



An Empirical Validation of Integrity Risk Factor Metric: An Object-Oriented Design Perspective

Shalini Chandra*
MCA Department & SRMCEM
India

Raees Ahmad Khan
DIT & BBA University
India

Abstract— Software is said to be secure if it is less prone to attack. Secure design leads building secure software. It is also required to check software design to assure that the design is safe or less prone to attack. In this paper, we proposed a methodology to scrutinize class hierarchy against security. Two security metrics and integrity state transition model has been developed for quantitative assessment of Integrity Risk. The methodology to check integrity of class hierarchy is implemented on online music store case study with experimental validation. This work is an effort to measure and rank security of software at the design stage of software development.

Keywords—software security, security attribute, object-oriented design, integrity, security quantification.

I. INTRODUCTION

Today, the most challenging job of software developer is to build secure software. Software design can not be neglected. Design will be secure only if it is completely free of open entry points or loopholes. Software security provides a better way to develop software in secure manner. For that there is need to develop robust design that makes software attacks difficult [10]. Software is said to be secure if it is comparatively less prone to attack. So, it is required that before code implementation of software, design should be checked for its safety. If loopholes exist, then effort should be made to remove it. Generally, software is developed in iterative manner; if design is verified against security, it will be easy to finalize more secure design.

One of the researchers had suggested that if any empirical relationship can be develop between security metrics and security problems, it can help to improve their products security at early stage of development [8]. In this work, a methodology is proposed to assess security level at design stage from Integrity perspective. This methodology helps to establish such a relationship at design stage. The methodology is implemented on three versions of the software developed by three different groups of student. We focused on empirical validation of the methodology using nine different class hierarchies of software. In [10], we have developed a methodology to quantify integrity risk on the basis of number of existing loop holes in a class hierarchy. An effort in this paper has been made to propose two security metrics along with integrity state transition model. Implementation of the methodology has been presented using online music store case study.

II. TERMINOLOGIES USED

Before implementation of the methodology some prerequisite requirements need to be fulfilled such as set of sensitive methods, validity check methods, don't care class, sensitive class etc. The following terminologies are used in the proposed methodology.

A. Sensitive Methods

Methods altering any confidential / sensitive information is called sensitive method. It may process attributes having sensitive information, modify only on the basis of its validity. If there are N classes in a class diagram then set of sensitive methods of class C_i is represented as $SM(C_i) = \{SM_{i1}, SM_{i2}, \dots\}$, where $i=1 \dots n$. In order to identify sensitive methods, some suggestive checks are proposed in table 1.

B. Validity Check Method

A method verifying the authenticity for accessing sensitive data by implementing a conditional check is known as validity check method. Set of validity check methods of class C_i is represented as $VM(C_i) = \{VM_{i1}, VM_{i2}, \dots\}$. In order to identify these methods, some suggestive checks are proposed in Table 3.

C. Don't Care Class

If a class doesn't have any sensitive method. The class is said to be don't care class. Set of don't care class is represented as NC.

D. Sensitive Class

A class having sensitive methods and needs protection is said to be sensitive class. Methods of the class may be of its own or inherited / coupled / aggregated to other classes. Set of sensitive class are represented as SC.

E. Risky Class

Class having sensitive methods without its corresponding validity check method is considered as risky class. Set of risky classes are represented as RC. In order to verify risky class, a checklist has been proposed in Table 2.

F. Safe Class

Class having sensitive methods with corresponding validity check method is considered as safe class. Validity check method may be of its own, inherited or coupled. Set of safe class are represented as SF.

G Super Class

The class from which any method or attribute has been taken or shared through coupling, inheritance, aggregation etc. is considered as super class/ source class. Set of all super classes for class C_i is defined as $SP(C_i)$.

H Subclass

The class where the methods and attributes of source class have been floated through inheritance, coupling, inheritance, aggregation etc. is called as subclass/target class. Set of all sub classes for class C_i is defined as $SB(C_i)$.

TABLE 1
CHECKS TO IDENTIFY SENSITIVE METHODS

S. No.	Checks	Y	N
1.	The method alters attributes having sensitive information (e.g. user id, user name, card id, card no. etc.)		
2.	The method alters any attribute carrying encrypted code, password / passcode etc.		
3.	The method alters any attribute having authorization code		
4.	The method alters any conditional check (e.g. expiry date).		
5.	The method alters transaction id etc.		

TABLE 2
CHECKS TO VERIFY A RISKY CLASS

S. No.	Checks	Y	N
1.	Class has any self checking method.		
2.	Class is not inheriting or sharing any validity check method corresponding to their sensitive methods.		
3.	Previous classes are safe.		
4.	Any corresponding validity check method exists, before accessing the class.		
5.	Class is sharing with already declared risky class.		

TABLE 3
CHECKS TO IDENTIFY VALIDITY CHECK METHODS

S. No.	Checks	Y	N
1.	The method process attributes having sensitive information (e.g. user id, user name, card id, card no. etc.)		
2.	The method process any attribute carrying encrypted code, password / passcode etc.		
3.	The method process any attribute having authorization code		
4.	The method process any conditional check (e.g. expiry date).		
5.	The method process transaction id etc.		

During integrity check of class hierarchy these terminologies have been used. As limitation of the methodology identification of sensitive methods and validity check methods have been required. In order to find out these methods some prescriptive checks have been proposed. These checks are developed for a specific type of application. In broader perspective UMLsec and SPARK's annotations may be used for identification of sensitive methods and validity check methods (which keeps any sensitive information needs protection) or any other such type of tools or mechanism. Main focus of the methodology is to produce quantitative results in order to assess security level of software among different versions of software design and provide basis for ranking software.

III. RESEARCH OBJECTIVES AND HYPOTHESIS

Lots of software systems are developed in the form of iterative processes. Recently, some metrics are developed to measure vulnerabilities of software in design stage. These metrics are capable enough to show the impact of Object Oriented (OO) characteristics on software security [1, 2, 3, 4, 5,15]. So, it strengthens the idea to develop such OO security metrics to measure specifically security attributes including confidentiality, integrity and availability in design stage of software development [13,14]. One of the important security design principle is minimized sharing. According to this principle computer resource should be shared between functions and processes unless it is necessary to do so.

Minimized sharing helps to simplify the design and implementation. The rule also concludes that no sensitive information should be shared unless that sharing has been explicitly requested and granted. In addition to this, internal sharing must be carefully designed to avoid in making loop holes [17]. We have concentrated on quantifying integrity risk in design stage using class hierarchy. In order to verify effectiveness and robustness of metrics, we examine the capabilities of metrics on three iterative versions of software class hierarchy. Further, following question may be posed as outcome of the work.

- Is there any relationship between software size and security?
- What are the main reasons of being software risky?
- Is only software size responsible to make software risky?
- Are sensitivity impacts on risk factors?

In order to address these questions, the following objectives are set fourth.

- To identify security metric to define the integrity risk factor of class hierarchy.
- To identify security metric to define the integrity risk factor of security pattern.
- To make it easy to understand the impact of dynamic behaviour of class during state transition.

This research work will test five hypotheses (the relevant null hypothesis is denoted as H_{n0} and alternate hypothesis is denoted as H_{n1} , where $n=0,1,2..$). The hypotheses are:

- Null Hypothesis H_{10} :** Integrity risk factor of hierarchy will not increase as number of risky classes increase.
Alternate Hypothesis H_{11} : Integrity risk factor of hierarchy will increase as number of risky classes increase.
- Null Hypothesis H_{20} :** Integrity risk factor of hierarchy will not decrease as number of safe classes increase.
Alternate Hypothesis H_{21} : Integrity risk factor of hierarchy will decrease as number of safe classes increase.
- Null Hypothesis H_{30} :** Integrity risk factor of hierarchy will not decrease as number of don't care classes increase.
Alternate Hypothesis H_{31} : Integrity risk factor of hierarchy will decrease as number of don't care classes increase.
- Null Hypothesis H_{40} :** Integrity risk factor of hierarchy will not decrease as number of sensitive classes increase.
Alternate Hypothesis H_{41} : Integrity risk factor of hierarchy will increase as number of sensitive classes increase.
- Null Hypothesis H_{50} :** A hierarchy with higher sensitiveness level has more impact on class level integrity risk as compared to security pattern integrity risk.
Alternate Hypothesis H_{51} : A hierarchy with higher sensitiveness level has less impact on class level integrity risk as compare to security pattern integrity risk.

IV. THE METHODOLOGY

Security experts have recognized that in most of the cases attacks are due to poorly designed and developed software. In many cases software is designed and developed without keeping security aspects in mind [12]. The main focus of the methodology is to find out existing loopholes in design. On the basis of number of existing loopholes, secure design will be decided. Lower value of security metrics gets high rank software design. Sometimes existing defects or weaknesses may not be ordinarily used and even they don't have any importance. But in a security context, defect or weakness does matter, because attackers may exploit these weaknesses and induce failures. Attackers may take advantage of it. Therefore, addressing these weaknesses in design phase of development life cycle is urgently needed in order to avoid any attempt of intruders. The methodology is similar to the concept of threat modelling i.e. to scrutinize architecture to discover and correct design-level security problems [8].

During integrity check process, sensitivity of class and method is checked. They are not to be tampered. Reusability is an important feature of object-oriented paradigm. It is advantageous, but with information flow the feature authorization of access privilege extends. Therefore, it is necessary to verify the sensitive information not to be made public. If it is, some restriction or protection is required at the point otherwise it will act as loop hole.

Several attributes and methods are used in a class. In a class, if any method modifies or alter any restricted / sensitive data, then the class may be considered as sensitive class. In coding phase it may easily be checked, whether the information is disclosed after its validity check through its program control flow. In design phase, a class having any self checking method or inheriting any validity check method corresponding to the sensitive method, may be verified. The validity check method ensures that the user has right to access facility of the resources. Permission may be granted to the user, depending on application security policy.

A. Integrity Check Process

In order to check integrity of sensitive class, every method of class need to go through the Integrity Check Process (ICP). For every sensitive method, its corresponding validity check method (authentication, authorization or validation, depending on the security context), is must. The validity check method may be of inherited, coupled or a method of its own class.

Classes altering sensitive methods have been considered as sensitive class SC. Set of sensitive methods of class C_i is $SM(C_i)$. For every sensitive method in $SM(C_i)$, its corresponding validity check method should be there. It may be in the same class or in any of its superclass. $SP(C_i)$, is the set of super classes of sensitive class SC. If any method of $SM(C_i)$ does not have its corresponding validity check method, the Integrity of that method may be violated easily, making the class risky.

Let,

$X = \{SM_{ij} \mid SM_{ij} \in SM(C_i)\}$

$SM(C_i) = \{SM(C_i) \cup SM(SP(C_i))\}$

Where, $SM(C_i)$ is the set of sensitive methods of a class C_i , $SM(SP(C_i))$ is the set of sensitive methods of all super classes of C_i where $0 < i \leq n$, n is no. of classes in a class hierarchy.

SM_{ij} is a sensitive method of Class C_i , need protection from unauthorized access, where, $0 < j \leq m$, m is no. of methods in class C_i .

$$Y = \{VM_{ik} \mid VM_{ik} \in VM(C_i)\}$$

$$VM(C_i) = \{VM(C_i) \cup VM(SP(C_i))\}$$

Where, $VM(C_i)$ is the set of validity check methods of a class C_i . $VM(SP(C_i))$ is the set of validity check methods of all super classes of C_i where $0 < i < n$, n is no. of classes in a class hierarchy.

VM_{ik} is the validity check methods for class C_i , where, $0 < k \leq p$ p is no. of methods in class C_i .

$$\text{Set } A = \left\{ \begin{array}{l} 1, \quad \forall \quad SM \in X \\ 0, \quad \text{otherwise} \end{array} \right\} \quad B = \left\{ \begin{array}{l} 1, \quad \text{if } Y \text{ exists corresponding to } X \\ 0, \quad \text{otherwise} \end{array} \right\}$$

$$\text{and } Z = \left\{ \begin{array}{l} 1 \quad \text{if } A = 1 \quad \& \quad B = 1; \text{method is safe} \\ 0 \quad \text{if } A = 1 \quad \& \quad B = 0; \text{method is risky} \end{array} \right\}$$

Interpretation

If $Z=1$, integrity of method is maintained. The sensitive method and its corresponding validity check method both exist. If $Z=0$, integrity of method is violated, the method becomes risky because the sensitive method exists but corresponding validity check method does not exist (or used by any other means).

The designed process illustrates the integrity check of a method based on the presence / absence of sensitive method and validity check method either its own or used by any means. If result of any input combination is false, then the whole class will be considered as risky making the complete hierarchy risky.

B. Integrity Check for Class Hierarchy (IC²H Algorithm)

Even though the problem of Integrity checks of a class hierarchy is relatively trivial, an attempt has been made to address the problem in the following section. For Integrity check of class hierarchy, algorithm has been proposed.

Procedure begin

```

Step 1: For each class  $C_i \in TC$ 
Step 2:    $C_i$  [status] =null
Step 3:   Do if  $C_i$  is sensitive
Step 4:     Then Enqueue (SC,  $C_i$ )
Step 5:     Else Enqueue (DC,  $C_i$ )
Step 6:   For each class  $C_i \in SC$ 
Step 7:     If  $C_i$  [status]! =checked
Step 8:        $C_i$  [status] =checked
Step 9:     Do cflag=null
Step 10:    Derive  $SMC_i$ 
Step 11:      For each method  $SM_{ij} \in SMC_i$ 
Step 12:        Do if ICP=false
Step 13:          Then cflag=false
Step 14:      If cflag=false
Step 15:        Then verify  $C_i$  with checklist
Step 16:        Enqueue (RC,  $C_i$ )
Step 17:         $Q \leftarrow 0$ 
Step 18:        Enqueue (Q,  $C_i$ )
Step 19:        While  $Q \neq \phi$ 
Step 20:          Do  $C_i \leftarrow$  Dequeue (Q)
Step 21:          For each class  $C_k \in SBC_i$ 
Step 22:            Enqueue(Q,  $C_k$ )
Step 23:          Repeat step 7 for  $C_k$ 
Step 24:   For each class  $C_i \in DC$ 
Step 25:     If count ( $SPC_i \cap RC$ ) >0
Step 26:     Then Enqueue (RC,  $C_i$ )
End
    
```

The algorithm first identifies set of sensitive classes and don't care class. To check integrity of sensitive classes (SC), it is required to check integrity of all sensitive methods (SM) of the same class. If, for any sensitive method, integrity check process (ICP) returns false ($Z=0$) then after verifying with checklist, the class is included as a risky class and its all subclass repeat the same integrity check procedure. If any subclass is found to be risky, the class is also recognized as a risky class. Moreover, even if any of super-class of the don't care (DC) class is risky then the class will be included in set of risky classes (RC).

From the integrity check process it is clear that don't care class does not require any protection but if it uses or shares method or attribute from any sensitive class, safe class or risky class, in the absence of validity check method, then it may also act as a leakage channel or risky class. Using the algorithm, probability of integrity violation of class hierarchy may be quantified in terms of Integrity Risk Factor (IRF) using the security metrics.

C. Integrity State Transition Model

In object-oriented software systems, through inheritance, coupling, and aggregation etc. classes are connected to each other. They share methods and attributes [11]. Because of sharing of methods and attributes, state of the class gets change. Fig 1 depicts the integrity state transition model for describing dynamic behaviour of class. As an advantage of the algorithm, dynamic behaviour of class can be derived or it can be derived from class hierarchy directly. From integrity point of view these states can be divided in four categories.

- Don't Care
- Sensitive
- Safe
- Risky

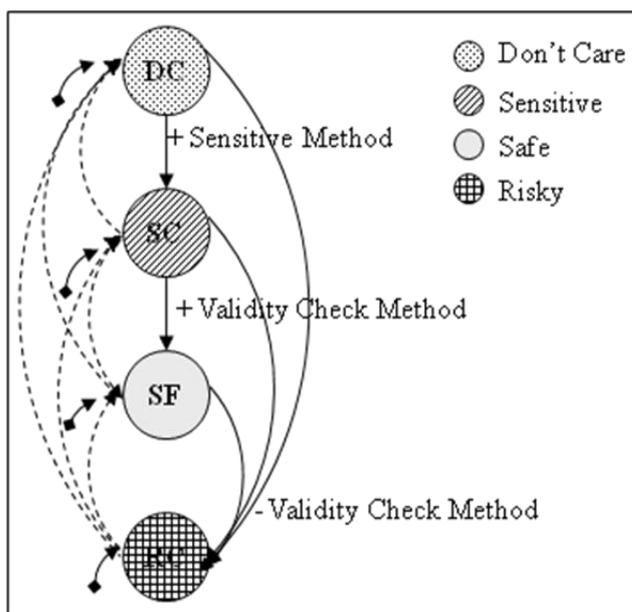


Fig. 1 Integrity state Transition Model (ISTM)

A class having no sensitive method will be at the state of Don't Care. On adding sensitive method, class at Don't Care state will be switched to sensitive state. A class at Sensitive state, on adding validity checks method reaches to safe state. A class at Safe state having any loss of validity checks method switches to risky state. Generally, there is no need to protect class at Don't Care, but if the class share with any sensitive, safe and risky class without protection, will be at risky state and it may increase leakage channels as a result integrity of class may hamper.

Total classes at don't care state give total number of don't care classes represented as DC. Total classes at sensitive states give total number of sensitive classes and represented as SC. Total classes at safe states give total number of safe classes and represented as SF. Total classes at risky state give total number of risky classes and represented as RC

D. Security Metrics

In this section, two security metrics have been established: Class Hierarchy Integrity Risk Factor (CHIRF) and Security Pattern Integrity Risk Factor (SPIRF). Integrity of software can be easily disturbed if sensitive data is computed incorrectly. To maintain integrity it is essential to prove the correctness of design and program at initial level. The objective of the development of these two metric shows the difference of impact of integrity components (total class, don't care class, sensitive class, safe class and risky class) on integrity risk factors. For example: if no. of don't care classes increases then automatically total number of classes increases but it will not risky from integrity point of view. In class hierarchy view, all classes of hierarchy are included and in security pattern view, only sensitive classes have been included which actually affect security of software. Probability of integrity violation increases with the increase of integrity risk factors. These metrics can be easily calculated using the outcome of the IC²H algorithm.

IV. D.1 Class Hierarchy Integrity Risk Factor (CHIRF)

This metric measures the overall class hierarchy integrity risk factor. The metric is defined as the ratio of total number of risky classes to total number of classes. CHIRF is the probability of integrity risk of complete class hierarchy. The CHIRF for class hierarchy is defined as follows:

$$\text{Class Hierarchy Integrity Risk Factor (CHIRF)} = \frac{\text{count}(RC)}{\text{count}(TC)}$$

The RC is set of risky classes in class hierarchy. TC is used to count total classes.

IV. D.2 Security Pattern Integrity Risk Factor (SPIRF)

The metric SPIRF measures probability of integrity risk of security pattern. SPIRF is defined as the ratio of total number of risky classes to total number of sensitive classes. The SPIRF for class hierarchy is defined as follows:

$$\text{Security Pattern Integrity Risk Factor (SPIRF)} = \frac{\text{count}(RC)}{\text{count}(SC)}$$

The RC is set of risky classes present in class hierarchy. SC is used to count total number of sensitive classes.

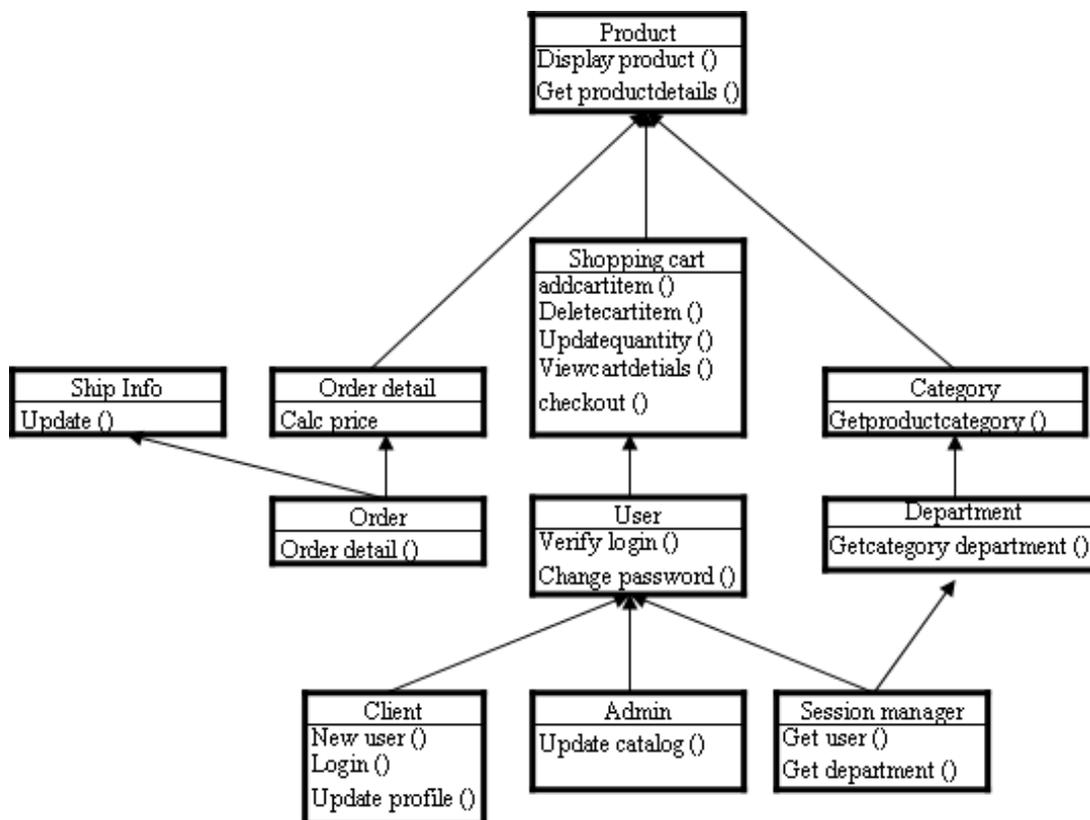


Fig. 2 1st Class Hierarchy

V. EXPERIMENTAL VALIDATION

A case study of online music store is used to implement the methodology proposed in the paper. Annotations serve to build integrity state transition model, and the numerical results shown in this section supports the claim of acceptability of the proposed methodology to assess integrity risk of class hierarchy.

In this section, a class hierarchy of online music store (version 1) is presented with some minor changes in original class hierarchy as in [7]. Fig 2 shows the class diagram in UML notation. In this case study, a music store web application with user interface has been designed. Application enables the user to browse, search, get song recommendations and buy the song items of their choice.

A. Implementation

Implementation of algorithm has been shown on first version of class hierarchy in fig 3. The classes *User*, *Client*, *Admin*, *Session Manager*, *Shopping Cart*, *Orders*, *Ship Info*, *Product*, *Category*, *Department*, and *Order detail* are denoted as $C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}$ and C_{11} respectively.

Total number of classes in version 1 is 11. First classes have been categorized into *Sensitive* and *Don't care* classes. As a result we get sensitive classes, $SC = \{C_1, C_2, C_3, C_5, C_6, C_7, C_{11}\}$ and don't care classes $DC = \{C_4, C_8, C_9, C_{10}\}$. Class C_4, C_8, C_9, C_{10} does not have any function altered sensitive data. The rest of the class has sensitive methods and is altering sensitive information.

For every sensitive method of sensitive class C_i , we go through ICP. At last *cflag* is *null* which states that there is no integrity risk for class C_i (ICP is *true* for every sensitive method of class C_i). Repeat the procedure for all sensitive classes. Complete implementation of the IC²H algorithm on first class hierarchy has been shown in figure 3.

In this class hierarchy, total 11 classes have been used. $C_4, C_8, C_9,$ and C_{10} classes do not have any method altering sensitive data. Therefore, number of don't care classes is 3. Rest of the classes has sensitive methods altering sensitive data. Number of sensitive classes are 7. After implementation of algorithm it has been found that in this class hierarchy, total 4 classes are at risky state and 7 classes are at safe state. It gives number of risky classes is 4 and number of safe classes is 7. As an outcome of the implementation, input details for further security quantification are $TC=11, DC=4, SC=7, SF=7$ and $RC=4$. Same algorithm has been applied on 9 samples of class hierarchy and results are given in table 4.

1st Implementation (class hierarchy version 1)
 TC={C₁,C₂,C₃,C₄,C₅,C₆,C₇,C₈,C₉,C₁₀,C₁₁}
 SC={C₁,C₂,C₃,C₅,C₆,C₇,C₁₁}
 DC={C₄,C₈,C₉,C₁₀}
 For class C₁
 SMC₁={SM(C₁) ∪ SM(SP(C₁))}={SM₁₂}
 VMC₁={VM(C₁) ∪ VM(SP(C₁))}={VM₁₁}
ICP=T
 For class C₂
 SMC₂={SM(C₂) ∪ SM(SP(C₂))}={SM₁₂,SM₂₃, SM₅₁, SM₅₂, SM₅₃, SM₅₅,SM₆₁,SM₇₁,SM₁₁₁}
 VMC₂={VM(C₂) ∪ VM(SP(C₂))}={VM₁₁,VM₂₂}
ICP=T
 For class C₃
 SMC₃={SM(C₃) ∪ SM(SP(C₃))}={SM₃₁}
 VMC₃={VM(C₃) ∪ VM(SP(C₃))}={VM₁₁}
ICP=T
 For class C₅
 SMC₅={SM(C₅) ∪ SM(SP(C₅))}={SM₅₁, SM₅₂, SM₅₃, SM₅₅}
 VMC₅={VM(C₅) ∪ VM(SP(C₅))}={ϕ}
ICP=F
 Q={C₅}, SBC₅={C₂}, RC={C₅}
 For class C₆
 SMC₆={SM(C₆) ∪ SM(SP(C₆))}={SM₆₁,SM₇₁,SM₁₁₁}
 VMC₆={VM(C₆) ∪ VM(SP(C₆))}={ϕ}
ICP=F
 Q={C₆}, SBC₆={C₂}, RC={C₅,C₆}
 For class C₇
 SMC₇={SM(C₇) ∪ SM(SP(C₇))}={SM₇₁}
 VMC₇={VM(C₇) ∪ VM(SP(C₇))}={ϕ}
ICP=F
 Q={C₇}, SBC₇={C₆,C₂}, RC={C₅,C₆,C₇}
 For class C₁₁
 SMC₁₁={SM(C₁₁) ∪ SM(SP(C₁₁))}={SM₁₁₁}
 VMC₁₁={VM(C₁₁) ∪ VM(SP(C₁₁))}={ϕ}
ICP=F
 Q={C₁₁}, SBC₆={C₆,C₂}, RC={C₅,C₆,C₇,C₁₁}

Fig. 3 C²H Algorithm Implementation

TABLE 4
 IC2H ALGORITHM IMPLEMENTATION ON 9 CLASS HIERARCHIES

I st Hierarchy	SC	C ₁	C ₂	C ₃	C ₅	C ₆	C ₇	C ₁₁		
Tot cl (11)	ICP (T/F)	T	T	T	F	F	F	F		
II nd Hierarchy	SC	C ₁	C ₂	C ₃	C ₅	C ₆	C ₇	C ₁₁	C ₁₂	
Tot cl (12)	ICP (T/F)	T	T	T	F	F	F	F	F	
III rd Hierarchy	SC	C ₁	C ₂	C ₃	C ₅	C ₆	C ₇	C ₁₁	C ₁₂	C ₁₃
Tot cl (15)	ICP (T/F)	T	T	T	F	F	F	F	F	F
IV th Hierarchy	SC	C ₁	C ₂	C ₃	C ₅	C ₆	C ₇	C ₁₁		
Tot cl (11)	ICP (T/F)	T	F	T	F	F	F	F		
V th Hierarchy	SC	C ₁	C ₂	C ₃	C ₅	C ₆	C ₇	C ₁₁	C ₁₂	
Tot cl (12)	ICP (T/F)	T	F	T	F	F	F	F	F	
VI th Hierarchy	SC	C ₁	C ₂	C ₃	C ₅	C ₆	C ₇	C ₁₁	C ₁₂	C ₁₃
Tot cl (15)	ICP (T/F)	T	F	T	F	F	F	F	F	F
VII th Hierarchy	SC	C ₁	C ₂	C ₃	C ₅	C ₆	C ₇	C ₁₁		
Tot cl (11)	ICP (T/F)	F	F	F	F	F	F	F		
VIII th Hierarchy	SC	C ₁	C ₂	C ₃	C ₅	C ₆	C ₇	C ₁₁	C ₁₂	
Tot cl (12)	ICP (T/F)	F	F	F	F	F	F	F	F	
XI th Hierarchy	SC	C ₁	C ₂	C ₃	C ₅	C ₆	C ₇	C ₁₁	C ₁₂	C ₁₃
Tot cl (15)	ICP (T/F)	F	F	F	F	F	F	F	F	F

B. State Modelling on Class Hierarchies

Integrity state transition model is simple and easy to implement on class hierarchies. Integrity state transitions of classes have been shown in table 5. Finally, the total outcome of states has been calculated. States of classes have been represented as 0 or 1. As the class comes at specific state, it is represented as 1 and 0 represents the class does not reach to the state. Outcome of the integrity state transition models has been presented in table 5.

TABLE 5
INTEGRITY STATE MODELLING AND METRIC VALUES OF RISK FACTORS

Class Hierarchy	Total Classes (TC)	Don't Care Classes (DC)	Sensitive Classes (SC)	Safe Classes (SF)	Risky Classes (RC)	CHIRF	SPIRF
1 st	11	4	7	7	4	0.363636	0.57143
2 nd	12	4	8	7	5	0.416667	0.62500
3 rd	15	6	9	9	6	0.400000	0.66667
4 th	11	4	7	6	5	0.454545	0.71429
5 th	12	4	8	6	6	0.500000	0.75000
6 th	15	6	9	8	7	0.466667	0.77778
7 th	11	1	10	1	10	0.909091	1.00000
8 th	12	1	11	1	11	0.916667	1.00000
9 th	15	3	12	3	12	0.800000	1.00000

TABLE 6

CORRELATION ANALYSIS SUMMARY BETWEEN INTEGRITY STATES AND RISK FACTORS INTERPRETATIONS

Metrics	TC	DC	SC	SF	RC	Sensitivity
CHIRF (p-value)	-0.065 (0.868)	-0.857 (0.003)	0.825 (0.006)	-0.957 (0.000)	0.930 (0.000)	0.968 (0.000)
SPIRF (p-value)	0.127 (0.744)	-0.713 (0.031)	0.874 (0.002)	-0.875 (0.002)	0.966 (0.000)	0.868 (0.002)

TABLE 7

CORRELATION COEFFICIENT

Coefficient	Relationship
.00 to .20	Negligible
.20 to .40	Low
.40 to .60	Moderate
.60 to .80	Substantial
.80 to 1.00	High

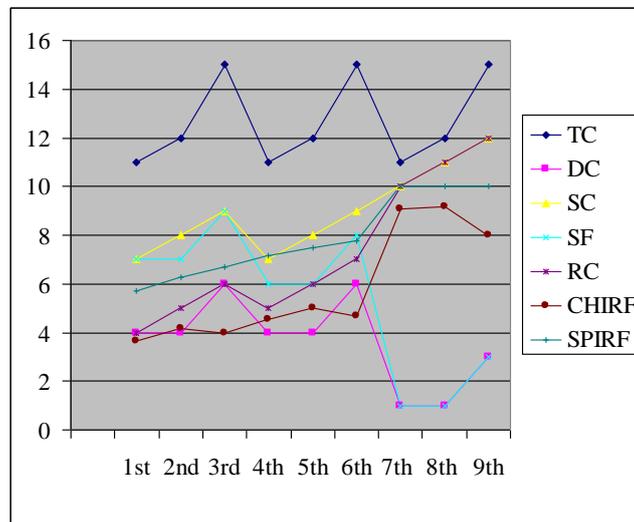


Fig. 4 Graphical Representation of Integrity States and Risk

C. Metric results

This section shows how proposed security metric is able to assess integrity risk of class hierarchy at the design stage of software development. Table 5 shows the results of applying security metrics to nine hierarchies. To make it easy to understand, results have been represented in graphical form in figure 4. The figure shows the variations of integrity risk factors CHIRF and SPIRF corresponding to don't care (DC), sensitive (SC), safe (SF) and risky (RC) classes.

VI. RESULTS ANALYSIS AND HYPOTHESIS TESTING

Results of the experiment are shown in table 5. On the basis of results the graph has been plotted. Graph reflects the relationship of effect of increment and decrement between class states and risk factor. These states affect integrity risk factor. Correlation coefficients between integrity risk factors and class states have been shown in table 6. In literature, so many ways available to interpret correlation coefficient depending upon research purpose. For evaluation of correlation coefficient magnitude we have used a method as available in [16]. Interpretation of magnitude of correlation coefficient is presented in table 7. To test Hypothesis H_{10} , we computed the correlation coefficient between integrity risk factor and risky classes. CHIRF have a strong positive correlation to risky classes and SPIRF have a positive correlation. It concludes that when the number of risky classes increases integrity risk factors will also increase. The results does not support null hypothesis H_{10} . Hence, we reject H_{10} hypothesis and accept alternative hypothesis H_{11} .

To test Hypothesis H_{20} , we computed the correlation coefficient between integrity risk factor and safe classes. CHIRF have a negative correlation to safe classes and SPIRF have a strong negative correlation. It concludes that when number integrity risk factors will decrease as the number of safe classes increase. The results does not support null hypothesis H_{20} . Hence, we reject the H_{20} hypothesis and accept alternative hypothesis H_{21} .

To test Hypothesis H_{30} , we computed the correlation coefficient between integrity risk factor and don't care classes. CHIRF have a negative correlation to don't care classes and SPIRF have a strong negative correlation. It concludes that integrity risk factors will decrease as the number of don't care classes increase. The results does not support hypothesis H_{30} . Hence, we reject H_{30} hypothesis and accept alternative hypothesis H_{31} .

To test Hypothesis H_{40} , we computed the correlation coefficient between integrity risk factor and sensitive classes. CHIRF have a strong positive correlation to sensitive classes and SPIRF have a positive correlation. It concludes that chances to increase integrity risk factors increase as the number of sensitive classes increase. The results does not support hypothesis H_{40} . Hence, we reject H_{40} hypothesis and accept alternative hypothesis H_{41} .

To test Hypothesis H_{50} , we computed the correlation coefficient between integrity risk factor and sensitivity of the class hierarchy. SPIRF have a strong positive correlation to sensitive classes and CHIRF have a positive correlation. This is because of the high correlation of Integrity Risk Factor with sensitive classes and risky classes. Sensitivity of the class hierarchy is completely dependent on both. The relationships between sensitive classes, risky classes and total classes have been shown in figure 4. The sensitivity level will increase only if the sensitive classes increase because don't care classes will also increase the total number of classes. Integrity risk will increase if sensitive classes and risky classes increase in parallel. Results conclude that when the sensitiveness level increases, chances to increase integrity risk factor will also increase. The results does not support hypothesis H_{50} . Hence, we reject hypothesis H_{50} and accept alternative hypothesis H_{51} .

VII. SIGNIFICANCE OF THE METHODOLOGY

For secure computing, it is very important that authenticity of the data should be maintained therefore it needs to be guaranteed. It has been concluded that common standard to ensure data integrity do not exist [18]. Therefore in the research work developed methodology provides a simple and easy way to quantify integrity. Objective of the methodology is to quantify integrity risk of software at design stage of software development. Simple and understandable view of the methodology makes it easy to implement. Quantitative estimation of software is required in order to assess security level of software. The changes in total classes, don't care classes, sensitive classes, safe classes, risky classes and corresponding change in the class hierarchy integrity risk factor and security pattern integrity risk factor has been clearly shown in the graph. These results provides basis to assess and control security level of software. Normally, software applications develop in iterative manner. Quantitative results of integrity risk factors help to identify more secure version of software design. It is concluded from the numerical results that integrity risk factor is highly affected from sensitive classes and risky classes. The graph has been plotted for all versions of class design. As shown in fig 4 numbers of risky classes is high where sensitive classes are higher. Additionally, if only total number of classes increases and number of sensitive classes is same then risk will be the same.

VIII. CONCLUSION

In this paper a modified version of the methodology to quantify integrity risk at design stage been discussed. An experimental case study is used to illustrate the applicability of the methodology. Experiments have been done on three iterative software processes. Quantitative results will help to choose better design or what considerations we should make to improve design quality. The goal of the work is to throw light on some facts which really affect software security. This work is concentrated on only integrity security attribute which is one of the essential security requirements. The main focus of the methodology is quantification of integrity risk in design stage. Identified set of sensitive classes and risky classes may use for further analysis or for improvements during design stage. There is possibility of improve methodology. This version of the methodology, it is cable enough to quantify integrity risk of class design including algorithm, state transitions and metrics. Further, with some modifications it is possible to identify or detect most risky classes and such classes which make other classes risky because risky classes act as disease which influence other one

very easily. So, it is required to identify and mitigate it. Future work will include remaining two attributes confidentiality, availability and its implementation.

REFERENCES

- [1] L. C. Briand, J. W. Daly, and J. K. Wüst, "A Unified Framework for Coupling" IEEE Transactions on Software Engineering, VOL. 25, NO. 1, Jan/Feb 1999, pp: 91-121.
- [2] B. Alshamari, C. Fidge and D. Corney, "Security Metrics for Object-oriented Class Designs", In: QSIC 2009 Proceedings of Ninth International Conference on Quality Software , August 24-25, 2009, Jeju, Korea.
- [3] B. Alshamari, C. Fidge and D. Corney, "Security Metrics for Object-oriented Designs", 21st Australian Software Engineering Conference, IEEE Computer Society
- [4] A. Agrawal, "A Vulnerability Metric for design Phase of Object-oriented software", IC3 2010,CCIS 94, Springer, pp:328-339.
- [5] A. Yadav and R. A. Khan "Measuring design Complexity- an Inherited Method Perspective", AVM Sigsoft Software Engineering Notes, VOL 34, No. 4, pp: 1-5.
- [6] K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai, "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics", IEEE Transactions on Software Engineering, VOL. 27, NO. 7, JULY 2001, pp: 630-650.
- [7] http://people.cis.ksu.edu/~reshma/798_ClassDiagram.htm
- [8] Y. Shin and L. Williams, "Is Complexity Really the Enemy of Software Security?" Proceedings of the 4th ACM Workshop on Quality of protection, 2008 Alexandria, Virginia, USA, pp: 47-50.
- [9] G. McGraw, "Software Security", IEEE Security and Privacy, pp: 80-83.
- [10] S. Chandra and R.A. Khan, "A Methodology to Check Integrity of a Class Hierarchy", <http://www.academypublisher.com/ijrte/vol02/no04/ijrte02048385.pdf>
- [11] T. Spyros. Halkidis, A. Chatzigeorgiou and G. Stephanides, "A qualitative analysis of software security patterns", Computers and Security, Volume 25, Issue 5, July 2006, Pages 379-392.
- [12] S. Chandra and R. A. Khan, "Object Oriented Software Security Estimation Life Cycle: Design phase perspective", Journal of Software Engineering, USA, pp: 39-46.
- [13] S. Chandra and R. A. Khan, "Software Security Metric Identification Framework (SSM)", In proceedings of International Conference on Advances in Computing, Communication and Control (ICAC3'09), 725-731.
- [14] S. Chandra, R.A. Khan, and A. Agrawal, "Software Security Factors in Design Phase", ICISTM 2009, CCIS 31, pp. 339-340, 2009.
- [15] A. Agrawal, S. Chandra, and R.A. Khan, "An Efficient measurement of Object Oriented Design Vulnerability", Proceeding of IEEE conference on Availability, Reliability and Security, 2009, IEEE Computer Society, pp: 618-622.
- [16] J. W. Best and J. V. Kahn, "Research in Education", PHI Publication.
- [17] T. V. Benzal, C. E. Irvine, T. E. Levin, G. Bhaskara, T. D. Nguyen, and P. C. Clark, "Design Principles for Security". September 2005.
- [18] C. Rong, S. T. Nguven, M.G. Jaatun, "Beyond lightning: A survey on security challenges in cloud computing", Computers & Electrical Engineering, Vol 39, Issue 1, January 2013, pp: 47-54.



Dr. R. A. Khan is currently working as a Associate Professor in the Department of Information Technology, Babasaheb Bhimrao Ambedkar University (A Central University), Lucknow, UP. His area of interest is Software Security, Software Quality and Software Testing. He has authored two books on software quality and software testing.



Dr. Shalini Chandra is currently working as Assistant professor in MCA Department, Shri Ramswaroop Memorial College of Engineering and Management, Lucknow. UP. India. Her research area is Software Security.