Suraksha: A Security Designers’ Workbench

Vineet Kumar Maurya¹, Santhosh Babu G¹, Jangam Ebenezer³, Muni Sekhar V¹,
Asoke K Talukder¹, Alwyn Roshan Pais¹,

Dept. of Computer Engineering, National Institute of Technology Karnataka, Surathkal
vineet.nitks@gmail.com, santhosh003@gmail.com, ebenezer.jangam@gmail.com,
munisek@gmail.com, asoke.talukder@gmail.com, alwyn.pais@gmail.com

Abstract — To design a secure software system, a security designer needs a workbench so that security of the system can be embedded into the system from the very early stages of Software Development Life Cycle (SDLC). Among the proposed approaches to elicit security aspects in the early stages of SDLC, researchers have suggested different techniques. However, a workbench that will facilitate a designer to build security right from the requirement analysis is not available. This paper presents a Security Designers’ Workbench to achieve this.

Keywords — Security Designers’ Workbench, Attack tree, Misuse case, Security requirement analysis, STRIDE, DREAD, Security Patterns, Security Development Lifecycle

I. INTRODUCTION

In 1968 October NATO Science Committee organized a conference on Software Engineering [1]. Delegates of this conference discussed the Software Crisis. They also discussed how this crisis could be addressed through Software Engineering. In last forty years different techniques in Software Engineering have been proposed to address this crisis. Today Software Engineering is a well-established discipline that includes engineering practices that includes requirement elicitation, design, construction, testing, and maintenance of software.

Combined with the growth of Internet and telecommunication, world is today experiencing a different type of crisis – this is the Software Security Crisis. Due to lack of security considerations, software that are being developed lack proper security considerations and quite often they are compromised. As a result the assets behind these software are not always secured and safe. To address this crisis we need Secure Software Engineering.

Security has been considered to be a nonfunctional requirement. The nonfunctional requirement is defined as – “A requirement definition that includes all requirements that are not functional”. It can also be defined as how the software will behave for a function that is not defined or known. The functional requirement is defined as – “A system or software requirement that specifies a function that a system or software system or its component must be capable of performing. These are requirements that define behavior of the system or the software”.

Till sometime ago, security for a software system was managed through perimetric security. It was not mandatory that applications should be security aware. That is why security has always been an afterthought. Now-a-days, it is obligatory to develop security aware applications. Security needs to be embedded into the system from the early stages of Software Development Life Cycle to develop a security aware application. Hence, security should be considered as functional requirement to avert the present Software Security Crisis.

There are several benefits if security is considered as functional requirement and embedded into the Software Development Life Cycle right from requirement analysis phase. The earlier is the focus on security, the more is the insight obtained into the security aspects of the system. Various advantages of identifying security aspects in early development stages are outlined in [2]. The early focus on security gives the security designer an opportunity to find out countermeasures also in the earlier phases of lifecycle. Developer also can be aware of security aspects of the system before developing the system. These factors pave way to build a secure system against different kinds of attacks. Unfortunately, there are no tools to support security designer to identify and implant security aspects from the very early phases of development.

In this paper we present a system named Suraksha, which is a Security Designer’s Workbench that comprises of interfaces that help designing secure software system. Suraksha in Sanskrit means security. This tool helps a security designer to elicit security requirement follows by security design.

In functional requirement analysis, we start with Use case; however, in security related non-functional requirement we start with Misuse cases. Misuse cases specify all the actions the system should not do as mentioned in [3, 4].
The organization of the paper is as follows. Section 2 explains the related work done in this area and available tool support. Section 3 gives the outline of the holistic approach recommended for a Security Designer. Section 4 provides information about various features of Security Designers’ Workbench Suraksha. We conclude in section 5.

II. RELATED WORK

Use cases are not sufficient to elicitate, communicate and document security requirements of a system. Hence, Misuse cases are proposed by Sindre and Opdahl [3] to represent the security requirements of a system. Textual representation of Misuse cases was proposed and explained in [3, 5]. They gave a five step procedure for Misuse case development [5]. Misuse cases can be either lightweight or extensive. If a Misuse case is lightweight, it does not aid developers; and, if it is extensive, it will be useful for later development stages [6].

According to Ian Alexander, an approach combining Use and Misuse cases can be applied at any level in systems engineering [7]. Moreover, it was stated that sometimes it is worth documenting Misuse case scenarios in detail and other times just the name of Misuse case is enough [8].

As said by Donald G. Firesmith [9], security use cases should be used to specify requirements that the system shall successfully protect itself from its relevant security threats while Misuse cases are suitable for analyzing and specifying security threats.

While introducing attack trees, Bruce Schneier [10] stated that attack trees provide a formal methodical way of describing the security of systems, based on varying attacks. Attack trees or threat trees provide insight into the attack or threat mentioned at root node. Child nodes of a node are connected using either AND or OR component.

Misuse case helps to identify the cases that need to be further analyzed. Attack tree provides the detailed information about different ways of performing an attack [10, 11]. Attack Trees and Misuse Cases provide complementary information. In the one hand, attack Trees provide a systematic way to symbolize attacks. On the other hand, a Misuse case that has additional information that is more accessible and useful to a developer. The combination of attack Tree and Misuse Case gives more information to designers as mentioned in [12].

STRIDE is a methodology for identifying possible threats [13, 14, 15]. It is used by Microsoft for threat modeling of their systems. Threats are identified by exploring the possibilities of Spoofing Identity, Tampering with Data, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege in the given case.

The DREAD methodology [15, 16] is another tool to determine possible threats and their impact. This acronym is also formed from the first letter of each category. DREAD modeling not only tries to identify a threat, but it also influences the thinking behind setting the risk rating, and is also used directly to mitigate the risks. While calculating the risk of a threat, its Damage Potential, Reproducibility, Exploitability, Affected Users and Discoverability of the threat are considered [16].

A design pattern is a formal way of documenting successful solutions to problems. The idea of design patterns was introduced by Christopher Alexander and has been adapted for various other disciplines. The goal of patterns in software engineering is to create a body of literature to help software community to map recurring problems into patterns through the software development lifecycle [17].

Thus researchers gave various kinds of approaches that are explained in various papers. But, the absence of holistic approach brings confusion and ambiguity in combining different approaches.

As explained in [17] by Asoke K Talukder, all of the above techniques when effectively combined together in security requirement analysis phase, can be used to reduce the attack surface. This literature also describes how to do the construction of the software in a secure fashion followed by security testing.

Regarding the available tool support, the present situation is not at all encouraging. Even though the combination of all the approaches is promising for security requirement analysis, hardly any tools support all the approaches. Even tools supporting individual techniques are rarely found. Misuse case modeling tool is not available and a few tools only support attack tree. Open source tools are hardly ever available for attack trees and Misuse cases. Absence of holistic approach and lack of open source tools are genuine matters of concern for a software designer.

III. OUR APPROACH

Mamadou, Jose, Susan and Debra anticipated that the combination of Misuse cases and attack trees will provide more information to the designer [12]. Moreover Sindre and Opdahl generalized the concept of Misuse cases [18]. They also suggested that combination of Misuse cases and attack trees is beneficial [6]. This combination forms an alternative to common criteria for specifying security requirements [12]. While the Common Criteria is comprehensive and thorough, it does not provide any tool to quantify and implement it. Furthermore, it does not provide a way to specify detailed
information such as preconditions in security objectives that might prove useful for the designers [12]. Considering STRIDE to identify threats and DREAD to calculate the risk of the threat, our approach reduces attack surface by effectively combining all the techniques together. Also concept of security patterns is used for formal documentation of recurring problems. Moreover, this is a Open source tool that allows the community to improvise it.

Step 1: At the very first step system objectives are identified. Assets are also identified along with their associated risks. An example of asset identification and prioritization is presented in Fig. 1. We followed the procedure explained in [19] to identify and prioritize assets. As a first step, a brainstorming session is conducted and all the valuable assets are listed. Next step is to examine various existing documents for other important assets. Once all the assets are listed, the assets are categorized and prioritized with respect to security. To perform this, an asset is taken and viewed from different perspectives i.e. customer, administrator and attacker. From each perspective, each asset gets assigned a number indicating the importance of confidentiality, integrity or availability for this asset. All the priorities of each asset are added and the asset with lowest sum is ranked as highest priority asset.

Figure 1: Assets identification for e-commerce Application

Step 2: Functional requirements of the system are analyzed using Use case and UML tools. Use case diagram specifying the functional requirements of the system is developed.

Step 3: For each actor in the above Use case, a misactor (one or more) is assumed and STRIDE [13, 14, 15] concept is applied in connection with each action and assets related to it. Here we use the same terminology and same graphical notations as suggested by Sindre [3]. The possibilities for Spoofing Identity, Tampering with Data, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege in the given case by misactor are explored. This yields a list of possible abstract threats. A light weight Misuse case is developed. This is explained in detail in Section IV.

Step 4: Each abstract threat in the above Misuse case diagram is considered as a root node and corresponding attack tree is developed to understand what are the AND and OR components in the threat path. This is shown in Figure 3. Here the user goes through each and every Misuse case. Suraksha has many libraries of threat models that help model the threat tree. User can use these trees as it is or enhance to create the threat forest.

Step 5: For each attack tree, DREAD is used to rate a threat. This is done by assigning values to each node beginning from leaf nodes. The values are percolated up after calculation at each node. Finally, some price is added to these threats and then compared with cost of assets. If it is too expensive to secure an asset compared to the cost of the asset, those threats need not be considered. DREAD tool is also used to prioritize the threat mitigation plan.

Step 6: After careful examination of each threat using DREAD, the threats are classified as ‘considered’ and ‘neglected’. For each threat considered, identification is given as a high risk or moderate risk or low risk.

Step 7: Detailed information about each threat is now available in the form of corresponding attack tree. Using the available information in the form of a forest (group of attack trees), Misuse case diagram in step 3 is modified and now extensive misuse case is developed.

Step 8: Following DREAD, the security requirement is finalized. Here all these threats that filter through the DREAD rating are considered as functional requirement for a security aware system. Protection of assets through security that was considered a non-functional requirement then becomes a functional requirement. Use case diagram and Misuse case diagram are represented together with these Misuse cases converted into a Use case and the misactor converted into an actor. Possible mitigation techniques are suggested in the form of security Use cases.

Step 9: In next step all the Security Patterns are considered and ensured that these patterns are included in design. If there are patterns but no use-case, the iteration starts from Step 1.
All the security patterns are listed below. Joseph Yoder and Jeffrey Barcalow were first to adapt security design patterns for information security. It is easy to document what the system is required to do. However, it is quite difficult and sometime impossible to define what a system is not supposed to do. The Yoder and Barcalow paper presented seven patterns in 1998 for security design [20]. They are:

1) **Single Access Point**: Providing a security module and a way to log into the system. This pattern suggests that keep only one way to enter into the system.

2) **Check Point**: Organizing security checks and their repercussions. Authentication and authorization are two basic entity of this pattern.

3) **Roles**: Organizing users with similar security privileges.

4) **Session**: Localizing global information in a multi-user environment.

5) **Full View with Errors**: Provide a full view to users, showing exceptions when needed.

6) **Limited View**: Allowing users to only see what they have access to.

7) **Secure Access Layer**: Integrating application security with low-level security.

To manage the security challenges of networked computers today, we need to look at many more design patterns. However, following patterns must be included in any security system [17].

8) **Least Privilege**: Privilege state should be shortest lived state.

9) **Journaling**: Keep a complete record of usage of resource.

10) **Exit Gracefully**: Designing systems to fail in a secure manner.

Finally, the attack surface is analyzed and above process is repeated until the attack surface is reduced to the required level.

The modified misuse case diagram serves as an input to threat modeling in the next phases of software development life cycle. Thus, this misuse case diagram enhances the process of threat modeling [21, 22]. Moreover, this misuse case diagram along with different attack trees will be available for a security designer.

As an example, we have presented the e-commerce application that was used by Sindre and Opdahl [3] throughout this paper. Two actors considered in this example are operator and customer. Operator or administrator has responsibility to maintain the system. Customer is authorized person to browse catalog and order goods. Crook is a malicious user who always tries to exploit the vulnerabilities of the system. Various misactions that can be performed by a crook are Denial of service, Spoofing identity and Information disclosure.

### IV. SURAKSHA: OPEN SOURCE TOOL

At National Institute of Technology Karnataka Surathkal, we have developed the Open Source Security Designers’ Workbench tool named Suraksha. The tool is freely downloadable from http://isea.nitk.ac.in/suraksha. Suraksha supports various important features like assets identification and prioritization, Textual representation of Misuse cases using Misuse case template, Co-representing Use and Misuse Cases, attack tree development, DREAD Rating, Support for security patterns etc.

**Figure 2**: Misuse-Case diagram for e-commerce Application

*Suraksha* offers support for asset identification and prioritization as represented in Figure 1. Assets are classified into two categories based on the mobility. They are static assets and assets in transit [17]. *Suraksha* facilitates user to list all the valuable assets. It promotes not only listing valuable assets but also prioritizing assets based on the values assigned by the user. User needs to enter values for Confidentiality, Integrity and Availability from the perspective of stakeholder and attacker. User need to choose one from three numbers 1, 2 and 3 provided using drop down box. Value 1 denotes high priority indicating that the importance of confidentiality, integrity or availability for the corresponding asset is more.
user didn’t select any value, default value is taken as 4. Once the assignment of all values is finished, the sum of all values for each asset is calculated. The lesser is the value of sum for an asset, the higher is the priority given to the asset.

User can easily add actor node, misactor node, use case node, misuse case node and can easily draw various relationship between them like extend, mitigate, threaten etc by selecting suitable item from the panel. To represent a system behavior, attempt should be to include both functional and non-functional requirements together. And, to represent this in graphical fashion, Use case and Misuse case should be combined to define the system. To represent Use cases and Misuse cases together, these need to be differentiated. Therefore, Use case is black in white and Misuse case is shown in an inverted format – white in black. Suraksha provides attractive GUI to draw Use cases and Misuse cases and to co-represent them. Within a short time interval, user can draw Misuse cases and Use cases easily using this tool. The size of the diagram can be adjusted as required. In the example presented in figure 2, operator and customer are actors and their corresponding permitted actions are shown. Crook is also represented along with some of the possible malicious actions.

Suraksha offers user friendly GUI for the user to document the textual representation of Misuse cases. Sindre and Opdahl focused on templates for Misuse cases in [23]. Suraksha uses the Misuse case template suggested by them. We considered the example used by them in [23]. The textual representation of the example Misuse case is shown in Figure 3. User can enter corresponding information against each field in the provided text boxes. After completion of giving all the necessary information, user can save the textual representation.

Suraksha provides attractive GUI for attack tree development as depicted in Figure 4. For each abstract threat mentioned in Misuse case, detailed information about the threat can be obtained by drawing an attack tree corresponding to the threat. User can draw an attack tree easily using this tool starting with abstract threat as root node. Various paths possible to achieve the goal (root node) are explored. User can easily draw all possibilities by creating children to a node and connect these children using AND or OR component. AND component is represented by straight line and OR component is represented using double line arc. User just need to select the required items from panel and can place the items in required position. To facilitate the designer, there are some standards threat models available in the library and can be used by the user. These threat models help to identify various attacks and their relationship. In real system these threats need to be mitigated. Also, the impact of these threats needs to be measured.
To measure the impact of each threat, *Suraksha* uses DREAD methodology. When a node in an attack tree is selected and right clicked, there is provision for the user to enter suitable values for Damage Potential, Reproducibility, Exploitability, Affected Users and Discoverability. By selecting proper radio button, user can choose either 0 or 5 or 10. After completion of giving information, risk associated with corresponding node is calculated automatically based on the formula mentioned below:

\[
\text{Risk DREAD} = \frac{(D+ R + E +A + D)}{5}
\]

Where,
- \(D\) = Damage Potential,
- \(R\) = Reproducibility,
- \(E\) = Exploitability,
- \(A\) = Affected Users,
- \(D\) = Discoverability.

The calculation always produces a number between 0 and 10; the higher the number, the more serious the risk. The corresponding snapshot with DREAD rating for the attack tree is shown in Figure 5.

![Figure 4: Attack tree for Intrude into Customer Account in e-commerce application](image1.png)

![Figure 5: Attack tree with DREAD Rating for Intrude into Customer Account in e-commerce Application](image2.png)

*Suraksha* provides unique provision to work with misuse cases and attack trees. For each misaction in lightweight Misuse case, an attack tree with misaction as root node can be created. It facilitates the user to think of various ways of performing the attack in the perspective of a hacker. Depending on rating and values obtained for different threats using attack trees, threats are filtered and corresponding extensive Misuse case diagram can be generated. Thus this tool reaps the benefits of using attack trees and Misuse cases together. This extensive Misuse case diagram can be saved and forms an excellent input for threat modeling.

Finally, it is very important to provide formal documentation for various problems along with their solutions. The purpose of this documentation is to use these solutions again when the same problems are encountered in future. For recurring problems this approach is very useful. *Suraksha* offers vital support for security designers to solve recurring problems using security patterns. A documented pattern contains details about the problem that the pattern addresses, the context in which this pattern should be used, the solution etc.
V. CONCLUSION

This paper focuses on a security designers’ workbench named Suraksha. This Open Source workbench tool is developed at the Department of Computer Engineering, National Institute of Technology Karnataka, Surathkal. The workbench allows a security professional to design security aware applications. The workbench allows user to analyze and capture security requirements through misuse case, followed by identification of threats through attack tree. Once the attack paths are known, the tool allows the user to rate different threats and to prioritize them. This results in a list of possible attacks. These attacks along with their corresponding countermeasures are included as part of functional requirement. This is then used to design the secure system through security patterns. Thus, this open source tool supports security designer to elicit security requirements in the early phases of Software Development Lifecycle (SDLC). As an example we analyzed a part of e-commerce application using this tool. Also, a holistic approach to use various techniques proposed by researchers is outlined.

REFERENCES


