Journal of Indian Research, Volume 1 No. 1, 2013, 73-84

Computer Science

DESIGN SECURE OBJECT ORIENTED SOFTWARE

M.U. Bokhari[*], Mahtab Alam[**], Prashant Johri[***]

ABSTRACT

The paper discusses joint work by scholars from three centres to develop a security mechanism for software development and maintenance. Security is an integral part of the widely distributed modern software systems. But, the software development process does not give primacy to this attribute. Generally, Security mechanisms are added to the existing systems as an afterthought, with all the consequent problems of insufficient security requirements, integration difficulties and mismatches between running system and the design models. We propose to integrate the design of security early into the system's development process. Object oriented software development is a new approach, which is gaining currency among developers. The standard language for modeling the design of Object-oriented systems is Unified Modeling Language (UML). The paper focuses on the security aspects of such an Object-oriented software in early design phase.

Keywords: Vulnerability, Threats, Unified Modeling Language, Object Oriented Software, Risk Assessment, Encapsulation, Polymorphism.

INTRODUCTION

Object Oriented Design (OOD) process is meant to design the classes identified at the requirement and analysis phase. During this phase ,the system is viewed as collection of objects and the operation of these classes are performed by the interaction of these objects through messages.At the design phase, a system must present unified security design that takes into account security principles. Many security properties cannot be verified by test activity alone; however verification through analyses and modeling at the design stage can increase the confidence that the specification provides a sound basis for developing a secure program [1]. There are several reasons for the lack of support of *security engineering*. Two most pertinent reasons include (i) security requirements are generally difficult to analyze and model and (ii) the lack of developer acceptance and expertise for secure software development. Security issues that must be addressed during design and development phases are not restricted to avoiding software vulnerabilities, such as buffer overflow errors, and meeting security requirements that were explicitly expressed at the beginning of the development process, but it also includes the Unified Modeling Language (UML). In recent years, UML has become a standard for object oriented modeling in the field of software engineering. Specification of the security policies using UML will ease the understanding of security requirements, so the developer acceptance for security aspects will grow. In this way, security problems will be earlier detected in the system development process, which reduces the costs and effort. Recent research concerns the integration of security engineering into the software development process [2][3].

* M.U. Bokhari is Chairman and Associate Professor, Dept. of Computer Science, Aligarh Muslim University, Aligarh,Email: mubokhari@gmail.com

** Mahtab Alam is Research Scholar, Computer Science, Mewar University, Chittorgarh, Mewar, Email: alam_mahtab@rediffmail.com

*** Prashant Johri is Professor, Dept. of Computer Science, Galgotia University, Greater Noida,Email: johri.prashat@gmail.com

SOFTWARE SECURITY

Security is a measure of the system's ability to resist unauthorized usage while still providing its services to legitimate users. An attempt to breach security is called an attack and can take a number of forms [4]. Security is fast becoming one of the most important quality attribute in Software Engineering. Critical information and transactions need to be safeguarded against malicious users and attackers. Copyright protected Commercial Off-the Shelf Components (COTS) may be used in software development. Due to the inaccessibility of source code, these components act as black boxes, whose credibility against various security attacks is not well established. Another common source of attacks is the people within an organization. *Security Engineering* is the multi-disciplinary technical field addressing such security issues. Security Engineering deals with the security and integrity of real-world systems. It deals with the development, integration, operation, administration, maintenance, and evolution of systems and applications as well as the development, delivery, and evolution of products. Security concerns must be addressed in the definition, management, and re-engineering of enterprises and business processes. Thereupon, Security engineering is delivered in a system, in a product, or as a service. Capability Maturity Models (CMMs) have been successful in setting standards and in maturing the software engineering processes. One of the main aims of Software Engineering Institute (SEI) has been to enable organizations to improve their software capability [5].

"*Systems Security Engineering Capability Maturity Model Version 3.0"* of the Software Engineering Institute, lists the goals of Security Engineering as follows:

- Gain understanding of the security risks associated with a software.
- Establish a balanced set of security needs in accordance with identified risks
- Transform security needs into security guidance to be integrated into the activities of other disciplines employed on a project and into descriptions of a system configuration or operation.
- Establish confidence or assurance in the correctness and effectiveness of security mechanisms
- Determine that operational impacts due to residual security vulnerabilities in a system or its operation are tolerable (acceptable risks).
- Integrate the efforts of all engineering disciplines and specialties into a combined understanding of the trustworthiness of a system .Security Engineering activities are practiced during all life cycle stages, such as Requirement stage, design stage, development stage, maintenance stage and testing stage.**OBJECT ORIENTED SOFTWARE DESIGN**

The Software design is the step in the software development process prior to construction. It is the step to think and plan holistically about the heuristic design to be built, and requirements for the tools or the rebuilt components needed to realize the goal . Moreover the selection of proper components and the process to properly fit those together is essential. The amount of time spent on the design of the software depends on the complexity of the software. The change in the programming paradigm from structure programming to object-oriented technology has spread to several application areas. Nowadays, object-oriented software design is the best programming paradigm, because it reduces problems to smaller and more manageable scale.

The Object Oriented Design consists of three big concepts, encapsulation, polymorphism, and inheritance, but the text "Fundamentals of Object-Oriented Design in UML" specifies that the following criteria are necessary for a language to be considered object oriented.

1. **Encapsulation** - the grouping of related ideas into unit. Encapsulating attributes and behaviors.
2. Inheritance : a class can inherit its behavior from a super class (parent class) that it extends.

3. Polymorphism : literally means “many forms”.
4. Information/implementation hiding : the use of encapsulation to keep implementation details from being externally visible.
5. State retention : the set of values an object holds.
6. Object identity : an object can be identified and treated as a distinct entity.
7. Message passing : the ability to send messages from one object to another.
8. Classes : the templates/blueprints from which objects are created.
9. Genericity : the construction of a class so that one or more of the classes it uses internally is supplied only at run time.
10. Test here
11. More testing here

SECURITY AT DESIGN PHASE

Security needs to be considered at the onset of a design rather than as an afterthought. One needs to know risks which come with awareness and knowing adversaries and their capabilities. Mitigating risks will involve building controls and security mechanisms that will impact policies, environment, and coding. If there are any weaknesses in the implementation, security becomes ineffective. Security need not be complex but failing to understand the impact of choices will obscure the difference between what is secure and what is just. The proper approach to designing a solution is one that meets business objects and that protects the confidentiality, integrity, and availability against identified risks with controls that are transparent to the user. Before digging into development and implementation, one needs to have some understanding of the requirements for security mechanisms.

Chenxi Wang [6] and William of University of Virginia, states in their work, “*Towards a Framework for Security Measurement*” that,

> *“Quantifying security is a difficult problem. The difficulties lie in that “computer security” is not a crisply defined term. Moreover, as computing systems grow larger and more complex, it is increasingly more difficult to make statements about any system-wide properties, even those better understood than security.”*

Cristian Oancea [7], in their document “*Integrating Security Policy Design in the Software Design*”state that

> *“There are at least two reasons for the lack of support for security engineering. The first reason is that security requirements are generally difficult to analyze and model. A second important reason is the lack of developer acceptance and expertise for secure software development. Acceptance is a problem because for software developers, security interferes with features and with time to market. Expertise is a problem because security policies are generally specified in terms of highly specialized security models that are not integrated with general software engineering models. This lack of integration is also detrimental to developer acceptance.”*

PRINCIPLES OF SECURE SOFTWARE DEVELOPMENT

While principles alone are not sufficient for secure software development, principles can help guide secure software development practices. Some of the earliest secure software development principles were proposed by Saltzer and Schroeder in 1974 [***8***]. These eight principles apply even today and are reproduced herein:

- *Economy of mechanism*
- *Keep the design as simple and small as possible*
- *Fail-safe defaults: Base access decisions on permission rather than exclusion.*
- *Complete mediation: Every access to every object must be checked for authority.*
- *Open design: The design should not be secret.*
- *Separation of privilege: Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.*
- *Least privilege: Every program and every user of the system should operate using the least set of privileges necessary to complete the job.*
- *Least common mechanism: Minimize the amount of mechanism common to more than one user and depended on by all users.*
- *Psychological acceptability: It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.* **[4]**

BEST PRACTICES FOR SECURE DESIGN

The collection of best practices to guide software design have been summarized after reviewing various documents ***[4][24][7][20][22][23]***. Some are listed as below:

(a) Perform Threat Analysis: Threat modeling is an engineering technique used to describe threats to the software application. Threat models are methodically developed in order to understand the risk to a system from malicious users or applications. Threat modeling allows development teams to anticipate attacks by understanding how an adversary chooses targets (assets), locates entry points and conducts an attack. A well-implemented threat model will profile the assets that need to be protected, how the threats to these assets, the attacks that could be used to realize a threat, and the conditions under which the attack may be successful. A threat model shows how adversaries view the system, its components, and where they are likely to attempt its exploitation. Not only is this important for identifying potential threats, but also in understanding what application ,defenses must be defeated in order for a threat or series of threats to be realized. For example, an e-commerce system may have the following threat, "A malicious user is able to gain an undeserved credit on their account." A threat model would specify this as:

- Threat: A malicious user is able to gain an undeserved credit on their account
- Attack: User sets item price field to a negative value
- Conditions: Server-side validation doesn't exist on item price field or server-side validation is non-existent.

A threat model will consist of multiple threats each of which can have multiple attacks. Each attack can then have multiple conditions. The conditions indicate how to defend against the threat in the design, and can be used during testing to generate security test cases.

(b) Identify Components Essential to Security: Designing the overall component structure of a system is a key activity in the design phase. Each component identified in the design must be carefully analyzed for its security needs. Some components will require protection. Other components will provide protection in the form of input validation, authentication, or cryptographic routines. These latter components must be the focus of extra attention from security engineers. For every component that implements a security function, engineers must carefully consider how that function is going to be implemented, reviewed and tested. Test planning for

security components is a crucial task that needs to be given attention as early as possible in the SDLC. Planning tests only after a component is implemented may be too late and multiply additional risk to the process.

(c) Artifacts of Secured Design Phase: The design phase builds upon the work done in the requirement phase by detailing initial conceptual solution, which is identified and prioritized business problems. There are three artifacts that come out of the design phase-Software Architecture, UML Object Model and Design Pattern ***[9].***

(d) UML Object Model: UML Object Model in design phase is used by the developer in order to show the key structural and behavioral parts of the application. UML Object Model fills the gap between requirement phase and implementation. In requirement phase, use case diagram are used in order to define a requirement, and UML Object Model consider each and every entity as an object (real world object). Thus, it also helps in coding. Breaking the application into separate slice that represents structural and intersections view is useful to drive pattern and visualized the application for review purpose ***[9].*** Reviewing object model is a good way for security team to find areas of interest to focus on application, for example:

- **State Changes:** There are many security implication to consider when state changes. When the state of sensitive data is changed then it should be recorded in audit log.
- **Context Change Trust Relationship Chain:** when application transfer data from one server to another then it is very critical from the point of view of security, as data can be stolen or modified during this process.
- **Persistence:** The data not only stored securely by application but also by other programs that deals with it, like domain controlled by DBA and System Admin.
- **Access points/Privileges Change Requirements:** Access points, authentication and authorization events are visible in design models. these provide a point of reference for the security to make sure that the security assumptions are made in the analysis phase requiring access levels for functionality usage are implemented in the design phase.
- **Identified Inputs:** In object model, the place of data input is well defined so that security team can get idea of placing input validation on data.
- **Understand Exception Scenarios:** Exceptions are the unavoidable part of every application but with the use of object model, exception handling is quite easy as it give clear understanding to the security team about probable exception conditions and a blueprint to fit in data flows in those conditions

UMLSEC

Given the widespread use of UML, it is important to include explicitly the security characteristics of a system within UML diagrams in a formalized manner. Such additional information can be added using stereotypes and attributes in the easiest case or by means of UML extensions if necessary ***[10].*** When design models (UML) are annotated with security related information, it is possible to carry on automatic verification of the design. Such tools enable the identification of defects and security threats as early as possible in the life cycle, thus reducing the cost of their correction. SecureUML is a proposal for a UML profile for security; more specifically, SecureUML extends UML to support authorization constraints ***[10].*** The capability of specifying significant information related to access control, and of deriving complete access control infrastructures (using EJB components), is presented ***[13].*** Basically, SecureUML defines a set of stereotypes and uses them to automatically produce the code that controls how users can access the different parts of the application. These stereotypes only consider the static parts of UML; dynamic ones are part of the future work.

A more ambitious UML profile is UMLsec ***[11].*** In this case, the approach is quite different: it extends state-charts and component diagrams to enforce the following security requirements: *fair exchange, secrecy/ confidentiality, secure information flow, and secure communication link.* The author describes the notation with a formal semantics based on Abstract State Machines language, to precisely analyze designed models. The main goals of UMLsec are ***[12]:***

- Evaluate UML specifications for vulnerabilities in design.
- Encapsulate established rules of security engineering.
- Make available to developers not specialized in security a simpler way to check UML models.
- Consider security from early design phases.
- Make verification cost-effective.

UMLsec offers mostly-used security requirements as stereotypes with tags (secrecy, integrity, etc…); it uses associated constraints to evaluate specifications and indicate possible vulnerabilities; it ensures that stated security requirements enforce given security policy and that UML specification satisfies the defined requirements. Among the mandatory requirements, UMLsec provides basic security requirements such as secrecy and integrity and it allows considering different threat scenarios depending on adversary strengths. Moreover it allows including important security concepts and mechanisms (e.g. access control) and it provides security primitives (e.g. (a)symmetric encryption).The main goal of UMLsec is to automatically check security properties on an extended UML model, stored in XMI standard format; then this model can be checked by a formal tool ***[14]*** to verify security requirements.

DESIGN PATTERNS

Design patterns are based on real world architecture. In the book "Design Patterns" [21], the author defines three types of patterns, which occur in the applications:

- **Creational Patters:** are responsible for constructing objects, creating their initial instance, and managing the lifespan of the object.
- **Structural Patterns:** defines objects composition in terms of its interfaces, structure and relationships.
- **Behavioral Patterns:** describe the ways that the object communicate process logic, handle state and interact with other objects.

The security team can utilize the design pattern format to develop its own security design pattern. They can design a pattern for access control where they can control authentication, authorization etc. The pattern approach is another tool that is already used in much organization and can be a boon to security team. The challenge to the security team is to find the balance between specificity, which helps developer deliver quality on time and reusability, which reaches across enterprise. The design of secured application is not an easy task; it requires deep understanding of various aspects of security, like security measurement, security categories, security policies etc. Effective management of any process requires quantification, measurement, and modeling. Metrics provide a quantitative basis for the development and validation of models of in overall process of software development. Metrics can be used to improve software productivity and quality. Quantifying security, however, is a difficult problem. The difficulties lie in that "computer security" is not a crisply defined term. We tend to know approximately what we mean by "security" and what we want it to do, but we seldom clearly state what security really means to us and how secure is "secure enough". Moreover, as computing systems grow larger and more complex, it is increasingly more difficult to make statements about any system-wide properties, even those better understood than security. Despite these problems, we believe it is a worthy effort to explore ways to measure attributes that are most important for secure application development. In this framework we are focusing on metrics and measurement of security in order to assess the security in

software development process and thus making our software development process secured. As object oriented programming languages are becoming more and more widely used, metrics are required that are specifically designed for object oriented software security. In recent years there has been an explosion of new, object oriented software metrics proposed in the literature. Unfortunately, many or most of the proposed metrics have not been sufficiently validated to measure what they suppose to measure and also there was no security consideration. An analysis of some of these metrics shows that they do not always satisfy basic properties of measurement theory. Our approach includes identification of threats; attacks and risk of design phase in object oriented software development, then identification of commonly accepted set of software security attributes, and then establishing a relationship between these attributes, so that we would be able to measure security at this phase of SDLC.

QUANTIFICATION OF SECURITY DESIGN

"*What is not measurable, make measurable*" — Galileo Galilei(1564 - 1642).

One cannot predict and control what one cannot measure [***11].*** With no certain way currently known to directly measure the security of a software product, identifying one or more surrogate measures may be possible. A surrogate measure of a product or process would produce data that correlates with the security properties of the products produced with the process***[4].***Measurement is necessary as it verifies that the product developers are capable of consistently and correctly using the process, that the process instructors properly train the developers to consistently and accurately use the process, that the correct process was properly used to develop the product, and that the process consistently produces secure software products. In addition to measuring all of the product work, measures are also required of all security-relevant product characteristics and properties. To perform in this way, software organizations must work to defined software engineering, security, and management goals and they must precisely measure and track their work throughout the product's lifetime. These measures must provide the data needed to estimate and plan the work, to establish team and personal goals, to assess project status, to report project progress to management, to verify the quality of all phases of the work, to assess the quality of all products produced, and to identify troublesome product characteristics and elements.

Such measures must also show how well the process was used and where and when corrective actions are needed. These measures can also be used to assess the quality of the training and coaching provided to the developers and their teams. Without such measures, it would be impossible to verify the consistent and proper use of the process or of the methods and practices it supports ***[15].***

(a) Selection of Units and Scales: Units and scale types determine how measurements can be achieved. For example, if we define the unit for availability as "requests served per hour", we effectively state that the measure of availability is to be derived from dividing the number of requests by a specified time frame. In choosing appropriate units and scale types, the following issues must be considered :

- **Plausibility:** The richer the scale type, the more information the measures represent. However, sometimes it is simply not possible to use a rich scale. For example, if sufficient information or measurement tools are not available, a less ambitious scale may be more appropriate.
- **Accuracy:** Accuracy is an important criterion in selecting a unit and a scale type. Sometimes a good reason to choose one unit or scale type over others is the potential measurement errors caused by one measure as opposed to others. ***[6]***

In the case of computer security, direct measurements of the end-to-end security properties are made impossible because of the scopes and structures of the modern computing systems. In these systems, security attributes are no longer functions of a single entity. They are more often functions of a host of objects and their interactions. To approximate the security strength of large systems, an estimation method must be used. There may be many estimation methods. For example, one can estimate system- reliability by sampling the history of the entire system or by doing so on each component and integrate them in some manner. What one requires is a model to relate the high-level security attributes to that of the low-level, more measurable

components ***[6].*** This methodology uses a decomposition method to develop such models — starting with high-level security properties of the system, work our way down to the basic components of the system and their interactions ***[6].***

(b) Software Measures and Measures of Secure Design: Following the specification of security related requirements, measurement objectives may be formulated that will provide insight into the satisfaction of the security requirements. Examples of measurement objectives include the following :

- What vulnerabilities have been detected in our products? Are our current development practices adequate to prevent the recurrence of the vulnerabilities?
- What process points are most vulnerable to the introduction of security-related risks (e.g., injecting reused code/modules into programs—where the variables could go unchecked, etc.)?
- What proportion of defects relate to security concerns and requirements? Do defect classification schemes include security categories?
- To what extent do practitioners comply with security-related processes and procedures?
- To what extent are security concerns addressed in the intermediate work products (requirements, design, etc.)? Have measures associated with security requirements and their implementation been defined and planned?
- What are the critical and vulnerable modules? Have vulnerabilities been identified and addressed? ***[24]***

Threat modeling or the attempt to identify likely types/sources of attack, can also form a significant guiding requirement to the development processes. A recent thesis by Stuart E. Schechter at Harvard's Department of Computer Science attempts to utilize economic models for valuing the discovery of vulnerabilities during development. His measurement of security strength depends most on threat scenarios to assign values to vulnerabilities in an effort to extend a market approach to the development process. Many risk and threat methodologies are available publicly, and Microsoft has published extensive materials that delineate the company's approach in analyzing and mitigating threat risks during the SDLC.

(c) Threat, Vulnerability and Risk identification: In order to start the process of security assessment, there is a need to identify common threat, vulnerability and risk in the software. In the figure 1.1 relationships between threat, vulnerability and risk is shown.

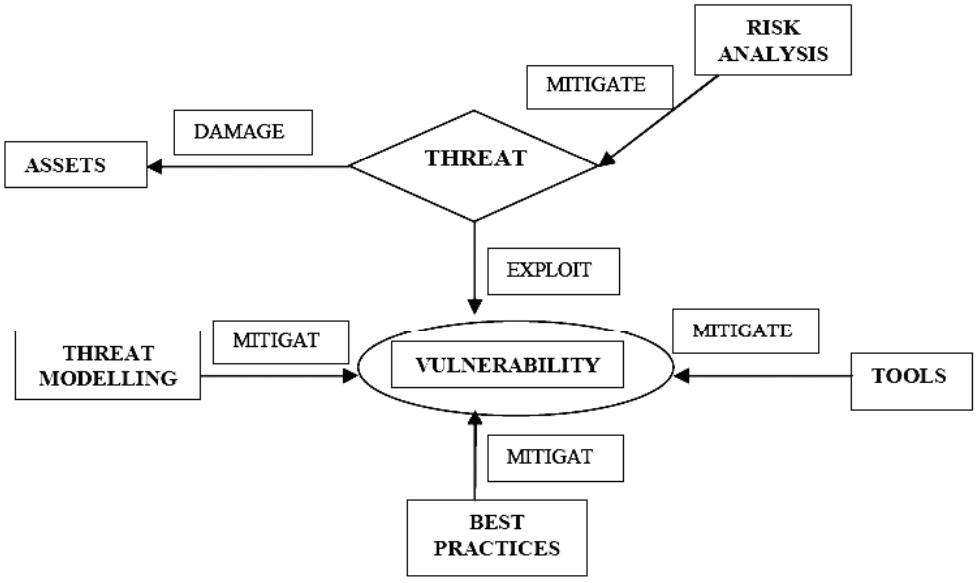

Figure-1, Relationship between Assets, threat, vulnerability and risk analysis

(d) Threats: Software threats can be general problems or an attack by one or more types of malicious programs *[16][17].*

- A threat is the possibility of a real attack which exploits a vulnerability [17]
- A threat entails the risk of an attack [17]
- Usually we cannot eliminate the threats, But we can lower the risk [17]
- We have to rate the risk of a threat to become a successful attack [17]

(e) Threats Categories: Types of common security threats that our application face, are as follows; these threats are often referred to by the acronym ***STRIDE***. These include:

1. **Spoofing identity:** an unauthorized user impersonating a valid user of the application.
2. **Tampering with data:** an attacker illegally changing or destroying data.
3. **Repudiability:** the ability of a user to deny that he or she performed an action.
4. **Information disclosure:** Sensitive data released to users or to locations that should not have access to it.
5. **Denial of service:** Acts of sabotage that make applications unavailable to users.
6. **Elevation of privilege:** A user illegally gaining an unacceptably high level of access to the application. ***[3][4][17][9].***

(f) Threat Modeling: Threat modeling is a security analysis methodology that can be used to identify risks, and guide subsequent design, coding, and testing decisions ***[4][24].*** The methodology is mainly used in the earliest phases of a project, using specifications, architectural views, data flow diagrams, activity diagrams, etc. But it can also be used with detailed design documents and code. Threat modeling addresses those threats with the potential of causing the maximum damage to an application. Overall, threat modeling involves identifying the key assets of an application, decomposing the application, identifying and categorizing the threats to each asset or component, rating the threats based on a risk ranking, and then developing threat mitigation strategies that are then implemented in designs, code, and test cases. Microsoft has defined a structured method for threat modeling, consisting of the following steps to identify assets

1. Create an architecture overview
2. Decompose the application
3. Identify the threats
4. Categorize the threats using the *STRIDE* model (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege)
5. Rank the threats using the *DREAD* categories (Damage potential, Reproducibility, Exploitability, Affected users, and Discoverability).
6. Develop threat mitigation strategies for the highest ranking threats [***5] [9][4][20].***
7. Other structured methods for threat modeling are available as well **[8].** Although some anecdotal evidence exists for the effectiveness of threat modeling in reducing security vulnerabilities, no empirical evidence is readily available.

(g) Limitation of Threat Model: There is no way through which we can discover all types of threats, using threat modeling.

- New technologies or attack strategies can result in new threat scenarios, or may uncover threats that had long been present but not discovered.

- Even when all the basic threats are known, there is no way to ensure that all of a node's wild scenarios have been discovered. If the scenario of interest has been modelled with existing threat diagrams we may find assurance in knowing these diagrams have stood the test of time. However, there is always the possibility that new attacks will appear [24].
- Finally, countermeasures are not always implemented correctly and even those that are act only to reduce threats not eliminate them. Keys may be guessed, well screened employees may be bribed, and components that functioned during a million consecutive tests may fail the next time they are used**[24].**

(h) Commonly Accepted Set of Security Attributes: The commonly accepted set of security attributes is as follows

- Confidentiality : The information requires protection from unauthorized disclosure.
- Integrity : The information must be protected from unauthorized, unanticipated, or unintentional modification.
- Authenticity : A third party must be able to verify that the content of a message has not been changed in transit.
- Non-repudiation : The origin or the receipt of a specific message must be verifiable by a third party.
- Accountability : A security goal that generates the requirement for actions of an entity to be traced uniquely to that entity.
- Availability : The information technology resource (system or data) must be available on a timely basis to meet mission requirements or to avoid substantial losses. Availability also includes ensuring that resources are used only for intended purposes. **:[20][2][18][19][22]**

(i) Relationship between Threat Type and Security Attributes: Strong authentication helps mitigate identity spoofing **[3].** Use of authorization techniques prevents data tampering and information disclosure threats **[3].** A checking of record of both authorized and unauthorized operations [Auditing] reduces repudiability **[3].** By proper availability of service, the denial of service can be avoided, and elevation of privileges can be controlled by proper techniques of confidentiality. The overall relationship between threat categories and their mitigations attributes are represented in Table-1.

Threat Category	Controlled by (Security Attributes)
Spoofing identity	Authentication, Authorization
Tempering with data	Integrity, Authorization
Repudiability	Auditing
Information disclosure	Integrity, Confidentiality
Denial of service	Availability
Elevation of privileges	Confidentiality, Integrity

Table-1 Threat categories and Security Attributes.

CONCLUSION

Software Design phase provides the blue print of the software application upon which the entire software depends. The full-proof success of secure software completely depends upon the secure design of the

software. Threat modeling is one of the best method techniques to assess the secure design. A lot of work is going on threat modeling approach. This paper presents such an approach to discover the threat related to the system and categorize them according to their consequences. A STRIDE System has been developed to assess post-categorization ranking.

REFERENCES

[1] David P. Gilliam, John D. Powell, "*Development of a Software Security Assessment Instrument to Reduce Software Security Risk*".2001. This is the result of a joint work of Jet Propulsion Laboratory ,California University and NASA. *http//:www.* trs-new.jpl.nasa.gov/dspace/bitstream/2014/12810/1/01-1121.pdf

[2] J. Jurjens. "*UMLsec: Extending UML for Secure Systems Development"*. In Proc. of UML 2002, Number 2460 in LNCS, pp.412–425.Springer, 2002.

[3] T. Lodderstedt, D. Basin, and J. Doser. "*SecureUML:A UML-Based Modeling Language for Model-Driven Security"*. In Proc. of 5th Int.Conf. on the Unified Modeling Language, Number 2460 in LNCS.Springer, 2002.

[4] P Clements Bass, R Kazman. "*Software Architecture in Practice*",Second Edition, SEI Series in Software Engineering.

[5] http://www.indusintelligence.com/IIP_SecThreats.aspx

[6] Chenxi Wang, William A. Wulf. "*Towards A Framework For Security Measurement"*, Department of computer science University of Virginia.

[7] Cristian Oancea , "*Integrating Security Policy Design in the Software Design"*. www.inf.fu-berlin.de/inst/instpub/exposeDok/expose**Oancea**.pdf,2003 last accessed on 1.12.2012

[8] Jerry Saltzer and Mike Schroeder. "*The Protection of Information in Computer Systems"*, Proceedings of the IEEE. Vol. 63, No. 9 (September 1975), pp. 1278-1308.

[9] Collaboration in Secured development process part2. Information Security Bulletin, July 2004.

[10] Paolo Falcarin and Maurizio Morisio. "*Developing Secure Software and Systems"*, Control and Computer Engineering Dept.Corso Duca degli Abruzzi, 24I-10129.

[11] ISO/IEC. ISO/IEC 13335: Information Technology- Guidelines for management of IT Security, 2001.

[12] Jan Jurjens. "*Secure Systems Development with UML*",2005.

[13] Tool for automatic J2EE deployment of secure UML, http://www.iosoftware. com/products/data/mda_security_factsheet_020902.pdf

[14] Ministry of Defence. *Defence Standard 00*{56 issue 2: Safety Management Requirements for Defence Systems, 1996.

[15] Improving Security across SDLC ,Task Force Report 2004

[16] Computer Knowledge: http://www.cknow.com/vtutor/SoftwareThreats.html

[17] Steven F Burns.*Threat Modeling: A Process To Ensure Application Security*, January 2005,http://www.sans.org/rr/whitepapers/securecode/1646.php

[18] Dr. Daniel Guinier. *Object-oriented software for auditing information systems security following a methodology for IS risk analysis and optimization per level*

[19] Marianne Swanson. *Security Self-Assessment Guide for Information Technology Systems.* NIST.Nov 2001

[20] A. S. Sodiya, S. A. Onashoga, and O. B. Ajayi. "*Towards Building Secure Software Systems"* ,University of Agriculture, Abeokuta, Nigeria.

[21] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *"Design patterns: Elements of Reusable Object oriented Software*", Additon –Wesley 1995

[22] Tim Grance,Joan Hash,Marc Stevens. *Recommendations of the National Institute of Standards and Technology*. NIST(National Institute of Standard and Technology)

[23] *Improving Security Across Software Development Life Cycle.* Task Force Report April 1 2004.

[24] James H Stock and Mark W. Watson. *Introduction to Econometrics*. Pearson Education, Boston, MA, 2003.