Scientific Research Publishing

Journal of Information Security, 2021, 12, 189-211
https://www.scirp.org/journal/jis
ISSN Online: 2153-1242
ISSN Print: 2153-1234

Process of Security Assurance Technique for Application Functional Logic in E-Commerce Systems

Faisal Nabi[1*], Jianming Yong[1], Xiaohui Tao[2], Muhammad Saqib Malhi[3], Muhammad Farhan[3], Umar Mahmood[3]

[1]School of Management and Enterprise, University of Southern Queensland, Queensland, Australia
[2]School of Sciences, University of Southern Queensland, Queensland, Australia
[3]School of IT and Engineering, Melbourne Institute of Technology, Melb Campus, Melbourne, Victoria, Australia
Email: *faisal.nabi@yahoo.com

How to cite this paper: Nabi, F., Yong, J.M., Tao, X.H., Malhi, M.S., Farhan, M. and Mahmood, U. (2021) Process of Security Assurance Technique for Application Functional Logic in E-Commerce Systems. *Journal of Information Security*, **12**, 189-211.
https://doi.org/10.4236/jis.2021.123010

Received: October 29, 2020
Accepted: April 10, 2021
Published: May 14, 2021

Abstract

Security practices such as Audits that often focus on penetration testing are performed to find flaws in some types of vulnerability & use tools, which have been tailored to resolve certain risks based on code errors, code conceptual assumptions bugs, etc. Most existing security practices in e-Commerce are dealt with as an auditing activity. They may have policies of security, which are enforced by auditors who enable a particular set of items to be reviewed, but also fail to find vulnerabilities, which have been established in compliance with application logic. In this paper, we will investigate the problem of business logic vulnerability in the component-based rapid development of e-commerce applications while reusing design specification of component. We propose secure application functional processing Logic Security technique for component-based e-commerce application, based on security requirement of e-business process and security assurance logical component behaviour specification approach to formulize and design a solution for business logic vulnerability phenomena.

Keywords

Business Logic Design Flaws, Components Integration Flaws, E-Commerce System, Assurance & Security, Model Based Design, Business Logic Attacks, Attack Pattern

1. Introduction

Application Business Logic: The business logic describes the particular "service" (such as Account Service by Account Component's business logic) offered

by business domain's business component, (this logical component can be part of sub-component or sub-system or system, both of which represent components in their own right) the steps required to complete or perform a particular action as defined by the business component-based application to automate business process; it uses logical component-ware and each component business logic makes a set of functionality by integrating these components business processing logic to make logical component-ware, which then makes overall application's business logic in the e-commerce system at middle tier [1].

Business logic layer within the application is developed with two kinds of components, Business Processing components and Business Entity Components. The "Business Process Components" handle the services or transactions that are requested by users through the user interface. They determine the operations of the business entity components that must be invoked and the order in which they must be executed. The "Business Entity Components" are persistent. They represent the business entity types of the application domain whose state must be stored by the application [2].

The middle tier contains wide range of components from different layers such as web components managed by web servers and business object components managed by application servers. The web component dynamically process user requests and constructs response to client application. The business object components implement business logic of a business domain. Both components are managed by an infrastructure environment such as J2EE or CORBA platform servers that provides important system infrastructure services for these components such as runtime environment for component compatibility, container management, security. Client components focus only on presentation logic, and business components provide business logic within a business domain as shown in Figure 1.

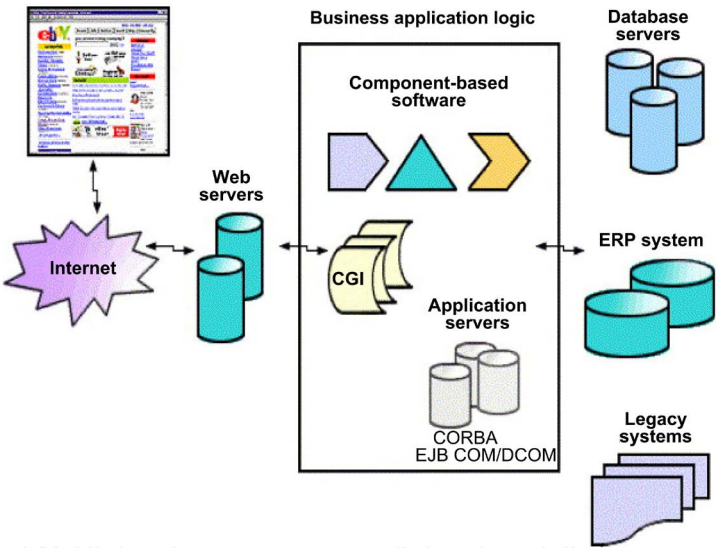

Figure 1. Middle tier of e-commerce servers that implements the business application logic [1] [3].

The middle tier of e-commerce servers implements the business application logic. The business application logic represents the functions or services that a particular e-commerce site provides. As a result, a given site may often employ custom-developed logic. As the demand for e-commerce services grows, the sophistication of the business application logic grows accordingly [1] [3]. Traditionally, e-commerce sites implement the middle tier of software on web servers using the common gateway interface (CGI). CGI scripts are programs that run on the web server machine as separate processes from the web server software. The web server invokes these general-purpose programs in response to user requests. The CGI scripts' main function is to process user input and performs some services (such as retrieving data from a database, or dynamically creating a web page) for the user because CGI scripts process untrusted user input, the security risks associated with the CGI (and other forms of middle tier software) are extremely high. Many attacks against web-based systems exploit CGI scripts [1] [4]. A very simple example of business application logic would be a customer adding an item to an online shopping basket & then being required to provide a name, address & payment details before being able to complete the purchase. application logic (also called business logic because it perform action as per defined business process which integrated through CBS application by developer, so it's called business logic, it's also noted that it does not refer to the general functionality of a web server, but to the specific operations of the application's function, such as product discounts, postage pricing rules, etc. Cyber-attacks are the core of any security assessment of Web based e-Commerce systems [1]. One of the more promising research fields in this context is related to the representation of the Vulnerabilities, Attack patterns Classification. Several models are proposed to represent these; these models usually provide a generic representation of attacks. According to the Purdue University Researchers, Conversely, the experience shows that attack profiles are strongly dependent upon several boundary conditions these conditions could be based on three well defined Areas: 1) Environmental (faults discovered by I. Krsul, 1998); 2) Coding (fault); 3) Configuration errors [5] [6]). Whereas in 2004 University of Luton researcher Faisal Nabi first time identified that design flaw could also be a cause of attack profile boundary condition of design specification in e-commerce systems (Faisal Nabi, 2004). In this paper, we will investigate design flaw attack profile boundary condition (that caused business logic attack) interms of reuse design specification in the CBS e-commerce systems.

2. Web Software Application Complexity & Component-Based-Development Risks

Modern web applications run large-scale software applications for e-commerce, Information distribution, entertainment, collaborative research work, surveys, & numerous other activities. They run distributed hardware platforms & heterogeneous Computer systems. The software that powers web applications is distri-

buted, is implemented in multiple languages & styles, incorporates much reuses & third-party components, is built with cutting edge technologies as stated (section above component based software) & must interface with users, other web sites & databases. Although. The word "heterogeneous" is often used for web software, it applies in so many ways that the synonymous term "diverse" is more general & familiar, & probably more appropriate [7]. The software components are often distributed geographically both during the development & deployment (diverse distribution), & communicates in numerous distinct & sometimes novel ways (diverse communication) [8]. Web-based software systems are created by combining a variety of components from various sources, such as custom-built special-purpose applications, customised commercial-off-the-shelf software components, and third-party software [7]. Much of the new complexity found with web-based applications also results from how the different Software components are integrated. Not only is the source unavailable might be hosted on computers at remote, even competing organization. To ensure high quality for the web systems composed of very loosely coupled components, which seriously required evaluate these Components connections [9].

Web software components are coupling more loosely than any previous software application [7]. AS it is stated above that e-commerce sites offer more than front-end servers, they usually run complex Middleware programmes such as CGI scripts, Java servlets, application servers & component-based-software such as EJB Java beans, Java 2 Enterprise Edition (J2EE), CORBA, COM & DCOM components-based solution. One reason for the emergence of this component-based software on e-commerce sites is the complexity of the software necessary to implement business application logic. This Complexity, in turn, introduces the more Software Flaws that can be exploited for malicious, gain [3].

The web's function & structure have changed drastically, particularly in the past couple of years, yet most software engineering researchers, educators, & practitioners have not yet grasped how fully this changes affects engineering principles & process [7], example of a changes in last couple of years idea use of web 2.0 feature Ajax (The Ajax engine is the client-side code that handles calls between the client & server. Typically this would be a library of JavaScript function included on the page [10], more prone it is to have flaws in that any attacker with basic skills can use proxy software(or call script functions directly)to bypass the intended logic/business logic due to complexities involved & since more application logic is being delegated to web browsers, this idea of Ajax is leading to open flaw which allows intruders to easily read the source code & look for weakness area in the system middle tier application logic. Sharing business logic client-side reveals source information of the complete system, which is too dangerous combining representation logic, rendering logic & business logic & resides business logic client & Application sever-side. For example, Ajax-enable application with multiple levels of user account it was found that the site employed one JavaScript include file for the entire client-side logic.

This meant that an anonymous user with trail account could see the logic behind the administrator-level service call. The locations of all administrator service script were disclosed, providing invitation a definitive map of application to a potential attacker to attack business logic in the middle tier. Therefore, in this scenario EASI framework also get failed to protect the system integrity & security. Another example, developing a simple script that allows one to use thousands of e-coupons or using a similar script to open thousands of brokerage accounts that can each receive small deposits from a bank—usually around five cents—to verify transactions. In the end, one could end up making tens of thousands as shown in Figure 2.

Web sites are now fully functional software systems that provide business-to-customer e-commerce, business-to-business ecommerce & many services to many users. The growing use of third-party software components & middleware represents one of the biggest changes in the e-commerce web software-Application systems so as security; integrity has threaten because of the flaws in the design, up to 50% of software defects leading to security problems are software architecture & design flaws [11]. In other words during the *high-level-design* stage of *software architecture design & technology architecture design decisions correspondence of web software structure that how various components will be integrated & interact, and which technologies will requirement define software function interpreted, failure in this cause* 50% *of software defects which then leading to security problem & threaten the internal software application integrity itself compromise because of software architecture & design flaws at the* high-level-design.

In commerce systems, especially e-commerce applications, relatively little security analysis are done at the business logic level. Most analysis at this level is focused on detecting what we would call mistakes in the implementation of a set

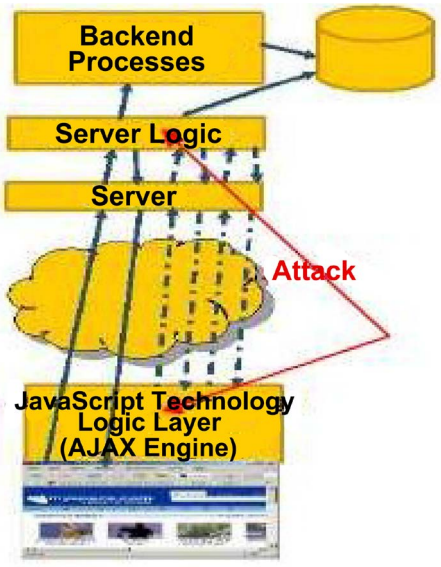

Figure 2. Application logic flaw resides both end client & server side.

of business rules, rather than detecting mistakes in their design. Examples of implementation error are the improper configuration of a CGI server, or the choice of CGI as an implementation language in the first place [1]. Examples of the business logic error are frauds in the business process, such as unsuitable transactions and abuse for personal enrichment through deliberate misuse or misapplication.

3. Research Design Strategy

Since our main scope of this research study is to focus on business component-based application logic problems and identify vulnerability that is because of mismatch between business process specification and logical component-ware specification at design level while using rapid development business component-based-software approach for business application logic in e-commerce system.

In the light of business logic vulnerability definition [12] [13], we have taken a case study from the industrial sector (Bank Case Study of business application logic), research process would go through stages as explained in the paper, these are based on vulnerability risk analysis in terms of security perspective, for that, it is very important to integrate the knowledge contained in the attack method with boundary knowledge related to vulnerability of the target Component-based-Software application e-commerce system. Our focus is to consider multi-layer specification based on components business event scenario using Bank Case Study. In this technique, process is divided into two phases: 1) High level view of system tier 2) Component layers. The High level view of system tier focus on the high level view of the design product, and component level layers will consider the design, test, and diagnostics specification for a separate entity component that take part in the system building.

4. Security Properties Violations [Problem Area] in Business Tier

The violation of Integrity & security within the web software application & components based software that develop rapid business application logic that can be custom-developed/COTS, because of flaws at design level in web software application, the use of components based software risks the cause of these logical vulnerabilities can subvert, misuse & circumvent the steps defined by function of the application that is not intended to do described by the function & business process specification [12].

Unfortunately, even simple flaws in the complex middleware layer can provide the leverage necessary to bypass even strong authentication schemes. Whereas most front-end & back-end systems are commercial-off-the shelf (COTS) software packages, a good portion of the middleware software is necessarily custom-development in order to implement every business's particular application logic. The most significant weak link in server-side systems is the middleware

layer. Therefore, a strong risk management plan will focus on providing rigorous software assurance for the middleware software [3].

Web Software Application Vulnerability

We define web software application vulnerability definition as web software application vulnerability includes mismatches between application software architectural/design logic & the assumptions about the environment made during the development/Implementation (code writing), operation of the programme and the environment in which the programme executes [13].

5. Logical Vulnerabilities in Application Layer

Since our main scope of this research study is to focus on investigation *web software application logic* problems & identify vulnerability that is because of mismatch between business process specification and component ware specification at design/Architecture level while using rapid development business component-based-software approach for business application logic in e-commerce systems. Our attack patterns are more specific to what components can pinpoint vulnerability in a system design. We will only target business application Logic vulnerabilities.

Problem Cause Definition & Explanation

Application Logic Attacks Operation:

Unlike, common application technical attacks, such as SQL injection or Buffer overflow, each application logic attack is usually unique, since it's not been mentioned or part of any taxonomy of web application attacks, and since it has to exploit a function or feature that is specific to the application. Since, application logic attacks are not based on characteristics like buffer overflow which can be characterize them as other technical vulnerabilities in the web application (SQL, SSI or buffer overflow).This makes it more difficult for automated vulnerabilities testing Tools to identify or detect such vulnerability class of attacks because they are caused by the flaws in application Logic & not necessarily faults/bugs in the actual code.

6. Bank Case Study Component-Based-Rapid Development

Brief Summary of the (Design Flaw of CBS Reuse):

This real time case defines a classical incident & serious mistake, which caused design flaw in application logic, while reusing the component and misconfiguration of the server—side component, cause security flaw in the business logic. The same component that was incorporated into the registration functionality also was used elsewhere within the application, including within the core functionality. The reason of this problem is gap between purpose of business process integration, and purpose of specification that seriously violated business logic design specification and their boundary conditions with respect to component-

based-software developed (CBSD), where CBSD application developer used rapid development approach in order to reuse the component but developer totally ignored the current application logic based on existing components logic, and their processing business logic. It shows that the assumption of the developer that just kept in view realization contract of component specification, which was depending on the component model specification that only provides support for deployment infrastructure. This refers to specifying the realization contracts in terms of component-based software model specification, but not usage contract specification. The usage contract translates service offered by a component interface specification. *An interface–focused-design* (*completely focus on interface as the main design abstraction that encourages designers to take into account system behaviour more abstractly*) *component-based development approach defines encapsulated behaviour accessible through well-defined Interfaces*. An interface is basically a description of combination a set of operations, and every operation represents service that acts based on component instances upon the client's invocation *request* for a particular service which is established through contracts (it can be a logical component or a building block that is used as a binding package of functionality to build logical component, either case is possible).

Therefore, this above explained case disclose clear violation, separation of business logic from implementation logic, case falls into purpose of specification re-using against its boundary conditions with respect to current component-based-software developed (CBSD) designed business logic. This converted the problem into logical vulnerability in the application layer. These are business component-based-rapidly developed (RDA) web software architecture and design flaws, which fall into the business component-based-software (BCBS) business process specification that refers to business process integration, which cannot be detected, by any code scanning tools for web application or any statically or dynamically analysis approach such as traditional Black Box Analysis tools for web application.

Detailed Description of the Case Study:

The Application Functionality & Business Process:

The application enabled existing customers who did not already use the online application to register to do so. New users were required to supply some basic personal information, to provide a degree of assurance of their identity. This information included name, address, and date of birth, but did not include anything secret such as an existing password or PIN number. When this information had been correctly entered, the application forwarded the registration request to back-end systems for processing. An information pack was mailed to the user's registered home address. This pack included instructions for activating their online access via a telephone call to the company's call center and a one-time password to use when first logging in to the application.

The Design Logic of Application:

The application's designers believed that this mechanism provided a very ro-

bust defence against unauthorized access to the application. The mechanism implemented three layers of protection:

1) A modest amount of personal data was required up front, to deter a malicious attacker or mischievous user from attempting to initiate the registration process on other users' behalf.

2) The process involved transmitting a key secret out-of-band to the customer's registered home address. Any attacker would need to have access to the victim's personal mail.

3) The customer was required to telephone the call center and authenticate himself there in the usual way, based on personal information and selected digits from a PIN number.

This design was indeed robust. The logic flaw lay in the actual implementation design of the mechanism. The developer was implementing the registration mechanism needed a way to store the personal data submitted by the user and correlate this with a unique customer identity within the company's database. Keen to reuse existing Component code, they came across the following class, which appeared to serve their purposes:

Event Trigger Component Cause of Business Logic Flaw;

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

After the user's information was captured, this object was instantiated, populated with the supplied information, and stored in the user's session. The application then verified the user's details, and if they were valid, retrieved that user's unique customer number, which was used in all of the company's systems. This number was added to the object, together with some other useful information about the user. The object was then transmitted to the relevant back-end system for the registration request to be processed. The developers assumed that making use of this code component was harmless and would not lead to any security problem. However, the assumption was flawed, with serious consequences.

Attack Pattern Birth:

The same *component (code)* that was incorporated into the registration functionality was also used "use case logic (+Process and Entity Type Logic)" within the application, including within the core functionality, which gave authenticated users access to "Account details component", "Statement's component", "Funds transfers component", and "Debit compoenet, Credit component and other information component". When a registered user successfully authenticated itself to the application, this same object was instantiated and saved in her session to store key information about her identity.

The majority of the functionality within the application referenced the information within this *CCustomer (Component)* object in order to carry out its actions because *CCustomer (Component)* object is candidate component (Process and Entity Type logic) within the majority of application functionality referenced the information within this component to carry out action—for example, the account details presented to the user on his/her main page were generated on the basis of the unique customer number contained within this component object. The way in the component code was already being employed within the application meant that the developers' assumption was flawed, and the manner in which they reused it did indeed open up a significant vulnerability. Although the vulnerability was serious, it was in fact relatively subtle to detect and exploit. Access to the main application functionality was protected by access controls at several layers, and a user needed to have a fully authenticated session to pass these controls as defined in Figure 3.

Bank Security Defence:

Figure 4 defined the application protected by access controls .Although the vulnerability stated above 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 needed to have a fully authenticated session to pass these controls and second security defence line was (security level 2: fraud management) as projected in the figure below.

Exploiting the Logic Flaw Scenario:

To exploit the logic flaw, therefore, an attacker needed to perform the following steps:

- Log in to the application using his own valid account credentials.

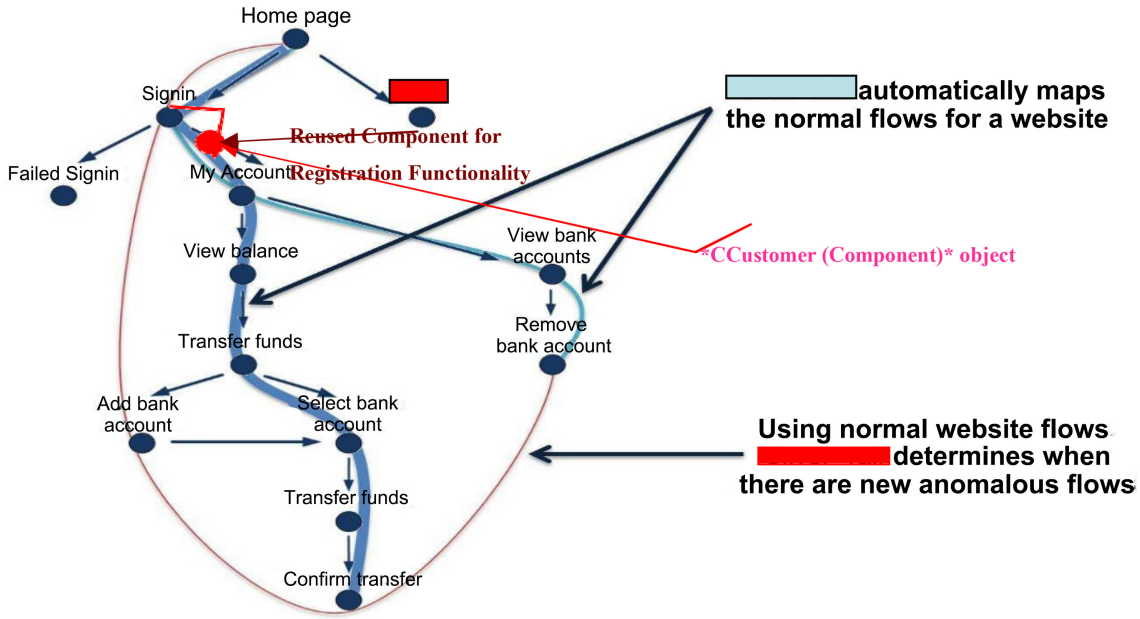

Figure 3. Attack birth its life cycle in bank service flow.

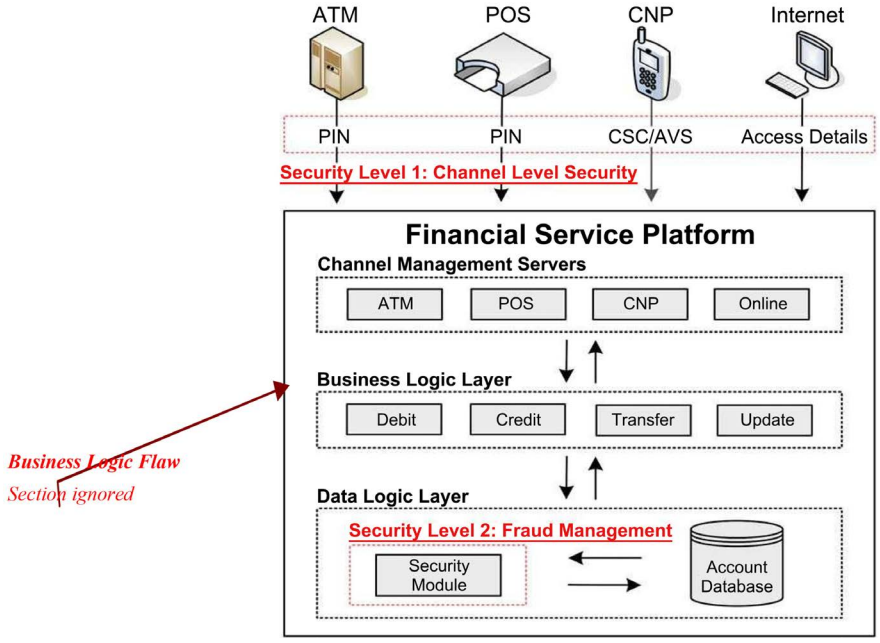

Figure 4. Bank application functionality protected by access controls & (COTS) security products.

- Using the resulting authenticated session, access the registration functionality and submit a different customer's personal information. This causes the application to overwrite the original "CCustomer" (Component) object in the attacker's session with a new object relating to the targeted customer.
- Return to the main application functionality and access the other customer's Account. A vulnerability of this kind is not straightforward to detect when probing the application from a Black-box perspective. However, it is also hard to identify when reviewing or writing the actual source code. Without a clear understanding of the application as a whole and the use made of different components in different areas, the flawed assumption made by developers may not be evident. Of course, clearly commented source code and design documentation would reduce the likelihood of such a defect being introduced or remaining undetected.

Attacking Method in This Scenario:

- In a complex application involving either horizontal or vertical privilege segregation, try to locate any instances where an individual user can accumulate an amount of state within their session which relates in some way to their identity.
- Try to step through one area of functionality, and then switch altogether to an unrelated area, to determine whether any accumulated state information has an effect on the application's behaviour.

6.1. Investigated Reason of Vulnerability in the Light "State of Art CBSD" in Business Logic

Component-based-rapid development approach is a method to make business

logic in the application layer of e-commerce system. Each component has business rules, which develop business logic in the component. Since business application logic is developed with business components, it develops logical componentware. Each component business logic makes a set by integrating these components to develop "logical component ware", and then business application logic is rapidly developed. Reuse of component in the e-commerce applications, means business component (that has a complete function, business logic based on business rules) designer starts design that which business component can be reused but it is also very important that designer must have complete knowledge about previous designed business logic in the system, so that no logical vulnerability could provide opportunity to intruder to violate the business application Logic in e-commerce system.

This case study describes, complete design based on component-oriented integration process and how developer made mistake during the integration, while ignoring business logic of the application's current functionality. This problem caused wrong analysis of existing component-based business application logic.

Business process integration, which depends on logical component's interface specification in business component, it refers to business function/processing logic specification. (Components are reusable units for composition. This statement captures the very fundamental concept of component-based development, that an application is made up and composed of a number of individual parts, and that these parts are specifically designed for integration in a number of different applications. It also captures the "idea that one component may be part of another component" [14], or part of a sub-system or system, both of which represent components in their own right). Integration of all this process, then develop overall business processing logic for a business component-based software to develop an application in the middle tier, which is connected with information system in the back-end systems of organization.

Since business process integration is made on the base of business functional concern; it cannot be dealt with technological point of view, because problem is not based on technical or technological specific principle of Integration, which is based on related to some particular physical component model such as (J2EE, CORBA, COM), but as it is stated, issue identifies that business processing designed solution based on business components and their business processing integration. This is the point, where the focus is set to the logical structure of the "business solution", this is the stage where logical problems occurs due to lack of paying attention to the design-based business component interfaces used and offered testing environment within the logical structure of the business solution, are known as Business Logic Vulnerabilities [12] [13] [15].

Moreover, during the investigation we also discovered that Bank developer was keen to reuse component totally ignored component specification, different components in multi-layers, their existing offered & used interfaces contract constrains, that assure the level of assurance while using component logic within the e-commerce system. Developer used the just physical component model spe-

cification, which is very serious violation of the principles that use case logic, processing logic of entity type logic of business component is not reused within the same application. As we demonstrated to justify this explanation by using one of the current, physical component-model specification technology J2EE and its reference architecture as defined in Figure 5(a).

Diagnostics specification for a separate entity component that takes part in the system building, Process of this course of exercise as stated above in the Section 3. This exercise confirms high level design view of e-business process to diagnostics specification of each layer and its participant components role, and their offered and usage Contracts for detailed Design for Test (DFT) process of verification and validation stage.

A logical component represents the simply binding packages of related functionality of a compoenetware system. A logical component is defined by interface

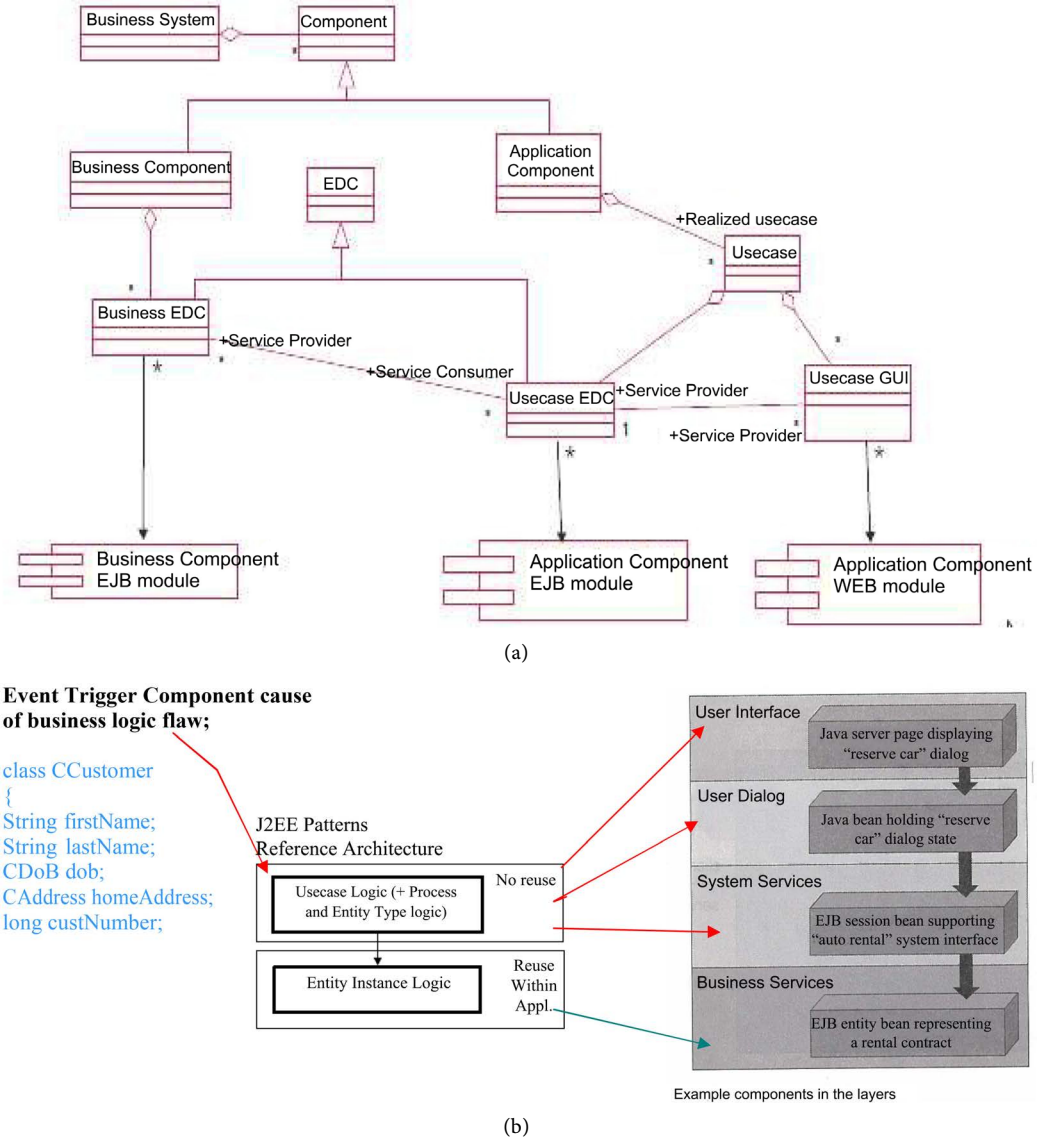

Figure 5. (a) System Multi-tier components in the layers; (b) System Multi-tier components in the layers.

specification based on interface information model derived from business type model, and integration is made on the base of usage specified interfaces contracts offered & used. Whereas a technical component is a unit that contains implementation logic of the software component unit based on physical component specification model. CORBA, NET, J2EE, and EJB are examples of current "physical" component technologies. Technical component integration is made on the base of realization contract specification related to the particular component specification model, these are building blocks of software component units as stated above based on the component model specification model artifacts as defined in **Figure 5(b)**.

6.2. Existing Methods and Approaches to Application Functional Logic Security

In modern times, the ruling perimeter security approach does not fit the e-business environment's security challenges. Due to the fact that e-business mode of doing business implies "openness" to the external world, while the perimeter security implies existence of boundaries, that separate between the organization and the external world.

Traditional security paradigm, which is as stated parameter security approach, does not relevant to e-business process. Therefore, business process appears to present most important change from traditional way to e-process [16].

E-business/e-commerce is the subject of a huge volume of ongoing research. Some of this relates to e-business information security, and just a small part (with regard to information security relates to business process identity security requirements of electronic processes.

Traditionally, enterprises have prioritized and focused their IT security strategies and budgets on protection of network perimeter and physical or logical access control to the application system environment. Following the common approaches, organizations security goals are defined to protect company's information systems by eliminating the external threats, and by providing logical access control and restrictions to the application, but business process can be attacked even when a very good network and infrastructure security programme is in place. For example, good network perimeter defence using firewalls, honey pot, 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 application interface level. This is reason why, e-commerce business process build on base of two blocks; business logic and information flow.

Therefore, there is a clear need for such technique or framework that could work for as an alternative approach for design e-business information systems security.

7. Proposed a Technique Secure Application Functional Processing Logic

We propose secure application functional process logic for e-commerce component-based application based on security requirements of e-process and security assurance logical component behaviour specification approach to formulize and design a solution for business logic vulnerability phenomena. First section of methodology follows security risk analysis in the CBSD rapid business logic and defensive strategy. In addition to this, we also propose in the second section, "A security Assurance Model process" to deal with logical component-ware reusing risks in application logic that cause logical vulnerability in e-commerce system to encounter in such situation while reusing component from its existing application logic. This would contribute to solve identified problem. Application logic represents translation of domain business logic that, in component-based developed application logic interoperates business process for particular domain problem.

Key Elements of security risk analysis in terms of Component-based e-commerce systems are: 1) Effect of attacks on system design; 2) Layer pattern; 3) Architectural risk analysis for component-based business logic security.

7.1. Effect of Attacks on System Design

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 above stated 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 modelling business scenario, which represent a basic end-to-end system function, also decomposed into sub-scenario, which identify 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 the 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.

7.1.1. Layers Pattern

Security encompasses all the architectural levels of a system. The layers architectural pattern is therefore the starting point of the design of secure systems. This pattern provides a structure where we can define patterns at all levels that together implement a secure system as defined in **Figure 6**. Its main idea is the decomposition of a system into hierarchical layers of abstraction, where the higher levels use the services of the lower levels. Here it provides a way to put

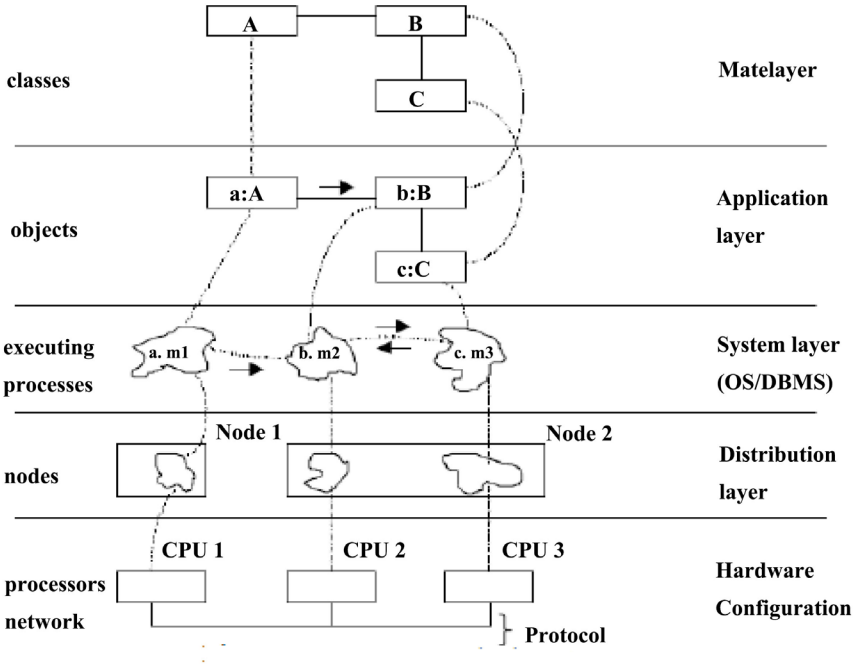

Figure 6. Layers architectural pattern.

things in perspective and to describe the mechanisms needed at each layer. Figure shows the specific set of layers we consider. This figure shows some of the participants at each layer and their correspondence across layers.

7.1.2. Architectural Risk Analysis for Component-Based Business Logic Security

Design flaws account for 50% of the security problems in the component-based-software system [17]. Architectural risk analysis is, at best, a good general-purpose yardstick by which we can judge our security design's effectiveness [14]. Because roughly 50 percent of security problems are the result of design flaws, performing a risk analysis at the design level is an important part of a solid good secure component-based-software system engineering.

To encompass the design stage, any risk analysis process should be tailored. The object of this tailoring exercise is to determine specific vulnerabilities and risks that exist for the software [17]. Architectural level risk analysis Approach n-tier Web application design model by the Cigital USA does not clarify the each layer in the tier and its components, as its very important for a functional decomposition of the application into major components, processes, data stores, and data communication flows, mapped against the environments across which the software will be deployed, allows for a review of threats and potential vulnerabilities, as its defined in the new proposed n-tier e-commerce web system architectural risk analysis & security management model as defined in **Figure 7**.

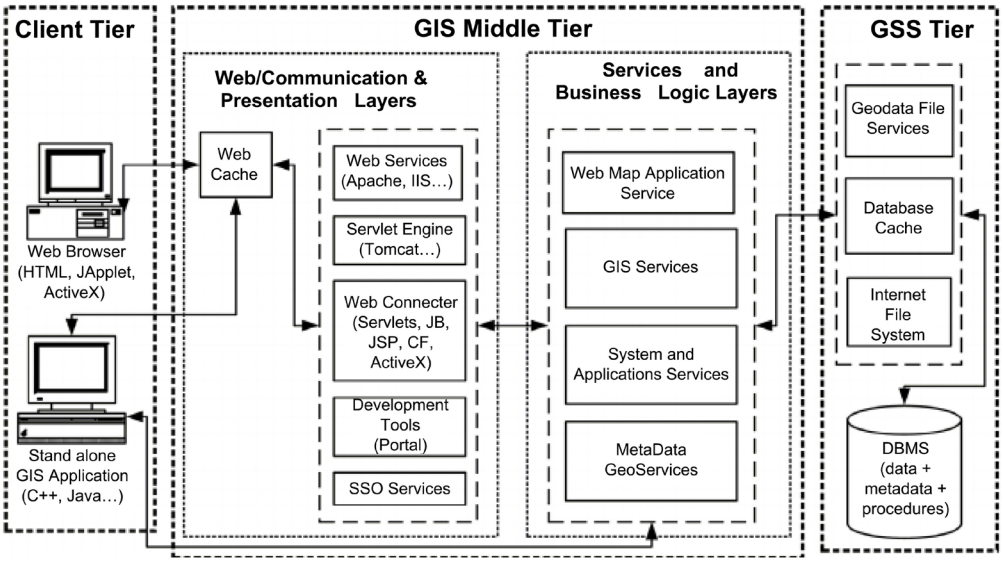

Figure 7. n-tier e-commerce web system architectural security management model.

It can contemplate using modelling languages, such as UML, to attempt to model risks; even the most rudimentary analysis approaches can yield meaningful results. Consider above model, which shows an n-tier deployment design model for web-based application issues. As we applied risk analysis principles to this level of design, we achieved immediately some useful conclusions about the security design of the application.

During the risk analysis process must consider the following:

1) The threats those are likely to want to attack the system.

2) The risks present in each tier's environment.

3) The kinds of vulnerabilities that might exist in each component, as well as the data flow.

4) The business impact of such technical risks, were they to be realized.

5) The probability of such a risk being realized.

7.2. Designed Defensive Strategy as a Solution to Deal Business Logic Concerns

This part of methodology will provide a strong risk management control plan focusing on providing rigours component ware assurance for rapid development of CBSD business application logic for e-commerce applications as projected in Figure 8.

Key elements of problem solution follow: 1) Strong risk management plan; 2) solution artefacts; 3) Security Characteristics of component-ware components and 4) Evaluation & validation of artefacts.

7.2.1. Strong Risk Management Plan

Ensure that every aspect of the application's design must be clearly & sufficiently detailed to understand every assumption and designed function logic within the application by designer.

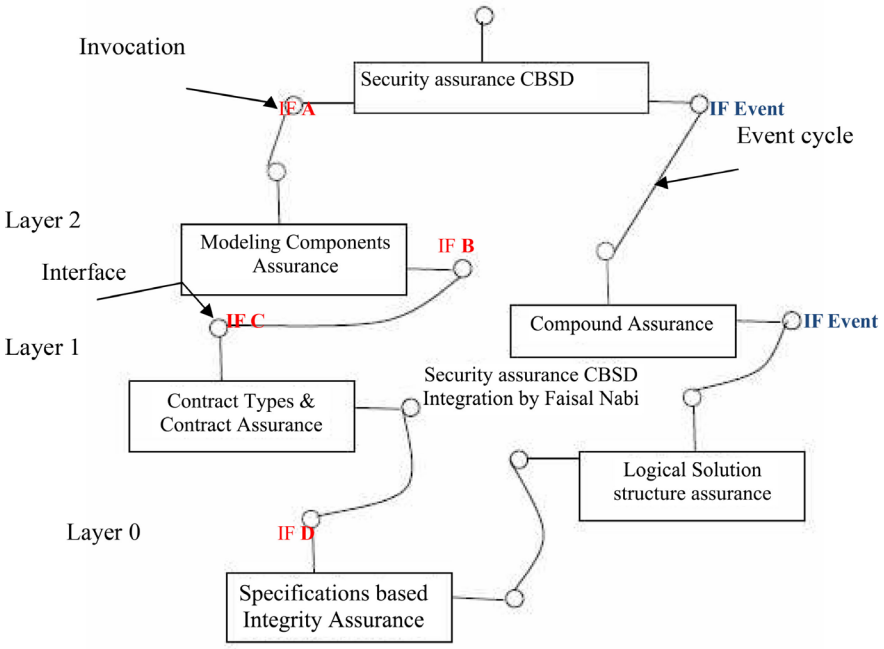

Figure 8. Security assurance CBSD model process.

Mandate that all CBSD should be clearly commented to include the following information throughout.

1) The purpose and intended use of each component (If Component code available information of code, if not, its functional business logic within the component through usage contract description).

2) The assumptions & logic made by each component about anything that is outside of its direct control.

3) 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

(Note: Client here dose 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).

7.2.2. Solution Artifacts

As that there is no unique signature by which logic flaws in component-based-rapid developed web software application can be identified, because there is no silver bullet so far developed which could protect.

Good Practice: Good practice that can be applied to significantly reduce the risk of logical flaws appearing within component-based-development and its logic.

Security Characteristics of Component Ware Components: Since a software component can be regarded as an IT product or system, it is natural to use the Common Criteria in assessing its security properties. The Common Criteria provide a framework for evaluating IT systems, and enumerate the specific secu-

rity requirements for such systems. The security requirements are divided into two categories:

1) Security functional requirements;

2) Security assurance requirements.

The Security Functional Requirements:

Describe the desired security behaviour or functions expected of an IT system to counter threats in the system's operating environment. These requirements are classified according to the security issues they address, and with varied levels of security strength. They include requirements in the following classes: security audit, communication, cryptographic support, user data protection, identification and authentication, security management, privacy, protection of system security functions (security meta-data), resource utilization, system access, and trusted path/channels.

The Security Assurance Requirements:

The security functional requirements mainly concern the development and operational process of the IT system, with the view that a more defined and rigorous process delivers higher confidence in the system's security behaviour and operation. These requirements are classified according to the process issues they address, and with varied levels of security strength. The process issues include: life cycle support, configuration management, development, tests, vulnerability assessment, guidance documents, delivery and operation, and assurance maintenance.

Therefore, in the Bank case security assurance requirement was also an issue. In the light of above stated security assurance CBSD process model. There are two further more important.

Moreover, There are two further more important artifacts which we dived to consider 1) Security focus review of application design (falls under Separation of business logic) 2) Security focus code reviewing review (Falls under the Implementation logic).

1) During Security-Focused Review of Application Design: Refers to tackle logical design flaws, during the security-focused review of design, must reflect upon every assumption made within the design of logical component ware stage and its business process integration, and try to imagine circumstances in which each assumption and logic might be violated. Focus particularly on any assumed condition that could conceivably be within the control of application user based on business process, rule and policy.

2) Security-Focused Code and Implementation Review: Refers to technical vulnerability issues that could be due to environment level, Infrastructure concerns or software artefact based multiple technology integration at the time of implementation. Carefully, think laterally about three key concerns at this stage;

a) The ways in which unexpected user behaviour and Input will be handled by the application.

b) The potential side effects of any dependencies and interoperation between different code components and physical component-model specification with

respect to integration between different application component functions and underlying middleware services.

7.2.3. Verification & Validation

It's true that in real-world applications making all the assumptions clear is obvious impossible but Model-based analysis and test generation leverage tools (such as Smartesting Tool, T-VEC Tool) that have been demonstrated to address several inadequacies of traditional testing [13]. They are systematic about providing requirement-based or design-based test coverage of complex software-intensive [18]. Not only are there tools that can systematically generate tests for each aspect of a requirement or design model, but they can identify defects (e.g., contradiction or inconsistencies). Detecting defects early during the requirement or design process can reduce cost by eliminating.

Proposed strategy and solution artefacts help to understand design and apply Model-based-Testing approach for component-based e-commerce applications assurance. So that the extracted test design from collected industrial case study of e-commerce system could follow a complete plan, such as, applying approach of modelling business scenario to generate test from extracted design test model as defined in **Figure 9**. This will cover integration strategies of components as compare to its requirement specifications, their offered and used interface focused design specifications, obtaining test scenarios from business events based on contracts involved between business components and their flow, and analyzing business data to achieve test cases. Therefore, our focus is to consider multi-layer specification based on components business event scenario using Bank Case Study. In this technique, process is divided into two phases 1) High level view of system tier 2) Component layers. The High level view of system tier focus on the high level view of the design product, and component level layers will

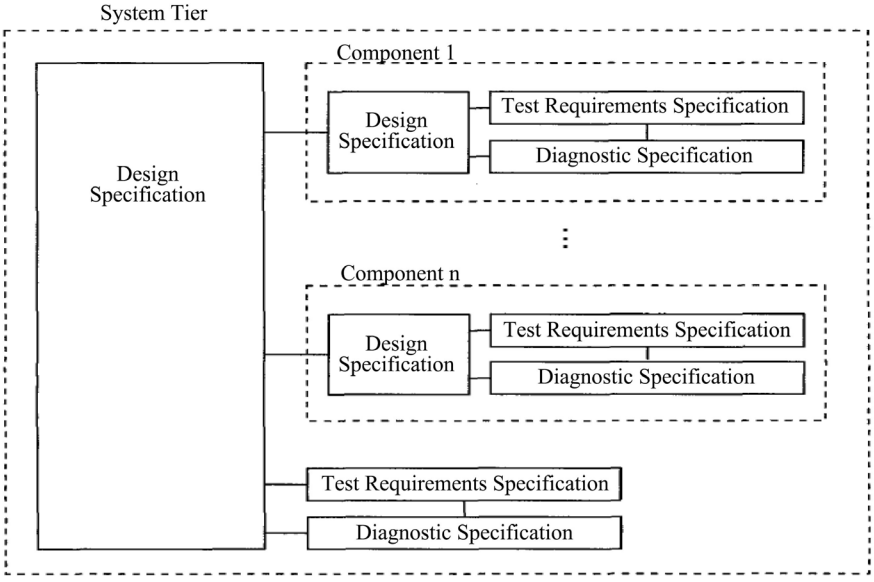

Figure 9. System middle tier & CBSD integration scheme for model based specifications.

consider the design, test, and diagnostics specification for a separate entity component that take part in the system building.

The most important point is to set focus on multi-tier specification .These will be system tier and the component tier. The system level tier represents a top view of the product, while the component level tier will account for the design, test, and diagnostics specification of the individual component that make up the system .This idea can be extended that a system can be nested as a component in the next higher order system, for the purpose of defining a design for test strategy, each tier will require a model based method for capturing design, test requirements, and diagnostic information.

Once, this test design is completed, next stage is to set the test bed Model-based DFT approach for components business process integration testing, which in return allow the validation and verification of the whole process, intems of system and component tier specifications that comprise the total Model based Design for test approach.

Model-based design for test approach for components business process integration testing, this will confirm the validation and verification of the whole process, in terms of system and component layer and tier specifications that encompass the overall Model based Design for test approach [19]. This approach based on the concept which confirms the philosophy of accuracy depends on precise construction; this reveals that discovering the design flaw in the product can be achieved early at design stage by using model based testing technique, for example integration flaws can be identified through "DFT method". Therefore, this philosophy invites researchers to apply Model-based approach that helps to refine and detection design flaw, those exist between the components interfaces, while interacting with other components in the system in order to deliver a service, trigger by the event, which call the particular service, composed with the business components based business process integration to develop "business process logic" in the e-commerce systems.

8. Lesson Learned

Therefore, in this case of bank developer completely ignored the purpose and type of behaviour specification of reused component in terms of requirement specification in each layer (*an n-tier CBS application*), component functional specification boundary conditions and knowledge of its defined interfaces within the systems, and ignored design specification for each layer component. This caused failure to meet the requirement specifications as compare to its functional specification based on design specification, for the purpose it was designed, based on its current logical component-based composition in the system. This gave birth to the design flaw in the component-ware. This all process of problem generated business logic vulnerability. This is a very serious violation of the principle "*specification purpose*" in component-oriented logical component-ware at the time of business interface-driven integration, while ignoring usage contract

type specifications. It's also case of "*Test by Contract*", which means not only design specification of component ignored but also contract establishment among the interfaces and their designed logic throughout the process, which created security assurance problem among the interface-focused designed components behaviour through e-process, while developing component-oriented business logic. The boundary condition of this attack falls in between functional specification and design specification. The attack triggering method is "*Event-based-generated*" e-process flow to violate business logic.

9. Contribution

Our contribution, proposed secure functional processing application logic for e-commerce component-based application, has covered the gap as stated above between traditional approaches and e-business process security requirements that will increase level of assurance during the practice of designing component-based rapid developed e-commerce applications from existing software components and deploying component based business logic into e-commerce system. Which reminds that focus on e-process security beside the functionality is also very important, because this functionality can be productive only when it works as per and within its functional control defined by the business logic in the e-commerce applications.

10. Conclusion

Much of the security today is addressed as an audit activity that mostly relies on penetration testing such testing activities often attempt to identify vulnerabilities that belong to certain categories of threats & use tools that are tailored around these threats. They may have security policies that auditors follow which require them to check a specific list of things, but they often fall short of identifying vulnerabilities that a result of the way the application logic has been custom developed. The fact is that many attacks that are reported today fall under what we define as application logic attacks. Therefore, our contribution of proposed methodology and approach will increase level of assurance during the practice of designing component-based rapidly developed web application software and deploying component based business logic into e-commerce system. This reminds us that focus on security besides the functionality is also very important because this functionality can be productive only when it works as per and within its functional control defined by the business logic in the e-commerce application.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Nabi, F. (2005) Secure Business Application Logic for e-Commerce Systems. *Elsevi-*

er Journal of Computer & Security, **24**, 208-217.
https://doi.org/10.1016/j.cose.2004.08.008

[2] J.Barrios, J. and Montilva C., J.A. (2003) A Methodological Framework for Business Modeling. 5*th International Conference on Enterprise Information System* (*ICEIS* 2003), Venezuela, 1 January 2003, 79-82.

[3] Anup, A.G. (2001) Security and Privacy in e-Business. John Wiley and Sons, Hoboken.

[4] Shishir, G. (1996) CGI Programming on the World Wide Web. O'Reilly and Associates, Newton, Massachusetts, USA.

[5] Krsul, I. (1998) Software Vulnerability Analysis. Purdue University. Purdue University Press, West Lafayette, USA.

[6] Aslam, T. (1995) A Taxonomy of Security Faults in the Unix Operating System. Purdue University, West Lafayette.

[7] Offut, J. (2002) Quality Attributes of Web Software Application. *IEEE Software*, **19**, 25-32. https://doi.org/10.1109/52.991329

[8] Cao, F., Bryant, B.R. Raje, R.R., Auguston, M., Olson, A.M. and Burt, C.C. (2002) Component Specification and Wrapper/Glue Code Generation with Two-Level Grammar Using Domain Specific Knowledge. *Proceedings of the* 4*th International Conference on Formal Engineering Methods*, Seattle, Washington, USA, 4-8 November 2002, 103-107.

[9] Dustin, J.E. (2001) Quality Web System: Performance, Security and Usability. Adition-Wesley, Boston.

[10] Ritchie, P. (2007) The Security Risks of Ajax/Web 2.0 Application. *Network Security*, **2007**, 4-8. https://doi.org/10.1016/S1353-4858(07)70025-9

[11] McGraw, G. (2006) Software Security: Building Security In. 2006 17*th International Symposium on Software Reliability Engineering*, Raleigh, NC, USA, 7-10 November 2006, 5-6. https://doi.org/10.1109/ISSRE.2006.43

[12] Nabi, F. (2008) Secure Framework Method for Business Application Logic Integrity in e-commerce Systems. *Annual Computer Security Application Conference* (*ACSAC*) 2008, California, 8-12 December 2008. https://www.acsac.org/2008/

[13] Faisal Nabi, M.N. (2017) A Process of Security Assurance Properties Unification for Application Logic. *International Journal of Electronics and Information Engineering*, **6**, 40-48.

[14] Kelly, T. (2019) An Assurance Framework for Independent Co-Assurance of Safety and Security. New York University Press, New York.

[15] Nabi, F. (2008) OWASP Testing Guide.
https://owasp.org/www-project-web-security-testing-guide/

[16] Nachtigal, S. (2007) E-Business Process Security Model. *International Confernece e-Commerce*, Minneapolis, USA, 23-26 December 2007, 34-40.

[17] McGraw, G. (2014) Risk Analysis in Software Design. *IEEE Security and Privacy*, **4**, 1540-7993.

[18] Mark, R., (2008) Model Based Testing Tools-Necessary for Complex system. *Software Productivity Consortium*, **6**, 34-42.

[19] Chechik, M., *et al.* (2019) Software Assurance in an Uncertain. In: Chechik, M., Salay, R., Viger, T., Kokaly, S. and Rahimi, M., Eds., *Fundamental Approaches to Software Engineering*, FASE 2019, Lecture Notes in Computer Science, Vol. 11424, 3-21.
https://doi.org/10.1007/978-3-030-16722-6_1