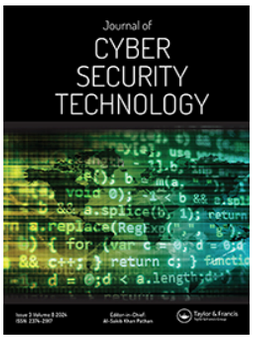

Journal of Cyber Security Technology

ISSN: (Print) (Online) Journal homepage: www.tandfonline.com/journals/tsec20

A Case Study of Cyber Subversion Attack based Design Flaw in Service Oriented Component Application Logic

Faisal Nabi, Xujuan Zhou, Umna Iftikhar & Hafiz Muhammad Attaullah

To cite this article: Faisal Nabi, Xujuan Zhou, Umna Iftikhar & Hafiz Muhammad Attaullah (2024) A Case Study of Cyber Subversion Attack based Design Flaw in Service Oriented Component Application Logic, Journal of Cyber Security Technology, 8:3, 204-228, DOI: 10.1080/23742917.2023.2261169

To link to this article: https://doi.org/10.1080/23742917.2023.2261169

JOURNAL OF CYBER SECURITY TECHNOLOGY
2024, VOL. 8, NO. 3, 204–228
https://doi.org/10.1080/23742917.2023.2261169

Taylor & Francis
Taylor & Francis Group

A Case Study of Cyber Subversion Attack based Design Flaw in Service Oriented Component Application Logic

Faisal Nabi[a], Xujuan Zhou[b], Umna Iftikhar[a] and Hafiz Muhammad Attaullah[a]

[a]Faculty Computer Science, Mohammad Ali Jinnah University, Karachi, Pakistan; [b]School of Business, University of Southern Queensland, Springfield, Australia

ABSTRACT

Modern e-commerce systems are more likely focused on mechanisms of security, such as secure transactional protocols, cryptographic schemes and parameter sanitization, and it is assumed that putting these in place will guarantee a secure e-commerce application. However, vulnerabilities in the business application logic itself are often ignored which can make the effect of these security mechanisms null and void. Essentially, the weakest link can be at the server rather than client because of business logic and insecure server-side business components, its security ignoring is another factor, which is done at developer's peril. This paper focuses on the weakest link (component's logic subversion) in the e-commerce system. We outline a logical attack (subversion attack, class Design Flaw) that would not be prevented by the deployment of the mechanisms commonly used in e-commerce systems. To further investigate this problem, we propose a security assurance methodology for service component-oriented application that will be practiced through threat modeling and component fault detection model with further modeling component and its application using unified modeling language secure-design approach with a valid technique (verification, validation model for security-by-design testing) for design flaw detection to avoid the business logic problem in component-based e-commerce applications from existing application logic.

ARTICLE HISTORY
Received 22 August 2023
Accepted 17 September 2023

KEYWORDS
Design flaws; subversion attack; e-commerce system; service component architecture assurance

1. Introduction

There is no clear difference between service-oriented architecture and component-based software architecture because, by the rule of implementation, it is the enhancement of components, where single components are represented as service, which is connected to develop a new business logic. Application business logic describes a particular 'service' (such as a shopping basket) offered by an application [1] [2]. We are interested in improving the security of applications built by integrating components

CONTACT Faisal Nabi faisal.nabi@yahoo.com Faculty Computer Science, Mohammad Ali Jinnah University, Karachi, Pakistan

together to develop a service [3] [2]. There have been several attacks at the application level that bypass the security mechanisms implemented.

At the level of the application infrastructure for secure software development, for example, an online shopping service that allowed customers to order goods and cancel but still receive the goods. This negates work done on improving the security of the infrastructure [4]. Techniques do exist for secure software development, but they are not necessarily aimed at component-oriented software and they do not take into account particular risks associated with rapid development techniques [5] [6]. Similarly, existing software testing techniques that assist by helping and verifying application design at design stage also develop tests that can be used to validate the implementation, which in fact still respect the constraints of the design. However, the biggest challenge is that these have not been used to evaluate the secure design of an application [3]. A business process can be attacked even when a very good network and infrastructure security programme are in place (Sharon Nachigal 2009). For example, good network perimeter defence using firewalls, honey pots, intrusion detection systems and other network security components must still ensure the applications can be accessed by legitimate users, and therefore at the same time can facilitate an opportunity for legitimate users to attack the organization business information systems by abusing the vulnerable e-commerce business process at the application interface level. The application logic and business process build on the basis of two blocks: business logic and information flow (Faisal Nabi, 2017). Most of the research community worked on information flow and developed a business-process-oriented security approach [7]. The challenges addressed in this approach related to e-business security but did not address the business processing logic of a component at the design stage, when components are integrated on the base of its business logic.

In the case of a Web, application and its services are custom developed, and the underlying components are referred to as core service logic (business logic). Typically, a façade component spearheads the core service logic by providing the technical interface that is used by runtime processors to translate message data to and from the messaging format supported by the WSDL. The design of core service logic (application logic) is affected by the application of service orientation. Service Loose Coupling, Service Autonomy, Service Statelessness and Service Composability in particular represent design principles that introduce several specific design characteristics into core service logic and its underlying implementation [6].

1.1. Problem statement

In this paper, we have discussed the challenges of design flaws that cause subversion attack in the component-based application logic, in n-tier applications.

The weakest link (component's logic subversion) in the e-commerce system is the subject of this essay. We describe a logic attack (subversion attack, class Design Flaw) that cannot be stopped by implementing the typical e-commerce system defenses. To further explore this issue, we suggest a security assurance methodology for service component-oriented applications that will be put into practice through threat modeling and component fault detection models with further modeling of the component and its application using a unified modeling language (UML) secure design approach with a valid technique (verification and validation (V&V) model for security by design testing) for design flaw detection to avoid the business logic issue in the component-based rapidly developing e-commerce.

1.2. Objective and contribution

This paper introduces three key novel contributions:

(1) Clarifying the Impact of Component Role Analysis on Design Flaws: The paper explores a significant question: how can the reuse of design specifications based on incorrect component role analysis lead to design flaws and subsequent security issues at the application logic level? This question is thoroughly examined, shedding light on the potential risks and their underlying causes.
(2) Real-world Case Study: A comprehensive real-world case study is presented, featuring an e-commerce component-based banking application. This case study serves to vividly illustrate the identified issues, providing a tangible context for understanding the implications of design flaws stemming from incorrect component role analysis.
(3) Practical Security Assurance Methodology: The paper proposes a practical approach for enhancing security assurance in service component-oriented applications. This is achieved through a comprehensive methodology encompassing threat modeling, component fault detection models and UML-based secure design techniques. Notably, the paper introduces a robust V&V model for security by design testing and then incorporates Java Crypto Algorithm (JCA) SHA 256 technique for enhanced security. This approach effectively addresses the challenge of customizing security assurance requirements in modern component-based applications, contributing to the detection and prevention of design flaw issues arising from potential attacks.

2. Impact of flaws in application business logic

The complexity of web applications increases the risk that developers may introduce flaws that can be exploited by attackers. We believe that a key problem is that web-based applications integrate numerous diverse

components from disparate sources, including custom-built special-purpose applications, customized 'commercial-off-the-shelf software components and third-party products' [5]. The integration of these components is not straightforward and essentially blind reuse can lead to the introduction of application-level flaws [8]. These types of problems are higher level than software faults such as buffer overflow problems and relate to the business logic, for example performing authentication at the right time or correctly implementing cancellation policies such that the application's overall security policy is enforced [4]. Examples from the literature include flaws allowing fraud to be committed leading to personal enrichment of attackers by exploiting flaws in business logic [9]. Although we commonly focus on low-level coding problems, these types of high-level software flaws are estimated to represent 50% of all software flaws that are security [10].

We define web software application vulnerability definition as 'web software application vulnerability includes mismatches between application software architectural/design logic and the assumptions about the environment made during the development/implementation (code writing), operation of the program, and the environment in which the program executes' [7]. For example, a component may have been designed by its developers with assumption that all accesses will be authenticated, but it may be reused in a context where pre-authentication takes place, and the assumption is that this logic has been implemented within the component. This is a trivial example, but our case study provides a more detailed example of problems of reuse when the users of components make assumptions that were not made by the developers.

3. Real-life example: Barclay Bank case study

This real-life case is a good example of a design flaw in application logic due to the reuse of a component-caused component subversion. In this example, the developer reused the same component that was already incorporated in the registration functionality elsewhere within the application and violated the assumptions of the component developer. This mistake leads to the introduction of an application-level flaw that allowed an attacker to access other clients' bank accounts. (Source of this case is taken from Barclay Bank IT section of application development and maintained through the method of walk-through technique, and further details are not provided because of privacy act of banking policy.)

3.1. Application functionality and business process

Customers who did not register for online application are encouraged to do so. For this purpose, the new users are required to provide basic personal information, such as name, address and date of birth, but not any secret information like passwords/PIN numbers, which ensures the degree of assurance of their identity. When this

information is processed, the application sends a back-end system a registration request to be processed. Upon successful registration, an information pack is mailed to the registered user's home address. This pack provides information about the online activation access via telephonic call to the company's call center and the use of a one-time password to log in the system.

The designer of the application believed that this mechanism would protect the unauthorized access to the application. The process of security is implemented in three ways of protection mechanism.

- At the initial stage, some modest amount of personal data are required in defence to judge a malicious attacker or troublesome user who attempts the initial registration process on behalf of other users.
- During this process, a key secret is transmitted out of hand: the registered customer's home address. The attacker needs to have access to the victim's personal mail.
- In order to authenticate himself, customer required to telephone call center as it is the normal way, based on personal information and selected digits from a PIN number. The design was supposed to be well defended but the logical flaw was in actual implementation of design mechanism. The personal data is to be stored which is based on correlate this with a unique customer identity in a company database, and the developer needs to develop the registration mechanism. The developer was eager to reuse existing component code that was already used within the application somewhere else.

```
class CCustomer
{
   String
   firstName;
   String
   lastName;
   CDoB dob;
   CAddress
   homeAddress;long
   custNumber;
   . . . }
```

After this process is completed, this object is instantiated, inhabited with provided information that is stored in the user session. In this process, application verifies user details, and if it matches, it retrieves that user's unique customer number which was used within company's system. This number is added in the Object with some other personal information. This Object communicates with the back-end system for completion of registration request to

be processed. The developer supposes that using this code would be harmless and would not be reason of any security problem. However, it was a serious mistake that caused flaws in the actual design.

3.2. Exploitation

The registration functionality incorporated with the CCustomer component that consist as '(use case logic + Process and Entity Type Logic)' within the application, including core functionally. This process allows the user to authenticate and grant access to the application components such as 'Account details component', 'Statement's component', 'Funds transfers component', 'Debit component, Credit component and other information component'. After having authenticated user itself to the application through registration process, the same Object instantiates and saves in the session key information related to the identity.

The components of application within functionally referenced information related to the *CCustomer (Component)* object in order to carry out its actions because *CCustomer (Component)* object is a candidate component (Process and Entity Type logic) within the majority of application. For example, account details shown on the main page of the user were generated based on customer unique number that contained within this component. In this way, composition or reuse of component code was already used within the application. It clearly shows that developer assumption leads to a flaw in reuse of application logic design. This caused the birth of vulnerability of subversion attack on application business logic. It was a serious mistake, and it is subtle to detect and exploit.

To exploit this logic flaw, an attacker may need to perform the following steps: (as shown in Figure 1)

- The first step is to 'log in' into the application using own valid account information.
- In a result of the authenticated session, and access of registration functionality, try to input some other customer' personal information. This will result in overwriting the original 'CCustomer' (Component) object in the attacker session with new object of targeted customer.
- After this process, get back to application functionality and try to target another customer's account. It is hard to detect without clearly understanding the application logic as a whole and the use of different kinds of components in the application logic layer. The flawed assumption by the developer caused subversion attack class vulnerability.

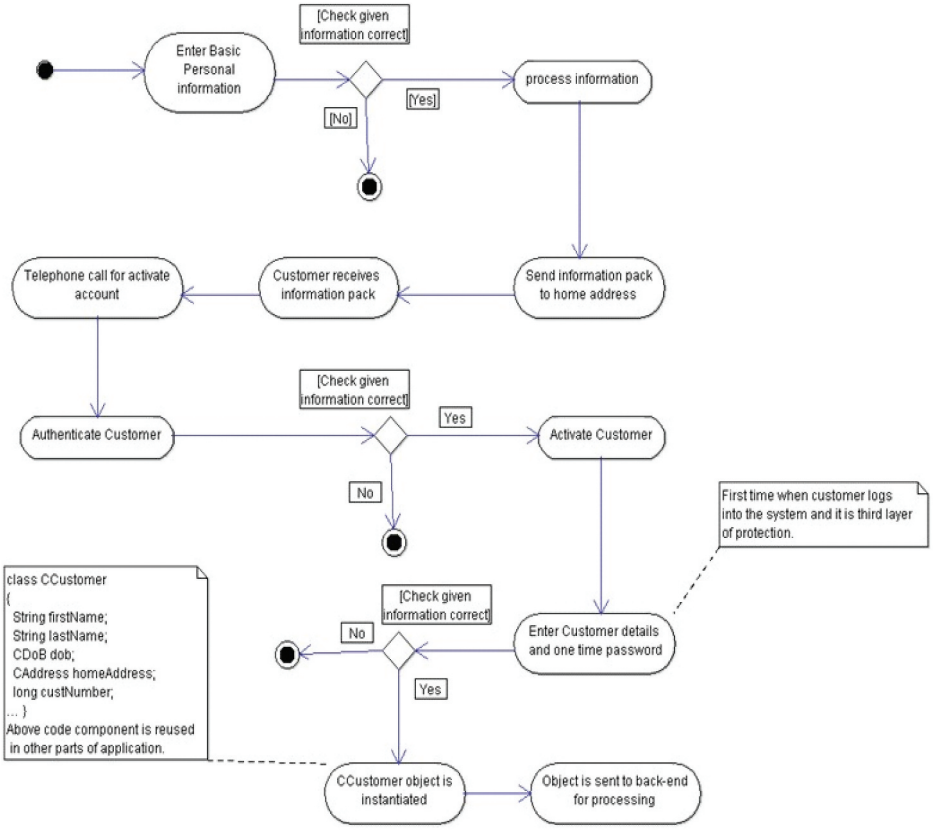

Figure 1. Customer information handling and reused component business process.

Although the vulnerability stated earlier was serious, it was in fact relatively subtle for intruders to detect and exploit. Access to the main application functionality was protected by access controls at several layers (channel level security) and a user needs to have a fully authenticated session to pass these controls. A second security defense was fraud management. This process is defined as table in (as shown in Tables 1 and 2).

Table 1. Advantages and disadvantages of existing works.

Aspect	Advantages	Disadvantages
Innovation	- Brings new ideas and concepts to the field.	- May lack practicality or feasibility.
Experience	- Builds on prior knowledge and expertise.	- May be influenced by outdated practices.
Efficiency	- Can streamline processes and save time.	- Implementation costs and learning curve.
Benchmarking	- Provides a basis for performance comparison.	- May limit creativity and originality.
Validation	- Tested and proven solutions reduce risk.	- May not address unique or novel challenges.
Collaboration	- Encourages collaboration and knowledge sharing.	- May lead to groupthink or lack of diversity.
Cost-Effective	- Can save resources compared to starting from scratch.	- May require licensing fees or royalties.

Table 2. Summarizing the main differences between various attacks.

Attack	Focus	Goal	Detection Difficulty
Subversion Attacks	Application Logic	Undermine intended functionality	Challenging
Circumvention Attacks	Security Controls	Bypass security measures	Requires expertise
Business Logic Attacks	Core Functionality	Exploit flaws in business processes	Challenging
Injection Attacks	Database Queries	Manipulate or extract data from databases	Moderate
Cross-Site Scripting (XSS) Attacks	User Input	Inject malicious scripts into software pages	Moderate
Denial of Service (DoS) Attacks	Resource Availability	Overwhelm and disrupt soft system services	Varies depending on sophistication
Phishing Attacks	User Trust	Deceive users into revealing sensitive information	Can be challenging

3.3. Overview of the proposed strategy as compared to other three industry methods

In light of the following, how might issues with the reuse of design specifications based on incorrect component logical role analysis result in design flaws at the application logic level security issues? First, we addressed the issue raised in the question when considering the case study situation for a banking application, and we then suggested a security assurance approach to resolve the issue. The technique of identifying vulnerability in the Banking Case is achieved via matching a sequence of components in a system's application design logic and problem caused by ignoring business process integration of components at the time of the application's business process logic [17-18].

All three methods of penetration, which are being practiced in the industry, rely on modern software development and are mostly based on SDLC and controlling security at an early stage but could not address the security by design methodology aspects of novelty, *which is motivated by the fact that logical flaws do not show attack patterns or signatures and, thus, their discovery cannot be automated*. Our main contribution is based on section contribution 1.3.

Our solution involves using Threat Modeling to tackle the issue, projecting design flaws related to subversion attacks and application logic vulnerabilities. This approach is supported by practical component and application modeling using UML for secure design. We employ a robust design flaw detection technique, the V&V model for security by design testing, to prevent customization challenges in security assurance requirements for modern component-based applications. This integrated approach enhances security and resilience against design flaws, ensuring more secure software systems.

4. Proposed security assurance methodology

A security assurance methodology is designed to overcome the problem of design flaw (service component-oriented application logic) in e-commerce

applications, which analyzes the security risks related to determine possible attacks on system design, and on the basis of that, three stages are defined to deal with the design-based flaw problem. The key element of this methodology consists of formulation of well-established existing approaches that help to design a new methodology: (1) threat modeling, (2) taxonomy of software vulnerability model and (3) component fault detection model. This further provides support to modeling the application and its components without fault, which then leads to designing security by design application modeling. This methodology is proved through the V&V security assurance process followed by V&V model for security by design testing approach, which is theoretically justified by designed model. Please note that the contingency and probability models are out of scope based on Case Scenario that is why it is not made a part of a solution strategy as shown in Figure 4.

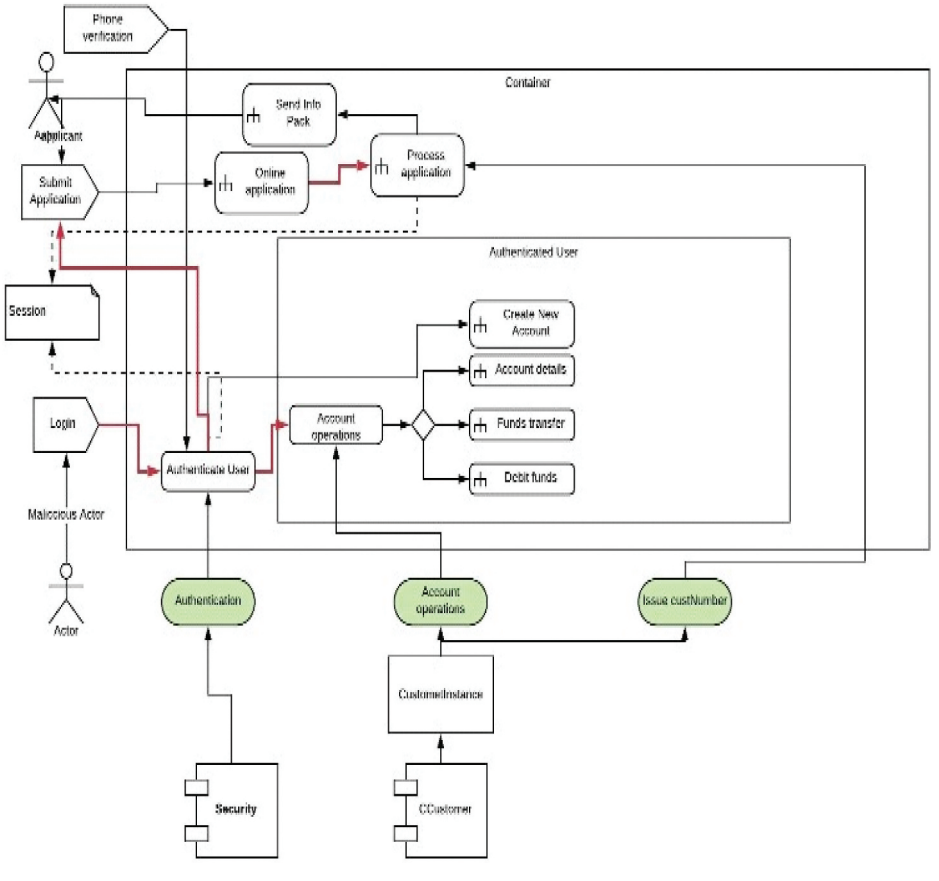

Figure 2. Event – attack – modeling and system exploitation in Barclay Bank.

4.1. Security risk analysis

The most important step for security risk analysis is to determine possible attacks on system design and their consequences when successful, such as the aforementioned case of e-commerce component-based web software application and discovered logical vulnerability in the application layer of business-tier.

As attacks are going to be very specific to the particular applications, one of the first steps in system design should be the analysis of the possible attacks on specific system and their consequences when successful, such as the aforementioned case of e-commerce component-based web software application and discovered logical vulnerability in the application layer of business-tier. The technique of identifying vulnerability achieved via mismatching a sequence of components in a system's application design logic and problem caused by ignoring business process integration of component at the time of application's business process logic (which can be mapped through scenario-based approach business process flow as mentioned in Figure 3, which represents a basic end-to-end system function, also decomposed into sub-scenario, which identifies functionality of important sub-system's component) that permits the sequence of 'Event Trigger' in the attack pattern to occur analysis of the description mentioned in light of case study and vulnerability attack pattern reveals the event that transpire, what component is used to exploit the vulnerability in Barclay Bank case. This analysis can be used to define the countermeasures that need and will also be useful later to evaluate the system security.

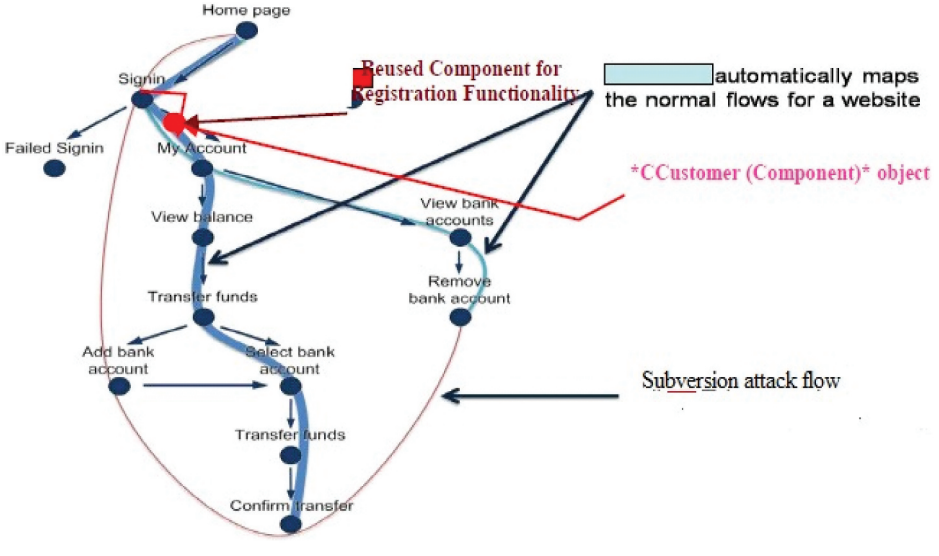

Figure 3. Subversion attack mapped through the business scenario-service flow.

4.2. Threat modeling of application logic vulnerability

Threat modeling is an engineering that can be used to help identify threats, attacks, vulnerabilities and countermeasures in the context of application scenario. In light of the aforementioned case study, we have examined class of vulnerabilities in the application logic, these attacks which we classified in this research Falls into the category of Design and Architecture Flaws based on Logical, Design and architectural division of application logic. We propose a *Method of Threat Modeling* approach to uncover the pattern of attack in the application logic from the root cause, integrating information from the case study of security breach. The threat model of an Attack Event and Attack, target method, is defined as a flaw in logic and flaw in design: At this stage, a logical attack is defined by some attacker access to targeted system under-attack that acts an illegitimate action by using an attack method (Subvert logic/Circumvent logic). A specification of an event attack method is to circumvent the logical flow of an application which may cause further two steps: one is subversion attack and second is circumvent actual application's logic that results in Business Logic Attack. The vulnerability is then described by some significant attack method (logical flaw or Design flaw) and used to identify the threat that can be associated with the components and to the whole system. It is also important to notice that growing use of third-party software components and middleware represents one of the biggest changes in the web-software-application systems so as Security; integrity has threat because of the flaws in the design (as shown in Figure 5).

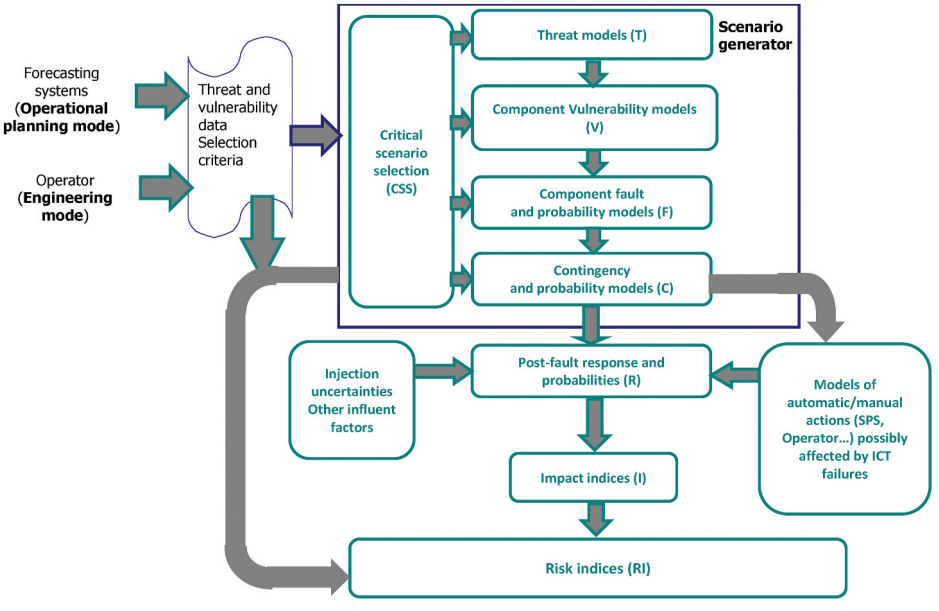

Figure 4. Security risk analysis model.

The model in Figure 5 is explained the relationship between CBS-based custom-developed components and COTS wrapped these components for web software application. This combination of development generates risk that attributes three main class of attributes: reliability, functionality and availability. These three pillars are main focal points for security of such applications, which often fall under integrity matters that cause threat. This threat gives birth to most serious flaws in design called flaws logic that can be in design or logic, which generates attack to subvert and circumvent the actual logic of application in web software. This is called business logic attack. Our model helps developers to identify flaws at design stage.

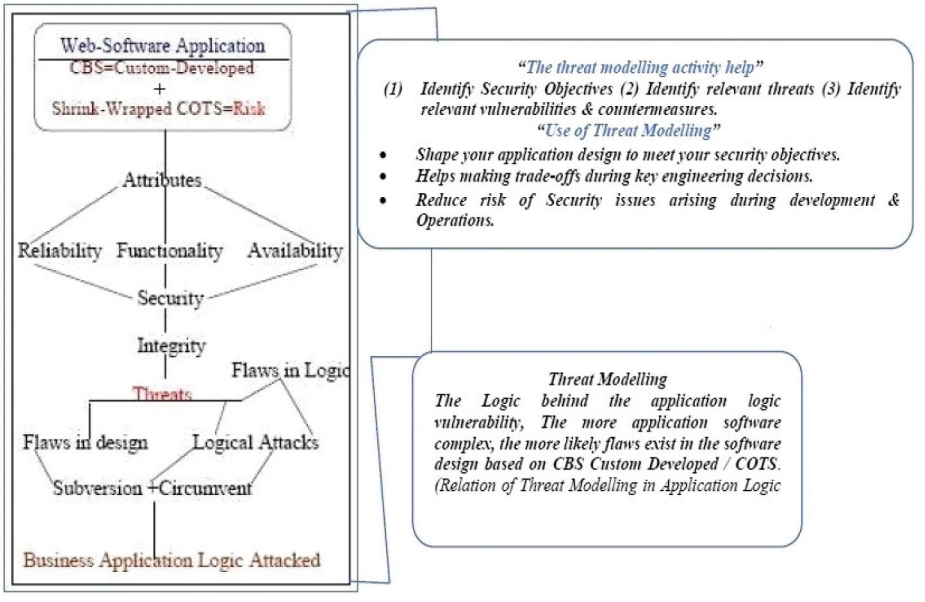

Figure 5. Threat modeling of subversion Attack.

4.3. Taxonomy of software vulnerabilities model

Software vulnerability defects cause security breach problem which as a result returns as software loopholes (Design Flaw, Coding Fault, Configuration Error). We believe, in light of our research work, that a risk assessment in component-based software web applications and systems is strongly connected with some concepts traditionally derived from the field of computer security, in particular 'Five Pillars' are the attributes/elements of interest that need to be defined: the concepts of Threat, Vulnerability, Attack, Risk and Asset [11]. Figure 6 clearly depicts the causes of security breach in relationship to missing question as explained. The taxonomy is based on five stages of software vulnerability, discovered by the researchers.

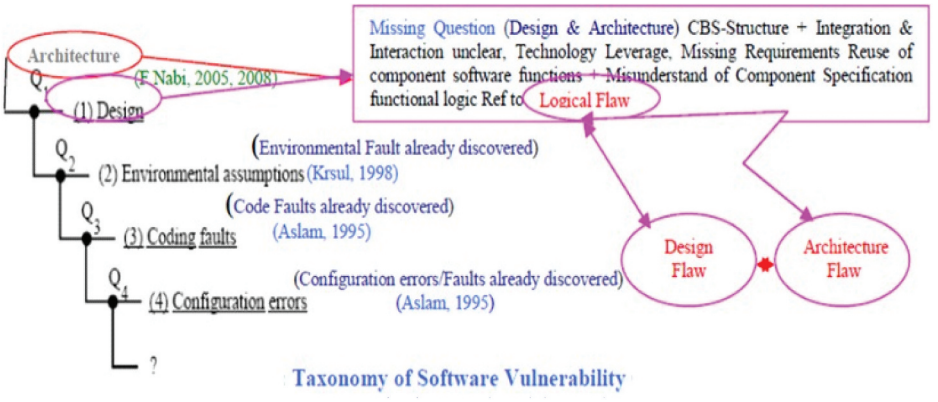

Figure 6. Taxonomy of software vulnerability mode.

Flaws are the software problems that exist in the software's design. A flaw may or may not represent vulnerability in the underlying software. Mitigating a flaw typically involves significantly more effort than simply modifying a few lines of code [3]. The problem does not lie solely in the implementation; the underlying design is *Flawed*. Therefore, any implementation that follows the design would contain the flaw [7]. For example, performing sensitive business logic in an untrusted client application is a design flaw that cannot be mitigated by a simple measure such as modifying array bounds.

4.4. Component fault detection model

Component fault detection is basically the state of the art in paradigm of security by design technique. However, fault detection is a preliminary element of fault tolerance technology. Therefore, fault detection and diagnosis-based fault tolerance technologies are hard to define separately. So many approaches have been introduced, such as fault detection method, based on the system's internal data exchange and inter-component communication, but no approach covers the system's broken component risk analysis, especially when code is not available. Therefore, under such circumstances, a technique is required that can model the component and system design on multi-tier specification through the UML modeling.

We designed a model for component fault detection scheme that detects faults on the base of component design specification which is tested through test requirements and diagnostic specification of component, as mentioned in the diagram. It divides the model into system tier and component tier, where component 1 to component N is modeled based on the stated specification, which then test and diagnose the overall design specification of the whole model for fault detection in system design or component.

Validity of this proposed CFDM is projected through the V&V model for security by design Technique in section 4.3.

4.4.1. Novelty of CFDM as a practical example

In order to test the component fault detection model, it is important to address the practical testing example. For this purpose, we have chosen an air control system. The applicability of the proposed CFD model fault detection is practiced through an air control system. This is based on two main components, operator user interface component and control station component, in which sub-components are component 1 which is ON and Component 2 which is Off connected with a contract point where there is another component 3 that supports the system having assumption of fault tolerance. The applicability of component function gets failed during the operation to communicate with the main component in the system as depicted in the model.

That shows fault in the system detected through UMLsec 2.0, keeping in view design specification based on test requirements and diagnosis specification of component and system. At this stage, where system design is considered as a whole model as depicted in the component and system fault detection model, which illustrates practical application of fault detect in component and system.

5. Modeling the application and its components without fault

CBSE (Component based Software Engineering) focus on different aspects of software engineering, for instance, different phases (in the design phase, components as reusable design parts; in the implementation phase, components confirmed to a specific component model, at run time, binary packages, distributed components), business aspects (business components, service components, COTS components), architectural issues (UML components) [12]. To enable the application and its components to be modeled, we must ensure that every aspect of the application's design is clearly and sufficiently detailed to understand every assumption and designed functional logic within the application by designer.

Mandate that all components of an application should be clearly commented to include the following information throughout to make security assurance of application and its underlying logic.

- The purpose and intended use of each component (if the component code is available, the code can also be provided, if not, its functional business logic within the component by defining a usage contract).

- The assumptions and logic made by each component about anything that is outside of its direct control.
- Reference to all client-component which makes use of the component clear documentation to this effect could have prevented the logic flaw within the online registration functionality in the Barclay Bank example (note: client here does not refer to the user-end of the client–server relationship but to other component (code) for which the component being considered is an immediate dependency).
- There are three main steps making up the risk analysis to assure security requirements for component-based applications. The first is to determine possible attacks on system design, the second is architectural risk analysis for component-based business logic security and the third is analysis at the component level.

6. Designing security by design application modeling

Designing of application for e-commerce distributed system in a tier is also very important since many attacks are caused by design flaws (subversion of attack) as mentioned in the aforementioned case study of e-commerce application. These logical flaws do not often refer to component-based flaws but also architectural, where component modeling has to set the logic of application while using business rules related to the business or activity. Therefore, it is important to clearly define architectural design of topology in which system is going to design to deploy by separating each tier clearly. The second stage focuses on the application logic design strategy andpolicy with that component must function under given business defined rule/policy. The third stage refers to design strategy for components in which dynamic web content is used to tailor an individual's interactions with a website and provide users with more interactive information. Dynamic content may be rendered in various forms, such as static HTML files, Java Script or JSP file, rendered using component-supported environment such as Java servlets in a J2EE that invokes business – logic application hosted middle tier to access back-end business data.

In a normal scenario, an application developer considers the approach, in which components are combined to be assembled for a particular business requirement and to create a solution. A composite application consists of new components that are created especially for business applications and existing components that are reused from other applications.

To theoretically justify the approach, the concept of Service Component Oriented system development technique is basically a collective set of specifications that propose programming model for developing applications and systems. This promotes the previous approaches to implementing services and supports the developing open standards like Web services.

Therefore, keeping in view theoretical justification, if we dissect a component structure, a simple component has two states: one is 'Service' and the other is 'Reference'. A service is considered that addresses interface for component which keeps one or more operations, whereas a reference is a dependency based on a service (functionality) that is required by another component.

Composite applications are created on the base of (SCA) stated specifications. The concept supports that a composite application is developed by combining one to more components, which then develops the business logic for a new application. A component consists of application program that creates business logic and related configuration information. The developer can deal with the same application program while adopting different configurations to form different components. A component role is to provide a service to other components, where in return it consumes functions offered by other services which uses service oriented interface [13].

Therefore, we have modeled a system (Figure 10) in a scenario, in which technique consist of system and component two different levels. The system level focuses on design product, while specification for individual component that participating in system development is modelled as UMLsec design approach. This is a multi-specification architecture of component-based system in which different layers of components are modelled based on their business rule and process, execution within the system. The component integration strategies at run time are compared to its requirement and design specifications.

Component-oriented programming concept promotes the implementation of service, where security by design technique supports the UML-based design modeling for system security early at design stage or while reusing the component from existing application. For this purpose, we have practiced UML-based security by design system modeling through the concept of Service Component oriented software engineering. In this exercise Business Domain Components and service Components are being modelled, which is based on multi-tier specification of architectural design of topology in which system going to design for deploy by separating each tier. The first tier consists of Business System Interface which relates to a service component (rendering logic) that correspond to the second tier Business Component and Application Component. From this stage mid- tier service is invoked from Business EDC and sub-component as realized use case. Furthermore, this process corresponds with the back-end service and application components. These components provide Consumer Service and Business Component service provider in order to process system function (defined as component's business logic), as depicted in the figure given below. The component integration strategies at run time are also clearly defined through inter-connections among all system and application components, while considering requirement and design specifications of service description.

7. Case-study-based research method

Case-study-based research plan is also called exploratory cases for experimentation [14]. The exploratory research case study investigates well-defined phenomena (business logic vulnerability) classified by scientific detailed research formulated event-based attack modeling approach for test generation that can be tested within the research environment using exploratory case study method. This sort of case study is very often applied as an exploratory research design exploring a completely new field of scientific investigation [15]. In light of given explanation of exploratory case study method, as shown in Figure 9, the research design will be further extend by exploring real life case study from that we take out the test design for service-oriented web-based banking system. This will follow further stages, such as the component integration strategies at run time will be compared to its requirement and design specifications (as can be seen in Figure 7b–8), while modeling component and its application. Further step is, considering attack event scenarios, which is based on connections between business components and its workflow, to analyze security risk related to case (as shown in Figure 2). Therefore, considering component secure method practice, multi-tire specification scenario modeling will be practiced using UMLsec 2.0.

There are three case studies that usually studied in computer science three main types of case study: intrinsic, instrumental and collective. We are taking collective-based exploratory case technique. The technique is further divided into two phases of process. The first phase represents system tier, while the second phase represents component tier. The first phase, system tier, focuses on design product, while the second phase considers the design, test and distinctive symptom specification for individual component that participates in system developing (as projected in Figure 8). The next phase is to consider the security assurance approach that helps to detect integration-based logical flaws in system which is proved through the validation and verification security assurance process followed by the V&V security using design testing model, which is theoretically justified by the designed model for security using design testing technique. This will validate the proposed solution called Security by Design approach for component-oriented service and its application.

7.1. A validation and verification method

To validate and verify the security assurance process, the technique would follow the V&V model for security using design testing approach that would be set as a test-bed model, in which there is no need of all component realizations. Normally, models are available at earlier stage as compared to realizations. This technique makes the process easier and makes it possible for earlier detection of design flaw, especially in the case of comparing/matching the real-time system testing.

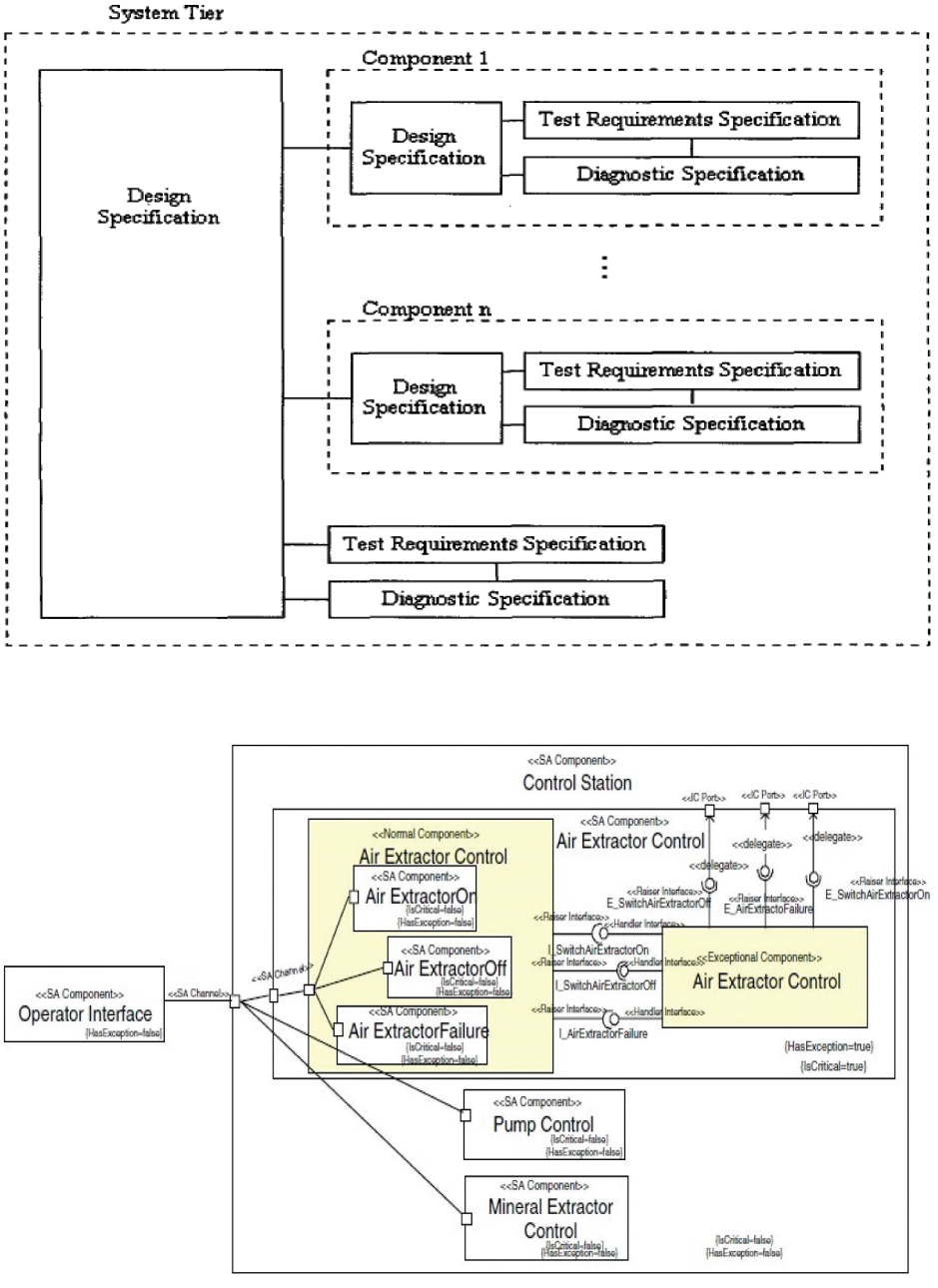

Figure 7. (a) Component fault detection model (CFDM).(b) Air Control system modeling by incorporating CFDM. The basic purpose of this example is to justify the aforementioned model in Figure 7a, keeping in view that CFDM incorporates the modeling application of Air Control system. The reason to do so is to exemplify the process of unification assurance process of component-based development system's in Java that helps to generate the security by design as well as logical functionality. One comparison of this component based system is to introduce Java-based design component ware software that is used in whether business prospective wise or some mission critical systems such as Mars Pollarding project 1999. Therefore, our technique is unique and novel, which is not only business-oriented software but also mission critical system application as well. This proves the significance of our proposed model in modern component-based systems.

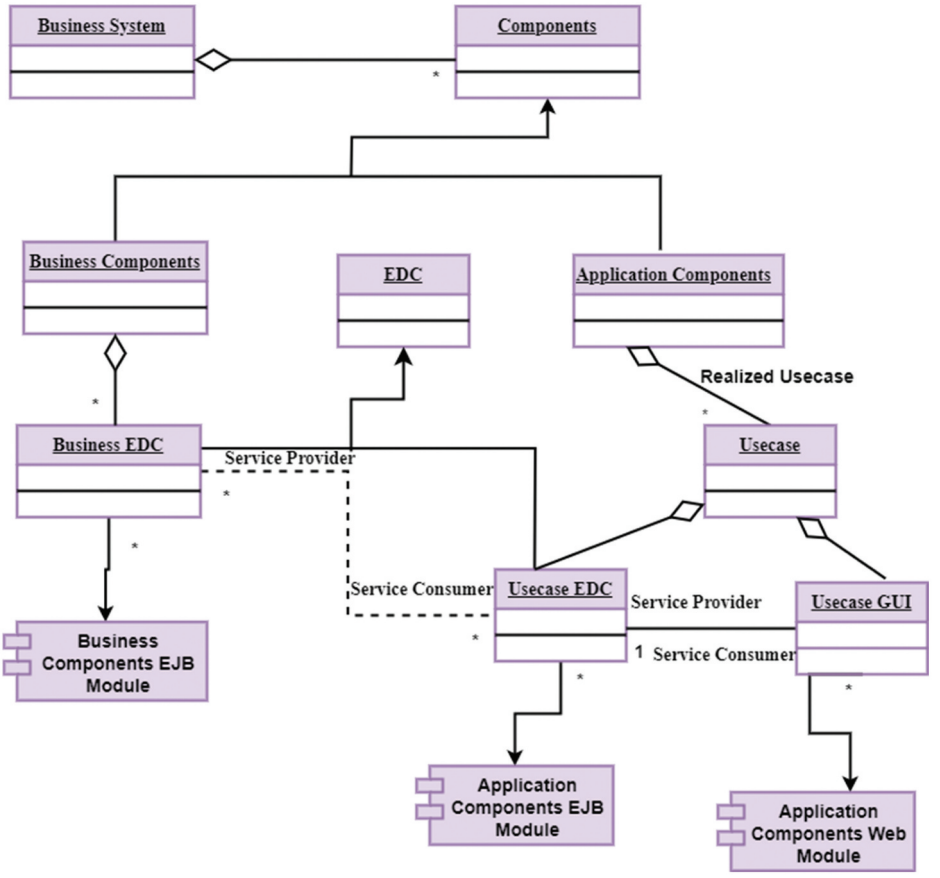

Figure 8. UMLsec 2.0 security by design approach multi-specification J2EE system modeling.

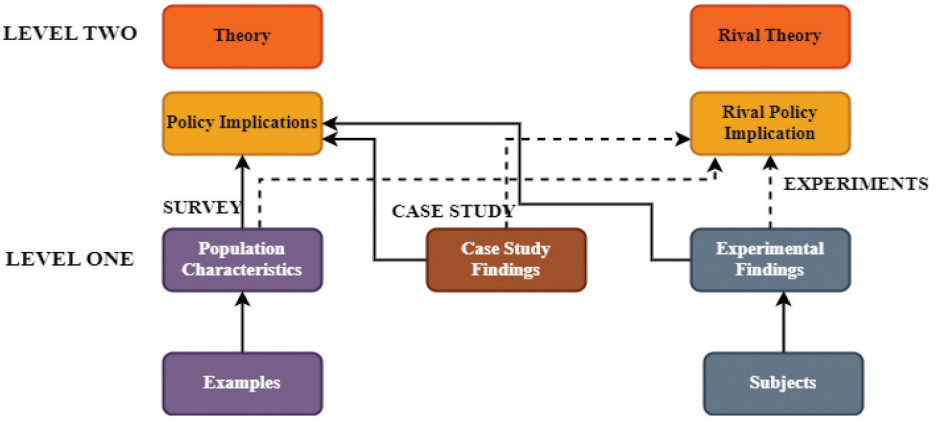

Figure 9. The case study model.

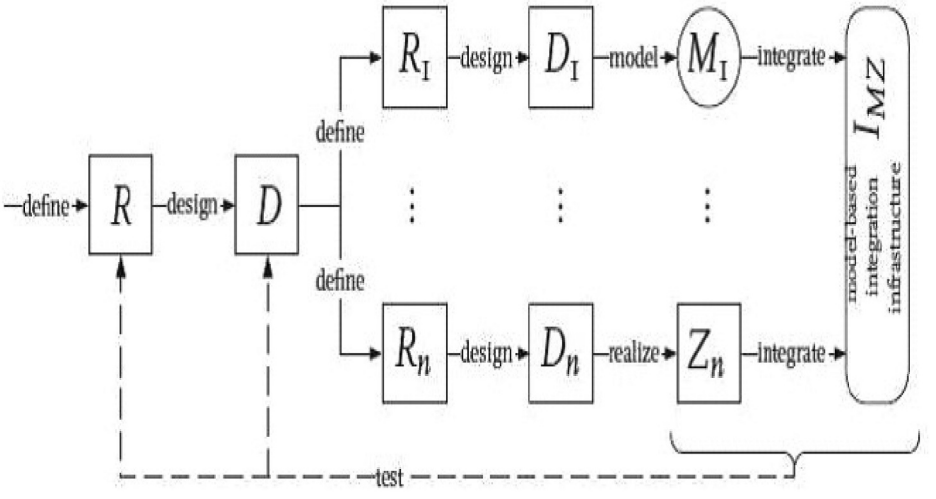

Figure 10. V&V model for security by design testing.

In addition, adaptation and configuration are normally allowed by model as compared to realizations that is appropriate for system testing in unrelated conditions for assurance purpose. It is easier that models are used as an alternative of realizations, especially when exceptional behavior testing, and when broken component are the condition.

In addition to that, test quality is improved and the ability of models at a great rate changes test conditions also improve the state of test execution. In return, it not only makes model-based testing simple and easy but also ensures less chances of risk during test process.

Theoretically justified by designed model security assurance process, diagram 11 illustrates the graphical display of V&V model for security using the design testing approach. It consider the components of C1 and Cn those are only illustrated, above the figure depict component *C*1 is shown/referred by *M*1 model, whereas *Cn* is shown/referred by realization *Zn*. *IMZ is* an infrastructure integration model, and both *M*1 and *Zn* need to be integrated, which refers to model and its realization so that equation is made {*M*1, *Zn*} *IMZ*. Therefore, this initial depict of the system considered for test at system early stage which extract from *R&D*, system requirements, and design, that is shown by the dashed arrow. The R system requirement and D design specifications are captured in order for M1 model and Zn realization to be tested using integration of IMZ for infrastructure integration model. This process of the model justified the security assurance service component-oriented applications in context of aforementioned process related to the methodology.

Hence, in light of V&V model, we have exemplified a case scenario of 'Domain Model System' in correspondence with the CFDM for its applicability into the proposed V&V model, in the context of aforementioned process

related to the methodology. That is demonstrated with the UML modeling to detect the flaw or fault in component or system design. The diagram illustrates model as Core System and Model Threat, which then constitute the component- and sub-components-based system. In this process, fault is detected keeping in view the equation {*M*1, *Zn*} *IMZ,* where all components realization is not required, especially when exceptional behavior testing and when broken component are the condition. The requirement R and design D of the Domain Model system information are gathered with specification of M1 and Zn realization with, respectively, C1 and Cn to be tested using integration IMZ. The threat model also explains the fault cause of error in the process of communication (event-based messaging) that gets failed and may cause of hazard vulnerability in underlying application logic. The technique of identifying vulnerability in the Domain Model System is achieved via matching a sequence of component and its sub-component in a system application logic and problem caused by ignoring business process integration of component at the run time of which resulted fault or design flaw.

In light of Figure 11a,b, the process clearly defines the integration of security functional requirements to enhance the security as defined encryption stages in this figure. This constitutes the application of security integration features that support the functional logic SSL support.

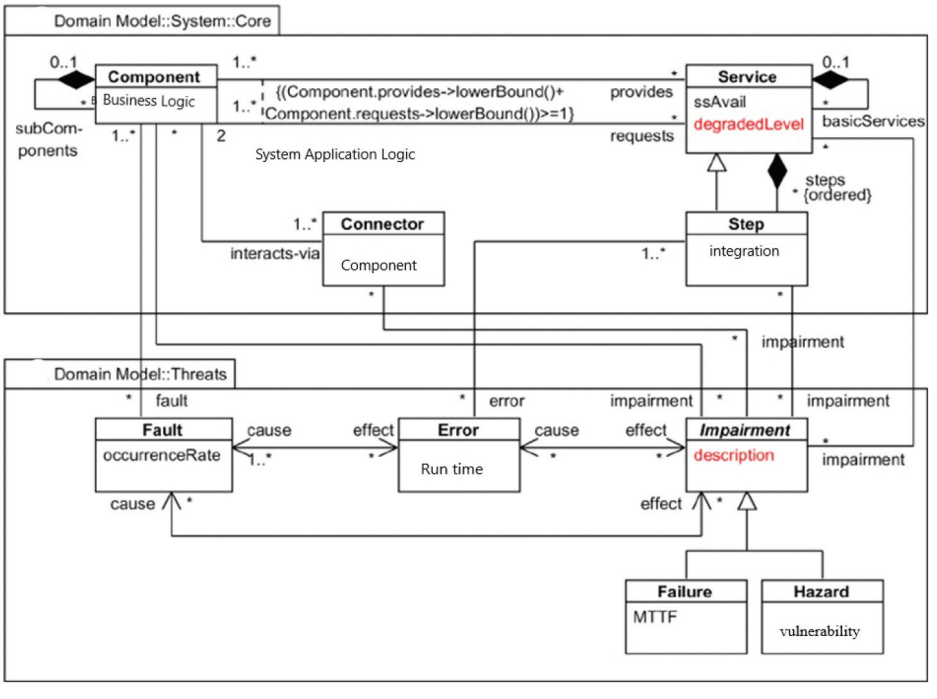

Figure 11. (a) Application of fault detection in correspondence CFDM.(a) Java crypto algorithm process to encrypt and decrypt business logic functional communication method.

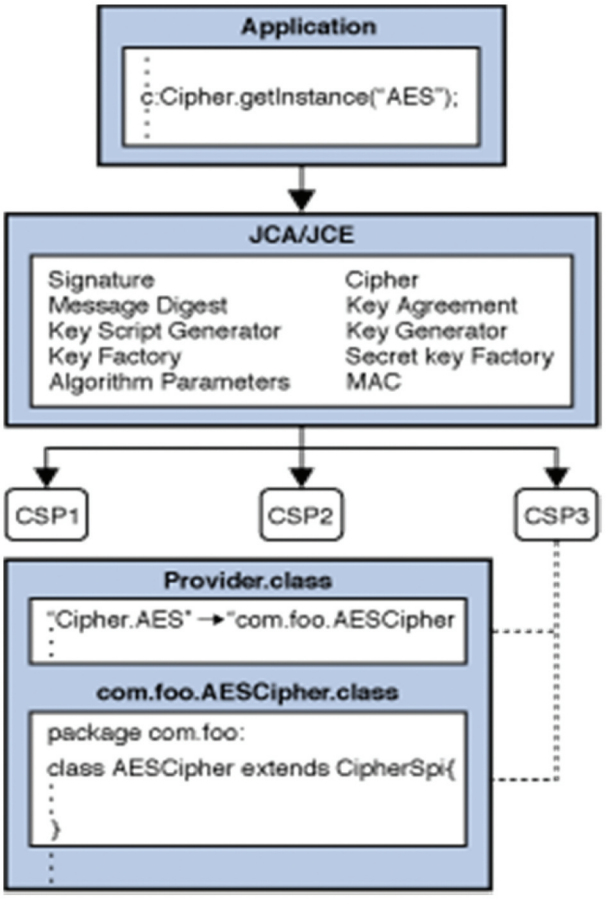

Figure 11. (Continued).

To use JCA, an application simply requests a particular type of object (such as a Message Digest) and a particular algorithm or service (such as the 'SHA-256' algorithm) and gets an implementation from one of the installed providers. Alternatively, the program can request the objects from a specific provider. Each provider has a name used to refer to it.

md = MessageDigest.getInstance('SHA-256');
md = MessageDigest.getInstance('SHA-256', 'ProviderC');

The following figures illustrate requesting an 'SHA-256' message digest implementation. The figures show three different providers that implement various message digest algorithms ('SHA-256', 'SHA-384' and 'SHA-512'). The providers are ordered by preference from left to right (1–3). In the first illustration, an application requests an SHA-256 algorithm implementation without specifying a provider name. The providers are searched in preference order, and the implementation from the first provider supplying that particular algorithm,

ProviderB, is returned. In the second figure, the application requests the SHA-256 algorithm implementation from a specific provider, ProviderC. This time, the implementation from ProviderC is returned, even though a provider with a higher preference order, ProviderB, also supplies an SHA-256 implementation.

7.2. Lessons learned from case study

The key point here is to consider that the component itself was correctly designed, as long as the design specification of the component matches with respect to boundary profile condition of each layer (*functional specification requirement*) in n-tier architecture of e-commerce application. This respects existing application logic in the overall system's function and its behavior [16]. The problem is that the designer of the component and the application developer are likely to be different individuals (indeed, this is the generally preferred scenario!). What is required over here, a technique of assurance unification process for validating the design before implementation to determine whether the overall application behavior is undesirable when the components are integrated with respect to component software artifact model and its functional processing logic based on interface driven design specification together to form a particular solution. This is quite different to technical implementation as the components integrated at this level are modeled through security risks, but their semantics and functional behavior were incompatible! The applicability of this learned lesson could also be beneficial for 'Department. of Defense Software Community' while designing component-based application solution by reusing component from existing application's logic, keeping in view design specification of different layers component and their role performance in the particular solution, such as Arian Rocket failure case (ESA), indicate that reused component software compatibility was proposed based on component software model artifacts but developer totally ignored the component interface driven established logical constraints by using 'used and offered interfaces' of component between the overall logical structure design by contract strategy during the composition, which caused failure to comply the solution and functional specific boundary profile condition compare to design specific boundary profile condition, result (ESA) Mission Arian failed.

8. Conclusion

The lesson derived from the case study is that it teaches the developers to always consider the key point that a component must be itself correctly designed, and as far as the design specification of the component is

concerned, it must be matched with respect to boundary profile condition of each layer *functional specification requirement* within the architecture, which respects existing application logic in the overall system's function and its behavior. Therefore, keeping this point in view, we have practiced a security assurance methodology for service component-oriented application through threat modeling and component fault detection model with further modeling component and its application using UML secure design approach and developed a valid technique for design flaw detection, which will increase level of assurance during the designing component-based rapidly developing web application software and deploying business logic into e-commerce system. The proposed approach would also support for the developers to design secure component-based application while re-using existing components from application's business logic. This will ensure a secure design-based modeling technique for security by design method.

Acknowledgments

The authors would like to thank Prof. Jianming Yong for support and cooperation.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

No funding is granted for this particular project.

ORCID

Faisal Nabi http://orcid.org/0000-0003-1804-2949

Availability of data and material

No data have been provided for that project as it is not allowed.

References

[1] Ghosh AK. Security & privacy for e-business. ISBN 0-471-384211-6: John Wiley & Sons; 2001.
[2] Nabi F, Yong J. Xiaohui Tao: A novel approach for component based application logic event attack modeling. Int J Netw Secur. 2020;22(3):435–441.
[3] Nabi F. Designing a framework method for secure. Business application logic integrity in e-commerce systems. Int J Netw Secur. 2011 Jan;12(1):29–41.

[4] Ritchie P. The security risks of Ajax/web 2.0 application. Network Security: Elsevier; 2007.
[5] Offutt J. Quality attributes of web software applications. IEEE software. March-April 2002;19(2):25–32. doi:10.1109/52.991329
[6] Nabi F, Yong J, Tao X. Xiaohui Tao: classification of logical vulnerability based on group attack method. J Ubiquitous Syst Pervasive Networks. 2021;14(1):19–26. doi:10.5383/JUSPN.14.01.004
[7] Agirre A, Parra J, Armentia A, et al. QoS aware middleware support for dynamically reconfigurable component based IoT applications. Int J Distrib Sens Netw. 2016;12(4):2702789. doi: 10.1155/2016/2702789
[8] Allan D, Web application security; automated scanning or manual penetration testing, watch paper from watch fire, 2006.
[9] McGraw G. Software security: building security in. Boston, MA:Addison-Wesley; 2006. Available from: http://www.buildingsecurityin.com.
[10] Zhang H, A formal security modeling and analysis in B2B e-commerce, [PhD thesis], Auckland University, 2006.
[11] Jones A, Ashenden D. Risk management for Computer security: protecting your network & information assets. Elsevier; 2005 March.
[12] Rodríguez M, Zalama E, González I. Integración automática de dispositivos en el Hogar Digital a través de la generación de adaptadores dirigida por modelos. Revista Iberoamericana de Automática e Informática Industrial RIAI. 2016;13(3):363–369. doi: 10.1016/j.riai.2016.03.007
[13] Agirre A, Armentia A, Estévez E, et al. A component-based approach for securing indoor home care applications. Sensors. 2018;18(2):46. doi: 10.3390/s18010046
[14] Singh K. Quantitative social research methods. SAGE Publications India Pvt Ltd; 2007. p. 64. doi: 10.4135/9789351507741
[15] Streb CK. Encyclopaedia of case study research. ISBN: 9781412956703. SAGE Publications, Inc, Print; 2012. pp. 1–10.
[16] Malohlava M, Hnetynka P, Bures T. SOFA 2 component framework and its ecosystem. Electron. Notes Theor Comput Sci. 2013;295:101–106. doi: 10.1016/j.entcs.2013.04.009
[17] Nabi F. Secure business application logic for e-commerce systems. Elsevier J Comput Secur. 2005;24(3):208–217. doi: 10.1016/j.cose.2004.08.008
[18] Agirre A, Marcos M, Estevez E Distributed applications management platform based on service component architecture. In Proceedings of the 17th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Krakow, Poland, 17–21 September 2012.