Proposing a Secure Component-based-Application Logic and Sys-tem's Integration Testing Approach

Faisal Nabi[1], Jianming Yong[1], and Xiaohui Tao[2]
(Corresponding author: Faisal Nabi)

School of Management and Enterprise, Information Systems Research Group
Toowoomba Campus, University of Southern Queensland, Australia[1]
(Email: faisal.nabi@yahoo.com)
Faculty of Health, Engineering and Sciences, University of Southern Queensland, Australia[2]
(Email: xtao@usq.edu.au)

(Received Mar. 30, 2018; Revised and Accepted Oct. 20, 2018; First Online Jan. 21, 2019)

Abstract

Software engineering moved from traditional methods of software enterprise applications to component based development for distributed system's applications. This new era has grown up for last few years, with component-based methods, for design and rapid development of systems, but fact is that , deployment of all secure software features of technology into practical e-commerce distributed systems are higher rated target for intruders. Although most of research has been conducted on web application services that use a large share of the present software, but on the other side Component Based Software in the middle tier ,which rapidly develops application logic, also open security breaching opportunities .This research paper focus on a burning issue for researchers and scientists ,a weakest link in component based distributed system, logical attacks, that cannot be detected with any intrusion detection system within the middle tier e-commerce distributed applications. We proposed An Approach of Secure Designing application logic for distributed system, while dealing with logically vulnerability issue.

Keywords: Application Architecture; Application Logic; Component-Based-Development; Design Flaw; Logical Attack; Web Software Risk

1 Introduction

Advent of the e-Commerce ushered in a new period pervaded by sense of boundless excitement & opportunities. However, need to think of risks as mere opportunities, the reason being that, in most business environment, the number or size of the risks taken usually to the number or size of the advantages to be gained. Today, vendors of e-Commerce systems are relied solely on secure transactional protocols such as SSL, TSL Nevertheless; the advancement of the security field has proved that vendors of e-commerce systems can not solely rely on secure transaction protocols such as SSL an encryption protocol promoted as proof of 100 % security by e-commerce vendors [1, 11].

Lost in the hype are the real security risks of e-commerce security is more than secure transactional protocols, cryptographic schemes / techniques, parameter security, Intrusion detection systems etc, these attributes make up only some part of security, privacy & client trust of e-commerce [4].

The software that executes on the either end of the transaction-server-side or client-side software poses real threats to the security, privacy in e-commerce systems. Two familiar adages play an important role in understanding to secure e-commerce systems (1) A chain is only as strong as its weakest link (2) in the presence of obstacles, the path of least resistance is always the path of choice [2]. Although, the security issues of the front-end & back-end software systems in e-commerce application warrant equal attention for complete security in e-commerce . This research paper focus is to represent the weakest link in e-commerce system which is based on CBS in middle Tier; we will also prove through experiment, logic subversion attack that cannot be detected with any intrusion detection system within the middle tier based application logic.

1.1 Contribution

Our work makes two important contributions, related to web insecure software development practices. This explained three categories of operational vulnerabilities; our target is Application Logic operational vulnerabilities that can be because of (1) design weaknesses, or (2) system configuration errors that may leads calling wrong component operation. We have proposed UML Based Secure Designing for Application Logic and system's integration testing model with applicability of system unification process for assurance purposes.

1.2 Type of Scientific Research

This research de-pends on exploratory research project that targets new idea about application vulnerability to scope out extent of business logic phenomena problem to generate idea about this phenomena. Proposing through UML based secure design modeling & system integration testing model with applicability of unification process for assurance.

2 Application Business Logic

The business logic describes the steps required to complete or perform a particular action as defined by the application developer, this is also called business logic because it contain business rules in e-commerce system in the middle tier. Modern web application implements business logic & its use changes the state of the business (as captured by the system) [12,13]. Web application executes business logic & so the most important models of the system focus on the business logic & business state, that refer to business rules as defined within an application of e-commerce to perform a particular action based on designing & implementation [12,13,19].

The middle tier of e-commerce servers that implements 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 [2,4].

3 Component-Based-Software Role in Business Logic & Concerns

A framework that provide a way to distribute a self-contained piece of software in forms, called "Objects" or in generally" Components" is encapsulated in a standard that can interoperate with other components in a framework such as JavaBeans, COM,DCOM & CORBA [2]. E-commerce sties offer more than front-end servers; however. They will usually run complex middleware programmes such as CGI, Java servlets, Application servers & Component-based-Software such as Enterprise Java Beens, Java 2 Enterprise Edition (J2EE), CORBA, COM & DCOM Components. Basically, Component-based-Software idea is to develop, purchases & reuse industrial-Strength Software in order to rapidly prototype business app-logic [17]. One of the more popular component frame works for e-commerce application is EJB, which support Component –Based Java Been. Other Component based technology models include the common object request broker architecture (CORBA) an open standard developed by OMG & Common object model by Microsoft & DCOM which support .Net environment [4].

The component frameworks are the glue that enables software components to provide services, business app-logic & uses standard infrastructure services such as naming, persistence, introspection & event handling, while hiding the details of the implementation by using welldefined interfaces [4]. The business application logic is coded in software "Components" that can be "Custom-Developed or purchased Commercial-off-the-shelf" [4]. In-addition to supporting the CGI functions, component –Based Software is expected to enable distributed B2B applications over the internet, and as that market for component–based software heats up, many standard business application logic components will be available for purchase off the shelf [4].

The application servers provide the infrastructural services for particular component models such as EJB, CORBA, COM, and DCOM. They also provide an interface for the business application logic to back-end services such as database management, enterprise resource planning (ERP), & legacy software system services [3, 4].

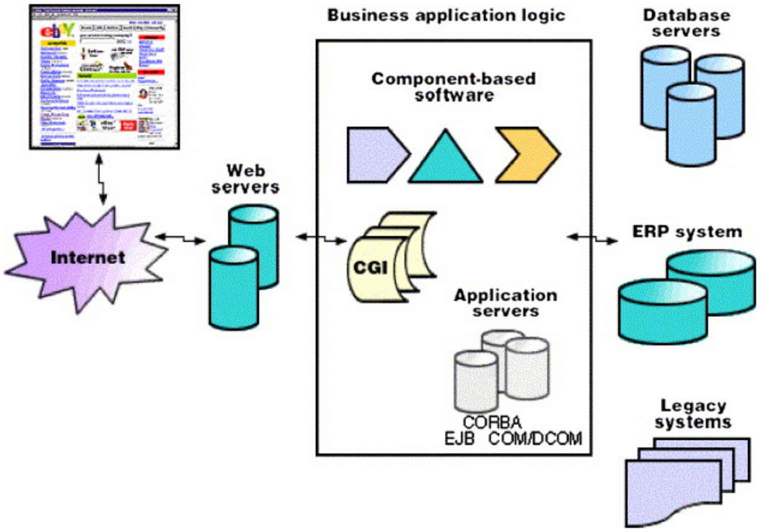


Figure 1: Component based business application logic

There is no doubt that component based software provide numerous benefits, but it poses security hazards similar to CGI scripts. CBS enables Software development in general –purpose programming language such as Java, C & C++.As these components execute with all rights & privileges of server process same, same like CGI they process untrusted user "Input" because Component based software can be used to build sophisticated large-scale applications, errors are unarguably more likely with CBS. Regardless of the implementation Application serv-ers-the security risks of server side software are higher & therefore server-side software must be carefully designed & implemented. One reason for the emergence of Components-based-software on e-commerce sites is the complexity of the software necessary to implement business application logic. This complexity, in turn, introduces more software flaws that can be exploited for malicious gain.

4 Web Software Application & 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 distributed, is implemented in multiple languages & styles , incorporates much reuses & third-party components , is built with cutting edge technologies as stated (section Component based Software) & must interface with users, other web sites & databases. Although. Server-side components are relatively new to the component market. Benefits enable the developer to provide solutions that run on a per server basis. These components serve many clients simultaneously without significant performance loss. Server-side components can also be upgraded efficiently removing the complexities of updating potentially thousands of desktop machines.

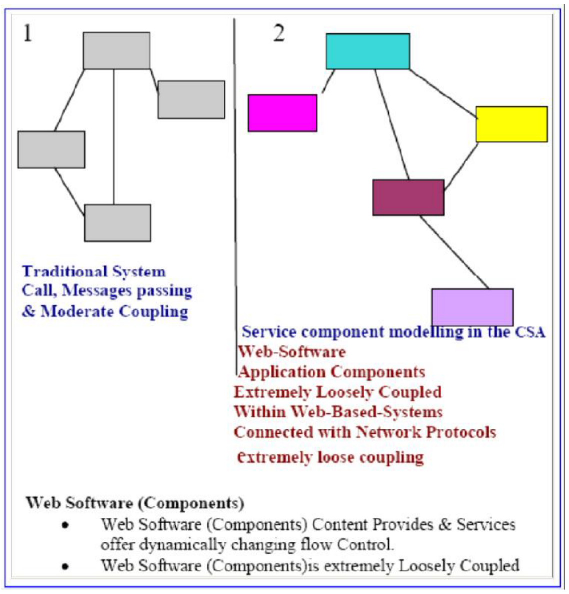
Component logic is often run on powerful servers as opposed to a desktop machine [9]. This makes the server-side component an excellent candidate for systems that require efficient throughput and performance [20]. The word “heterogeneous” is often used for web software, it applies in so many ways that the synonymous term “diverse” is more general & familiar, & probably more appropriate [5]. The software components are often distributed geographically both during the development & deployment (diverse distribution), & communicates in numerous distinct & sometimes novel ways (diverse communication) [10].

Web-based-software systems by integrating numerous diverse components from disparate sources, including custom-built special-purpose applications, customized “Commercial off-the-shelf Software Components & third-party products [5]. 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 [6].

Web software Components are coupling more loosely than any previous software application(Jeff Offutt,2002).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, 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, 22].

The web‘s function & structure have changed drastically, particularly in the past couple of [5], 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 [7], 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 appli-



Web Software (Components)

- Web Software (Components) Content Provides & Services offer dynamically changing flow Control.
- Web Software (Components)is extremely Loosely Coupled

Figure 2: Traditional tightly coupled software system vs. extremely loosely coupled web software systems

cation logic is being delegated to web browser , 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 server-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. Web sites are now fully functional software systems that provide business-to-customer e-commerce, busi-ness-to-business e-commerce & 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 threat because of the Flaws in the design.

The business application logic is a key weak link in security of many online sites. Typically, application subversion attacks as well as data driven attacks exploit weakness in this web app-software.

5 Security Properties Violations in Middle Tier

The violation in the middle tier is real caused based on business rule, in a way, those are deployed, basically indicate serious violation and related Integrity & security. Component based software that develops rapidly business application logic can be Custom-Developed/COTS that may have flaws in design of its web software application. The use of component based software risks, cause of these logical vulnerabilities that may lead towards subversion, misuse or circumvent the steps deployed by the application function.

The major cause of web insecurity is insecure software development practices. Operational security vulnerabilities generally have three main causes:

1) Design weaknesses;

2) Implementation/coding vulnerabilities;

3) System configuration errors.

Addressing design weaknesses ,especially important because these weaknesses are not corrected easily after a system has been deployed. We are working on operational vulnerabilities in the middle tier of web based information system that is composed with Components, not on traditional software techniques as used to be in the past.

As that, it is explained three categories of operational vulnerabilities, our target is Business Application Logic operational vulnerabilities that can be because of (1) design weaknesses, or (3) system configuration errors that may leads calling wrong component operation.

Unfortunately, even simple flaw 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].

"A software system's security & its integrity only as secure as its weakest component" [14].

Security problems originate from flaw in software design, and configuration management. These flaws are leveraged by the users of the software by malicious or accidently providing a level of access & privilege that would not otherwise be granted by the programme [1].

A flaw become the cause to represent vulnerability in the underlying software mitigating a flaw typically involves significantly more effort than simply modifying a few lines of code. Please note another point that problem does not solely in the implementation, the implementation that follows the design flaws & based on component-based (COTS) software that might contain the flaws. For example , the classical example, for instance performing sensitive business logic in a tainted client application is a design flaw that cannot be mitigated by simple measure such as modification array bounds.

Designing software behave, is a process that involves indentify & codifying policy & logic, then enforcing that policy & logic with reasonable technology to perform certain function or activity. There is no silver bullet for software security. Advance technology for scanning code is good at finding implementation-level mistakes, but there is no substitute for experience [15, 16].

6 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 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 Logic & not necessarily flaws in the actual Code.

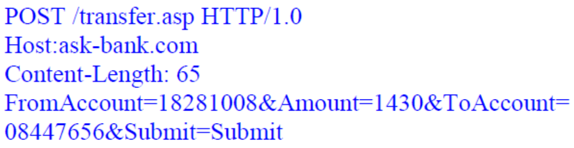
When application logic attacks are successful, it is often because developers do not build sufficient process validation & controls into application logic. This lack of functional flow control of logic allows attacker to perform certain steps incorrectly or out of order of the defined Logic.

An experiment conducted to demonstrate attacking on application's business logic in the scenario of the (SOAP) by injecting code in the SOAP message. In this case, as we know all that (SOAP) is a message-based communications technology that uses the XML format to encapsulate data. It can be used to share information and transmit messages between systems, even if these run on different operating systems and architectures. Its primary use is in web services, and in the context of a browser accessed web application, you are most likely to experience SOAP in the communications that occur between "Application Components" [8].

SOAP is often used in large-scale enterprise applications where individual tasks are performed by different computers to improve performance (W3C.org). It is also often found where a web application has been deployed as a front end to an existing application. In this situation, communications between different components may be implemented using SOAP to ensure modularity and interoperability. Because XML is an interpreted language, SOAP is potentially vulnerable to code injection in a similar way as the other examples [18].

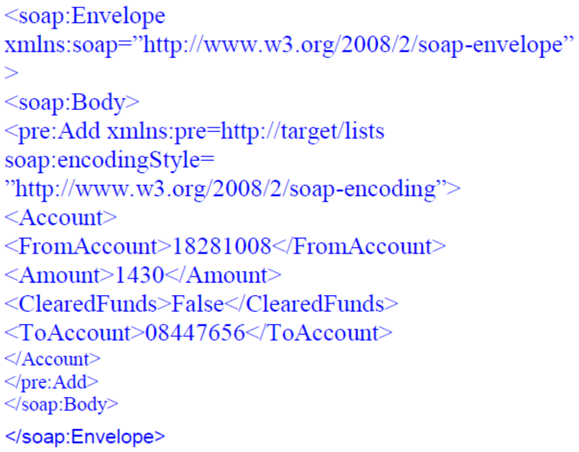
XML elements are represented syntactically, using the "Metacharacters" <> and /. If user supplied data con-taining these characters is inserted directly into a SOAP message, an attacker may be able to interfere with the structure of the message and so interfere with the application's logic or cause other undesirable effects [19].

A "Banking Application" in which a user initiates a funds transfer using an HTTP request like the following:



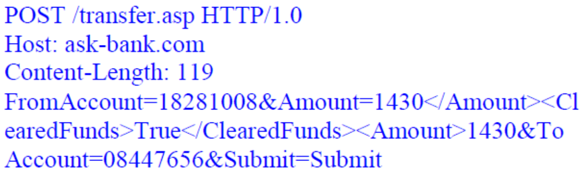
```
POST /transfer.asp HTTP/1.0
Host:ask-bank.com
Content-Length: 65
FromAccount=18281008&Amount=1430&ToAccount=
08447656&Submit=Submit
```

In the course of processing this request, the following SOAP message is sent between two of the application's back-end components:



```
<soap:Envelope
xmlns:soap="http://www.w3.org/2008/2/soap-envelope"
>
<soap:Body>
<pre:Add xmlns:pre=http://target/lists
soap:encodingStyle=
"http://www.w3.org/2008/2/soap-encoding">
<Account>
<FromAccount>18281008</FromAccount>
<Amount>1430</Amount>
<ClearedFunds>False</ClearedFunds>
<ToAccount>08447656</ToAccount>
</Account>
</pre:Add>
</soap:Body>
</soap:Envelope>
```

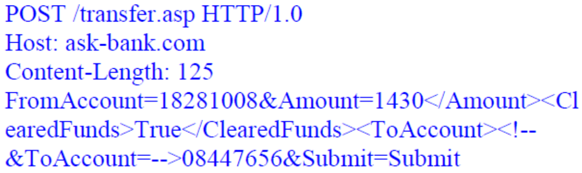
Look how the XML elements in the message correspond to the parameters in the HTTP request, and also the addition of the ClearedFunds (Component). At this point in the application's logic, it has determined that there are insufficient funds available to perform the requested transfer, and has set the value of this Component to False, with the result that the component which receives the SOAP message does not act upon it. In this situation, there are various ways in which you could seek to inject into the SOAP message, and so interfere with the application's logic. For example, submitting the following request will cause an additional ClearedFunds (Component) to be inserted into the message before the original element (while preserving the SQL's syntactic validity). If the application processes the first ClearedFunds (Component) that it encounters, then you may succeed in performing a transfer when no funds are available:



```
POST /transfer.asp HTTP/1.0
Host: ask-bank.com
Content-Length: 119
FromAccount=18281008&Amount=1430</Amount><Cl
earedFunds>True</ClearedFunds><Amount>1430&To
Account=08447656&Submit=Submit
```

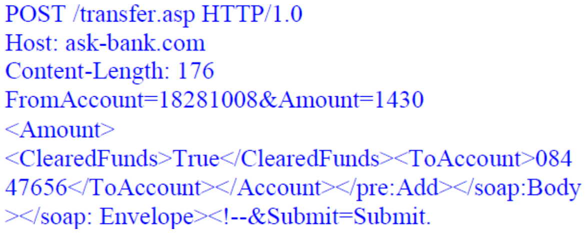
If, on the other hand, the application processes the last ClearedFunds (Component) that it encounters, you could inject a similar attack into the ToAccount parameter.

A different type of attack would be to use XML com-ments to remove part of the original SOAP message altogether, and replace the removed elements with your own. For example, the following request injects a ClearedFunds (Component) via the Amount parameter, provides the opening tag for the ToAccount (Component), opens a comment, and closes the comment in the ToAccount parameter, thus preserving the syntactic validity of the XML:



```
POST /transfer.asp HTTP/1.0
Host: ask-bank.com
Content-Length: 125
FromAccount=18281008&Amount=1430</Amount><Cl
earedFunds>True</ClearedFunds><ToAccount><!--
&ToAccount=-->08447656&Submit=Submit
```

A further type of attack would be to attempt to complete the entire SOAP message from within an injected parameter and comment out the remainder of the message. However, because the opening comment will not be matched by a closing comment, this attack produces strictly invalid XML, which will be rejected by many XML parsers:



```
POST /transfer.asp HTTP/1.0
Host: ask-bank.com
Content-Length: 176
FromAccount=18281008&Amount=1430
<Amount>
<ClearedFunds>True</ClearedFunds><ToAccount>084
47656</ToAccount></Account></pre:Add></soap:Body
></soap: Envelope><!--&Submit=Submit.
```

In most situations, you will need to know the structure of the XML that surrounds your data, in order to supply crafted input which modifies the message without invalidating it. In all of the preceding tests, look for any error messages that reveal any details about the SOAP message being processed. This will disclose the entire message, enabling you to construct crafted values to exploit the vulnerability.

Experiment Result:

Hence, we have derived the result from this experiment, that application's component logic can be subverted, even if, application follows absolutely correct functionality. This course of action proved the claim that such attack subversion of application logic cannot be detected, even if EASI frame work of security deployed, failed to filter or catch this security breach attempt. Through this experiment research, it is concluded that business logic layer in n-tier applications need for such a design strategy is motivated by the fact that logical flaws do not show patterns or signatures and, thus, their discovery cannot be automated.

7 Previous Model of Security EASI for e-Commerce System

Actually previous model EASI does not meet the com-plete solution since it's also based on COTS Security Products these security services are used through API,s of Security services as mentioned within the Model EASI [21] provides a common security framework to integrate many different security solutions to securely connecting Web servers to back-office data Stores. The key security services of this model authentication, authorization, cryptography, accountability, and security administration.

Therefore, since the nature of business application logic vulnerabilities are varied, as explained above in the experiment and in Section 6.1,7, which is reason why explained above EASI framework does not control those problems in the business application logic to be attacked or violation its integrity & logical functions.

8 Proposed UML Based Secure Designing for Application Logic

One of the first steps in system design should be the analysis of the possible attacks to the specific system and their consequences when successful. This analysis can be used to define the countermeasures that need and will also be useful later to evaluate the system security.

Identifying the components that needs to be secured, is a very important factor and first stage in the designing a secure environment for system. Next mechanisms that can be used to secure those components need to be identified. It is then necessary to understand which mechanisms are to be put together to secure the components thus giving rise to a secure development scenario.

In the n-tier distributed–computing environment, front-end presents presentation logic which invoke the business logic for the submitted request then the business logic layer hosted application interacts with the data tier & its logic for requested enquiry and computes the results that will be delivered to the presentation-logic layer.

Typically , security senility increases its flow from the first layer towards the last such partitioning into zones helps define the security requirement for the environment & the design of the topology to the host the components. It is also need to make sure that every aspect of the application's design is clearly mentioned in the sufficient detail to understand every assumption made by the designer and all such assumption must be explicitly on the record within design plan.

For example, sources code is clearly commented purpose & intended uses of each component and assumption made by each component about anything that is outside of its direct functional control. It is also important to reference to all client code which makes use of the component and clear to it effect could have prevented the Logic flaw within the online registration functionality as defined in the example in that case "client" here refers not to the user end of the client-server relationship but to other code for which the component being considered is an immediate dependency. When implementing functions that update session data on the basis of input received from the user or actions performed by the user, reflect carefully on any impact that the updated data may have on other functionality within the application unexpexted side effects can occur in entirely unrelated functionality defined by a different programmer or even a different development team.

8.1 UML Based J2EE Secure Design Modeling for Application Logic

Designing of application for e-commerce distributed system in a tier is also very important since many attacks are cause of design flaws in the e-commerce systems such logical flaws do not often refer to component based flaws but also architectural, component modeling to set the logic of application while using business rules related to the particular business or activity. Therefore , it is very important to define clearly architectural design of topology in which system going to design for deploy by separating each tier clearly , second stage focus on the appli-cation logic design strategy & policy with that components have to function under given business defined rule/policy , third stage refer to design strategy for components which dynamic web content is used to tailor an individual's interactions with a web site & provide users with more interactive information. Dynamic content may be rendered in various form, such as static HTML files, Java Script or JSP file rendered using component supported environment such as Java servlets in a J2EE invokes business –logic application hosted middle tier to access back-end business data.

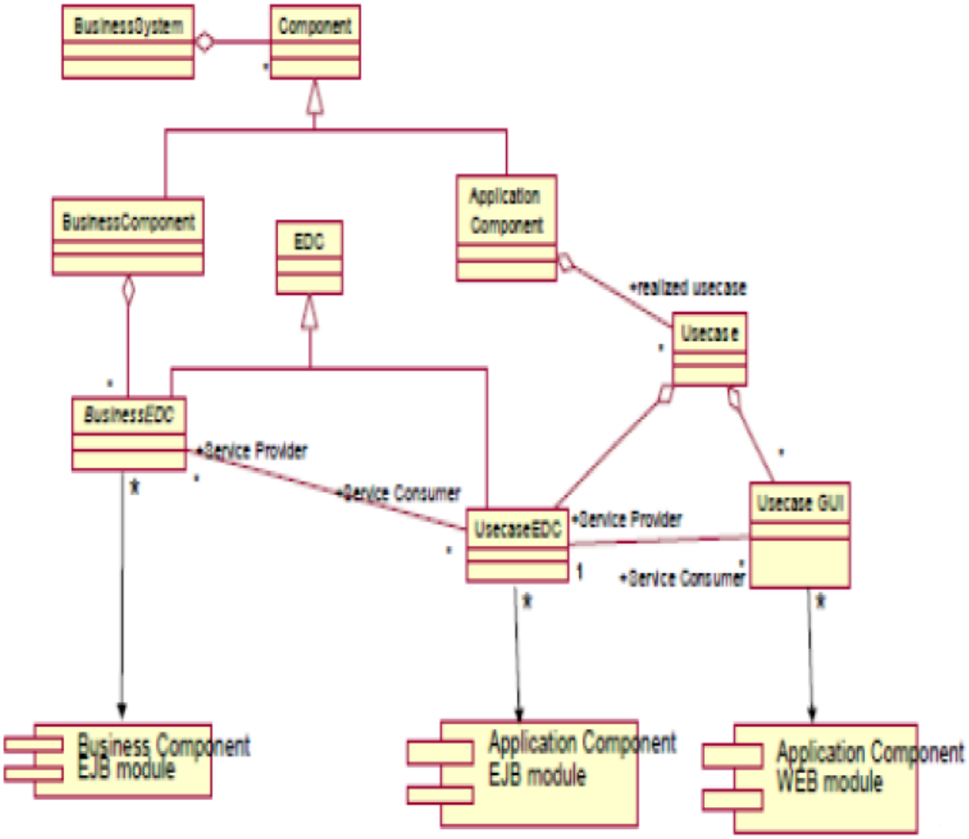


Figure 3: UML based secure design J2EE component based application logic modeling

In the above given example of attack as it is stated that always follow right principles of web application software engineering in the right technology environment support for component interoperability & security. it is noticed in the example that threat to the application integrity & application dependability based on the design flaw & engineering of the component while defining logic for e-commerce system web application software by merging all logic and invoke the direct access using servlets through JDBC to the back-end having bypass the middleware process or without encapsulating the business logic in an enterprise bean, it also define screening as a shield to the organizational resources by restricting unauthorized people to access valuable data or temper the resources.

8.1.1 Validation for the proposed UML Based Secure Design J2EE Component Based Applica-tion Logic

Business-level- Process Integration is based on business component's integration, which takes part to process a certain job/task event. This comes into being because of business logic inside a component. When we talk about business process integration, it means integrating business component's business processing logic, this integration completely rely upon "Business" component's Interface. A component interface is self-descriptor that provides specification, which defines component offers, what service such as, Account service provided by Account Component interface (as given above example experiment study), this is called Component interface specification, it reflects component's business process functionality, and indicates a component offer a particular service [24].

By using this specification, we can derive a component's syntax and semantics to determine its provided and required interfaces. The provided interface is a collection of functionality and behavior that collectively define the service a component provides to its associated clients. It may be seen as the entry point for controlling the component, and it determines what a component can do. If we look at a component from the point of view of its provided interface, it takes the role of a server. If we look at it from the point of view of its required interface, the component takes the role of a client. Provided and required interfaces define a component's provided and required contracts [25].

8.1.2 Integration and System Testing Approach

Components and integration level testing is therefore often confronted with the problem that the service provided by the component or a group of components under test (the SUT) requires functionality of components which are not ready for integration. Delays of the component and integration level testing process can be avoided by the development of emulators for the missing functionality and an elaborated project plan which considers the integration order of the components. This integration level testing shows, how the UML Testing Profile (UTP by (OMG)) can be utilized for this kind of testing [23].

Since model-based integration and system testing does not require all component realizations .Models can usually be available earlier than realizations, this model-based I & T technique enables earlier detection and prevention of problems when compared or matched to real system testing.

Furthermore, models usually allow easier adaptation and configuration than realizations, which means that they are well suited for system testing under different conditions. Especially for exceptional behavior testing, creating the non-nominal test conditions, e.g., a broken component, is usually easier and less expensive when models are used instead of realizations. Besides that this improves the coverage and thus the quality of tests, the ability to rapidly change test conditions using models also improves the efficiency of test execution. As such, model-based system testing not only allows earlier testing, but it also increases the risk reduction rate for the test phase.

Figure 4 contains a graphical representation of mod-el-based integration and system testing. When only the depicted components C1 and Cn are considered, the figure shows that component C1 is represented by model M1, while component Cn is represented by realization Zn. Using the model-based integration infrastructure IMZ, model M1 and realization Zn are integrated, yielding the model-based integrated system {M1, Zn} IMZ.This early representation of the system is then tested on the system level using tests derived from the system requirements R and the system design D, which is graphically represented by the dashed arrow .

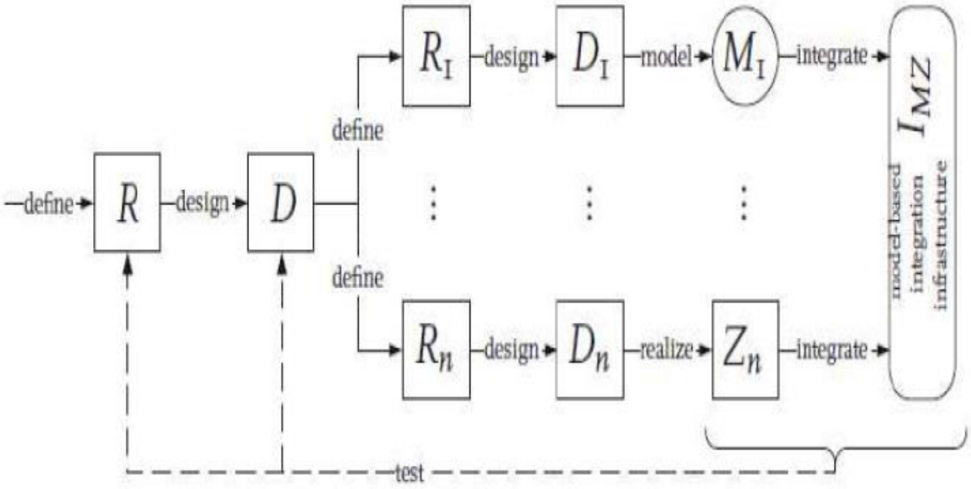


Figure 4: Integration and system testing model

8.2 Systems Unification Approach for Application Process & Assurance Properties

In cooperating our design model through model-based approach in component-based-software development, would promotes as a means to achieve cost-effective, high quality of assurance in the development and platform-independent design. This approach is a means of realizing the "correct by construction philosophy" where by flaws in a product design are discovered early at the design stage, such as components integration flaws. By using this approach, we can extract the integration flaw/fault existing between the components interfaces, interacting with each other in the system, that is composed with the component's integration on the bases of their "business process functionality".

Component's realization contract artifacts help to understand design and applicability of its transformation into model-based application process for assurance properties unification that can be achieved by its interconnection relationship behavioral study.

Application of extracted test design for an e-commerce system should follow a complete plan.

1) Scenario-based-approach for modeling business scenario to generate test scenarios from extracted Test design.

2) Selecting integration strategies of components matching with requirement specifications & their offered and used interface, design drivers and stubs.

3) Derive test scenarios from business scenarios and business flows of components, and derive test cases by analyzing business data. Limits of Conducted Practice:

The above proposed secure design strategy successfully, practiced at above mention e-commerce industry. Due to ethical right and their company policy, snapshots of integration or component test (Diagnostic specification) out come not allowed exposing publically.

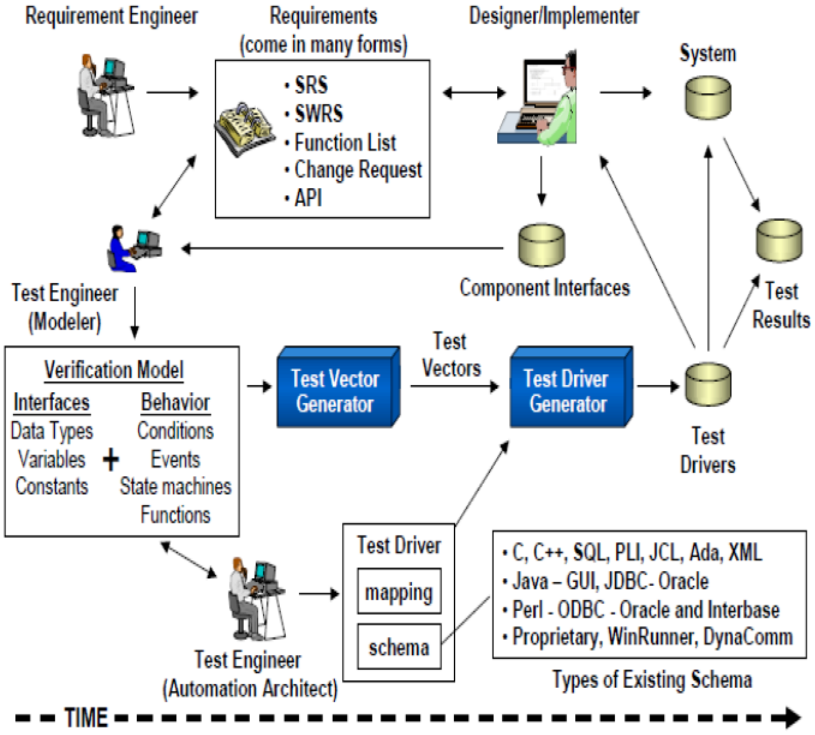


Figure 5: System s unification approach for application process & assurance properties

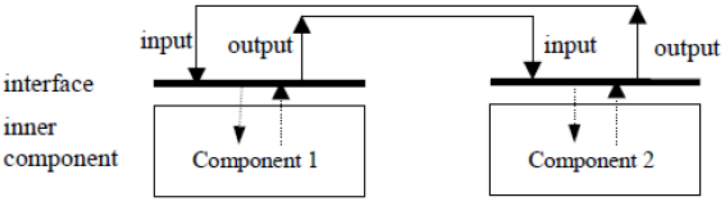


Figure 6: Component’s interconnection relationship design by contract

9 Conclusion

Attacking an application's logic involves a mixture of systematic probing and lateral thinking. As we have identified, there are various key points' checks that one should always carry out to the application's behavior.

Often, the way an application responds to these actions will point towards some defective assumption that can violate, to malicious effect.

Therefore, common sense is a appropriate tool while designing secure web application software and deploying component based business logic into the system. Application system developer must focus on security besides the functionality, because this functionality can be productive, when it works as per and within its functional control defined business policy into the e-commerce systems.

References

[1] A. K. Ghosh, *E-Commerce Security: Weak Links Best Defence*, John Wiley & Sons, New York, 1998.

[2] A. K. Ghosh, *Security and Privacy in Ecommerce*, John Wiley & Sons, 2000.

[3] A. K. Ghosh, *Security & Privacy for EBusiness*, John Wiley & Sons, 2001.

[4] F. Nabi, "Secure business application logic for e-commerce systems," *Elsevier Journal of Computers & Security*, 2005.

[5] J. Offutt, "Quality attributes of web software applications," *IEEE Software*, vol. 19, no. 2, pp. 25-32, 2002.

[6] E. Dustin, J. Rashka, D. McDiarmid, *Quality Web System: Performance, Security & Usabilty*, Adition-Wesley, Boston, 2001.

[7] P. Ritchie, The Security Risks Of Ajax/Web 2.0 Application, Elsevier, Network Security, 2007.

[8] C. V. Berghe, J. Riordan, F. Piessens, "A vulnerability taxonomy methodology applied to web services," *IBM Zurich Research Laboratory*, 2005.

[9] R. R. Raje, B. R. Bryant, M. Auguston, A. M. Olson, C. C. Burt, "A unified approach for the integration of distributed heterogeneous software components," in *Proceedings of Monterey Workshop Engineering Automation for Software Intensive System Integration*, 2001.

[10] F. Cao, B. R. Bryant, R. R. Raje, M. Auguston, A. M. Olson, and C. C. Burt, "Component specification and wrapper/glue code generation with two-level grammar using domain specific knowledge," *Lecture Notes Computer Science*, vol. 2495, Springer-Verlag, 2002.

[11] G. Simson, S. Gene, *Web Security And Commerce*, O'Reilly Publishing, 1997.

[12] M. Hung, Y. Zou, *Extracting Business Process From The Three-Tier Architecture System*, Queen's University Kingston, ON, K7L 3N6, Canada 2005.

[13] M. Hung and Y. Zou, "A framework for exacting workflows from e-commerce systems," in *Proceedings of Software Technology and Engineering Practice*, 2005.

[14] J. Viega, G. McGraw, *Building Secure Software*, John Wiley, 2006.

[15] G. Hoglund, G. McGraw, *Exploiting Software*, Adition-wesley, 2004.

[16] D. Verdon, G. McGraw, "Risk analysis in software design," *IEEE Security & Privacy*, vol. 2, no. 4, pp. 79-84, 2004.

[17] D. Allan, *Web Application Security: Automated Scanning or Manual Penetration Testing*, Jan. 2008. (`ftp://ftp.software.ibm.com/software/rational/web/whitepapers/r_wp_autoscan.pdf`)

[18] M. McIntosh and P. Austel, "XML signature element wrapping attacks and countermeasures," in *Workshop on Secure Web Services*, 2005.

[19] M. A. Rahaman, A. Schaad, and M. Rits, "Towards secure soap message exchange in a SOA," in *Workshop on Secure Web Services*, 2006.

[20] G. J. Brahnmath, R. R. Raje, A. M. Olson, M. Au-guston, B. R. Bryant, C. C. Burt, "A quality of service catalog for software components," in *Proceedings of the Southeastern Software Engineering Conference*, 2002.

[21] R. Heffner, *Planning Assumption: Giga's Model for Enterprise Application Security Integration*, Giga Information Group, June 22, 2001. (http://www.gigaweb.com)

[22] D. Schmidt, M. Stal, H. Rohnert, F. Buschmann, *Pattern-Oriented Software Architecture*, Vol. 2, Patterns for Concurrent and Networked Objects, Wiley, 2000.

[23] P. Baker, *et al.*, *Component and Integration Level Testing*, Springer Belin Heidelberg, 2007.

[24] M. Juric, R. Nagappan, R. Leander, S. J. Basha, *Professional J2EE EAI*, Wrox Press, 2002.

[25] H. G. Gross, *Component-Based-Software Testing with UML*, Springer, 2005.

Biography

Faisal Nabi is a PhD researcher at university of southern Queensland Australia. He has published some interesting papers on component-based application logic vulnerability. His main area of research is e-commerce security, specially focus on rapidly developed COTS based application design (security by design).

Jianming Yong is an Associate Professor at University of Southern Queensland.He did his PhD from SwinburneUT. His main area of research interest are Advanced Networking, Internet technology, M-Commerce,E-business, Data Integration, Workflow systems, Information system security, Network management. He is member of IEEE.

Xiaohui Tao is Senior Lecturer in Faculty of Health, Engineering and Sciences, University of Southern Queensland (USQ), Australia. Before joined USQ, I was a Research Associate with the e-Discovery Lab, Faculty of Science and Technology at Queensland University of Technology (QUT), Australia, and completed PhD in QUT as well.His research interests include Information Retrieval, Text Mining, and Knowledge Engineering.