ISSN No. 0976-5697

Volume 7, No. 1, January-February 2016

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Threat Modeling for a Secured Software Development

S. Shanmuga Priya
Senior Assistant Professor
Department of Computer Science and Engineering
New Horizon College of Engineering
Bangalore, Karnataka, India

S. S. Arya
Assistant Professor
Department of Computer Science and Engineering
New Horizon College of Engineering
Bangalore, Karnataka, India

Abstract: Security system evaluation is an important and essential activity which could be conducted at different stages in the life cycle of a software development. Understanding the threats to the software is critical step in creating a secure product. Recent researches have demonstrated that threat modeling can provide a basis for building flawless software that can withstand any potential attack. Threat modeling plays an important role in developing secure software which describes a threat subjected on to a system and the harm that could arise on vulnerabilities. The vulnerabilities or loop holes in the securities arise as flaws in requirement specification or designing or results in incomplete implementation or throw out several bugs in testing stage. Increase in the complexity of the software, possibly introduces more design errors which leads to security vulnerabilities and increases security attacks. Hence, it is insisted that the security issues must no more be considered as non-functional requirements and isolated to single phase alone in Software Development Life Cycle (SDLC). These security issues can be well identified using the threat modeling followed in almost all the phases in SDLC.

Keywords: Software Development Life Cycle (SDLC), Threat Modeling, Threat, Requirement, Design, Development, Testing.

I. INTRODUCTION

Threat modeling aims at identifying the critical assets of the system, analyze those, document it and prioritize the system's vulnerability issues. It also helps in identifying the entry point and exit point of the system. The entry point is by which an attacker could enter into the system for attacking and through the exit point he leaves the system after attacking. These entry point and exit point are needed to be identified much in advance during the design so that there won't be any loopholes left for the attackers to invade the system. Threat modeling must be addressed early in the Software Development Life Cycle, so that the organization might minimize or eliminate threats at a proper time and prevents it getting dissipated to the subsequent phases. After collecting the requirements from the user, the designer should take at most attention while designing the software. The architect must have a crucial thinking in all perspectives before designing the software. The architect must understand the assets of the system, the implementation details, the architecture need to protect the system being attacked etc. The designer must think in terms of the attacker, the possible assets that can be attacked, the ways by which those assets can be attacked are to be analyzed properly. After collecting the various attack possibilities, a visualization diagram can be drawn with the available information by focusing on the ways by which an attacker could possibly attack the system. Most probably, Data Flow Diagram (DFD) or UML is the obvious choice in depicting the understandings about the system and is used for producing the attack visualization as well as data flow. DFD is preferred for a structured programming whereas UML is used for the object oriented programming. However, researchers are focusing on the technique of making the visualization of DFD in UML. This might help the developer to develop a secured code and tester tests the system in a more effective way to give a secured system to the end user.

The major aim of the attackers is to steal the valuable information from the organization. As a result of these attacks, it happens for the customer as well as the organization to lose their confidential information and is no longer secure. Hence, these criminal issues must be focused rite from the initial stage of any Software Development Life Cycle. In the SDLC, the security is considered as a non-functional requirement and hence often it loses it identity and never finds a consideration inside the development environment. In order to build flawless software, it is recommended that the security must be considered as a functional requirement. The different stakeholders like the architects, developer, tester, project manager, business manager are the major beneficiary from the threat modeling.

The paper is organized as follows. Section II list out the security objectives, Section III gives the threats and security policies, Section IV throws the insight on the threat modeling, Section V suggests the various possibilities on when to do threat modeling, Section VI briefs out the overall threat modeling process, Section VII gives the threat modeling throughout secured software development life cycle process, Section VIII gives the threat modeling for security requirement elicitation, Section IX gives the threat modeling for design level security, Section X describes the security development considerations, Section XI gives the security testing and Section XII concludes.

II. SECURITY OBJECTIVES

The six security objectives that must be measured in any system to enforce security as given by Shawn Hernan et al. [1] are:

a) Confidentiality (C), which includes protection of the system against unauthorized information disclosure,
b) Integrity (I), which includes preventing unauthorized information changes that affects the software,
c) Availability (A), that includes providing the required service for the legitimate user. In addition to the basic CIA objectives,
d) Authentication,
e) Authorization and
f) Repudiations.

Identifying security objectives helps in understanding the goals of the attacker and the area that needs keen observation to protect against the attack can also be explored.

III. THREATS AND SECURITY PROPERTIES

A way to ensure that the software under construction meets the security objectives is to employ threat modeling using STRIDE classification (Microsoft Model) an acronym for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege [1]. The following table gives the threats mapped to the objectives which protect the software.

Table 1. Threat Mapping to Security Objectives

Threat	Security Property
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-Repudiation
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

IV. THREAT MODELING

Threat modeling provides a systematic way in identifying the threats that could affect the system. Identified threats must be analyzed well in advance which aids to uncover the vulnerabilities in the system. Analyzing the risks of the threats, countermeasures could be provided that addresses the threats and mitigates the risks. Swiderske and Snyder [2] listed the purpose of threat modeling as:

a) Understand the threat profile of a system.
b) Provide a basis for secure design and implementation.
c) Discover vulnerabilities.
d) Provide feedback for the application security life cycle.

A. Threat Modeling Approaches

By threat modeling approach, threats are represented using a separate model designed by professionals by taking various security attacks and its consequences into account. Generally there are three approaches to threat modeling in practice. They are:

a) Software – Centric,
b) Attacker – Centric represented as Attack Trees [3,4] and
c) Systems – Centric (or) Threat – Centric represented as Fault Tree [3].

Each of these approaches can be followed separately depending upon the system under construction. However, in order to have an effective solution, it is recommended to follow a combination of all the three approaches, called a hybrid approach in threat modeling.

B. Threat Visualization

The threat visualization (representation) must be both attacker – centric and system – centric. Drake Patrick Mirembe and Maybin Myeba [4] have analyzed the various representations of threat models. The various threat representation techniques are:

a) Fault Trees
b) Attack Trees
c) Attack Suites
d) Attack Nets
e) Mitigation Tree.

a) Fault Trees

Fault trees published in 1960, are a graphical representation of system failures [2]. The nodes in the tree represent the event and edges represented as a casual-effect relationship between events. Leaf node is linked to the higher nodes in the hierarchy via logic gates. Non-leaf nodes represent hazards identified. The fault tree lacks in expressiveness due to their inability to capture atomic details about the threat like attacker tools, familiarity, skill, motivation and goals.

b) Attack Trees

Schneier was the first to coin the term attack tree [5]. He proposed attack tree as a formal way to describe the security of the system based on varying attacks. The root node represents the goal of an attacker and leaf nodes represent the different ways by which the goal of an attacker could be achieved. Nodes are decomposed by AND and OR relationship. Values like cost that needed to be spared to achieve the goal or probability to do a task can be assigned to the nodes. The value of the root node says whether the system's goal is vulnerable to attack. Attacker's characteristics must be analyzed in order to determine which part of the attack tree need close consideration from rest parts. The advantage of attack tree is it helps to study the system from attacker's point of view and helps in analyzing the system by evaluating the impact of applying countermeasures. The major limitations of the attack tree are: the designer and developer should have a sound knowledge about the attacker's characteristics to model the tree. Moreover, Schneier does not discuss on how the attack trees could be linked to other development artifacts such as designer, developer or tester of the software.

c) Attack Suite

Attack suite is the enhancement made to the attack trees and defined by using algebraic semantics. The attack is characterized as a finite non-empty, multi-set of components.

A universal set N containing components with various combinations results into different attacks. The set N is given as:

$$N = \{x_0, x_1, x_2, x3,\ldots x_{n-1}\}$$

where x_i, $0< i < n-1$ gives the unique component. The major drawback of the attack suite is it sometimes introduces more complexity in representing the threat model and makes it hard to understand. Hence it is widely not in use.

d) Attack Nets

J. P. McDermott [6] has proposed the attack net approach to penetration testing. Attack nets provide a graphical means to show how a collection of flaws may be combined to achieve a significant system penetration. An attack net retains the essential benefits of attack tree and also provides the alternatives and refinement of the attack tree approach. Attack nets can model more sophisticated attacks that may combine several flaws. They are used to organize the development of plausible attack scenario.

e) Mitigation Tree

Guifre Ruiz et al. [7] proposed a new data structure know as mitigation tree to detect threats in software designs, which is similar to attack trees but with a slight variation. Attack trees are built in a destructive way, whereas mitigation trees are built in a constructive way. Mitigation Tree has the goal of mitigating the determined threat and each branch contains the set of software specification or features, for the design and implementation activities needed to accomplish the goal of the root. In addition, each feature contains an estimated cost of carrying it out.

V. WHEN TO DO THREAT MODELING

Irrespective of the nature of the software being built either simple or complex, security issues must be addressed as early as possible, in every phases of the Software Development Life Cycle (SDLC). Threat modeling must be a continuous iterative process. The reasons are two folded. First, it is impossible to identify all possible threats in single pass. Secondly, the changing business requirements must be enhanced and adapted as the system is being built.

Figure 1 represents the threat modeling scope in any software development life cycle. It emphasizes that threat modeling must be an iterative process and it can be introduced as early as possible. Threat modeling is not only associated with design phase, it can also be considered as an important part of requirement phase and can be executed continuously throughout SDLC. In the design phase, threat modeling covers the vulnerabilities that could be introduced due to lack in the security requirement specification. In code development phase, vulnerabilities could arise due to poor implementation. If the security concerns are addressed earlier in software development, it leads to effective, less cost, and less time consumption in building a flawless secured system.

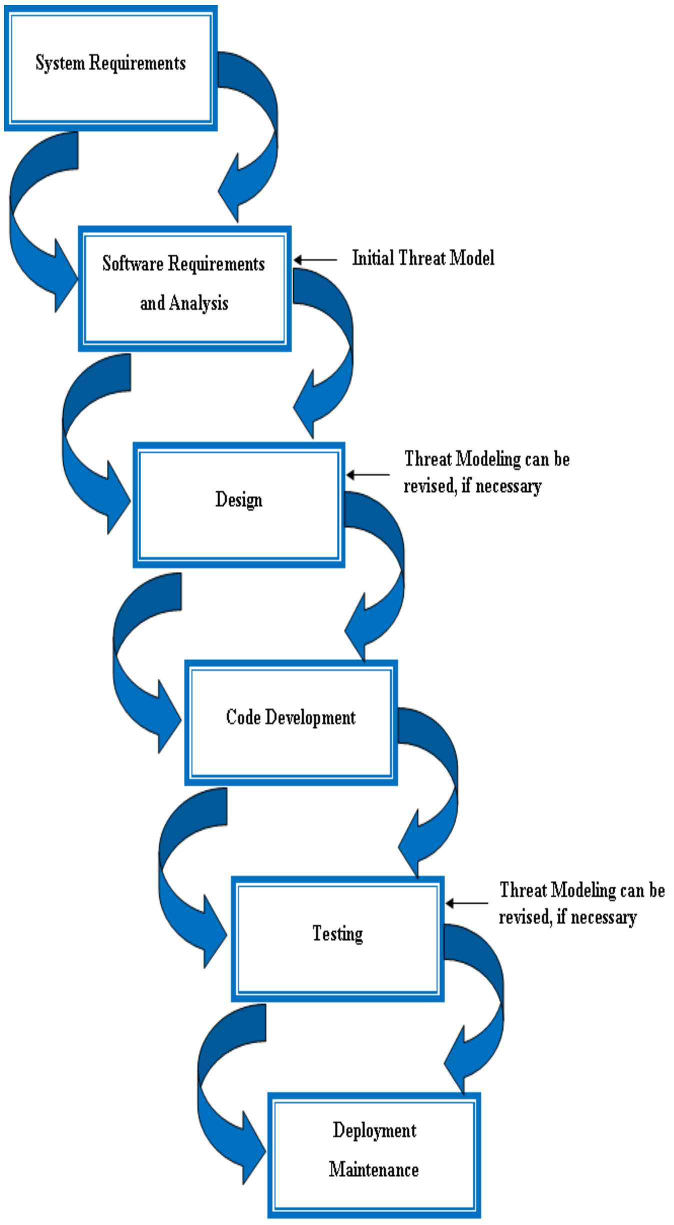
System Requirements
Software Requirements and Analysis
Initial Threat Model
Design
Threat Modeling can be revised, if necessary
Code Development
Testing
Threat Modeling can be revised, if necessary
Deployment Maintenance

Figure 1 Threat Modeling in Software Development Life Cycle

VI. THREAT MODELING PROCESS

There are various threats modeling process in existence.

A. *Suvda's Threat Modeling Process*

Suvda Myagmar et al. [8] have investigated on threat modeling which can be used as a foundation for specifying security requirements, upon which rest of the security system is built. They have also presented three case studies: Software-Defined Radio, a network traffic monitoring tool and a cluster security monitoring tool.

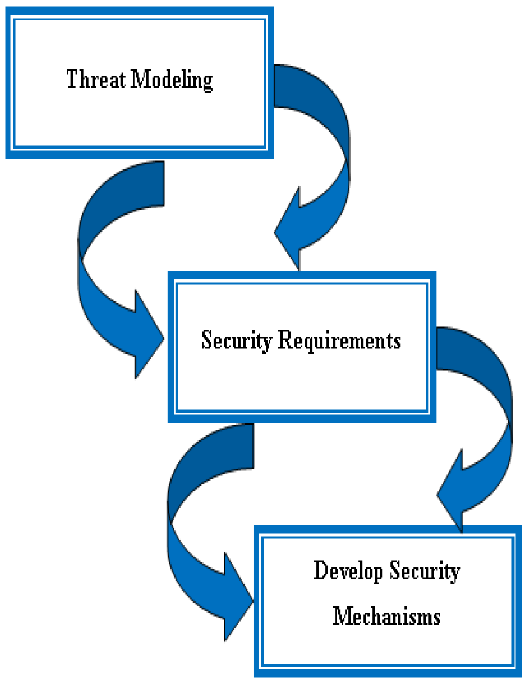

Figure 2 System Security Engineering

Figure 2 shows the view of system security engineering. Threat modeling is used to identify all possible threats to a system. In security requirement formation, the identified threats are analyzed based on their dangerous impact and decisions could be made whether to mitigate it or not. Based on the risk associated secure mechanisms can be developed which could be followed in rest phases of SDLC.

The threat modeling proposed by Suvda Myagmar et al. [8] consists of 3 high-level steps:

a) Characterizing the system,
b) Identifying assets and access points and
c) Identifying threats.

Characterizing the system involves creation of a system model after understanding the system components. Identifying the assets and access point involves in identifying the valuable assets of a system that must be secured and identifies the potential ways by which the attacker might enter and exit a system. The threat to a system can come from either inside or outside the system, authorized or unauthorized that could violate the security objectives.

B. *Microsoft Threat Modeling Process*

Figure 3 gives the threat modeling process proposed by Microsoft which can be performed as a six-stage process:

a) Identify Assets: It identifies valuable assets that the system needs to protect.
b) Create an Architecture overview: It involves the tasks as identifies what the application might do, create an architecture diagram and identify the technologies that could be used for implementing the system.
c) Decompose the application involves the tasks as: Identify trust boundaries, identify data flow (DFD – Use case diagram can be used), identify entry points, identify privilege code and document the security profile.
d) Identify the threats involves two basic approaches as: Use STRIDE to identify threats and use categorized threat list.
e) Document the treat : This could be achieved by filling a template that shows several threat attributes, such as threat description, threat target, risk associated, attack techniques, countermeasure etc.
f) Rating the threat: It is achieved by calculating the product of probability of the threat occurred and damage potential.

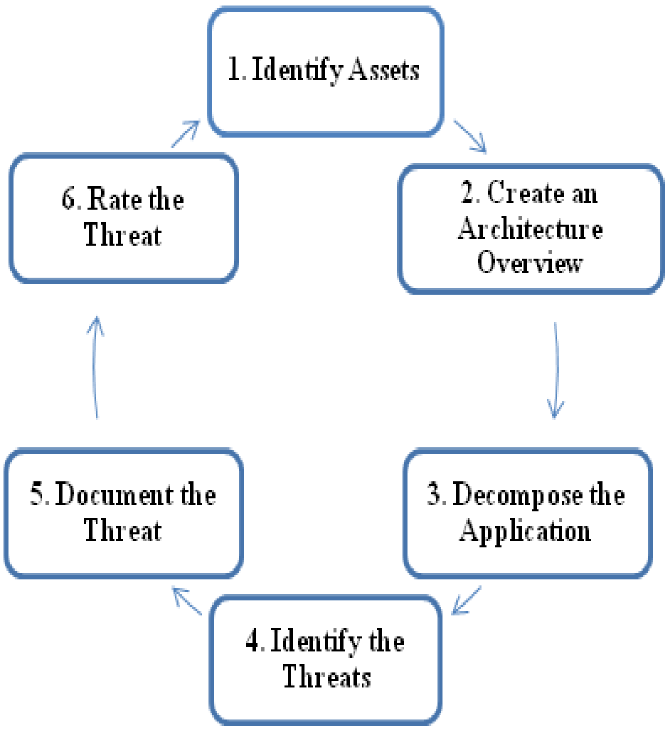

Figure 3 Microsoft Threat Modeling Process

At Microsoft, DREAD (acronym for Damage potential, Reproducibility, Exploitability, Affected users, Discoverability) model is used to calculate the risk and threats and are rated as High (1), Medium (2) and Low (3). Based on the obtained rating, decisions can be made on whether to mitigate the threat or not.

C. *An Approach to Threat Modeling in Web Application Security Analysis*

In [9], Sreenivasa Rao and Kumar proposed a study on question-driven approach to Threat Modeling. This approach helps in identifying security design problems in the application design process. An application overview was created to identify relevant threats. This was achieved by using deployment diagram. Further, in order to identify the threats more clearly, an in depth knowledge and understanding of the application is essential. To do these, the role of the application, key usage scenarios, technologies involved in building the application and various application security mechanisms were identified in detail. The application must be decomposed in order to understand the data flow through the application from entry to exit, entry points and exit points must be identified through which an attacker must enter and leave a system. The threats and attacks to an application were identified by using two basic approaches: starting with common threats and attacks, then using a query-driven approach in which the STRIDE model could be used to post questions related to the architecture and design of the application. Threat identification task involves in identifying the common threats and attack, then identify threats along use case and data flow.

VII. THREAT MODELING THROUGHOUT SECURED SOFTWARE DEVELOPMENT LIFE CYCLE PROCESS

According to studies made, the security design flaws are widespread. These defects give a high impact in business and are easily exploitable which brings threat to the entire system. It is estimated that fixing the design flaws is costly. IBM Systems Sciences Institute studies say that the cost spent on fixing flaw during design phase is 7 times cheaper than fixing flaws during implementation and 100 times cheaper than during production. So following secured software development process at the beginning is a cost effective solution. The various phases in Security Software Development Life Cycle (SSDLC) are:

a) Security Requirement Elicitation
b) Design Level Security
c) Security Development Consideration
d) Security Testing

Threat Modeling finds its place in all the phases of SSDLC.

VIII. THREAT MODELING FOR SECURITY REQUIREMENT ELICITATION

Requirement Engineering is considered as the main building block for any software, as this phase directly deals with the customer in understanding the needs that a product must satisfy. The requirements vary depending upon the project to be built. Traditional requirement engineering considers the functional requirements and it pay less attention to the non-functional requirements like security, reliability, robustness, and scalability, so on. The security issues that could arise in a software product are mostly attended when there is a demand, if any, from the customer. Most probably after the product is deployed, the end user after starting to use the product might come out with certain defects in concern with security. These security issues must be considered as functional properties of the system being built and must be addressed in the beginning stage itself. As this security requirement elicitation phase is considered to be the foundation of a product on which the other phases are built, it is advisable to have a strong foundation.

A. Categories of Security Requirements

In [10] Paco Hope and Peter White classified security requirements into three categories as:

a) *Functional Security Requirements* is a security description, which could be derived from misuse case that is integrated into each functional requirement. Representatively, this requirement category also says what shall not happen.

b) *Non – Functional Security Requirements* usually derived from the architectural principles and the standard practices followed. This brings out the properties that are security related architectural requirements, like "robustness" or "minimal performance and scalability".

c) *Derived Security Requirements* is just like a hybrid combination that is derived from functional and non-functional security requirements.

In [9] Malik Imran Daud, has categorized the security requirements as:

a) Functional Security Requirements,
b) Non-Functional Security Requirements,
c) Derived Security Requirements,
d) User Stories (Agile Software), and
e) Abuse Case.

B. Threat Modeling for Security Requirement Elicitation

In [8] Suvda Myagmar et al. proposed threat modeling and used it for specifying security requirements. Threat modeling involves identifying the various threats possible on a system and a deep understanding on the identified threats. The loopholes that leads to vulnerability, if exists in a system could be possibly exploited by the attackers. These issues must be well identified in the beginning itself possibly during gathering and analysis of security requirements, so that a decision can be made on whether to mitigate or accept the risks involved with those identified threats. This early analysis aids to turn out for a secured system.

In [13] Lee M. Clagett gave the threat modeling process initiated by identifying the system assets and possible threats to those assets. A threat exists when an entry point leads to the access of an asset. Attacks to achieve the threat can be represented using different diagrams that help in producing a visual effect. Possible diagrams could be misuse case [25, 26], abuse case [14, 15] use case diagrams, state chart diagram, petri nets etc. This representation of attacks on a system paves a way to decide whether the threat could be mitigated or not.

As security is constantly changing one, there are various difficulties in gathering the security requirements. Few things that must be taken into account during gathering and analyzing requirements are software security requirements must be state in positive tone, must be stated in language and platform independent way, and must be verifiable and testable.

IX. THREAT MODELING FOR DESIGN LEVEL SECURITY

The design level security is the next phase followed by security requirement elicitation. The architects play a major role in designing the system. The other stakeholders involving in this phase are designers and developers. The stakeholders make a complete study on the requirement specification. They bring out the secure design elements, the architecture, secure design reviews needed at different levels, and conduct threat modeling as per the specified requirements. Every time traceability is done between the requirement gathered and the design being carried out in order to ensure that the design goes as per the customer demand. The designer is supposed to prepare a design specification that technically focuses on how the system is to be implemented. The various functional and non-functional requirement specifications are necessarily to be tackled in bringing out the essential security features to be implemented. This gives a guideline to the developers to implement a secure software and trustworthy system. Figure 4 shows the overview of design phase of secure software development life cycle that must be considered effectively and efficiently.

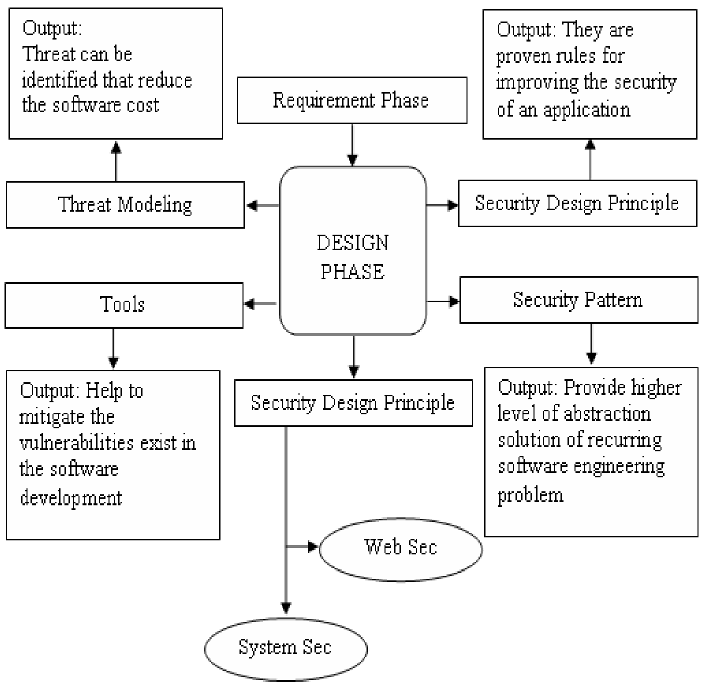
Output: Threat can be identified that reduce the software cost
Requirement Phase
Output: They are proven rules for improving the security of an application
Threat Modeling
DESIGN PHASE
Security Design Principle
Tools
Security Pattern
Output: Help to mitigate the vulnerabilities exist in the software development
Security Design Principle
Output: Provide higher level of abstraction solution of recurring software engineering problem
Web Sec
System Sec

Figure 4 Overview of design phase of secure software development life cycle [16]

A. *Security Design Principles*

Various security design principles are followed up which need to be known in advance as they provide a clear guideline on how to design the secured system. Few general guidelines are:

a) From the security requirement specification, analyze and identify the valuable assets of the system that need to be protected.
b) Make a clear understanding on the goal of the system to be implemented.
c) Understand the mechanism and the environment in which the developed software is going to be in use.
d) Identify the threats that are possible on the software.
e) Understand the threat and categorize them as active threat or passive threat, so that according to the nature, the threats can be resolved by prioritizing it.
f) Carefully make an analysis on the malware (i.e., Virus, Trojans, Worms or other malicious software) in existence could compromise the software or data associated could be damaged. If so, make additional steps in solving the issues.
g) Identify the vulnerabilities and analyze the consequences of those.
h) Identify the enemies of the system, in what way they could attack the system, the various skills that an attacker must possess in order to attack the system, the various possible entry point that exist for attacking the system and the various possible exit point by which the attacker could leave after attacking the system must be analyzed.

The Security Design Principles as described by Saltzer and Schroeder [17] are:

a) Principle of Least Privilege,
b) Principle of Fail-Safe Defaults,
c) Principle of Economy of Mechanism aka KISS Principle,
d) Principle of Complete Mediation,
e) Principle of Least Common Mechanism and
f) Principle of Psychological Acceptability.

B. *Threat Modeling for Design Level Security*

Threat modeling is an iterative process followed for modeling security threats. It helps to identify design flaws that could be exploited by the attackers and also facilitate in taking the countermeasures to be implemented that could mitigate the identified threats. The threat modeling can be followed in all the SSDLC phases, but it must be considered more serious and essential in designing phase. The major steps of threat modeling in the design level security are:

a) Identify security problems,
b) Investigate potential threats,
c) Investigate potential vulnerabilities, and
d) Provide solutions for the identified vulnerabilities.

The architect, designers and the program managers could get participated in threat modeling, so that they all could share their views from their side and provide solutions from their point of view and could come out with a feasible solution that could be put into practice. These when put into practice, helps in planning for security test easily, helps in reducing the software support cost as well, because the vulnerabilities were identified well in advanced during the design phase itself and developed accordingly. Such product when gets into production, security defects might be reduced greatly which in turn might improve the quality of the product and brings customer satisfaction.

Many researchers advocates that it is best practice to follow a diagrammatic representation by using any formal model (mathematical model) like Petri Nets or semi formal model like UML diagrams (especially State Chart Diagram, Activity Diagram, Use Case Diagram, Sequence Diagram) could be followed in producing the architecture diagram of the system during design phase. These models give a mental model i.e., visual representation of the system that is going to be developed. One of the major advantages of this diagrammatic representation is it helps all the stake holders to understand the system uniquely rather than everyone having their own illusion of what the system is completely about. The systematic approach followed to create a threat model was proposed by Meire J.D et al. [18]. It is an iterative approach which can be used throughout the SSDLC. The Figure 5 shows the iterative threat modeling process.

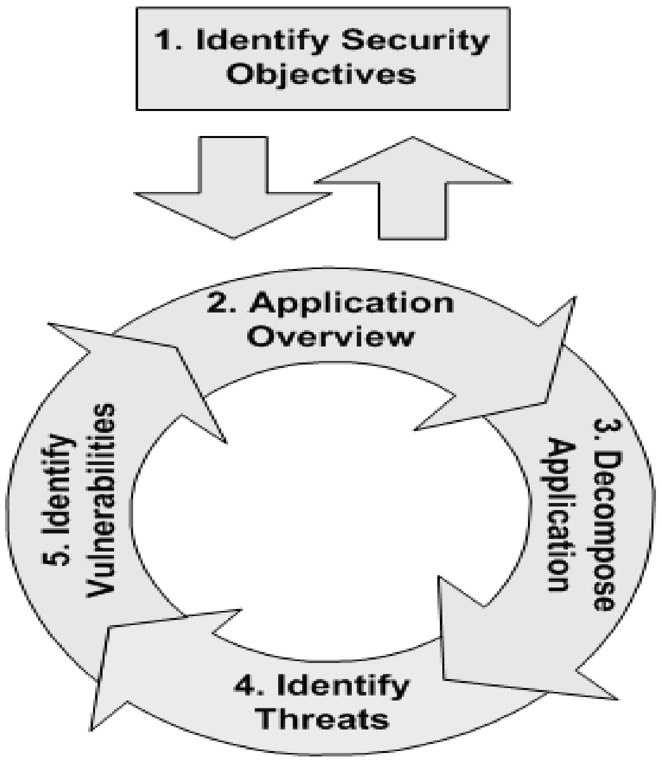

Figure 5 Iterative threat modeling process [16]

The five threat modeling steps given are:

a) Identify security objectives,
b) Create an application overview,
c) Decompose your application,
d) Identify threats, and
e) Identify vulnerabilities.

C. Security Patterns for Design Phase

In [19], Joseph Yoder and Jeffrey Barceló were first to adapt design patterns for information security. Even the design needs to be documented and it’s a good practice to document list out what the system is required to do as well as what the system is not supposed to do. This gives a clear idea for the developer to implement the system perfectly by considering both does and don’ts of the system. It also helps the tester to test the system efficiently by developing test cases which covers both does and don’ts of the system. The security patterns produced are:

a) Single Access Point,
b) Check Point,
c) Role,
d) Session,
e) Full View with Errors,
f) Limited View,
g) Secure Access Layer,
h) Level Privilege,
i) Journaling, and
j) Exit Gracefully.

At the end of the design, the attack surface is analyzed. When the attack surface area is high, above process is repeated until the attack surface is reduced to the minimum level.

In [20] Nobukazu Yoshioka et al. proposed security patterns for each phase of SSDLC in terms of security concepts. They have also shown the patterns for requirement phase, design phase and implementation phase. They have also added the methodologies that help in developing secure software when those security patters are adopted.

X. SECURITY DEVELOPMENT CONSIDERATIONS

When the architecture designed for the SSDLC produced by using the threat modeling gives a good layout for the next phase, development. During this phase, the program developer implements the code and test the developed code. When the security requirements and the designing are laid out properly, but if the software is poorly coded, it leads to vulnerabilities that could result in non-secure software which is an undesirable one. The threat modeling gives important guidance and the developers must pay attention to ensure the code correctness. Developer must focus on the testing to ensure that the possible threats identified during the threat modeling in design phase are blocked or mitigated. This kind of testing will improve to block the entry and exit point for the attackers. In connection to these developments, the developer must have an eye on the various other possible threat attacks in order to produce an up to date product.

A. Impementation Elements of Secure Development Life Cycle

In [21], Steve Lipner and Michael Howard have given the essential elements for Secure Development Life Cycle which can be applied in the implementation phase. When the coding standards are followed, it helps the developers to avoid the flaws being injected into the software which in turn can lead to secured, flawless, vulnerable free software.

In [22] Agrawal and Khan gave a software vulnerability detection and analysis framework (SVDAF) which is independent to development life cycle. The produced framework aims on vulnerability analysis which is used to analyze the vulnerability inputs that can be supplied at each phases and a report on these vulnerabilities are given as a feedback to the SDLC so that the inputs can be modified accordingly.

The common security bugs that could occur in the construction phase are:

a) Incorrect or incomplete input validation,
b) Poor or missing exception handling,
c) Race Condition,
d) Buffer Overflows,
e) SQL Injection,
f) Cross-Site Scripting (XSS), and
g) Integer Overflow.

XI. SECURITY TESTING

In traditional testing, the software tester might have some limitations in testing. But in security testing for a product, the tester must not have any boundaries in testing in order to uncover different classes of the errors. A good tester must take minimum time to discover the errors with minimum effort. It is advisable for the software tester to play the role of an attacker while testing the software. The tester can give inputs like the manner how an attacker do so that he could break the system. This kind of testing could help in making a tight secured code so that it might prevent the attackers being intruding the system. Security testing must be risk-based rather than being requirement based. The tester must be trained with security aspects of the software, only then it will be easy for him to test the software. Lack of security awareness won't bring all the expected security acts into existence. The security testing must not be limited to the testing phase alone, instead it must be followed throughout the SDLC phases. Concentrating in the security issues at early stage is more costly in software development so sometimes security bugs remain undetected. But early stage concentration in these issues might cut down the later cost involved in mending the software after production.

The various security testing methods that are widely in focus are:

a) Vulnerability Assessment,
b) Negative Testing,
c) Penetration Testing,
d) Ethical Hacking,
e) Fuzz Testing, and
f) Fault Injection

Aaron Marback et al. [23] gave a security testing approach which is three step activities. First a threat model was constructed using threat tree for visualization; secondly security test sequences were generated from the threat tree and finally by taking the valid and invalid inputs in to account executable test cases were created.

Linzhang et al. [24] proposed threats modeled for achieving a linkage between the models (used for designing, done by using UML Sequence Diagram), code implementation and the security testing. They extracted threat traces from design-level model, made a decision on the kind of information collected during runtime, produced instrumented code for it by using the guide information, and later carried out security testing. The execution traces were analyzed and verified whether it contains any security violation. If it contains any reports were prepared and actions were proposed to mitigate the threat in the proposed system.

XII. CONCLUSION

There are different software engineering approaches like waterfall model, spiral model, prototyping model, rapid application development model, incremental model are followed for developing software. Even though these are efficient software application development, somewhere around the corner, security issues were neglected and hence much of the products fail in the market. Security issues become a major concern in SSDLC. This paper depicts the necessity of implementing threat modeling as a security-analysis methodology and spotted out its importance in each SSDLC phases. This could give awareness to the software builders the importance of threat modeling for making flawless software.

XIII. REFERENCES

[1] Shawn Hernan, Scott Lambert, Tomasz Ostwald, Adam Shostack, "Uncover Security Design Flaws using the STRIDE Approach" msdn .microsoft.com, Nov. 2006. Available:http://msdn.microsoft.com/en-us/magazine/ cc163519 .aspx

[1] F. Swiderski and W. Snyder. Threat Modeling. Microsoft Press, 2004.

[2] Drake Patrick Mirembe and Maybin Myeba, Fault Tree Analysis Tool, Jan 2008.

[3] Drake Patrick Mirembe, Maybin Muyeba, "Threat Modeling Revisited: Improving Expressiveness of Attack," Second UKSIM European Symposium on Computer Modeling and Simulation, pp.93-98, 2008.

[4] Bruce Schneier, "Modeling Security Threats: Attack Trees", Dr. Dobb's Journal of Software Tools 24, pp.1-9, December 1999.

[5] McDermott, J.P. "Attack Net Penetration Testing.", Proceeding of the Workshop on New Security Paradigm, Sept. 2000.

[6] Guifre Ruiz, Elisa Heymann, , Eduardo Cesar and Barton P. Miller, "Automating Threat Modeling through the Software Development Life-Cycle", Sep. 2012. http://research.cs.wisc.edu/mist/papers/Guifre-sep2012.pdf

[7] Suvda Myagmar, Adam J. Lee, and William Yurcik, "Threat Modeling as a Basis for Security Requirements", IEEE Symposium on Requirements Engineering for Information Security (SREIS '05), Paris, France, Aug 2005.

[8] B. Sreenivasa Rao and N. Kumar, "An Approach to Threat Modeling in Web Application Security Analysis", Journal of Computer Applications ISSN: 0974 – 1925, Volume-5, Issue EICA2012-5, February 10, 2012.

[9] Paco Hope and Peter White, "Software Security Requirement the Foundation for Security", Cigital Inc., Available: http://sqgne.org/presentations/2007-08/Hope-Sep-2007.pdf.

[10] Malik Imran Daud, "Secure Software Development Model: A Guide for Secure Software Life Cycle", Proceedings of the International MultiConference of Engineers and Computer Scientists, Vol. I, IMECS, Hong Kong, March 17-19, 2010.

[11] Suvda Myagmar, Adam J. Lee, and William Yurcik, "Threat Modeling as a Basis for Security Requirements", IEEE Symposium on Requirements Engineering for Information Security (SREIS), August 2005.

[12] Lee M. Clagett, "Security Requirements for the Prevention of Modern Software Vulnerabilities and a Process for Incorporation into Classic Software Development Lifecycles", Thesis dissertation.

[13] Chun Wei (Johnny), Sia, "Misuse Cases and Abuse Cases in Eliciting Security Requirements", 25 Oct 2005.

[14] Martyn Fetcher, Howard Chivers, Jim Austin, "Combining Functional and Security Requirements' Processes", ROLLS ROYCE PLC-REPORT-PNR, Vol. 93025, 2005.

[15] Swapnesh Taterh, Yadav K. P., Sharma S. K., "Threat Modeling and Security Pattern used in Design Phase of Secure Software Development Life Cycle", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, Issue 4, April 2012.

[16] Saltzer, Jerome H. and Schroeder, Michael D., "The Protection of Information in Computer Systems", Proceedings of the IEEE 63, pp. 1278-1308, September 1975.

[17] Meier J.D., Alex Mackman, Blaine Wastell, "Threat Modeling Web Applications Patterns & Practices Library", Microsoft Corporation, May 2005. Available: http://msdn.microsoft.com/en-us/library/ff648006.aspx.

[18] Joseph W. Yoder and Jeffrey Barcalow (1997), "Architectural Patterns for Enabling Application Security", Proceedings of 4th Conference on Patterns Languages of Programs (PLoP'97) Monticello, Illinois.

[19] Nobukazu Yoshioka, Hironori Washizaki and Katsuhiasa Maruyama, "A Survey on Security Ptterns — Progress in Informatics", No.5, pp.35-47, 2006.

[20] Steve Lipner and Michael Howard, "The Trustworthy Computing Security Development Lifecycle", Security Engineering and Communications, Security Business and Technology Unit, Microsoft Corporation, March 2005.

[21] Agrawal A. and Khan R. A., “A Framework to Detect and Analyze Software Vulnerabilities – Development Phase Perspective”, International Journal of Recent Trends in Engineering, Vol. 2, No. 2, November 2009.

[22] Aaron Marback, Hyunsook Do, Ke He, Samuel Kondamarri and Dianxiang Xu, “A Threat Model-Based Approach to Security Testing”, Software-Practice and Experience, Vol. 43, pp. 241–258.

[23] Linzhang Wang, Eric Wong, Dianxiang Xu, “A Threat Model Driven Approach for Security Testing”, 29th International Conference on Software Engineering Workshops(ICSEW'07), 2007.

[24] Alexander .I, “ Misuse Cases: Use Cases with Hostile Intent”, *IEEE* Software 2003, Vol. 20(1), pp.58–66.

[25] Sindre .G, Opdahl .A,” Eliciting Security Requirements with Misuse Cases”, In Proceedings of TOOLS Pacific, pp. 120–131, 2000.