Analysis of Security Failure-Tolerant Requirements

Michael Shin
Texas Tech University
Department of Computer Science
Lubbock, Texas, USA
michael.shin@ttu.edu

Don Pathirage
Texas Tech University
Department of Computer Science
Lubbock, Texas, USA
don.pathirage@ttu.edu

Dongsoo Jang
Texas Tech University
Department of Computer Science
Lubbock, Texas, USA
dongsoo.jang@ttu.edu

Abstract - This paper describes an approach to analyzing security failure-tolerant (SFT) requirements that are specified by means of SFT use cases, along with security use cases and application use cases for application systems. The SFT requirements are analyzed with the analysis model that consists of the static model and dynamic model. A meta-modeling approach is taken to specify the static and dynamic models for analysis of SFT requirements. Threats are identified in the analysis of SFT requirements, and SFT countermeasures against the threats are specified in the analysis model. An online shopping system is used for illustrating our approach.

Keywords – Security Failure-Tolerant Requirements; Analysis of SFT Requirements; Static Model; Dynamic Model; Threat; Meta-Model

I. INTRODUCTION

Security services seem to be unbreakable and they can protect security assets in applications from attacks. However, this appearance is not a reality. Security services, such as authentication, encryption or non-repudiation, are incorporated into applications in order to achieve security goals of the applications. Although the applications are designed by means of security services, they are still vulnerable because the security services can always be broken down as attack skills are getting crafty [1, 2].

Several approaches [3, 4, 5, 6, 7, 8, 9, 10] have been developed to make applications secure in software development. Most of the approaches have focused on specifying and designing applications using security services in order to build secure applications. Security requirements are specified with Unified Modeling Language (UML) [11, 12] and its extended notation [3, 4], separately from application requirements [7]. Secure software architecture is designed by means of secure connectors [8, 9, 10, 13] that encapsulate security services. However, less attention has been paid to the tolerance of broken security services.

Security failure-tolerant (SFT) requirements in our previous research [14] are specified to make applications tolerable when security services are breached. SFT requirements are modeled as SFT use cases, along with application use cases and security use cases, against the threats identified in application use cases. The SFT approach aims at reducing the possibility of security damage to security assets in the applications from the breaches of security services.

This paper is an extension of our research [14] by analyzing SFT requirements specification using the analysis model that represents the static and dynamic views of applications. The research in [14] specifies SFT requirements using SFT, security and application use cases against threats. This paper describes the analysis of the use cases and finds new threats, which are not identified in SFT requirements specification, and models the threats in the analysis model. In addition, this paper attempts to develop security and SFT countermeasures against the threats.

This paper is organized as follows: Section 2 describes related work for our research. Section 3 describes SFT requirements specification. Section 4 describes the meta-model for the analysis model of SFT requirements. Section 5 describes the analysis of SFT requirements in terms of the static and dynamic models. Section 6 describes the validation of our approach. Section 7 describes conclusions and future work.

II. RELATED WORK

Related work focuses on threat modeling, secure software development, and efforts to mitigate security failures so that applications become more secure. There is some research on security, but the research does not provide an adequate solution for developing secure software systems that tolerate the failures of security services.

Threat Modeling. Threats in a system have been modeled by several approaches, which include attack trees [1], data flow diagrams [15, 16], and UML-based modeling [17, 18, 19, 20, 21]. Attack trees in [1] provide an approach to modeling and analyzing the threats of systems, and the threats are analyzed in terms of attackers' capabilities. The design models in the research [15, 16] are specified with data flow diagrams, and the threats to the models are identified and analyzed using scenarios of each function in a system. Several threat-modeling approaches, such as misuse cases [17, 18, 19], abuse cases [20] and HAZOP (Hazard and Operability Analysis) [21], have been developed for object-oriented software systems. The approaches model threats using the use case model in UML and specifies security requirements against them.

The misuse case model [17, 18, 19] extends the use case model to misuse cases, along with actions that systems should take to support security requirements. An inverse of a use case is a misuse case, which is a negative requirement of a system that should not occur. The scenario of each possible attack is modeled using a misuse case. A use case description is analyzed to identify misuse cases and their actors.

DOI reference number: 10.18293/SEKE2018-0122

Secure Software Development. Some research for developing secure software has been done in terms of secure requirements and design. The studies in [3, 4] proposed a UML-based modeling language for the model-driven development of secure, distributed systems. The research in [22] illustrates an ontology-based approach that uses predefined pattern-based templates to aid requirements engineers in the formulation of security requirements. Security patterns in [5, 6] address a broad range of security issues that should be taken into account in the stages of the software development lifecycle.

In earlier work by a coauthor in [7], an approach is described to model complex applications by modeling application requirements and design separately from security requirements and design using the UML notation. In later work by a coauthor in [8], an approach is described for modeling the evolution of a non-secure application to a secure application in terms of the requirements and software architecture. The recent work by coauthors in [9, 10] proposes the design of reusable secure connectors using a component-based approach in which reusable secure connectors are structured into reusable security components and communication components.

Mitigation of Security Failures. Security failures can be mitigated by several approaches, such as layered security (defense in depth) [23], intrusion tolerance [24], and self-protection [25]. Layered security [23] addresses multiple facets of a security on a network. It is made up of multiple layers of complementary security technologies, so that all the technologies work together to provide the required level of protection.

The study in [24] presents the systematical notion of intrusion tolerance by rearranging the concepts and design of intrusion tolerance. Also the work in [26] presents a way to combine preventive maintenance with an existing intrusion tolerance system to improve the system security.

Self-protection [25] is a part of autonomic computing in which a self-protection component controls the security of a system without human interaction. To defend a system against malicious attacks or cascading failures, a self-protection system automatically prevents the attacks or failures.

III. SPECIFICATION OF SFT REQUIREMENTS

SFT requirements [14] are specified against threats to application systems. The threats can be identified by considering security assets described in the use case description. The threats are represented using the use case notation in the use case model. A threat use case may not have a specific actor because an attacker can be any malicious persons or parties. A threat point [14] is defined in the use case description for an application use case where a security asset is contaminated if a security service is broken and there is not any SFT service to protect the asset. The threat to *make order request* use case in the online shopping system [27] is modeled in Fig. 1a in which the *release ID and password* threat threatens the *make order request* use case at the *ID and password* threat point.

The requirements of security services for an application system are specified with security use cases [7, 14] separately from application use cases. When an application system requires security services, the security use cases are extended from the application use cases at extension points. An extension point is a location in an application use case where a security use case extends an application use case if the system requires the security service. An application use case designates an extension point in the use case description where a security use case extends the application use case. The security use case for the *make order request* application use case is depicted in Fig. 1b in which the *check keystroke logging* security use case is extended from the *make order request* application use case if the system requires the *check keystroke logging* security service.

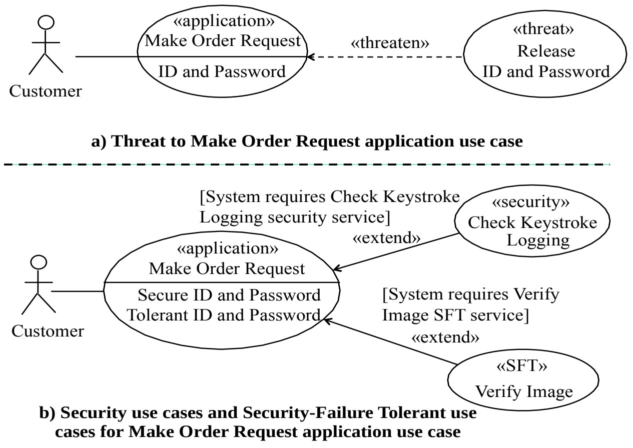

Fig. 1 Threat and Security use case and SFT use case for Make Order Request application use case

SFT requirements for security services are modeled using SFT use cases [14], which tolerate breaches of security services. By careful separation of concerns, SFT requirements are captured in SFT use cases separately from security use cases and application use cases. A SFT use case extends an application use case at an extension point if the system requires the SFT service. The *verify image* SFT use case (Fig. 1b) extends the *make order request* application use case at the *tolerant ID and password* extension point if the system requires the SFT service. The *verify image* SFT use case verifies that an image selected by a customer is matched with the image that the customer registered in the system. Even though the customer ID and password are released to an attacker due to failure of *check keystroke logging* security use case, the attacker would have to know of the customer's image registered in the system in order to make a malicious purchase order.

IV. META-MODEL FOR ANALYSIS OF SFT REQUIREMENTS

SFT requirements are analyzed from the static model for defining structural relationships between classes and dynamic model for defining how objects participate in use cases. SFT requirements are specified using the use cases that describe the requirements of SFT applications. The static model is developed using the class diagram in UML that determines the classes supporting each use case of SFT requirements and relationships between classes. The dynamic model is developed using the communication diagram in UML that describes the sequences of message communication between objects for each use case.

The static and dynamic models for SFT requirements analysis are depicted in Fig. 2 using a meta-model, which is extended from the meta-model for the class diagram and communication diagram in UML. A meta-model describes the meta-classes and relationships between the meta-classes. Any class diagrams and communication diagrams instantiated from the meta-model for SFT requirements analysis should follow the meta-classes and relationships between the meta-classes defined in the meta-model. The meta-model (Fig. 2) is simplified by representing the underlying meta-classes and their relationships associated with SFT requirements analysis.

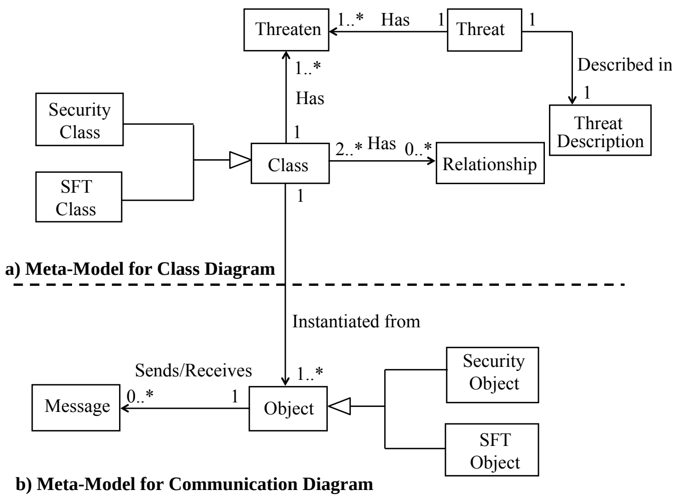

Fig. 2 Meta-model for SFT Requirements Analysis

The meta-model for the class diagram (Fig. 2a) of SFT requirements consists of class meta-class and relationship meta-class. There is a relationship meta-class between class meta-classes. A class meta-class is specialized to security class meta-class or SFT class meta-class. A threat meta-class threatens a class meta-class through a threaten meta-class. A threat meta-class is described in a threat description meta-class. Similarly, the meta-model for the communication diagram (Fig. 2b) of SFT requirements is represented by means of object meta-class and message meta-class in which an object meta-class may send a message meta-class to or receive a message meta-class from other object meta-classes. An object meta-class can be specialized to a security object meta-class or SFT object meta-class. An object meta-class in the meta-model for the communication diagram is instantiated from a class meta-class in the meta-model for the class diagram.

V. Analysis of SFT Requirements

A. Static Modeling of SFT Requirements

The static model for SFT requirements defines the structural relationships between application classes, security classes and SFT classes. Application classes support application use cases, whereas security classes are involved in realizing security use cases. SFT classes are needed to implement SFT use cases. The structural relationships between application, security, and SFT classes depict the static view between the classes. The static model for *make order request* application use case (Fig. 1b) is depicted in Fig. 3 using the class diagram, which includes *Customer Interface*, *Customer Account* and *Delivery Order* classes (Fig. 3) for *make order request* application use case (Fig. 1b), *Keystroke Logging Checker* security class (Fig. 3) for *check keystroke logging* security use case (Fig. 1b), and *Image Verifier* SFT class (Fig. 3) for *verify image* SFT use case (Fig. 1b). The *Customer Interface* class checks keystroke logging attack using *Keystroke Logging Checker* security class, and it verifies the image selected by a customer using *Image Verifier* SFT class.

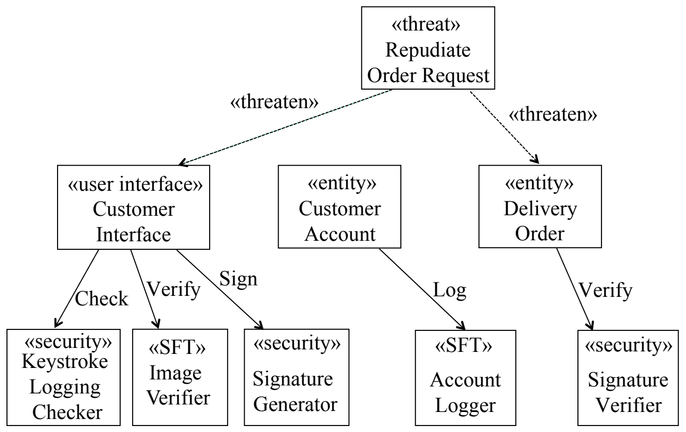
«threat» Repudiate Order Request
«threaten»
«threaten»
«user interface» Customer Interface
«entity» Customer Account
«entity» Delivery Order
Check
Verify
Sign
Log
Verify
«security» Keystroke Logging Checker
«SFT» Image Verifier
«security» Signature Generator
«SFT» Account Logger
«security» Signature Verifier

Fig. 3 Static model for Make Order Request application use case

B. Dynamic Modeling of SFT Requirements

The dynamic model for SFT requirements determines how security objects and SFT objects participate in the sequence of message communication between application objects. Security objects are invoked by application objects if the application objects require security services. Also, application objects invoke SFT objects when they need SFT services in order to tolerate the breaches of security objects. Security objects and SFT objects are represented on the communication diagram along with application objects.

The dynamic model for *make order request* application use case is depicted in Fig. 4, which describes how security and SFT objects are integrated into the sequence of message communication between application objects. The *Keystroke Logging Checker* security object checks malicious keystroke logging software (messages M1.1 and M1.2 in Fig. 4) when a customer initiates an order service (message M1 in Fig. 4). If there is no keystroke logging software installed, *Image Verifier* SFT object verifies whether the image selected by a customer is matched with the customer's image stored in the system (messages M1.3 through M1.8 in Fig. 4). A customer's order request is sent to *Purchase Order Manager* business logic object (messages M2 and M3 in Fig. 4), which requests a customer's account information from the Customer Account entity object (message M4 in Fig. 4). The *Purchase Order Manager* business logic object requests the authorization of a customer's credit card payment from a bank via *Bank Interface* object when it receives a customer's account information from the *Customer Account* entity object (messages M5 through M6 in Fig. 4). If the bank approves a customer's credit card payment (message M7 in Fig. 4), *Purchase Order Manager* business logic object stores an order in the *Delivery Order* entity object (messages M8 and M9 in Fig. 4) and sends a confirmation email to a customer via *Email* entity object (message M10 in Fig. 4).

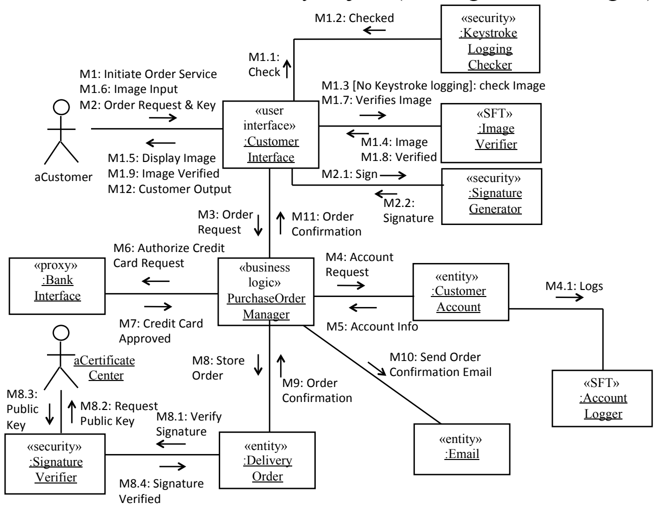

Fig. 4 Dynamic model for SFT Make Order Request application use case

C. *Threat Modeling of SFT Requirements Analysis*

Threats in SFT requirements specification focus on the security assets that should be protected from attacks. A security asset can be a security-relevant input to an application, secure data maintained in an application, and a system itself on which an application is running [17]. A security-relevant input to an application is a user's input to the application or an input from an external system or device to the application in which the inputs require security. An account identification (ID) or password entered by a user to an application can be an example of a security-relevant input to an application. Secure data stored in an application can be a target of an attack. An example of secure data can be the credit card information maintained by an online shopping system or a patient's medical records stored in a healthcare system. Also, a system on which an application is running should be a security asset when the system's availability affects the application's availability.

New threats are found in the analysis of SFT requirements in terms of confidentiality, integrity, or non-repudiation of messages, which can be sent from an object to another. While an object communicates a message with another object, the message might require confidentiality. Also, some messages should not be tampered with in an interaction between objects. In addition, when an object sends a message to another, it might need to prove who sends the message, so as to protect the non-repudiation security.

New threats can be identified by examining the message sequence between objects described in the communication diagrams for SFT requirements analysis. Each use case is refined by means of a communication diagram, which represents the business logic using objects and message sequence between objects. Some messages involved in the business logic might need to be secure from the application perspective. The *order request* (message M2 in Fig. 4) is created by a customer and is sent to the *Delivery Order* entity object through the *Purchase Order Manager* business logic object (messages M3 and M8 in Fig. 4) after the customer's credit card payment is approved by a bank. The customer might deny the *order request* later due to this or that reason after the customer made the *order request*. The *order request* message is under a repudiation threat from the application perspective.

The new threats identified in an analysis of SFT requirements are modeled in the static model, which describes application classes, security classes, and SFT classes. A threat is represented by means of the class notation with the stereotype «threat», and a threat class has a «threaten» dependency relationship with a class (Fig. 2). The *Repudiate Order Request* threat is modeled in the static model (Fig. 3) in which the *Repudiate Order Request* threat threatens the *Customer Interface* class and the *Delivery Order* entity class. This means that the *Customer Interface* object and the Delivery Order entity object in the dynamic model (Fig. 4) are under the *Repudiate Order Request* threat.

Each new threat is defined in a short threat description that describes threat name, description, security asset, security goal, security class, and SFT class. The security class and SFT class are the security countermeasure against the new threat. An alternative to a short threat description, a threat can be analyzed and specified in detail in terms of threat attributes, threat effect, and security concern [2]. The following is the short threat description for the *Repudiate Order Request* threat:

- Threat Name: Repudiate Order Request
- Description: Customer can repudiate the order request.
- Security Asset: Order Request
- Security Goal: Non-repudiation of order request
- Security class:
  - Signature Generator security class
  - Signature Verifier security class
- SFT class: Account Logger SFT class

As the analysis model of SFT requirements reveals a new threat, a security service and its SFT service against the threat are incorporated into the analysis model. The security and SFT services are modeled through classes in the static model. A digital signature security service is taken against the *Repudiate Order Request* threat, being implemented using the *Signature Generator* security class (Fig. 3), which signs a customer's order request using the customer private key, and the *Signature Verifier* security class (Fig. 3), which verifies the order request signed by the customer using the customer's public key. Also, an *Account Logging* SFT service is created against the *Repudiate Order Request* threat, which is mitigated by the *Account Logger* SFT class (Fig. 3) to record a customer's access to the customer account.

Security and SFT objects against a threat identified in the analysis of SFT requirements are incorporated into the dynamic model. The *Signature Generator* security object signs a customer's order request using the customer private key (messages M2.1 and M2.2 in Fig. 4) before the *order request* is sent by the *Customer Interface* object to the *Purchase Order Manager* business logic object. The signed order request is verified by the *Signature Verifier* security object (messages M8.1 through M8.4 in Fig. 4) using the customer public key obtained from a certificate authority before the *Delivery Order* entity object creates a new order request. Also, the *Account Logger* SFT object logs a customer's access to the *Customer Account* entity object (message M4.1 in Fig. 4) for proving the customer's transaction later.

VI. Validation

A. Implementation

The dynamic model for the *make order request* application use case (Fig. 4) was implemented along with the *keystroke logging checker*, *signature generator, signature verifier* security objects, as well as *image verifier* and *account logger* SFT objects. The *make order request* application use case was implemented on an online hosting sever to generate real world results. The *keystroke logging checker* security object (Fig. 4) was implemented as an antivirus agent, which was developed in C# for this paper and ran on a customer's local machine once every 24 hours. The antivirus agent updates the security status of the customer's local machine in an online database. The antivirus agent checks if the client's machine installed anti-keystroke logging software, such as Symantec Endpoint Protection. The online shopping system connects to the online database and checks the security status of the client's machine, which has been updated by the *antivirus agent*. If the database indicates that the client's machine is secure from keystroke logging attack, the system allows the customer to proceed with the order. If the system detects that the client's machine does not have an *antivirus agent* or does not have up to date anti-keystroke logging software, it displays an appropriate warning massage and stops the *make order request* use case. However, no SFT service was implemented to protect the database. We assumed that the database would remain secure for the simplicity of our approach.

The *Image Verifier* SFT object (Fig. 4) accesses an image repository in the online shopping system and displays a random set of four images, including a customer's personal image. The system verifies that the image selected by the customer is matched with the customer's personal image stored in the system. If the customer chooses a wrong image, the system prompts a different set of images. If the customer selects an incorrect image consecutively twice, the customer's account is locked.

The *signature generator* and *signature verifier* security objects were implemented for the non-repudiation security service. The *signature generator* security object computes a signature for an order request using a *secure hash algorithm 1* (SHA1) to generate a hash value, followed by an encryption of the hash value using the customer's private key. The *signature verifier* security object verifies that the signature is correct for the order request using the customer's public key.

The *account logger* SFT object (Fig. 4) was implemented to tolerate the breach of a digital signature. The *account logger* SFT object logs the customer's activities, including the transaction time, customer's information, and order details. The log data is used later to confirm the validity of the order request.

B. Performance Analysis of SFT Requirements

This section describes the performance analysis of our approach to see how much performance overhead occurs due to SFT use cases that are specified for application and security use cases. The computational performance of our approach was measured using three approaches: (1) with *standalone application object approach*, for running the application use case with no security; (2) with *security object approach*, for running the application use case with security objects; (3) with *security object and SFT object approach*, for running the application use case with security objects and SFT objects.

The performance was evaluated by measuring the average time taken to complete the execution of three approaches, which were run 20 times each to calculate the average execution time, so that the performance evaluation would not be dependent on a few exceptional running times. The average execution time was calculated by measuring the run time of the program per session, but it excluded the time that a customer interacted with the system. Also, we assumed that the *antivirus agent* was already installed on the client's machine. Table 1 shows the average execution time of the approaches and performance comparison.

The second column of Table 1 shows that the average execution time is 1.33 seconds for the *make order request* application use case (Fig. 4). The third column of Table 1 shows that the average execution time for the *make order request* application use case with *security objects* (Fig. 4) is 1.57 seconds. The fourth column shows that the corresponding average execution time for application use cases with both security and SFT objects takes 1.64 seconds.

Table 1. Average execution time of approaches and performance comparison

Application use case	With standalone application object approach	With security object approach	With security object and SFT object approach	Time difference between with standalone application object approach and with security object approach	Time difference between with standalone application object approach and with security object and SFT object approach	Time difference between with security object approach and with security object and SFT object approach
Make order request use case (Fig. 4)	1.33 s	1.57 s	1.64 s	0.24 s ≈ 18%	0.31 s ≈ 23%	0.07 s ≈ 3%

The fifth column of Table 1 indicates that there is the time difference between the *with standalone application object approach* and the *with security object approach*. Time difference for the *make order request* application use case is 0.24 seconds (17 %) because the *with security object approach* provides the application use case with security services. The security services in the *with security object approach* consume 17% more processing time for logging account, and generating and verifying a digital signature in the system; the *with standalone application object approach* is faster because it provides no security services.

Similarly, when an application use case is deployed with both security and SFT objects, the average execution time is increased further, as shown in the sixth column of Table 1. The *make order request* application use case (Fig. 4) with security and SFT objects takes 0.31 seconds, which is a 23% increase in run time.

The last column indicates that the time difference between *with security object approach* and *with security object and SFT object approach* is 0.07 seconds for the *make order request* application use case. This result indicates that *with security object and SFT object approach* takes more execution time (3%). However, the *with security object and SFT object approach* makes the system more secure compared to the *with security object approach* alone.

VII. Conclusions and Future Work

This paper has described an approach to analyzing SFT requirements. SFT requirements were analyzed by means of the analysis model, which was represented using the class diagram and communication diagram. The meta-model for the class diagram and communication diagram was developed to specify the static and dynamic models for an analysis of SFT requirements. New threats were identified in the analysis of SFT requirements, and security and SFT objects against the threats were specified in the analysis model. Our approach can be used by requirements engineers to specify security requirements for applications, as well as SFT requirements against the failures of security requirements.

This paper can be strengthened with further research. The SFT requirements specification and analysis can be extended to the SFT software architecture that describes the components and their interaction for SFT applications. New threats could be identified in the SFT software architecture and, if so, it is necessary for both security and SFT services to be incorporated into the software architecture. Also, we can investigate how both security services and SFT services are encapsulated in secure connectors [8, 9, 10, 13], along with communication patterns.

References

[1] B. Schneier, "Attack trees: Modeling security threats," Dr.Dobbs Journal, pages 21–29, December 1999.

[2] M. E. Shin, S. Dorbala, and D. Jang, "Threat Modeling for Security Failure-Tolerant Requirements", ASE/IEEE International Conference on Privacy, Security, Risk and Trust (PASSAT2013), Washington D.C., USA, 2013.

[3] T. Lodderstedt, D. Basin, J. Doser, "SecureUML: A UML-Based Modeling Language for Model-Driven Security", Fifth International Conference on the Unified Modeling Language, London, UK., 2002.

[4] J. Jürjens, "UMLsec: Extending UML for Secure Systems Development", Fifth International Conference on the Unified Modeling Language, London, UK, 2002.

[5] M. Schumacher, E. B. Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad, "Security Patterns", Wiley, 2006.

[6] E. B. Fernandez, "Security Patterns in Practice", Wiley, 2013.

[7] H. Gomaa and M. E. Shin, "Modeling Complex Systems by Separating Application and Security Concerns", 9th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2004), Italy, April, 2004.

[8] M. E. Shin and H. Gomaa, "Software Modeling of Evolution to a Secure Application: From Requirements Model to Software Architecture", Science of Computer Programming, Volume 66, Issue 1, pp. 60-70, 2007.

[9] M. E. Shin, B. Malhotra, H. Gomaa, and T. Kang, "Connectors for Secure Software Architectures", 24th International Conference on Software Engineering and Knowledge Engineering (SEKE'2012), San Francisco, July 1-3, 2012.

[10] M. E. Shin, H. Gomaa, D. Pathirage, C. Baker, and B. Malhotra, "Design of Secure Software Architectures with Secure Connectors", International Journal of Software Engineering and Knowledge Engineering, Vol. 26, No. 5, pp 769–805, 2016.

[11] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language User Guide", Second Edition, Addison Wesley, Reading MA, 2005.

[12] J. Rumbaugh, G. Booch, and I. Jacobson, "The Unified Modeling Language Reference Manual (2nd Edition)," Addison Wesley, Reading MA, 2004.

[13] M. E. Shin, H. Gomaa, and D. Pathirge, Model-based Design of Reusable Secure Connectors," 4th International Workshop on Interplay of Model-Driven and Component Based Software Engineering (ModComp2017), September 17, Austin, Texas, USA, 2017.

[14] M. Shin and D. Pathirage, "Security Requirements for Tolerating Security Failures," 29th International Conference on Software Engineering and Knowledge Engineering, Pittsburgh, USA, July 5-7, 2017.

[15] P. Torr, "Demystifying the Threat-Modeling Process," IEEE Security and Privacy, vol. 03, no. 5, pp. 66-70, Sept/Oct, 2005.

[16] M. Abi-Antoun, D. Wang and P. Torr, "Checking Threat Modeling Data Flow Diagrams for Implementation Conformance and Security", ASE 2007, 21 pages, 2006.

[17] G. Sindre and L. Opdahl, "Eliciting Security Requirements with Misuse Cases," Requirements Engineering, Volume 10 Issue 1, January 2005, pp. 34 - 44.

[18] P. Hope, G. McGraw, and A. I. Anton, "Misuse and Abuse Cases: Getting Past the Positive," IEEE Software, 2003.

[19] I. Alexander, "Misuse Cases: Use Cases with Hostile Intent," IEEE Software, vol.20, no. 1, pp. 58-66, 2003.

[20] J. McDermott and C. Fox, "Using Abuse Case Models for Security Requirements Analysis," In Proceedings of 15th Annual Computer Security Applications Conference (ACSAC`99), pp. 55-64, Phoenix, Arizona, December, 1999.

[21] T. Srivatanakul, "Security Analysis with Deviational Techniques," PhD thesis, Department of Computer Science, University of York, UK, 2005.

[22] D. Olawande, G. Sindre, and T. Stalhane, "Pattern-based security requirements specification using ontologies and boilerplates", *IEEE Second International Workshop on Requirements Patterns (RePa),* 2012.

[23] S. Gantz, "Layered Security Architecture: Establishing Authentication, Authorization, and Accountability", securityarchitecture.com/docs/, 2008.

[24] P. E. Veríssimo, N. F. Neves, and M. P. Correia, "Intrusion-Tolerant Architectures: Concepts and Design", Architecting Dependable Systems, Springer-Verlag, Berlin, Heidelberg, 2003.

[25] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology", http://people.scs.carleton.ca/~soma/biosec/readings/autonomic_computing.pdf, 2001.

[26] Iman El Mir, D. S. Kim, and A. Haqiq. "Security modeling and analysis of a self-cleansing intrusion tolerance technique." IEEE 11th International Conference on *Information Assurance and Security (IAS),* 2015.

[27] H. Gomaa, "Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures", Cambridge University Press, 2011.