Adopting Knowledge Based Security System for Software Development Life Cycle

Jalal Alowibdi[1]
[1]Department of Computer Science, College of Engineering, University of Illinois at Chicago
jalowi2@uic.edu

Abstract— *The high-demand from the software industry led to the development of many Software Development Life Cycle (SDLC) models that help produce high quality software within budget and time constraints. Most of these SDLC models do not completely cover security as early as possible in the development cycle. Since security is a major concern to the users and the developers, adopting it at the early stages of the SDLC could help to ensure integrity, accessibility and confidentiality in future systems. It is still unclear how to achieve a perfectly secured software system by modifying the SDLC models. In this paper, the Knowledge Based Security System (KBSS) model is proposed to help in modeling and specifying security at all stages of SDLC in an effort to achieve a maximally secured software system. KBSS is a system that categorizes, clusters, monitors, alerts, and controls the Security Knowledge Management by the knowledge of the Security Expert Team, who are able to identify, collect, organize, manage, retrieve, provide and store all aspects of security functions and issues.*

Keywords: computer security, software life-cycle, knowledge-based software engineering

1. Introduction

Software Systems (SS) play an integral role in many organizations and companies. SS often hold private and confidential information about organizations and companies who are using them. Ensuring the integrity, accessibility and confidentiality of such information in SS is a major concern in the software industry. Protecting the infrastructure of SS from threats and vulnerabilities could reduce the overall risk of "cyber attacks" [1]. Technically, including the security at the early stages of the Software Development Life Cycle (SDLC) helps reduce the overall risk of cyber-attacks as well as the cost and the time of restoring system operations that might arise after the deployment of the SS. Focusing on security at the early stages of the SDLC might produce well protected SS that prevent cyber-attacks on threats, vulnerabilities, and defects including man in the middle, denial of service, buffer overflow, SQL injections, and password attacks. It is critical and hard to guarantee secured SS all the time. However, applying a "Learn from Mistakes" approach could ensure just that. The approach is to build a Knowledge Base Security System (KBSS) that covers the most common threats, vulnerabilities and defects available up-to-date in the industry. Understanding these security issues, including abuse cases, as early as possible in the SDLC could help the developers to devote appropriate attention to security issues.

In this research, we seek to study different models that have been used in the industry. In addition, we attempt to model the security requirements using KBSS in order to provide up-to-date security functions and issues. These security functions and issues must consider all stages of SDLC as Iterative and Incremental Development Process (IIDP). This paper is organized as follows: Section 2 provides the related work in security at the early stages. Section 3 introduces the relation between security and the SDLC at all stages. The proposed model of KBSS within the SDLC at all stages, using IIDP model, discusses in Section 4. Section 5 discusses future work.

2. Related Work

A great deal of research has been conducted in the field of adopting security at the early stages of system development. Most of the research has been focused on security at the early stages of the SDLC, ideally at the requirements engineering stage. Very little research has been conducted on security at all stages. In this section, we seek to explore an initial concrete study of different models that have been used in the past. There are several studies related to security in the SDLC which can be summarized as follows:

McGraw [2] focuses on the security of the software at the early stages of the SDLC. Based on his research, he applies security concerns in the traditional waterfall model of the SDLC. Rigorously, it starts with the security requirements at the requirement stage to clarify the following questions: "How to protect? What to protect? and Whom to protect from?" Also, he describes the abuse cases. At the design and analysis stage, he identifies the security privilege and documents possible attacks, by a so-called risk analysis. At the code stage, he concentrates on implementation flaws, discovering the common vulnerabilities, and fixing bugs. Finally, he intends to determine feasibility of an attack by testing the overall functionality, called penetration testing.

Viega [3] used Comprehensive Lightweight Application Security Process (CLASP) to build security requirements.

Clearly, CLASP is "a set of process components that design to help the development teams to build and improve the security of the SS" [3]. The CLASP approach consists of four basic steps. These steps can be summarized as follows: (1) Identify system roles and resources; (2) Categorize the resources and the roles that make the process manageable (e.g., user-highly-confidential, user-confidential, user-low-confidential, system-private, and public); (3) Identify the resource interactions that relate to the roles and resource categorization; (4) Write the requirement specification. Also, the author defines the security services of CLASP as follow: Authorization, Authentication and Integrity, Confidentiality, Availability, and Accountability. His recommendation is to use the extension of the SMART+ requirements. Those requirements are Specific, Measurable, Attainable, Reasonable, Traceable, and + Appropriate.

Mead and Stenshney [4] defined the Security Quality Requirements Engineering (SQUARE) method, which "provides achievement for eliciting, categorizing, and prioritizing security requirements for the SS" [4]. Moreover, the SQUARE methodology builds security concepts into the early stages of the SDLC by documenting and analyzing security. The SQUARE method is divided into nine steps each having its own input, technique, participants and output. Rigorously, SQUARE starts with common security definitions and then identifies the goals of these security definitions. Once the organization has defined the common goals, it starts transforming them into deliverable requirements. It follows by choosing the elicitation techniques. These requirements can be categorized to meet the business goals. Next, risk assessment is performed with respect to the categorized requirements. The categorized requirements are prioritized. Then, this is followed by the inspection by the stakeholder as final step.

Romero-Mariona, Ziv, and Richardson [5] surveyed several approaches supporting the security requirements engineering in later stages of the SDLC. They explore six main areas to be considered in Later Stage Support (LSS) which represent support for the security requirements. First, security requirements integration provides support for integrating security requirements in the later stages of the SDLC. Second, constraint consideration expresses new constraints for security purposes. Third, security-oriented test cases provide validation of the security requirements. Fourth, test cases are developed alongside the security requirements. Fifth, the efforts to produce secure software are estimated. Sixth, the integration of other type of requirements focuses on non-security requirements.

Vetterling and Wimmel [6] combine a phase-oriented software development with Common Criteria (CC). They apply a so-called Target Of Evaluation (TOE) to assess the security requirements of the product based on the CC. The security requirement of the CC can be simplified into security functional requirements and security assurance requirements. Both are important to meet security objectives of the TOE. Moreover, the authors list the products of different assurance classes of the CC. Those classes are security target, configuration management plan, design and representation, life cycle documentation, test documentation, vulnerability assessment, guidance documents, and delivery and operation documentations.

Table 1 summarizes the previous survey by comparing the methodologies with respect to the following criteria: Defined Security Objectives (DSO), Adopted Method (AM), Target Development Process (TDP), Target Stage (TS), Iterative and Incremental Approach (IIA), Documented Security Functions and Issues (DSFI), Adopted Monitoring and Alerting Security System (AMASS), Used Heuristic Security Approach (UHSA), and Used Deterministic Security Approach (UDSA). Respectively, the proposed Knowledge Based Security System KBSS methodology is corresponding to all criteria.

Table 1: Summary compares the related works.

	M [2]	V [3]	M, S [4]	R, Z, R [5]	V, W [6]
DSO	YES	YES	YES	YES	YES
AM	NONE	SMART+	SQUARE	LSS	CC
TDP	waterfall	NONE	NONE	NONE	NONE
TS	ALL	REQ	REQ	REQ	REQ
IIA	NONE	NONE	NONE	NONE	NONE
DSFI	NONE	NONE	NONE	NONE	NONE
AMASS	NONE	NONE	NONE	NONE	NONE
UHSA	YES	NONE	NONE	NONE	NONE
UDSA	NONE	NONE	NONE	NONE	NONE

3. Security and the SDLC

Traditionally, many users think that a SS product is just code. we view a SS as a complete process methodology. It starts from the requirements stage and ends with deployment. The most famous SDLC is the waterfall model. There are many other models that have been used in the industry such as extreme programming, rapid application development, iterative and incremental development, the rational unified process, the spiral model, scrum development, and agile software development. However, most SDLC models do not completely cover security as much as they could. Additionally, the requirements stage is the earliest stage of the SDLC in all models. There are many sub-topics under the requirements to be considered within our proposed KBSS model as inputs and/or outputs of the requirements stage. Among them are stakeholder identification and system prototypes. These sub-topics might help in describing the security of the SS for our KBSS model. Good requirements engineering leads to a successful project within budget and time constraints. Security concerns are usually included in the non-functional requirements of a SS. However, our research focuses on security as part of functional requirements and constraints as well.

The increase of the SS vulnerabilities has made the software industry rethink security in their SDLC models. A major challenge in SS security is anticipating all plausible future attacks and ensuring that these attacks will not succeed. When an attack is discovered or expected, an appropriate solution is provided. The technical details of preventing the attacks are in continuous evolution. We have to extend our knowledge by understanding existent cyber-threats and by guessing future possible cyber-attacks. As long as there are malicious users trying to attack different SS using different techniques, there will be a challenge of providing 100% secured SS. In addition, we must also provide an iterative procedure to ensure SS security by incrementally expanding our KBSS.

3.1 Security Objectives and Attacks

Ideally, there are many security objectives to be considered, simulated and identified to have a secured SS. Each SS has its own security objectives that might be shared or not. Identifying the common objectives at the early stages of the SDLC lead to ensure the SS security. Getting familiar with the objectives will help to understand them from different aspects of our proposed KBSS. The common shared objectives are:

- Authentication is the concept of confirming the correctness of the identity of a user, and assuring the process of the system is trusted [7], [10]. An example of authentication is as follows: A user X plans to use the software system Y, Y must ask X for information that is provided by X and then verified by Y to authenticate X claiming identity is valid and trusted.
- Authorization is the concept of defining and specifying access rights or policies such as permit or deny to a user and/or a process of system [3], [7]. Ideally, authorization is a kind of approval or permission. For example, if the user X shows proper identification to the software system Y, Y authenticates X. Then, Y would grant authorization to X to access a set of certain information. Other information that is not related to X would not be allowed to access.
- Confidentiality is the concept of ensuring that the data is protected, secured, viewable and accessible only to the authorized user and preventing unauthorized user [7], [12]. An example of confidentiality is as follows: When the user X intends to access data D from the software system Y, Y must ensure that X has a permission to access D.
- Availability is the concept of ensuring the SS is reliable, runnable, serviceable and accessible to legitimate users [7]. Also, it is the concept of protecting the SS from unauthorized users who intend to make the SS unavailable. For example, when the user X decides to access the software system Y, Y must be available and ready for X.
- Accountability is the concept of ensuring that the interaction and communication between the users and the SS is trusted, traceable, reliable and accountable [7], [8]. For example, when the user X plans to use the software system Y, X would provide Y with information and Y would respond to X. The interaction and communication between X and Y must be accountable.
- Integrity is the concept of ensuring the quality, correctness and consistency of the data during the processing operations such as data transfer, data storage, data retrieval, and data quality. Also, it is the concept of protecting the data of SS from unauthorized users who intend to destroy the data during operation [7], [11]. For example, a user X intends to exchange data and information from the software system Y, the provided data must be accurate, correct, and complete. Y assures that the data does not get corrupted or deleted accidentally by unauthorized users who intend to change or delete the data during operation.
- Non-Repudiation is the concept of ensuring the communications of send and receive between the SS and legitimate user without being deniable [7], [9]. For example, a user X asks the software system Y for data D. Y sends X the required D, and X receives D from Y, where Y non-repudiation objective is to prevent X from denying receiving D in future.
- Non-Intrusion is the concept of preventing unauthorized user who attempts to break into the SS by identifying possible security breaches [13]. For example, a user X intends to compromise the software system Y using common known threats and attacks techniques that break Y. Y must assure that all the common fragility attacks techniques are perfectly secured and fixed toward preventing X's attempt, such as Buffer Overflows, SQL Injections, Design Flaws.
- Assurance is the concept of covering all security objectives by ensuring their workability and correctness. The SS is not going to have an error, failure or shutdown during an unauthorized user who attempts on attacking the SS by using common known threats and attacks. Assurance might raise a flag and alert the SS when the attack is detected [7]. For example, when the user X attempts to access the software system Y, Y must check its security using assurance A. A provides a checkpoint to check the correctness, stability, and workability of each security objective in Y. Then, A provides Y with information about the status of Y's security. Based on the provided information, Y decides whether to enable or disable X.

There are many relationships that can be defined between users, the SS, and security objectives in our KBSS. To apply security objectives correctly and produce secured SS, we must first have our SS to check its security objectives' correctness via assurance while the user is requesting to

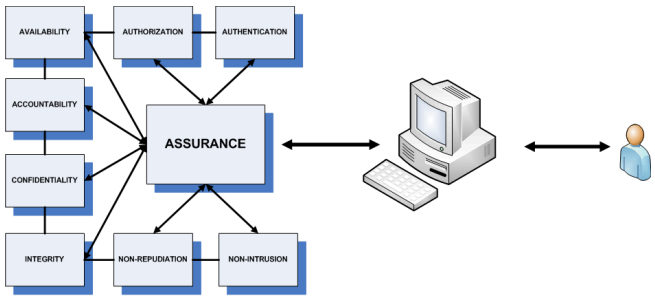

Fig. 1: Diagram illustrates the relationship between the user, the software system, and the security objectives, when the user intends to access a secured software system

access the SS. This methodology is preventing the user to access the SS until the SS approves its security consistency and correctness, ensuring loyalty of the user and preventing unauthorized user from accessing the SS. To simplify the scenario, when the user X requests to access the software system Y. Then, Y must check its security using Assurance A, which is the most significant security objective, that is responsible for checking Y's security correctness, stability and workability. Then, A checks X's identity using authentication, grants authorization to X to access certain information and makes data confidential and integral between X and Y. Otherwise, A rejects X. In the meantime, A ensures that the communications between X and Y is accountable, non-repudiative, and non-intrusive. Then, A provides Y with information about the status of Y's security. Finally, Y processes the provided information from A and decides whether to enable/disable X. When Y enables X to access the SS, Y keeps monitoring A iteratively while X is accessing Y. If A finds that one of the security objectives is compromised, A raises a flag and alerts Y. Figure 1 shows the relation between the user, the SS, and the security objectives.

We have introduced the most common objectives to ensure secured SS on one hand. On the other hand, we seek to clarify the possible attacks that compromise the SS security by taking advantage of the vulnerabilities of the SS. Technically, the attackers can target one or more security objective to compromise the stability of the SS. Knowing the common attacks and applying the appropriate security prevents the SS from being attacked. The assurance objective is responsible for assuring all security objectives of the SS where attacking one of them is correspond to the assurance. There are many examples of attacks against the weakness of the SS. Among them are: fraud attacks, phishing attacks, man in the middle, denial of service, worms, virus, spy-ware attacks, spam-ware attacks, cookies attacks, replay attacks, session attacks, hijack attacks, eavesdrop attacks, identity attacks and malicious attacks. The common possible and shared attacks are:

- Attack on authentication and authorization is the concept of attempting to break into the SS using the weakness of the authentication and authorization functions that mislead the identity and become an unauthorized user [3], [7], [10].
- Attack on confidentiality and integrity is the concept of attempting to access, read, modify and delete certain data [7], [11], [12].
- Attack on availability and accountability is the concept of attempting to make the SS or particular part of it unavailable to its users that leads the SS to be unaccountable. The accountability might result from every attack to the SS security objectives [7], [8].
- Attack on non-Repudiation and non-Intrusion is the concept of attempting to access the users logs by modifying and deleting the tracking data that misleads the trust between X and Y [7], [9], [13]. Also, we might consider the attack that takes advantage of newly discovered attacks on other organizations and uses it against other SS.
- Attack on assurance is the concept of attempting to compromise different objectives in one attack [7].

4. Knowledge Based Security System

The Knowledge Based (KB) of security is a special kind of database for Security Knowledge Management (SKM). SKM provides the most of our knowledge and up-to-date of having secured SS. SKM is the concept of systemically collecting, organizing, retrieving, enabling, storing and identifying the repository of security knowledge in an organization that can be available to other organizations.

Our proposed KBSS is a system that categorizes, clusters, monitors, alerts and controls the SKM by Security Expert Team (SET), who are able to identify, collect, organize, manage, retrieve, provide and store all aspects of security functions and issues. KBSS uses heuristic and deterministic based approach that helps in providing the SS with numerous security functions and issues that can be beneficial to any organization. This system collects its data by SET using various resources such as security organizations' knowledge, security expert's knowledge and heuristic and deterministic security knowledge within the organization. KBSS keeps monitoring the security updates that are released by SET. Once there is a new released security, the KBSS starts releasing its alert. Precisely, KBSS uses different inputs and/or outputs behavior description. For example, in the requirements engineering stage, KBSS uses various forms for representing security behavior such as natural language, user stories, or requirements specifications. In the analysis and design stage, KBSS uses Unified Modeling Language (UML) for representing the behavior description of security and the same is applied to other stages. These inputs and/or outputs can be inserted into the KBSS by SET, who are able to find the security and provide the appropriate solution in different stages of KBSS. Also, KBSS is the bridge between

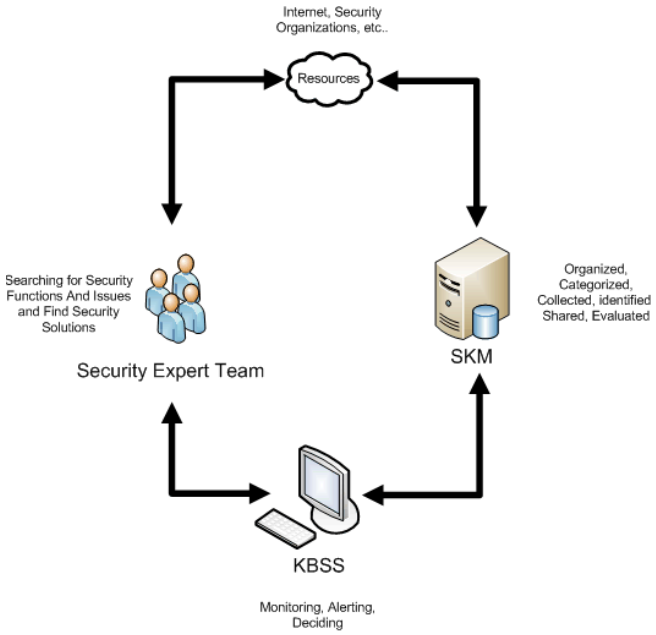

Fig. 2: Diagram illustrates the interactions between the SET and the SKM that controls the KBSS

the SET and the SKB, which holds all security knowledge to various domain. Figure 2 depicts the KBSS interactions.

Technically, there are four main actors. Each one of them depends heavily on his neighbor. The SET is an assigned team in the organization. Its responsibility is to collect, define, monitor, decide, and provide up-to-date security functions and issues with their appropriate solutions. The SET uses their knowledge of expertise and accesses the resources to collect security functions and issues, and insert them into the SKB via KBSS. Also, it is responsible for finding the appropriate solutions to security functions and issues that are not being recognized by the security organizations. Additionally, the resources can be classified as Internet's resources, security organizations' resources, Books' and Publications' resources, heuristic security within the organizations' resources and any valid security resources. SKM is special kind of database that systemically collects, organizes, retrieves, enables, stores and identifies the repository of security knowledge in an organization to be available to other organizations. SKM can be automatically accessing the resources and do the same job as the SET but with limited solutions. For example, the SET has the ability to find an appropriate solution that has not been recognized before where the SKM has not. We have to program the SKM to be an expert and intelligent to do decision as the SET. Keep in mind that the SET cannot access the SKM directly, which needs KBSS. KBSS is the main actor whose responsibility is to monitor, decide, and alert the development organizations that adopting it in the SDLC. Also, it is responsible for categorizing and clustering the SKM and provides recommendations to the organization with specific domain.

4.1 KBSS with SDLC

KBSS is trying to secure all well-known and expected security functions and issues within SDLC. Those functions and issues might be used as iterative and incremental approach toward SDLC. They can be simplified once they get discovered. Also, they must be alerted and adopted to the stage that the developer team has reached or just alerted and postponed to the next iteration. We adopt the KBSS in SDLC ideally with an Iterative and Incremental Development Process (IIDP). We are taking advantage of IIDP for adding security functions and issues with their appropriate solutions.

Simply, you can consider our KBSS methodology as a regular SS that keeps monitoring security functions and issues with appropriate solution. Once we start developing a SS, we must first add the domain of the SS to the KBSS. The KBSS is going to keep tracking our development processes with target concentration to the security perspective. When the KBSS is initialized with a specific domain, the KBSS is going to provide the developer with the most current initial security functions and issues. They might be represented as a natural language and provided at the beginning of the development processes. The KBSS will give appropriate security feedback highlighting issues and required functionality incrementally throughout the life cycle. As they are started at the requirements stage and end up with the evaluation stage, the KBSS moves forward iteratively and incrementally using IIDP compared to the waterfall model that stops at the last stage. The methodology adopts a checkpoint at every stage of the SDLC. Figure 3 shows the adaptation of the KBSS in SDLC.

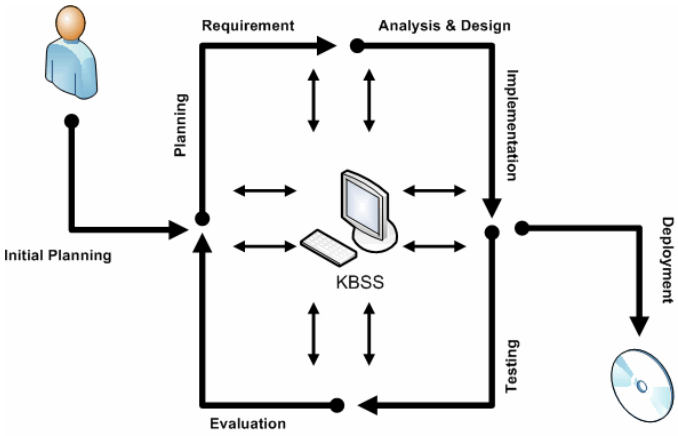

Fig. 3: Diagram illustrates the application of the KBSS to the Iterative and Incremental Development Process

Security updates are issued almost daily in response to new threats, which exploit previously-unknown vulnerabilities to attack SS. The newly-discovered vulnerabilities must be corrected in order to ensure that SS are completely secured. Many such updates could be issued while developing a given SS, requiring that the SS be modified appropriately in order for the SS to be secure. If a SS is developed with KBSS, a so-called security guard (checkpoint) will be evaluated at the end of each iteration of the SDLC. KBSS is a partially automated toolset that continuously monitors the security level of the SS under development after the development team specifies the domain of the SS. Simply, once we plan to develop a SS, the KBSS works as follows:

1) In the initial planning, we must add the domain of the SS to be developed to the KBSS.
2) The KBSS evaluates the domain, puts its first security checkpoint after the initial planning and extracts all the prospective security functions and issues with their appropriate solutions that might relate to the domain of SS. Also, it defines all the possible well-known and expected attacks and presents them to the developer in natural language for consideration. Passively, the KBSS automatically updates at discrete time intervals. whenever it gets updated with new security functions and issues, it starts alerting the developer of those functions and issues. The developer will consider them as an incremental and iteratively extension to the SS. Also, this checkpoint contains some of the issues that will be addressed in future stages of development. In addition, we could use one of the available tools, if there is any tool available, for converting the planning story to requirement specifications such as natural language to be used as requirements for this stage in our KBSS.
3) At time of processing the analysis and design stage, there is another security checkpoint for KBSS to evaluate the requirements security and provide the most recent security functions and issues with their respective resolutions. Also, it defines all the possible well-known and expected attacks that can be adopted to this stage such as abuses cases. For example, for accessing online banking system, the user must enter a valid user name and password. Once we have another user using the system from different machine, we must detect that and start using another type of verification such as security questions that previously set up by the user. This scenario is an example of security function and issue with its respective appropriate solution. Mainly, KBSS is using UML to represent security functions and issues at this stage. KBSS is an automated system that keeps updating him continuously. When KBSS detects an update, it directly raises a flag, alerts and contacts the developer team with that respective update. The developer will consider it as incremental and iterative feature to add to the SS. We might adopt one of the available tools [14], [15] for modeling the requirements in our KBSS. As a result, when there are new security functions and issues arise, KBSS will provide them to the developer team as UML to be added directly as incremental feature to the analysis and design stage.
4) At time of processing the implementation stage, there is another security checkpoint for KBSS to evaluate the analysis and design security and provide the most current security functions and issues with their respective appropriate solutions. Also, It defines all the possible well-known and expected attacks that can be adopted at this stage such as securing all objects and classes of the codes we implemented. KBSS is using algorithms or pseudo code to represent security functions and issues at this stage. Also, we could use one of the available tools [16], [17] that converting UML to code in our KBSS to validate the security releases issued at this stage.
5) At time of processing the testing stage, there is another security checkpoint for the KBSS to evaluate the implementation security and provide the most current security functions and issues with respected resolutions. It defines all the possible well-known and expected attacked that can be adopted such as buffer overflows, SQL injections, and design flaws. The KBSS will consider the black box testing, white box testing and gray box testing as testing security solutions to the developer team. The black box approach occurs when we have no advanced knowledge of the infrastructure to be tested. However, the white box approach happens when we have advanced knowledge of the infrastructure to be tested. The gray box approach is a hybrid between the two. Also, we could have the KBSS to provide the developer team with some tips, advises and techniques on testing and evaluating security at the testing stage.
6) Then, at time of processing the deployment stage, there is another security checkpoint for the KBSS to evaluate the testing security and provide the most current security functions and issues with their respective appropriate solutions. It defines all the possible well-known and expected attacks to the SS functionality that can be adopted at this stage. Also, KBSS is going to check if all the previous security functions and issues with their respective resolutions have been correctly applied. Otherwise, KBSS is going to flag, alert, and contact the developer team with the missing security functions and issues with their respective resolutions in order to be applied before deploying the SS.
7) Then, the first incremental released of the SS is deploying with respect to all aspects of security con-

cerns. Also, the KBSS automatically keeps tracking the updates with discrete time interval. When there are updates, it starts alerting the developer with those updates to be considered as incremental development processes.

8) Finally, for any release for security functions and issues to respective domain, we need to do step 2 to 6 repeatedly on one hand. On the other hand, we are going to adopt some of the available tools that are going to convert the planning stories security to requirements specification such as a natural language. Then, they are going to be modeled to UML and then to implementation codes. Also, they are going to provide the developers with tips, advises, and techniques on testing based on the provided UML. Finally, they are going to be provided to the developer team as incremental releases that are ready to use.

4.2 KBSS Objectives

The KBSS is successfully achieved to produce a secured SS when adopts its methodology at all stages of the SDLC ideally, as IIDP. Also, the KBSS is categorizes, clusters, monitors, alerts and controls the SKM, that contains many security functions and issues with their appropriate solutions to different domains, using the knowledge of the SET. In addition, the KBSS keeps tracking the security concerns within the SDLC using IIDP model. The KBSS can be incrementally added the security functions and issues with their appropriate solutions to the SS development as being developed and iteratively rework on the security concerns. KBSS is automatically monitoring and alerting the developers with security functions and issues as soon as they are released and discovered. Assuming the developers of the SS have valid subscription to our KBSS and their domains are registered in KBSS, KBSS is security solutions that can be beneficial to organization and any other organizations, who are planning to provide secured SS.

5. Conclusion

We have identified many major security objectives. To have secured SS, the SS must adopt security objectives correctly. We have realized that the attackers mostly target the weakness of security objectives. KBSS can discover and resolve the weakness of the security objectives. Moreover, we have recommended KBSS that categorizes, clusters, monitors, alerts and controls the SKM by the SET, who is able to identify, collect, organize, manage, retrieve, provide and store all aspects of security functions and issues. Additionally, we have offered evidential model of adopting the KBSS with SDLC ideally with IIDP. The basic idea of the adaptation is to have security checkpoints. Those security checkpoints help feeding SS with the most current security functions and issues with appropriate solutions.

In future work, we plan to use Bayesian Probabilistic System (BPS) for organizing KBSS. BPS helps ensuring the possibilities of what part of the system will be most likely to be attacked. Ideally, adding most of the knowledge of our common cyber-attacks against cyber-protects in BPS will help to provide connections between domains, the common cyber-attacks, and cyber-protects. The BPS will find the risk of possible attacks to specific domain.

References

[1] J. Wang, H. Wang, M. Gup, and M. Xia, "Security Metrics for Software Systems," in *47th Annual Southeast Regional Conference (ACM-SE 47)*, Article 47, ACM 2009.

[2] G. McGraw, "Building Security in Software Security," in *IEEE Security and Privacy Journal*, vol. 2, no. 2, pp. 80–83, IEEE 2004.

[3] J. Viega, "Building Security Requirements with CLASP," in *Workshop on Software Engineering for Secure Systems-Building Trustworthy Application*, ACM 2005.

[4] N. Mead and T. Stehney, "Security Quality Requirements Engineering (SQUARE) Methodology," in *Workshop on Software Engineering for Secure Systems-Building Trustworthy Application*, ACM 2005.

[5] J. Romero-Mariona, H. Ziv, and D. Richardson, "Later Stages support for Security Requirements," in *Richard Tapia Celebration of Diversity in Computing Conference: Intellect, Initiative, Insight, and Innovations*, pp. 103–107, ACM 2009.

[6] M. Vetterling and G. Wimmel, "Secure Systems Development Based on the Common Criteria: The PalME Project," in *Special Interest Group on Software Engineering*, pp. 129–138, ACM 2002.

[7] L. Ma and J. Tsai, "Attacks and Countermeasures in Software System Security," in *Handbook of Software Engineering and Knowledge Engineering*, Volume III, 2005.

[8] A. Yumerefendi and J. Chase, "Trust but Verify: Accountability for Network Services," in *11th Workshop on ACM SIGOPS European Workshop*, Article 37, ACM, 2004.

[9] M. Hwang, L. Li, and C. Lee, "A Key Authentication Scheme with Non-Repudiation," in *SIGOPS Operating Systems Review*, vol. 38, no. 3, pp. 75–78, ACM 2004.

[10] D. Mellado, J. Rodriguez, E. Fernandez-Mendine, and M. Piattini, "Automated Support for Security Requirements Engineering in Software Product Line Domain Engineering," in *International Conference on Availability, Reliability and Security*, pp. 16–19, IEEE 2009.

[11] L. Corman, "Data Integrity and Security of the Corporate Data Base: The Dilemma of End User Computing," in *SIGMIS Database*, vol. 19, no. 3–4, pp. 1-5, ACM 1988.

[12] P. Bennison and P. Lasher, "Data Security Issues Relating to End of Life Equipment," in *International Symposium on Electronics and the Environment Conference Record*, pp. 317–320, IEEE 2004.

[13] Z. Li, A. Das, J. Zhou, "Theoretical Basis for Intrusion Detection," in *Sixth Annual IEEE SMC on Information Assurance Workshop*, pp. 184–192, IEEE 2005.

[14] K. Subramanian, D. Liu, B. Far, A. Eberlein, "UCDA: Use Case Driven Development Assistant Tool for Class Model Generation," in *16th International Conference on Software Engineering and Knowledge Engineering*, pp. 324–329, Banff, Alberta, Canada 2004.

[15] R. Gaizauskas, H.M. Harmain, "CM-Builder: An Automated NL-Based CASE Tool," in *15th International Conference on Automated Software Engineering*, pp. 45–53, IEEE 2000.

[16] F. Do Nascimento, M. Oliveira, M. Wehrmeister, C. Pereira, F. Wagner, "MDA-based Approach for Embedded Software Generation from a UML/MOF Repository," in *19th Annual Symposium on Integrated Circuits and Systems Design*, pp.143–148, ACM 2006.

[17] W. Harrison, C. Barton, M. Raghavachari, "Mapping UML Designs to Java," in *15th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, pp. 178–187, ACM 2000.