Security Requirements for Tolerating Security Failures

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

Abstract— This paper describes security failure-tolerant requirements, which tolerate the failures of security services that protect applications from security attacks. A security service, such as authentication, confidentiality or integrity security service, can be always broken down as advanced attack skills are coined. There is no security service that is forever secure. This paper describes an approach to developing the security failure-tolerant use case that specifies the security requirements for tolerating the breaches of security services. A security failure-tolerant use case is modeled along with application use case and security use case, and specified with application use case description. Threats to applications are identified and modeled to develop security failure-tolerant requirements. Online shopping system is used for illustrating security failure-tolerant requirements.

Keywords - Security Requirements; Security Failure-Tolerant use case; Security use case; Application use case

I. Introduction

Secure applications are designed with the security services that are made to achieve security goals, such as authentication, authorization, confidentiality, integrity, availability and non-repudiation. The security services seem to be unbreakable enough to protect security assets in the applications from attacks. However, in reality, although applications are designed with unbreakable security services, the security services are always broken down as attack skills are getting crafty [2, 15]. To make applications more secure, it is necessary for security services to be tolerated when they are broken down.

Several approaches [7, 8, 9, 10, 11] were developed to make applications secure in software development. Most of the approaches have focused on specifying and designing applications with security services in order to make applications secure. Security requirements are specified with Unified Modeling Language (UML) [1] and its extended notation [7, 8], separately from application requirements [10]. Secure software architecture is designed using secure connectors [11] that encapsulate security services. However, less attention has been paid to the tolerance of broken security services in terms of security requirements for secure applications.

This paper describes an approach to developing security failure-tolerant requirements that tolerate broken security services. The tolerance of breached security services is specified on the assumption that any security services can be broken down. The security failure-tolerant approach aims at reducing the possibility of security damage to security assets in the applications from the breaches of security services. The security failure-tolerant approach is adopted from fault-tolerant approach to minimizing the system damage from a fault of systems. The proposed approach can delay attacks until the security failure-tolerant use cases are compromised. Although the security failure-tolerant approach might not be the ultimate solution to security, it can be a solution to make applications more secure by mitigating the breaches of security services.

II. Related Work

Threat Modeling. Threats in a system have been modeled by several approaches, which include attack trees [2], data flow diagrams [3], and UML-based modeling [4, 5, 6]. Attack trees in [2] provide an approach to modeling and analyzing the threats of systems, and the threats are analyzed in terms of attacker's capabilities. The design models in [3] are specified with data flow diagram, 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 [4], abuse cases [5], and HAZOP (Hazard and Operability Analysis) [6], have been developed for object-oriented software systems. The approaches model threats using the use case model in UML and capture security requirements for them.

Secure Software Development. Several researches for developing secure software have been done in terms of secure requirements and design. The studies in [7, 8] proposed a new modeling language based on UML for the model-driven development of secure, distributed systems. The research in [17] illustrates an ontology-based approach that uses predefined pattern-based templates to aid requirements engineers in the formulation of security requirements. Security patterns in [9] address the broad range of security issues that should be taken into account in the stages of software development lifecycle.

Mitigation of Security Failures. Security failures can be mitigated by several approaches, such as layered security (defense in depth) [12], intrusion tolerance [16], and self-protection [13]. The layered security [12] 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.

DOI reference number: 10.18293/SEKE2017-098

III. Security Requirements for Security Failure-Tolerance

A. Threat Modeling

Threats to a security failure-tolerant application focus on the security assets in the application that should be protected from attacks. A security asset can be a security relevant input to applications, secure data maintained in an application, and the system itself on which an application is running [4]. The security relevant input to applications is a user's input to applications or the input from an external system or external devices to applications in which the inputs require security. An account identification (ID) or password entered by a user to an application can be an example of the security relevant input to applications. A secure data stored in an application can be a target of an attack. The example of a secure data can be the credit card information maintained by an electronic commerce application or a patient's medical record 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 an application's availability.

The security assets in a security failure-tolerant application can be identified by analyzing the use case descriptions for each application use case. The use case description describes application business logic in terms of the actor's inputs to a system and the system's responses to the actor. Also, a use case description addresses the data stored in the application and the actions applied to the data so as to process the actor's input and generate a response to the actor. The actor's input or the data stored in an application is a security asset if it requires security.

The *make order request* use case in online shopping application [14] receives a customer order request, checking the sufficient credit to pay for the requested items and creates a delivery order for the customer if the credit is sufficient. The use case description for *make order request* use case is described as follows:

Use case name: Make Order Request
Summary: Customer enters an order request to purchase items. The customer's credit card is checked for validity and sufficient credit to pay for the requested items.
Actor: Customer
Precondition: Customer has selected one or more catalog items.
Main sequence:

1. Customer selects the order request service.
   <Secure ID and Password>
2. System prompts the input for order request to customer.
3. Customer provides a purchase order request and customer account ID and password to pay for the purchase **<Threat point: ID and Password>**.
   <Tolerant ID and Password>
   <Secure Credit Card>
   <Tolerant Credit Card>
4. System retrieves customer account information, including the customer's credit card details **<Threat point: Credit Card>**.
5. System checks the customer's credit card for the purchase amount and, if approved, creates a credit card purchase authorization number.
6. System creates a delivery order containing order details, customer ID, and credit card authorization number.
7. System confirms approval of purchase and displays order information to customer.
8. System sends email confirmation to customer.

Alternative sequences:
Step 4: If customer does not have an account, the system prompts the customer to provide information in order to create a new account.
Step 5: If authorization of the customer's credit card is denied, the system prompts the customer to enter a different credit card number.
Threat and Security:

- Threat at Step 3: Release ID and Password
    - Security Asset: ID and Password
    - Description: ID and Password can be released to attackers
    - Security goal: Confidentiality of ID and Password
    - Security use case: Check Keystroke Logging security use case
    - Security failure-tolerant use case: Verify Image security failure-tolerant use case
- Threat at Step 4: Release Credit Card
    - Security Asset: Credit Card
    - Description: Customer credit card information might be released
    - Security goal: Confidentiality of Credit Card
    - Security use case: Check Malicious Code security use case
    - Security failure-tolerant use case: Fraud Monitor security failure-tolerance use case

Post-condition: System has created a delivery order for the customer.

The customer account ID and password at step 3 in the *make order request* use case description can be a security asset in terms of a customer input that requires security. Also, the customer's credit card details at step 4 are another security asset stored in the application so that the system processes the customer's purchase request.

A threat identified is modeled with application use cases in the use case model. A threat threatens an application use case at a threat point. In this paper, a threat is represented using the use case notation in the use case model, but a threat use case does not have a specific actor because an attacker can be any malicious person. Also the threat use case does not have a common scenario as to how to realize the threat. This is because an attacker can realize a threat in an unpredictable way. A threat point is a point in the application use case where a threat can occur. Also, a threat point is a step in the use case description for an application use case where a security asset is jeopardized if a security service is broken and there is no any security failure-tolerant service to protect the asset.

The threats to *make order request* use case are modeled in Fig. 1 in which the *release ID and password* threat threatens the *make order request* use case at the *ID and password* threat point. Similarly, the *release credit card* threat threatens the *make order request* use case at the *credit card* threat point. A threat point is designated in the use case description by means of <threat point> with the threat point name. The *ID and Password* threat point is designated at step 3 as <threat point: ID and Password> in the *make order request* use case description. Also the *credit card* threat point is presented with <threat point: Credit Card> at step 4 in the same use case description.

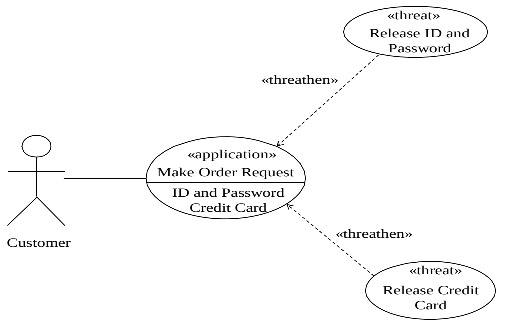

Fig. 1 Threats to Make Order Request application use case

Each threat is specified to analyze security concerns so that a security failure-tolerant service is developed along with a security service. A threat can be described in the use case description for an application use case. A threat is described shortly in the threat and security section of the use case description in terms of threat name, security asset, threat description, and security goal. In the *make order request* use case description above, the release ID and password threat is specified with security asset (ID and password), description (ID and Password can be released to attackers), and security goal (Confidentiality of ID and Password). Similarly, the release credit card threat is specified in the use case description. As 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 [15].

B. Security Requirements Modeling

Security requirements of security services for an application system are specified with security use cases [10] separately from non-secure application use cases. When the 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 application requires the security use case. An application use case provides an extension point where a security use case extends the application use case.

The security use cases for the *make order request* application use case are depicted in Fig. 2 in which the *check keystroke logging* security use case and *check malicious code* security use case are provided for the non-secure *make order request* application use case. A user's computer might be infected with malicious keystroke logging code that records user credentials and sends them to a third party location to do further harm. The *check keystroke logging* security use case mitigates the leak of user's ID and password using anti-malware software on the user's computer. The *make order request* application use case is extended to the *check keystroke logging* security use case at the *secure ID and password* extension point if the application use case requires the security use case. The *secure ID and password* extension point is described in the use case description for *make order request* application use case. The check keystroke logging security use case is specified as follows:

Security use case: Check Keystroke Logging
Summary: System checks a keystroke logging attack to protect customer input.
Actor:
Precondition: Anti-Keystroke Logging software is running.
Description:
1. System checks a keystroke logging attack.
2. If system detects a keystroke logging software, system displays a warning message "Keystroke Logging Attack" and removes the keystroke logging software.
3. System logs a keystroke logging attack.

Alternatives:
Post-condition: keystroke logging software has been checked.

The *check malicious code* security use case is another security measure that has been employed to protect the *make order request* application use case. Malicious code may get in the application system and it can release user's credit card information to an attacker. When the *make order request* use case requires the *check malicious code* use case, the *check malicious code* security use case is extended from the *make order request* application use case at the *secure credit card* extension point, which is designated in the *make order request* application use case.

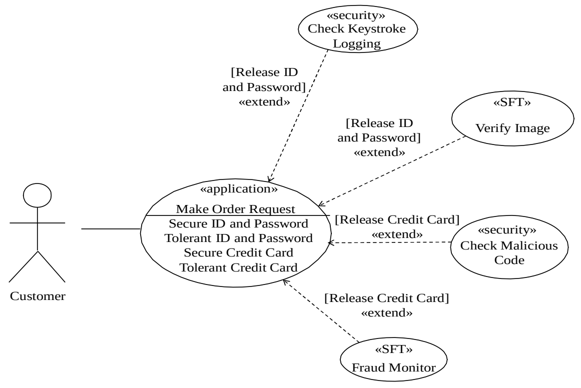

Fig. 2 Security use cases for make order request use case

C. Security Failure-Tolerant Requirements Modeling

Security failure-tolerant requirements are modeled with security failure-tolerant use cases, which tolerate the breaches of security services for applications. By careful separation of concerns, the security failure-tolerant requirements are captured in security failure-tolerant use cases separately from security use cases and application use cases. When an application use case requires a security failure-tolerant use case, the security failure-tolerant use case tolerates the breach of security use case.

Fig. 2 depicts *verify image* and *fraud monitor* security failure-tolerant use cases, which tolerate the breaches of *check keystroke logging* and *check malicious code* security use cases for *make order request* application use case, respectively. The *verify image* security failure-tolerant use case verifies that an image selected by the 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 should know of the customer's image registered in the system in order to make a malicious purchase order. The *fraud monitor* security failure-tolerant use case activates a service to monitor credit card fraud so that it prevents the damage caused by the release of credit card information. Malicious code hidden in the system might release the customer's credit card information to an attacker if the *check malicious code* security service fails to detect malicious code. However, the *fraud monitor* security failure-tolerant use case tolerates the attacker's fraud of released credit card. Providing a credit monitoring service to its customers can ensure that even if credit card information gets released, customer will be protected from further damage.

A security failure-tolerant use case is extended from an application use case at an extension point if the application requires tolerating the breaches of security service. An extension point for a security failure-tolerant use case is a location in an application use case where the security failure-tolerant use case extends the application use case. An extension point of a security failure-tolerant use case is distinguished from that of a security use case. For example, the *verify image* security failure-tolerant use case extends the *make order request* use case at the *tolerant ID and password* extension point (Fig 2), where the *check keystroke logging* security use case is extended from the *make order request* use case at the *secure ID and password* extension point (Fig. 2). The *tolerant ID and password* extension point is designated in the *make order request* use case description. The *verify image* security failure-tolerant use case is described as follows:

Tolerant use case: Verify Image
Summary: Customer clicks an image rather than keystroking his/her ID and password and system verifies the image.
Actor: Customer
Precondition: Customer's personal image is stored in the system.
Description:
1. System displays multiple images, which includes the image that customer has selected while registering for the system.
2. Customer selects an image that he/she has selected when registering for the system.
3. System verifies that the image selected by the customer is matched with the customer's image stored in the system.
4. If the images are the same, system approves that the customer makes an order.

Alternatives:
Step 4: If the customer selects the incorrect image consecutively for 2 times, the customer account is locked.
Post-condition: System has verified an image selected by a customer.

IV. Conclusions and Future Work

This paper presumes that security services are broken all the time in a real-world setting. On this assumption, first our approach has identified threats associated with security assets in terms of security relevant user's input, secure data stored in the application, and the system on which an application is running. Second we constructed security use cases against the threats so that the application would be protected from the threats identified. Finally, security failure-tolerant use cases have been specified to tolerate the breaches of security use cases.

The security failure-tolerance can be envisioned with further research. The security failure-tolerant requirements can be extended to security failure-tolerant analysis modeling that describes the static modeling and dynamic modeling. Also, this research can be extended to develop a framework for security failure-tolerant requirements in which security failure-tolerant use cases are categorized with security use cases in terms of security goals.

References

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

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

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

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

[5] 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.

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

[7] 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.

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

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

[10] 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.

[11] 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.

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

[13] 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.

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

[15] 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.

[16] I. E. 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.

[17] 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.