EVALUATION BASED SECURE SOFTWARE ARCHITECTURE AND DESIGN FRAMEWORK

Mubashirah Majeed¹, S.M.K Quadri²

Department of Computer Sciences University of Kashmir, Srinagar, India
Department of Computer Sciences, Jamia Milia Islamia, New Delhi, India
mubashirah.majeed@gmail.com, quadrismk@hotmail.com

ABSTRACT

In this paper we are proposing a hybrid secure software development lifecycle (H-SSDLC). The proposed H-SSDLC provides the early indications for the software developer to think about the security at the very early stage of software development lifecycle. An empirical evaluation based on the most important security attributes like dependency, availability, confidentiality and integrity is carried on a system to determine the security level of the components. The calculated security (severity) levels provide the early indications for the developer to take the necessary counter measures to mitigate the security flaws early in the software development process. Our proposed framework uses the best of the techniques at each level of development to achieve the more secure system.

Keywords: vulnerabilities, Confidentiality, secure software, Security Requirements, Component Based Design

[1] INTRODUCTION

Developing the secure software is a process to build the software that can withstand both the intentional and unintentional malicious intents. In the real environment developing the security of the software has always been treated as a side line property that is to be carried out after the completion of software development in the form of securing the operational environment placing the right protection mechanisms like firewalls in place etc. Presently most of the software security often fails since its development is generally based on ad-hoc foundations [1]. With the increase in network connectivity, complexity of code and programming languages and the
sophistication and easy access to the attack tools the demand of reducing the security vulnerabilities in the software is the most urgent need. The traditional development process used for development of software is not able to focus on the security concerns and lack in the knowledge of security resulting in the security flaws. As mentioned earlier the security is unnoticed in early phases of software lifecycle. A good software engineering approach is to think about the security right from the infancy stages of the software development lifecycle [2]. It is evident that fixing a bug in the software after the development phases is ten times costlier than fixing it in the early development stages and also fixing a bug after the completion of development process of system the patches to be applied to fix the security holes. These patches lead to further vulnerabilities if attacker came to know about the application of patches. The traditional software development life cycle must be enhanced to accommodate the security concerns right from the beginning of the development process. Such secure software development Process (SSDP) ensures the security of the system at each progressive stage of the software development.

In this paper a hybrid approach (H-SSDP) is proposed for secure software development process. The H-SSDP is based on the process evaluation of the security and countermeasures (readjustment) at the early architectural and design phases of the development. The proposed H-SSDP is iterative in nature to achieve the required level of assurance.

[2] PROPOSED H-SSDP

The proposed H-SSDP is based on the evaluation and readjustment strategy. Below diagram (1) depicts the overall H-SSDP.
2.1 Security Requirements: every software development process is based on the strong foundations known as the security requirements. From the early studies it is evident that most of the security flaws are exploited in the software due to the use of improper security requirement specification methods. The most current software requirement specifications fall within the following three categories [3],(i) totally silent regarding security (ii) merely specify vague security goals or (iii) specify commonly used security mechanisms as architectural constraints. In the first case security is not considered during software development at all. In the second case the security requirements are unstructured and very hard to evaluate, and in the last case security requirements are binded early in the design phase which results in improper specification of requirements. In security software development process the system is evaluated for the specific security requirements. From our experience and literature, we believe that the all the specified security requirement for any type of system should ultimately come under one or more well-
defined and well-established security attributes as shown in previous chapter figure(3.2). In our proposed framework we are evaluating the system against the basic security attributes which are confidentiality, integrity, availability and dependency because we predict that the evaluation of the system against such attributes will increase the scalability of the system.

2.2 Software Architecture and Design: In this section of proposed secure software development process we present the earlier tools and techniques used for architecture and design specification. Since our evaluation framework aim at the software architectural and design phase, so our foremost focus is to identify the commonly used software architecture and design techniques that have been adopted by the developers in real practices and their feasibility in security evaluation.

Constructing a software system by composing prefabricated or newly developed components is always an initiative and attractive vision for software development [4]. There is a consensus on the fact that for any large software systems the overall structure, i.e. the high-level organization of computational elements and interaction between those elements is the critical aspect of the design [5]. Software architecture is the depiction of the overall system, its components and the interfaces among these components. It aids on providing the help for understanding the system. The good system always begins with good design. The quality of software is highly dependent on the architecture defined in the early stages of development process. Architecture influences both functional and non-functional requirements of the system. Modifying or correcting the wrong design decisions take lot of efforts, therefore it is important to analyze the specified architecture at the early development stages [6]. Various software architectural and design modeling techniques like Object oriented Analysis and design (OOAD), Enterprise Architecture Framework (EA), Service oriented Architecture and Design (SOAD), Component Based Design (CBD) have been presented and adopted among them Component Based Design (CBD) is the most common. Selection of the appropriate modeling technique for the evaluation of the quality attributes is very important. Choosing particular software architectural and design approach for the evaluation of quality attributes in general and for the evaluation of security in particular, is highly dependent up on the levels of abstraction provided by each of them.

In our proposed system security evaluation process we are adopting the Component Based Design (CBD). The position of Component Based Architecture and Design (CBAD) in the architectural hierarchy are neither too coarse-grained like services that hide the internal functional units nor too fine-grained to make the evaluation process too complex to bind the lower implementation level details. From the literature survey and our experience, The Component layer is the best suited for the evaluation process of any quality attribute of the system.

2.3 Security Architecture Representation with Dataflow Dependencies: UML is one of the dominant modeling languages used in the software design for specification, visualization, construction of software artifacts. UML is equipped with verities of diagrams like class diagrams, interactive diagrams, component diagrams to specify the functional and behavioral aspects of the software. In our proposed system design we have adopted the UML modeling.

The UML components interact, coordinate, and communicate with one another for information/data flow. In this system the components requires or provides the services to other components, so their exists dependencies among the components. The dependencies among the components are depicted by the arrows. The solid arrows represents the direct interaction among
the components while as the dashed arrows depicts the intermediate interaction among the components of the system. Similar to the object-oriented systems, in which the object is the basic building block, in CBSs, component is the basic, but usually a black box building block. As new components get added in the system they affect the whole or part of a system. The newly added components interact with the existing components for the required services. Mainly there exist two types of dependencies:
- Direct dependency is direct interaction among components
- Indirect dependency is intermediate interaction among components

2.4 Security Evaluation Baseline: Every evaluation mechanism must have a certain baseline to strive for. The security requirements specification becomes the ultimate criteria to strive for but considering this requirement as a baseline for evaluating security poses the various difficulties. One major hindrance is that security requirements are environment and system dependent. In such case the evaluation framework becomes very complex. Second it is very difficult to quantitatively evaluate the system against the specified requirements, because the requirements normally are theoretical representation of what is to be achieved. In the proposed H-SSDP we believe that all security requirements ultimately come under the umbrella of the following four well defined well established security attributes.

- Dependency
- Confidentiality
- Integrity
- Availability.

In the proposed H-SSDP these four security attributes are the parameter for the evaluation and prioritization (with respect to security) of the systems under study.

2.5 Evaluation of Software Architecture and Design: Architecture & design phase is the most critical phase of the software development. It acts as a blueprint for overall system’s functionality and behavior. The quality of a software system is heavily dependent on the quality of its architecture & design. A security flaw or hole at this level can lead to the vulnerabilities at the next levels of system life cycle and can cause the severe damage. Considering the security at this level reduces the cost and efforts required at the further stages of the system life cycle. The security evaluation at architecture and design level mitigates the risk of vulnerabilities early in the development phase. Security in traditional software development lifecycles were always given an afterthought, such processes needs modification to make the development process more secure and reliable.

In our proposed secure software development lifecycle we are evaluating the system at the architecture and design level for the early indicator of security risks. When the security flaws are considered at the early design phase of software it reduces the cost and efforts made at the next higher level. There are various architecture and design specifications but selection of the one depends on the level of applicability. The evaluation stage after the architecture and design specification acts as a corner stone for our secure software development process (SSDP). The aim of this evaluation of architecture for security is twofold. It can act as a mechanism for the readjustment (countermeasures) of software architecture and design to minimize the security risks. Secondly it can act as a base to create the security protection mechanism from the scratch or from the application of existing counter
measures at the appropriate level and place in the system. The proposed framework is productive in nature and is very generic that can be applicable to any system regard less of the size and nature.

As depicted in the SSDP diagram (1) the evaluation of the software can be repetitive in nature and a threshold level is to be achieved.

2.6 Quantization of the components: As mentioned above, the proposed system is analyzed and predicted for the early indication of security against the most important security attributes viz Confidentiality, Integrity, Availability and Dependency. The system gives the security criticality level for individual components and also for the overall system. The components of the system are grouped according to the level of criticality or sereneness in the system on a scale of (0-10). We have categorized the component according to the resultant security indicators into five levels (Level1-Level5) with Level1 the most severe and Level5 the least severe. The categorization of the components is based on the following ranges of output values.
- Components having the resultant value between (0-1) are at level 1
- Components having the resultant value between (1-2) are at level 2
- Components having the resultant value between (2-3) are at level 3.
- Components having the resultant value between (3-4) are at level 4
- Components having the resultant value above 4 are at level 5.

2.7 Countermeasures and Corrective measures: Software is always suspected to malicious intents. So security measures are needed to prevent the assets from these malicious intents. The security measures acts as tool to evaluate the security level of the system and to take the necessary prevention measures to mitigate the security threats. Various tools and techniques are proposed by different authors to assess the security. These tools and techniques are applicable at different levels of the system. Counter measures can be the various security protection mechanisms applied at the various system levels to ensure the security attributes (confidentiality, integrity, availability and dependency) of a system.

Confidentiality
1. Cryptography
2. Defense in Depth

Integrity
1. Mirroring
2. File system integrity checker:
3. Code signing:
4. Tripwire
5. Caspar

Availability
1. Denial of service

[3] FRAMEWORK FOR SECURE SOFTWARE ARCHITECTURE & DESIGN

In this section and the upcoming subsections we will present the proposed framework for secure software architecture & design. The proposed framework is predictive in nature and based on the mathematical modeling techniques. At the very high level abstraction the proposed framework is based on the following two stages.
1. Identification & prediction of the components responsible for the security of the overall system.
2. Countermeasure to be adopted based on the stage 1.

A system composed of several components (functional units) can be represented in graphical form. In our case we use UML based component modeling, because of its various advantages over the other modeling techniques, as mentioned earlier. In order to simplify the process we take into account the system composed of multiple component for illustrative purpose as shown in figure below using Visual paradigm for UML 9.0. As depicted in Figure (2) there exists dependencies that occurs due to the interconnection of provided and required interface (includes both data and interface dependencies) for the provided and required functionalities. Beside these implicit dependencies, the dependencies are also specified explicitly by dashed arrow lines. The direction of the line depicts the source component on the tail of the line which depends up on the destination component on the arrow head.

The dependencies among the components is depicted in an adjacency matrix (AM) with values 0’s and 1’s. In the matrix form, the components are arranged as rows and columns with index i and j respectively. If a component Ci in row i is dependent on the component Cj in column j then the corresponding element at i*j is marked as 1, otherwise it is marked as 0. In general, the values for each of the elements of adjacency matrix $DM(n*n)=d_{ij}(n*n)$.

Where:

$$d_{ij} = \begin{cases} 1, & \text{if } C_i \rightarrow C_j \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

According to equation (1) the values for the direct dependency matrix of figure (2) is as:
The above given matrix represents the all possible direct dependencies among the components of the system. In order to calculate the indirect dependencies among the components we use Warshal’s Algorithm for transitive closer [7] and the resultant complete functional dependency matrix is as:

\[
\begin{align*}
C_{\text{FDM}} &= \begin{bmatrix}
C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 & C_8 & C_9 \\
C_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
C_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
C_3 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
C_4 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
C_5 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
C_6 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
C_7 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
C_8 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
C_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{align*}
\]

Figure 2.1: Functional Dependency Matrix

Besides the components direct and indirect associations there exists other types of associations among the components, Input-Dependency and Output-Dependency.

**Input-Dependency:** Input-dependency denoted by InDep of a component (Ci) is defined as the number of other components in the system which requires the services of the component (Ci).

Mathematically:

\[
\text{InDep}(c_i) = \sum_{j=1}^{n} \text{FDM}(j|i) \quad \text{--------- (2)}
\]

Where \( \text{InDep}(c_i) \) is the input-dependency of the component \( c_i \), 
\( \text{FDM} \) is the functional dependency matrix, \( n \) is total number of components in the system.

**Output-Dependency:** Output-dependency denoted by OutDep of a component (Ci) is defined as the other components in the system upon which component (Ci) depends (directly or indirectly) for their provided functionalities.

Mathematically:

\[
\text{OutDep}(c_i) = \sum_{j=1}^{n} \text{FDM}(i|j) \quad \text{--------- (3)}
\]

Where
OutDep(ci) is the output-dependency of the component ci, FDM is the functional dependency matrix, n is total number of components in the system. Two more terms related to the dependency are complete Input-dependency and complete Output-dependency are:

**Complete input-dependency**: denoted by $ClInDep(C_j)$ is defined as the number of components in the Input-dependency of component(Cj). It can be easily calculated by counting the number of 1's in the corresponding column j of the complete dependency matrix (CFDM).

Mathematically:

$$ClInDep(ci) = \sum_{j=1}^{n} CFDM(j,i)$$  (4)

Where

- ClInDep is the complete Input-dependency of a component (ci).
- CFDM is the complete dependency matrix.
- n is the total no of components in the system.

**Complete Output-dependency**: denoted by $ClOutDep(C_i)$ is defined as the number of other components in the system upon which component (Ci) is dependent. $ClOutDep$ of a component can be calculated by counting the number of 1’s in the corresponding row i of the complete dependency matrix (CFDM).

Mathematically:

$$ClOutDep(ci) = \sum_{j=1}^{n} CFDM(i,j)$$  (5)

Where

- ClOutDep is the complete Output-dependency of a component (ci).
- CFDM is the complete dependency matrix.
- n is the total number of components in the system.

Based on the above mentioned dependencies we will evaluate the system for the dependency of the individual components.

### 3.1 Dependency evaluation

Dependency in a system is one of the main concerns to analyze the security of a system. When a system functions in isolation i.e., it does not interact with any other system there are less chances of security breaches. In complex systems where various components interact with other components for providing the best services to the clients, the dependency level of each component is necessarily to be evaluated. The level of dependency of a component in a system gives the indication of its criticality in the system.

As mentioned above a component exhibits two types of dependencies **Input Dependency** and **Output Dependency**, equation (2) and equation (3) respectively. The total dependency of a component (direct and indirect) is calculated by using the following formula:

$$Dep (C_i) = \frac{10}{ClOutDep(ci) + ClInDep(ci)}$$  (6)

Where

- ClOutDep is the complete out-dependency of component (Ci).
- ClInDep is the complete Input-dependency of component (Ci).

and ClOutDep(ci), ClInDep(ci)>0

The resultant values are calculated in a range (0-10). Lower the resultant values higher will be the effect of the component.
3.2 Availability evaluation: the services provided by the software or its components in a network should remain available to others in appropriate manner. In current networked environment the systems are changed from standalone application to large distributed architectures. In case of remotely connected components, the provided functionalities of components are accessed by remote procedure call (RPC). In a system where the components are functionally dependent on one another; the one component may invoke a call and wait for the provided services of second component which in turn requires the functionality of third component and so on. Such a dependency chain will certainly result in delayed response time at each hop of the dependency. The two main factors responsible for the delay at different hops are: Processing Delay and Transmission Delay.

**Processing Delay:** Time taken by a component to successfully process the request of its clients and return the result from the invocation to the end result returned.

**Transmission delay:** when a remotely located component is called for a service. The transmission delay is the time alone over the network excluding processing delay.

The measurement of availability especially results in to find the availability critical components, so that the alternative corrective measures can be applied. In order to calculate the availability critical components we need the following terms:

- \( \text{degOutput}(C_i) \)
- \( \text{degInput}(C_i) \)
- Processing Delay (P).
- Transmission Delay (T).

Where \( \text{degOutput}(C_i) \) is the maximum number of components that \( C_i \) call for their provided services and \( \text{degInput}(C_j) \) for a component \( C_i \) is the maximum number of components that may require the services of component \( C_i \). As the dependency level of a component (both in and out) increases, it will certainly affect the availability of the component and the overall system, because of the processing delay and transmission delay of each of the component in the dependency graph.

Mathematically:

\[
Avl(C_i) = \frac{\text{ClnDep}(C_i) + \sum_{j=1}^{10} \text{COutDep}(C_i) + \sum_{j=0}^{\text{Pj} + T_j}}{\sum_{j=0}^{\text{Pj} + T_j}}
\]  

Where \( \text{ClnDep}(C_i) \) is the complete Input-dependency of component \( C_i \), \( \text{COutDep}(C_i) \) is the complete out-put dependency of the component \( C_i \) and

\( \text{ClnDep}(C_i), \text{COutDep}(C_i) > 0. \)

Pj is the processing delay of the jth component in \( \text{COutDep}(C_i) \).

Tj is the transmission delay involved of jth component in \( \text{ClnDep}(C_i) \).

Note that the index of j starts from 0 to take in account the processing and transmission delay of \( C_i \) itself.
3.3 Confidentiality Evaluation. Confidentiality of a system or a network ensures that the unauthorised disclosure of data/information should not occur. Our proposed evaluation method aim to assess, predict and provide the quantitative indicators about the level of confidentiality of each of the component. In order to calculate the confidentiality criticality of components we need the following parameters.

- **Entry points**: denoted by $E_p$, are the required interfaces of a component through which the In-flow of data/information takes place.
- **Exit points**: denoted by $E_x$, are the provided interfaces through which the Out-flow of information takes places.
- Number of components that can likely make an In-flow (write operation) directly or indirectly on $C_i$ is $C_{OutDep}(C_i)$.
- Number of components that can likely responsible for the Out-flow (read operation) of data/information directly or indirectly from $C_i$ is $C_{InDep}(C_i)$.

Mathematically:

$$
\text{ConfId}(C_i) = \frac{10}{\text{CIndep}(C_i)^2 \times E_x / \text{COutdep}(C_i) \times E_p}
$$

Where:

- $C_{Indep}(C_i)$, $C_{Outdep}(C_i)$, $E_x$, $E_p > 0$
- $\text{ConfId}(C_i)$ is the calculated confidentiality of the component $(C_i)$.
- $\text{CIndep}(C_i)$ is the complete in dependency of the component $(C_i)$ (maximum number of components capable of reading data from component $(C_i)$).
- $\text{COutdep}(C_i)$ is the complete out dependency of the component $(C_i)$ (maximum number of components capable of performing data flow into the component $(C_i)$).
- $E_x$ is the number of exit point of the component $(C_i)$.
- $E_p$ is the number of entry point of the component $(C_i)$.

In order to calculate the results in a specified range of 0 to 10, the whole equation is dived by 10.

In above equation (6), lower the resultant value, higher will be the impact of the confidentiality breach of the system.

3.4 Integrity Evaluation: Integrity is the other main pillar of security for a system, component or a network. The main concern of integrity is to ensure that the unauthorised modification of data/information should not take place. The integrity parameter of a component is evaluated by using the following parameters:

Let $C_i$ be a component whose integrity level is to be evaluated.

The possible number of components that are responsible for the In-flow (write) of data/information will be the $C_{OutDep}(C_i)$ of component $C_i$. as mentioned early the possible number of components

on which the component $C_i$ dependents for their provided services directly or indirectly is the $C_{OutDep}(C_i)$ of component $C_i$. Let $C_i$ be a component whose integrity level is to be evaluated. The number of ports or read-interfaces through which the inflow of data can take place is $E_p$.

Mathematically:
The proposed method provides the early indication of level of integrity in the system. With the lower values on the scale higher will be the integrity breach.


The empirical evaluation is carried out on an “online inventory system”. The component level design and architecture of the system is prepared using UML 2.0 component based design and architectural notations along with the interfaces (provided and required) and the dependencies among the components in “Visual Paradigm for UML 9.0” design tool figure(2). Table(1) below shows the evaluated data collected from the system.

As depicted in the figure (2) there are 16 fine grained components in the system composition. Besides the component representation we have also depicted the dependencies among the components with arrow lines, the direction of arrow specifies the source and destination of a particular dependency.

![Component Based Architecture and Design of “online Inventory Store”](image)

4.1 Evaluation of the system

According to equation (1) the values for the direct dependency matrix of figure (2) is as:
In order to calculate the indirect dependencies among the components we use Warshal’s Algorithm for transitive closer [7] and the resultant full dependency matrix is as:

**Figure 2.1: Functional Dependency Matrix.**

Based on the data collected from figure (2), the resulting values are depicted in the tabular form as under

**Figure 2.2: Complete Functional Dependency Matrix**

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
<th>c4</th>
<th>c5</th>
<th>c6</th>
<th>c7</th>
<th>c8</th>
<th>c9</th>
<th>c10</th>
<th>c11</th>
<th>c12</th>
<th>c13</th>
<th>c14</th>
<th>c15</th>
<th>c16</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>c15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 1: Data Collection from Figure (2)**
The values of the entries of table (1) are calculated according to the above mentioned equations. Based resultant values the criticality level of the components with respect to the security attributes (Dependency, Availability, Confidentiality and Integrity) are shown as under:

### Table 2. Security attributes indicators

<table>
<thead>
<tr>
<th>Component No</th>
<th>Dependency</th>
<th>Availability</th>
<th>Confidentiality</th>
<th>Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.3333</td>
<td>2.3981</td>
<td>1.0000</td>
<td>5.0000</td>
</tr>
<tr>
<td>2</td>
<td>2.5000</td>
<td>0.5817</td>
<td>1.0000</td>
<td>10.0000</td>
</tr>
<tr>
<td>3</td>
<td>0.7692</td>
<td>4.7393</td>
<td>4.6154</td>
<td>0.4167</td>
</tr>
<tr>
<td>4</td>
<td>0.8333</td>
<td>4.6083</td>
<td>4.4000</td>
<td>0.4545</td>
</tr>
<tr>
<td>5</td>
<td>1.0000</td>
<td>2.3866</td>
<td>3.3333</td>
<td>0.6250</td>
</tr>
<tr>
<td>6</td>
<td>1.2500</td>
<td>0.2068</td>
<td>0.1000</td>
<td>5.0000</td>
</tr>
<tr>
<td>7</td>
<td>1.0000</td>
<td>1.9268</td>
<td>2.8000</td>
<td>1.4286</td>
</tr>
<tr>
<td>8</td>
<td>1.4286</td>
<td>1.0834</td>
<td>0.2632</td>
<td>5.0000</td>
</tr>
<tr>
<td>9</td>
<td>3.3333</td>
<td>1.3947</td>
<td>3.3333</td>
<td>2.5000</td>
</tr>
<tr>
<td>10</td>
<td>1.0000</td>
<td>0.6549</td>
<td>0.5000</td>
<td>1.2500</td>
</tr>
<tr>
<td>11</td>
<td>1.6667</td>
<td>0.6592</td>
<td>4.1667</td>
<td>1.0000</td>
</tr>
<tr>
<td>12</td>
<td>1.0000</td>
<td>0.9001</td>
<td>3.0000</td>
<td>0.3704</td>
</tr>
<tr>
<td>13</td>
<td>1.1111</td>
<td>0.9588</td>
<td>0.1515</td>
<td>5.0000</td>
</tr>
<tr>
<td>14</td>
<td>1.2500</td>
<td>1.0753</td>
<td>0.1961</td>
<td>5.0000</td>
</tr>
<tr>
<td>15</td>
<td>1.1111</td>
<td>0.9662</td>
<td>0.1538</td>
<td>10.0000</td>
</tr>
<tr>
<td>16</td>
<td>1.2500</td>
<td>1.1947</td>
<td>0.2000</td>
<td>10.0000</td>
</tr>
</tbody>
</table>
[5] RESULT AND DISCUSSION

5.1 Prioritization of modules for enforcing security mechanisms

Table (4) depicts the resultant security indicators in a range of (0-10) for each of the component in the system composition, with respect to the four major attributes of security (Dependency, Availability, Confidentiality and Integrity). We have categorized the component according to the resultant security indicators into five levels (level1-level5), with (level1) the most severe and the (level5) least severe with respect to each of the security attributes. The categorization of the components is based on the ranges of output values depicted in table (3).

Table 3. Component severity levels.

<table>
<thead>
<tr>
<th>Level</th>
<th>Value in range</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>0 to 1</td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td>1 to 2</td>
<td></td>
</tr>
<tr>
<td>Level 3</td>
<td>2 to 3</td>
<td></td>
</tr>
<tr>
<td>Level 4</td>
<td>3 to 4</td>
<td></td>
</tr>
<tr>
<td>Level 5</td>
<td>4 above</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Security attributes</th>
<th>Component No</th>
<th>Total no of components</th>
<th>security level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependency</td>
<td>3,4,5,7,10,12</td>
<td>6</td>
<td>L1</td>
</tr>
<tr>
<td>Availability</td>
<td>2,6,10,11,12,13,15</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Confidentiality</td>
<td>1,2,6,8,10,13,14,15,16</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Integrity</td>
<td>3,4,5,11,12</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependency</td>
<td>6,8,11,13,14,15,16</td>
<td>7</td>
<td>L2</td>
</tr>
<tr>
<td>Availability</td>
<td>7,8,9,14,16</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Confidentiality</td>
<td>=</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Integrity</td>
<td>7,10</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependency</td>
<td>2</td>
<td>1</td>
<td>L3</td>
</tr>
<tr>
<td>Availability</td>
<td>1,5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Confidentiality</td>
<td>7,12</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Integrity</td>
<td>9</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependency</td>
<td>1,9</td>
<td>1</td>
<td>L4</td>
</tr>
<tr>
<td>Availability</td>
<td>=</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Confidentiality</td>
<td>5,9</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Integrity</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependency</td>
<td>=</td>
<td>0</td>
<td>L5</td>
</tr>
<tr>
<td>Availability</td>
<td>3,4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Confidentiality</td>
<td>3,4,11</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Integrity</td>
<td>1,2,6,8,13,14,15,16</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Component security levels
As shown in table (4) most of the components fall under Level 1 of severity i.e. between (0-1). The components falling under Level 1 are the most severe with respect to the specified security attributes. Level 3 is the moderate level and Level 5 is in the safe zone. The output of the security evaluation can act as the input to the decision making in order to apply the necessary corrective and preventive mechanism with varying level of priority.

1.2 Readjustment: Now the decision team or the developer have the visible security profile of the system. The security attribute indication provides him an early alarm about the system security. Figure (3) depicts the security severity of the components. Based on the security severity of the components the software developer will adopt the appropriate security countermeasures to eliminate the criticality of the components.
REFERENCES


