASCA_k$, $k \in \{1,\ldots,asca\}$ such that $ASCA \subseteq SCA \subseteq A$. Then, *ASCAM* is expressed as:

$$ASCAM(D) = \frac{\sum_{k=1}^{asca}(ASCA_k \times v)}{|SCA| \times \max(V)} \quad (1)$$

B. Accessibility of Security-Critical Methods Metric (ASCMM)

As it's shown previously that the design property of data encapsulation in object-oriented designs is concerned with having restrictions on data accessibility inside a given program [8]. Another possible way of accessing data is through methods which have access to attributes. Similar to direct accessibility of security-critical attributes, accessing security-critical attributes indirectly through methods which interact with them can have the same security impact. Therefore, it is recommended to have less accessibility to methods which interact with security-critical attributes in order to satisfy the security design principle of "*reducing the attack surface size*" [19].

This metric aims to measure the proportion of methods which have access or interaction with security-critical attributes and are accessible from outside their class in a given object-oriented design with respect to their security levels. Therefore, this metric is defined as; "*The ratio of the number of criticality of non-private security-critical methods in a given design to the total number of criticality of security-critical methods in that design*".

Consider the set of methods in a design D as M_i, $i \in \{1,\ldots,m\}$, set of security-critical methods in D as SCM_j, $j \in \{1,\ldots,scm\}$ such that $SCM \subseteq M$, and set of non-private (accessible) security-critical methods in D as $ASCM_k$, $k \in \{1,\ldots,ascm\}$ such that $ASCM \subseteq SCM \subseteq M$ (Given a set S, let the magnitude operator $|S|$ returns the size of the set.) Then, *ASCMM* is expressed as:

$$ASCMM(D) = \frac{\sum_{k=1}^{ascm}(ASCM_k \times v)}{|SCM| \times \max(V)} \quad (2)$$

C. Cohesiveness of Security-Critical Methods Metric (CSCMM)

In terms of object-oriented design properties, it is recommended to have a cohesive design which increases the interactions of attributes with methods inside their class. In regard to security, it is recommended to decrease cohesiveness of interactions between security-critical attributes and methods inside their classes in order to have more secure programs [19]. Cohesiveness is about privileges over security-critical attributes and the fewer number of such interactions, the more this adheres to the security design principle of least privilege [19].

The main aim of this metric is to measures the degree of potential flow of security-critical data caused by interactions between security-critical attributes and methods in a given object-oriented design taking into consideration the security levels of these attributes. Therefore, this metric is defined as; "*The ratio of the number of methods which interact with security-critical attributes for every class to the maximum number of methods which could interact with attributes for every class in a specific program's design*".

To calculate this metric, the number of interactions with every security-critical attribute is multiplied by its security level for every class. Then, the sum of these values are divided by the total number of possible ways of interactions with attributes in that class. This can be calculated by multiplying the total number of methods in that class by the total number of attributes in the same class. Then, it is multiplied by the maximum value of the security model used in context. The values of all of the classes' cohesiveness is summed, and then divided by the number of classes in the design in order to take the design's average cohesiveness.

Consider the set of classes in a design D as C_i, $i \in \{1,\ldots,c\}$, set of methods in the same design as M_j, $j \in \{1,\ldots,m\}$, set of attributes in D design as A_k, $k \in \{1,\ldots,a\}$, and set of security-critical attributes in D as SCA_l, $l \in \{1,\ldots,sca\}$ such that $SCA \subseteq A$. Let $\alpha(SCA_l)$ be the number of methods which access security-critical attribute SCA_l. Then, *CSCMM* is expressed as:

$$CSCMM(D) = \frac{\sum_{i=1}^{c} \frac{\sum_{l=1}^{sca} \alpha(SCA_l) \times v}{|M| \times |SCA| \times \max(V)}}{|C|} \quad (3)$$

D. Coupling of Security-Critical Classes Metric (CSCCM)

Coupling is one of the most common object-oriented design properties that have been widely examined in several studies. This property is defined by the number of interactions an object has with others inside the program [26]. Many studies have shown that it's recommended for object-oriented programs to be loosely-coupled between its components in order to be more reusable, understandable, and extensible [8] [26].

With regard to information security, it has been shown that there is a high correlation between coupling and insecurity of programs. The study of Liu and Traore [27] has shown that successful attacks in many cases are caused by highly-coupled objects. Furthermore, the studies of Alshammari et al. [19] [20] have shown that loosely-coupled programs can decrease the potential flow of security-critical information, and thus creating more secure programs. This satisfies the security design principle of "*least privilege*" [12].

Therefore, this model focuses on the effect of coupling on programs security, and it defines a security metric that fits the multilevel security model described here. This coupling metric aims to measure the potential occurrence of links between security-critical attributes and classes in a given design taking into consideration the security level of each link. Therefore, this metric is defined as; "*The ratio of criticality of the number of associations of all classes with security-critical attributes to the maximum number of associations which could occur with security-critical attributes for every class in a specific program's design*".

Consider the set of classes in a design D as C_i, $i \in \{1,\ldots,c\}$, set of attributes in the design D as A_k, $k \in \{1,\ldots,a\}$, and set of security-critical attributes in D as SCA_l, $l \in \{1,\ldots,sca\}$ such that $SCA \subseteq A$. Let $\beta(SCA_l)$ be the number of classes which are associated with security-critical attribute SCA_l. Then, *CSCCM* is expressed as:

$$CSCCM(D) = \frac{\sum_{l=1}^{sca} \beta\left(SCA_l\right) \times v}{|C-1| \times |SCA| \times \max(V)} \quad (4)$$

E. Security-Critical Design Size Metric (SCDSM)

Design size is one of the object-oriented properties that must be considered from an early stage of a program's development life-cycle since it has a major impact on its reusability and functionality [8]. Due to this importance, Bansiya and Davis [8] defined a metric that is related to this property. This metric is called Design Size in Classes (DSC) which measures of the number of classes in a specific design [8].

In terms of programs' security, the effect of the design size property has been studied in a number of studies. This include the work of Chowdhury et al. [7] which defined a metric to measure the ratio of critical code segments in a particular program's code. The work of Alshammari et al. [19] [20] have also studied this property. They show that it is desirable to have a small design size of security-critical classes in order to adhere to the security design principle of "*reducing the attack surface size*" [20].

The model defined in this paper considers the importance of the design size property on security, and hence it defines a metric for this purpose. The main goal of this metric is to measure the proportion of security-critical classes taking into consideration their security levels. Therefore, this metric is defined as; "*The ratio of criticality of the number of security-critical classes to the total number of security-critical classes in a specific program's design*".

Consider the set of classes in a design D as C_i, $i \in \{1,\ldots,c\}$, set of attributes in the design D as A_k, $k \in \{1,\ldots,a\}$, set of security-critical attributes in D as SCA_l, $l \in \{1,\ldots,sca\}$ such that $SCA \subseteq A$, and set of classes with security-critical classes defined in them in the design D as SCC_l, $l \in \{1,\ldots,scc\}$ such that $SCC \subseteq C$. Then, $SCDS$ is defined as:

$$SCDSM\left(D\right) = \frac{\sum_{l=1}^{scc} \left(SCC_l \times v\right)}{|C| \times \max(v)} \quad (5)$$

VII. Model Case Study

This section shows how to apply the multilevel security design metrics defined here for a specific banking system. To measure the security of a given program with regard to these metrics, a complete UML class diagram needs to be provided by the system's developers. It is also required to provide the security matrix defined by this model as explained in the previous section. This matrix shows the interactions of every attribute with other methods and classes, and the security level of each interaction.

A. Banking System UML Class Diagram

The UML class diagram shown in Figure 3 is part of a banking system extracted to show how to measure the security of a certain program with regard to the model defined by this paper. This diagram consists of classes that are responsible for storing information about clients of a certain bank. The class diagram consists of five classes; Customers, Credit Cards, Debit Cards, Loans, and Cheques. Each of these classes is responsible for a specific task, and it contains a number of attributes and methods to surf this purpose.

For instance, the Customers class is responsible for storing personal information about clients such as their name, date of birth, national ID, and ebanking password. It also defines a number of methods that are related to retrieve specific data in response to requests from either inside or outside the class. The Loans class stores information about the customer's loans such as the loan amount. It has also functions that are responsible for returning this information once requested. The Cheques class provides information about the customer's cheques, and defines functions that access such information. The Credit Card and Debit Card classes define attributes that store cards' numbers and their passwords. They also contain methods that return and update this information.

B. Banking System Matrix

This section explains how to construct the security matrix associated with the banking system illustrated in this paper. This matrix shows the interactions between attributes and methods in the entire system. It also shows those classes which directly access certain attributes from outside their classes. This security matrix has to include a security value for every interaction. This value is the same as the security level of the attribute which the interaction has occurred with.

For the purpose of this case study, let's suppose that the security model used by in this context is $\{0,1,2,3,4\}$. Attributes which their security value is 0 indicate that they are not security-critical. Other than that, attributes are security-critical and their criticality level depend on their security value. This means as this value increases, the security criticality of a certain attribute increases. For instance, attributes assigned the security value of 1 indicate that they have the least security-critical data, while attributes which their security value is 4 show the highest security-critical data. Moreover, the matrix shows the accessibility of every attribute, method, and class in the system. This information is extracted from access modifiers shown in the UML class diagram of the system.

Table II. Banking System's Attributes Security Levels

Class	Attribute	Security Level
Customers	name	0
Customers	dob	2
Customers	nationalID	3
Customers	portalPassword	4
Credit Cards	cardNo	4
Credit Cards	cardPassword	4
Debit Cards	cardNo	3
Debit Cards	cardPassword	4
Cheques	chequeNo	1
Cheques	chequeAmount	2
Loans	interestRate	0
Loans	loanAmount	1

In the UML class diagram in Figure 3, there are five classes and each one of them has a number of attributes with different security levels. Let's suppose that the system's analyst has provided Table II which shows the security levels for all attributes in the system. These values are used in order to construct the security matrix of the banking system as shown in Figure 4 (which needs to be constructed by the system's analyst).

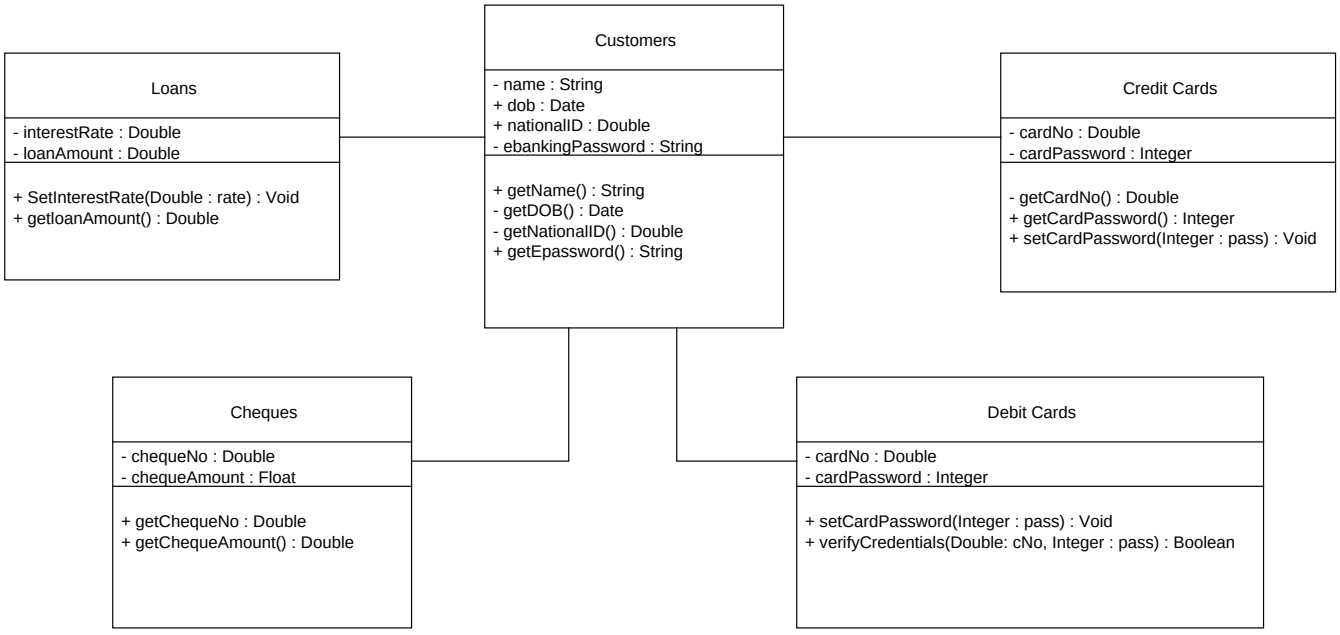

Figure 3. Banking System UML Class Diagram

C. Model Security Metrics Results

The main goal of this section is to show how to calculate the security metrics defined by this model based on the security matrix shown in Figure 4. The purpose and result of each metric are explained here in order to identify their impact on security of the banking system examined here.

The data encapsulation metrics measure the ratio of accessible security-critical attributes and methods in a given class with regard to the total number of security-critical attributes and methods in that class taking into consideration their security values. This means that when calculating these metrics, the security level of each attribute and method has been considered. These values can be easily obtained from the security matrix shown in Figure 4. Therefore, the ASCAM is computed by counting the number of non-private security-critical attributes multiplied by each of their security values. In Design 3, there are eight security-critical attributes out of ten attributes defined in the banking system UML class diagram. Out of these eight security-critical attributes, there are two attributes which are accessible from outside their classes (i.e., `Customers.dob` and `Customers.nationalID`). This information is shown in the security matrix since these two attributes have the (+) symbol in front of them which is an access modifier for public attributes. The security value associated with `Customers.dob` is 2 and with `Customers.nationalID` is 3. Therefore,, the total security value of accessible security-critical attributes in this case is 5. The the total security value of possible accessibility of security-critical attributes is counted by multiplying the total number of security-critical attributes in the design (i.e., 8) to the maximum value of criticality in the security model used in context (i.e., 4) which is 32. Hence, the ASCAM is computed by dividing 5 to 32, which is 0.16.

With regard to the ASCMM which measures the accessibility of security-critical methods in a certain design, it is computed by counting the number of non-private security-critical methods multiplied by their security values for each of them to the total number of possible accessibility of security-critical methods in the same design multiplied by the maximum security value defined in the model. The security value of a certain method is the same as the security value of the attribute it interacts with, which is shown in the security matrix. If there is a method that interacts with more than one attribute of different security levels, then the security level of this method is the same as the highest attribute that it interacts with. The security matrix in Figure 4 shows that there are eleven security-critical methods which eight of them are non-private security-critical methods. Out of these eight methods, there are two methods which their security value is 1; `Cheques.getChequeNo` and `Loans.getLoanAmount`. There is one method which its security value is 2 (i.e., `Cheques.getChequeAmount`). There is one method which its security value is 3; `Customers.getNationalID`. There are five methods which their security value is 4; `Customers.getPortalPassword`, two methods in Credit Cards class `CreditCards.getCardPassword` and `CreditCards.setCardPassword`, and two methods in Debit Cards class`DebitCards.setCardPassword` and `DebitCards.verifyCredentials`. The total possible security value of accessible security-critical methods is counted by multiplying the total number of security-critical methods (i.e., 11) to the maximum value of criticality in the security model used in context (i.e., 4), which is 44. Hence, the ASCMM is computed by dividing 27 to 44, which is 0.61.

In terms of the metric which measures the cohesiveness of security-critical methods in a given design (i.e., CSCMM), it computes the interactions of methods in a given program with security-critical attributes in the same program taking

		Attributes											
		- Customers. name (0)	+ Customers. dob (2)	+ Customers. nationalID (3)	- Customers. portalPassword (4)	- CreditCards. cardNo (4)	- CreditCards. cardPassword (4)	- DebitCards. cardNo (3)	- DebitCards. cardPassword (4)	- Cheques. chequeNo (1)	- Cheques. chequeAmount (2)	- Loans. interestRate (0)	- Loans. loanAmount (1)
Classes	+ Customers	0	0	3	0	0	0	0	0	0	0	0	0
	+ Credit Cards	0	2	3	0	0	0	0	0	0	0	0	0
	+ Debit Cards	0	2	3	0	0	0	0	0	0	0	0	0
	+ Cheques	0	0	3	0	0	0	0	0	0	0	0	0
	+ Loans	0	0	3	0	0	0	0	0	0	0	0	0
Methods	+ Customer.getName	0	0	0	0	0	0	0	0	0	0	0	0
	- Customer.getDOB	0	2	0	0	0	0	0	0	0	0	0	0
	- Customer.getNationalID	0	0	3	0	0	0	0	0	0	0	0	0
	+ Customer.getPortalPassword	0	0	0	4	0	0	0	0	0	0	0	0
	- CreditCards.getCardNo	0	0	0	0	4	0	0	0	0	0	0	0
	+ CreditCards.getCardPassword	0	0	0	0	0	4	0	0	0	0	0	0
	+ CreditCards.setCardPassword	0	0	0	4	0	4	0	0	0	0	0	0
	+ DebitCards.setCardPassword	0	0	0	0	0	0	0	4	0	0	0	0
	+ DebitCards.verifyCredentials	0	0	0	4	0	0	3	4	0	0	0	0
	+ Cheques.getChequeNo	0	0	0	0	0	0	0	0	1	0	0	0
	+ Cheques.getChequeAmount	0	0	0	0	0	0	0	0	0	2	0	0
	+ Loans.setInterestRate	0	0	0	0	0	0	0	0	0	0	0	0
	+ Loans. getLoanAmount	0	0	0	0	0	0	0	0	0	0	0	1

Figure 4. Banking System Security Matrix

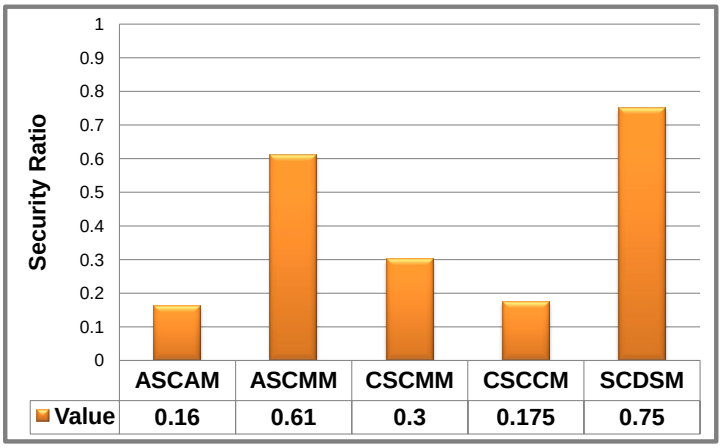

Figure 5. Banking System Results Chart

into consideration the security values of these attributes. The main aim of this metric is to have a low value of cohesiveness as possible in order to meet the requirements of the security design principle of least privilege. From the design used in this context, there are 11 security-critical methods interacting with different security-critical attributes of different security levels. The criticality of interactions is calculated by summing the number of interactions in every class, and then multiplying each interaction by its security value. This result is divided by the number of methods in the same class multiplied by the number of security-critical attributes and by the maximum security value in the model. In the Customers class case, the criticality of interactions of methods in this class with its security-critical attributes is 9. This is extracted from the security matrix in Figure 4 since the criticality of interactions from method `Customers.getDOB` is 2, the criticality of interactions from method `Customers.getNationalID` is 3, and the criticality of interactions from method `Customers.getPortalPassword` is 4. Moreover, the total number of possible interactions from inside the Customers class with its security-critical attributes is 48. The same process has to be done for all classes f the program, then the sum is divided by the total number of classes in that program. Hence, the cohesiveness of the Credit Cards class is 0.33, Debit Cards class is 0.6875, Cheques class is 0.1875, and Loans class is 0.125. Finally, the CSCMM is computed by dividing the sum of the cohesiveness of all classes (i.e., 1.5175) to the number of classes in the design (i.e., 5). which gives 0.3035 and this is the result of the CSCMM for the design shown here.

With regard to the security metric of coupling, it aims

to reduce the number of associations of classes with other security-critical inside a given program. The lower number of such associations, the more secure a program is in terms of the least privilege security design principle. This metric is computed by counting the number of outsider classes which interact with security-critical attributes, and each interaction is multiplied by the security level of these attribute. The security level of each interaction is determined by the security level of the attribute a class is interacting with. Then, this number is divided by the total number of possible associations with security-critical attributes. In Figure 4, there are five security-critical classes which have ten security-critical attributes of different security levels. The number of classes that interact with the security-critical attribute `Customers.dob` is two, and it's security level is two. Hence, the criticality of these interactions when those two numbers are multiplied by each other is four. The number of classes which interact with security-critical attribute `Customers.nationalID` is four, and it's security level is three. Thus, the criticality of these interactions is twelve. The number of classes which interact with security-critical attribute `Customers.portalPassword` is three, and it's security level is four. Thus, the criticality of these interactions is twelve. Finally, the criticality of the total number of links with security-critical classes is 28. The criticality of the possible number of classes that may interact with security-critical classes in this design is 160 (number of classes less by one multiplied by number of security-critical attributes and then multiplied by the maximum security level in this case). Therefore, the CSCCM is computed by dividing 28 over 160, which is 0.175.

The design size metric aims to measure the proportion of security-critical classes in a given program. A lower value of this metric is desirable in order to meet the requirements of the security design principle of reducing the attack surface size. To compute this metric, each security-critical class is multiplied by its security level, and the sum of this is divided over the total number of classes multiplied by the maximum security value in a certain design. The security value of a certain class is the same as the highest security value of the attributes it includes. In the design shown in Figure 4, there are five classes that all of them have security-critical attributes. There are three security-critical classes which their security value is 4 ; Customers, Credit Cards, and Debit Cards. There are one security-critical class which its security value is 2 (i.e., Cheques) and one class which its security value is 1 which is Loans. The sum of these security values is 15. The total security values of all classes in a certain design is computed by multiplying the number of classes (5 in this context) by the maximum security value in a given design (4 in this context). Hence, The SCDSM is computed by dividing 15 over 20 which is 0.75.

D. Results Discussion

Figure 5 shows the results of the five security metrics defined and studied by this paper. This part examines these results in order to show which ones contribute to a more secure design. As shown in the previous section, the results of these metrics vary and this effects the security of the program in reference to the relevant security design principle. For instance, the result of ASCAM is the lowest one among the results of the five metrics defined by this model. Since ASCAM measures the information flow with regard to the accessibility of security-critical attributes in order to meet the requirements of the security design principle of reducing the attack surface size, then this result indicates that it is the best metric which makes the program secure in this regard.

On the other hand, SCDSM measures the information flow of security-critical data in a given object-oriented design with regard to the size of the security-critical classes in that design. Therefore, it is desirable to have as few as possible of security-critical classes to satisfy the security design principle of reducing the attack surface size. However, the result of SCDSM in the case study shown here is the highest one among all of the five metrics, and hence the examined program is the least secure with this regard.

In terms of the metrics which measures the information flow in a given design in order to meet the requirements of the security design principle of granting least privilege, the results of these metrics show that CSCCM is lower than CSCMM. This indicates that CSCCM is more secure than CSCMM in terms of the least privilege security design principle in this case.

VIII. Conclusion

This work has developed a generic model for quantifying security of multilevel object-oriented designs. It concentrates on defining five security design metrics with regard to four different quality design properties for object-oriented programs. These metrics aim to assess the potential information flow of security-critical data within a given program with regard to security design principles of least privilege and attack surface size. This model differs from previous ones as it takes into consideration the criticality of the system's components which is defined by their security levels. Further contribution of this work includes the definition of a security matrix that is developed in order to make it easy for software designers to extract the results of these metrics from the system's UML class diagram. At the end of this paper, a case study of a typical object-oriented program is examined. It has shown in this case study how to apply these metrics, construct the security matrix of that design, and compare the results of these metrics to identify which metrics makes programs more secure than others.

Future extensions of this work include studying the effect of other software quality properties such as polymorphism and inheritance on the security of object-oriented programs. Future work also includes studying these metrics on a lower level of modularity such as at the code-level to identify the impact of these metrics and others on multilevel security-critical programs.

Acknowledgment

I wish to thank the anonymous reviewers for their helpful suggestions. The author gratefully acknowledges Aljouf University for funding this research through grant 35/344.

References

[1] M. Howard and D. LeBlanc, *Writing Secure Code*. Redmond, Wash.: Microsoft Press, 2002.

[2] G. McGraw, *Software Security: Building Security In*. Upper Saddle River, NJ: Addison-Wesley, 2006.

[3] V. B. Livshits and M. S. Lam, "Finding security vulnerabilities in Java applications with static analysis," in *SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium*, (Berkeley, CA, USA), pp. 18–18, USENIX Association, 2005.

[4] S. F. Smith and M. Thober, "Improving usability of information flow security in Java," in *Proceedings of the 2007 workshop on Programming languages and analysis for security*, PLAS '07, (New York, NY, USA), pp. 11–20, ACM, 2007.

[5] G. Smith, *Principles of Secure Information Flow Analysis*, vol. 27, pp. 291–307. Springer, 2007.

[6] Y. Liu and A. Milanova, "Static analysis for inference of explicit information flow," in *Proceedings of the 8th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, PASTE '08, (New York, NY, USA), pp. 50–56, ACM, 2008.

[7] I. Chowdhury, B. Chan, and M. Zulkernine, "Security metrics for source code structures," in *Proceedings of the Fourth International Workshop on Software Engineering for Secure Systems*, (Leipzig, Germany), pp. 57–64, ACM, 2008.

[8] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering*, vol. 28, pp. 4–17, 2002.

[9] M. Howard, "Attack surface: Mitigate security risks by minimizing the code you expose to untrusted users," *Microsoft MSDN Magazine*, vol. 11, 2004.

[10] P. K. Manadhata, K. M. C. Tan, R. A. Maxion, and J. M. Wing, "An approach to measuring a system's attack surface," Tech. Rep. CMU-CS-07-146, Carnegie Mellon University, Pittsburgh, PA, August 2007.

[11] P. Manadhata and J. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, p. 1, 2010.

[12] M. Bishop, *Computer Security: Art and Science*. Boston: Addison-Wesley, 2003.

[13] K. Maruyama, "Secure refactoring - improving the security level of existing code," in *Proceedings of the Second International Conference on Software and Data Technologies (ICSOFT 2007)* (J. Filipe, B. Shishkov, and M. Helfert, eds.), (Barcelona, Spain), pp. 222–229, 2007.

[14] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, pp. 476–493, 1994.

[15] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics: A Practical Guide*. Englewood Cliffs, NJ: PTR Prentice Hall, 1994.

[16] V. R. Basili, "Gqm approach has evolved to include models," *IEEE Software*, vol. 11, pp. 1–8, 1994.

[17] F. B. e Abreu, M. Goulão, and R. Esteves, "Toward the design quality evaluation of object-oriented software systems," in *Proceedings of the 5th International conference on Software Quality, Austin Texas*, 1995.

[18] B. Henderson-Sellers, *Object-oriented metrics: measures of complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[19] B. Alshammari, C. J. Fidge, and D. Corney, "Security metrics for object-oriented class designs," in *Proceedings of the Ninth International Conference on Quality Software (QSIC 2009)*, (Jeju, Korea), pp. 11–20, IEEE, 2009.

[20] B. Alshammari, C. J. Fidge, and D. Corney, "Security metrics for object-oriented designs," in *Proceedings of the Twenty-First Australian Software Engineering Conference (ASWEC 2010), Auckland, 6–9 April* (J. Noble and C. J. Fidge, eds.), (California, USA), pp. 55–64, IEEE Computer Society, 2010.

[21] Igor Kotenko and Elena Doynikova, "Comprehensive multilevel security risk assessment of distributed information systems," *International Journal of Computing*, vol. 12, no. 3, pp. 217–225, 2013.

[22] D. Ourston and R. J. Mooney, "Theory refinement combining analytical and empirical methods," *Artificial Intelligence*, vol. 66, no. 2, pp. 273–309, 1994.

[23] J. Jurjens, "Using UMLsec and goal trees for secure systems development," in *Proceedings of the 2002 ACM Symposium on Applied Computing Madrid*, (Madrid, Spain), ACM, 2002.

[24] J. Barnes, *High Integrity Software: The SPARK Approach to Safety and Security*. London, Great Britain: Addison-Wesley, 2003.

[25] P. K. Manadhata, *An Attack Surface Metric*. PhD thesis, Computer Science Department, Carnegie Mellon University, December 2008.

[26] S. Bennett, S. McRobb, and R. Farmer, *Object-Oriented Systems Analysis and Design Using UML*. Maidenhead: McGraw-Hill Higher Education, third ed., 2006.

[27] M. Y. Liu and I. Traore, "Empirical relation between coupling and attackability in software systems: a case study on DOS," in *Proceedings of the 2006 Workshop on Programming Languages and Analysis for Security, Ottawa*, (Ottawa, Ontario, Canada), pp. 57–64, ACM, 2006.