Analysis of Security Vulnerabilities Using Misuse Pattern Testing Approach

Yuan Yifan*, Somjai Boonsiri
Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University, Bangkok Thailand.

* Corresponding author. Tel.: +66 944501131; email: Yifan.Y@student.chula.ac.th.
Manuscript submitted December 15, 2014; accepted March 10, 2015.
doi: 10.17706/jsw.10.5.650-658

Abstract: Vulnerability detection is commonly been executed during the testing phase of software development. Current methods are not able to detect system or software security vulnerabilities of certain types of attacks during the early stages of software development. These attacks include both the ones we did not anticipate as well as the ones unknown during the design phase. In this paper we propose a method to detect the security vulnerabilities during the design phase of software development. This approach simulates attacks according to the misuse patterns using model testing method. With this approach, we are able to analyze system security vulnerabilities during the design stage of the system development. The practical example provides evidences the feasibility of our method.

Key words: Vulnerability, security patterns, misuse patterns, UML, USE, model testing.

1. Introduction

With the rapid growth of open networks and distributed systems, security has become one of the most vital issues while developing a system or a software. There are several approaches to build secure systems such as Microsoft's Security Development Lifecycle (SDL) [1], QWASP's CLASP [2] etc. However most of those approaches are based on secure coding. It must be admitted that code-based security is valuable. However, analyzing and patching security vulnerabilities in the code are usually high-cost tasks during the testing phase of software development. Using a better design according to the security requirements is able to reduce the chance of the occurrence on finding security vulnerabilities in the late stage of software development. The common security approaches mentioned above cannot be implemented during the design stage of the software and system development. They can be a good complement to model-based work.

One of the useful model-based method to make a secure design is using security patterns. Security patterns, a term initially been introduced in 1998 [3], are reusable packages with the knowledge of security experts. Security pattern has certainly proved its values in the industry. With these patterns, software engineers are able to build secure programs and systems without prior expert security experiences. Many patterns have been proposed. Some of the examples are shown in [4].

However, security patterns have some apparent limitations [5]. One of the limitations of security patterns is that they are focused on threats instead of vulnerabilities. One vulnerability could result in multiple attacks. The misuse pattern has been proposed in order to help identify which security patterns should be used to stop or mitigate a specific type of attacks and also to understand the underlying principle of attacks.

But merely using the misuse patterns still does not address the focus on vulnerabilities. Additionally, even the identification of vulnerabilities in the existing design model of our system can be difficult. The common vulnerability analysis methods are generally based on static analysis, such as the code analysis [6]. However these methods cannot be implemented in the design stage of system development.

Therefore, here we propose a systematic method to detect the system security vulnerabilities through the model testing of misuse patterns. This method helps to uncover existing vulnerabilities that expose our system to potential threat of a certain type of attacks. This method can also indicate what security patterns should be used to mitigate such risks. Additionally, drawing benefits from the contributions of model testing, we are able to implement the process of analyses in the design stage of system development. From a cost-benefit perspective, this is valuable because improving the design itself often has a lower cost than debugging and fixing a system during runtime.

Reference [7] proposed a systematic method of validating security patterns by model testing. This gives us a good guide of validating the misuse patterns in our new method. Our method consists of three main tasks:

- Validating current countermeasures: check whether the current countermeasures in the system model are working appropriately.
- Validating misuse pattern: validate whether this misuse can be successfully implemented under the protection of countermeasures on our system
- Vulnerabilities analysis: if the result of Step 2 is yes; analyze how and what components are exploited by the misuse to achieve its success and find out the corresponding vulnerabilities

In Section II we introduce some essential background. Section III describes the concrete process of our method. In Section IV, we apply our method in a practical case. We describe the future works in Section V and end with conclusions.

2. Background

2.1. Misuse Pattern

A misuse is an unauthenticated use of assets in a system or software. The structure of misuse patterns is similar to that of security patterns; however, the misuse pattern id described from the perspective of an attacker.

The purpose of inventing misuse patterns is not to make it easier to implement a misuse, but to understand a misuse. It is necessary to obtain an understanding of the possible threats in order to design a secure system. A systematic approach to identify the threats has been proposed in [8]. Many useful misuse patterns have been proposed such as the denial of service attack in VoIP [9] and worm [10]. Misuse patterns help us to learn how the components could be used by attackers to reach their misuse objectives and how an attack is performed. We subsequently analyze the security vulnerabilities and find means to counter the attack.

2.2. UML-Based Specification Environment

Unified modelling language (UML) [11] is now a standard for software development. UML-based models, with its sub-language object constraint language (OCL) [12], have been widely used in the system and software development.

In order to check the quality of UML design models, we need to define some method to validate the models. However, the UML tools do not offer much support on the methods of validating UML models and their OCL.

The UML-based specification environment (USE) tool [13] runs tests to validate models based on UML

and OCL. It supports analysts, designers and developers in executing UML models and checking OCL constraints, and thus, enables them to employ model-driven techniques for software production.

3. Process of Our Approach

Fig. 1 shows the process of our method, which is partitioned into six steps:

1) The purpose of our method is to test whether a simulated misuse can be implemented successfully on the target system. To accomplish this task, the first thing we need to do is list the misuse requirements so that we could verify the test results of the misuse model in the subsequent step. Additionally, the corresponding security requirements are also needed to verify the countermeasure modules in the victim system model.
2) It is necessary to know what security countermeasures are working on the target system before we validate the misuse pattern on it. Also, we need to ensure that the security design pattern model of these countermeasures are applied appropriately on the target system. The countermeasure models are tested in USE with their test cases. The process goes through only if all of the existing countermeasure models are operating adequately.
3) The accomplishment of Step 2 indicates that the preparations are completed. In this step, we apply the selected misuse pattern and test it in USE. This is the core step in validating misuse pattern. Be confirmed that the new model in this step should be a combination of the existing countermeasure models and the misuse model.
4) The system is considered vulnerable to the misuse if the result of testing misuse pattern in Step 3 satisfies the misuse requirements in Step 1. Therefore, in this step we then analyze the model created in Step 3 to understand how and what components are utilized by the attacker to achieve this misuse, and also determine the corresponding vulnerabilities. Otherwise, the process ends at Step 3 if the security requirements are still satisfied after we applied the misuse pattern. This indicates that the system is not vulnerable to the specific misuse.
5) According to the vulnerabilities found in Step 4, we offer some new countermeasures against the applied misuse.

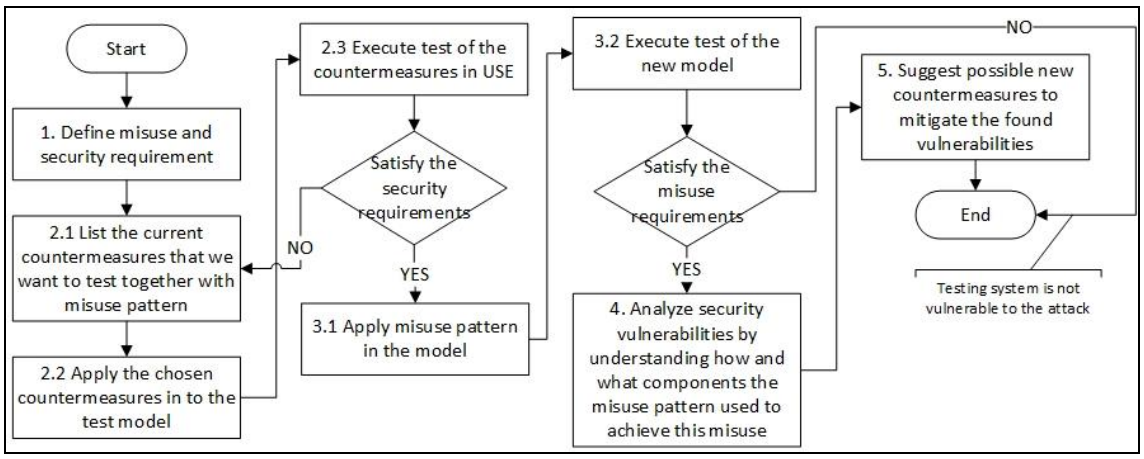

Fig. 1. The process of our method.

4. Practical Example

In order to demonstrate the integrated method of analyzing system vulnerabilities, here we use a Web server in the design stage as a victim system, which is employing credential based I&A as countermeasure. Typically, the combination of a username and a secret password is considered as an appropriate credential. Different credentials are usually chosen according to the different security situations. Herein, for the

explicit expression of our process in this example, we use the combination of username and password as credentials in our experiment.

For attacking model, we use the principal spoofing misuse pattern [14] to simulate the attack on the victim system which we have mentioned above.

4.1. Step1: Misuse Requirements and Security Requirements

Table 1 and Table 2 show a comparison of the security requirements and misuse requirements. The security requirements of I&A only allow regular users to access the protected assets. The situation is changed in the misuse requirements in Table 2 where the spoofing attack requires the possibility of accessing protected assets as an irregular user.

Table 1. Security requirements of I&A

Actions \ Condition	Regular user	Irregular user
Allow to access assets that need I&A	×	
Reject to access assets that need I&A		×

Table 2. Misuse requirements of principal spoofing in web services

Actions \ Condition	Regular user	Irregular user
Allow to access assets that need I&A	×	×
Reject to access assets that need I&A		

4.2. Step2: Validate System Current Countermeasures

Table 3. Security design requirements of credential pattern

Actions \ Condition	With a valid credential	Invalid/without credential
Consider a regular user	×	
Consider an irregular user		×

Table 3 shows the security design requirements of the credential pattern which is implemented on the target system, while Fig.2 shows the requirements in an OCL statement. The table and the OCL statement indicate that a user is considered as a regular user only if it has a valid credential in the system. A regular user is given some rights to access the protected assets such as their personal information. The user could have more rights depending on the role of the user.

```
class Authenticator
attributes
   Requester:Principal
constraints
   inv Authentication:
      if self.Requester.credential.validity = true then
         self.Requester.regular_user = true
      else
         self.Requester.regular_user = false
      endif
end
```

Fig. 2. The constraints in credential pattern.

```
--Create objects
!create user_1:Principal
!create credential_1:Credential
--Insert association
!insert(user_1, credential_1) into Own
--Initialize test case
!set credential_1.validity := 'valid'
!set user_1.name := 'Regular_user'
!set user_1.regular_user := true
!set user_1.credential := credential_1
--Execute
!openter user_1 access_assets()
```

Fig. 3. The credential pattern test case.

A test case of a regular user is shown in Fig.3. The statements in the test case can be executed in USE. In this environment, we are able to create objects of our existing classes and can also assign values to the attributes in the objects. However, the values we assigned to the objects have to satisfy the constraints in OCL otherwise the OCL check will output errors. Fig.3 shows the test case of regular users. Additionally, we also need to test the case of an irregular users in order to compare the outcomes with the requirements in Table 3.

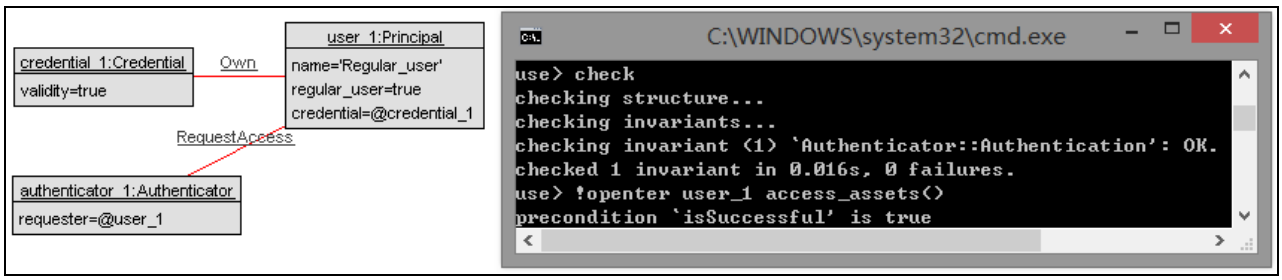

Fig. 4. The USE test result of credential pattern.

Fig. 4 shows the USE test result of acting as a regular user. The USE test outputs true while the value of credential validity is "true", and the test outputs false if the validity is "false". The results satisfy the security design requirements in Table 3, hence, the pattern is applied appropriately. The satisfaction of security requirements in Table 1 indicates that this input model of countermeasure works appropriately and offers protection for our assets. This means that only the users holding valid credentials can access the protected assets.

4.3. Step 3: Apply and Validate Misuse Pattern

In this step, we change the perspective from a defender into an attacker. From Table 2, we have determined that the aim of principal spoofing is to illegally access regular users' personal data or other protected assets. Therefore, the approach to successfully attain illegal access is to impersonate the identity of a regular user.

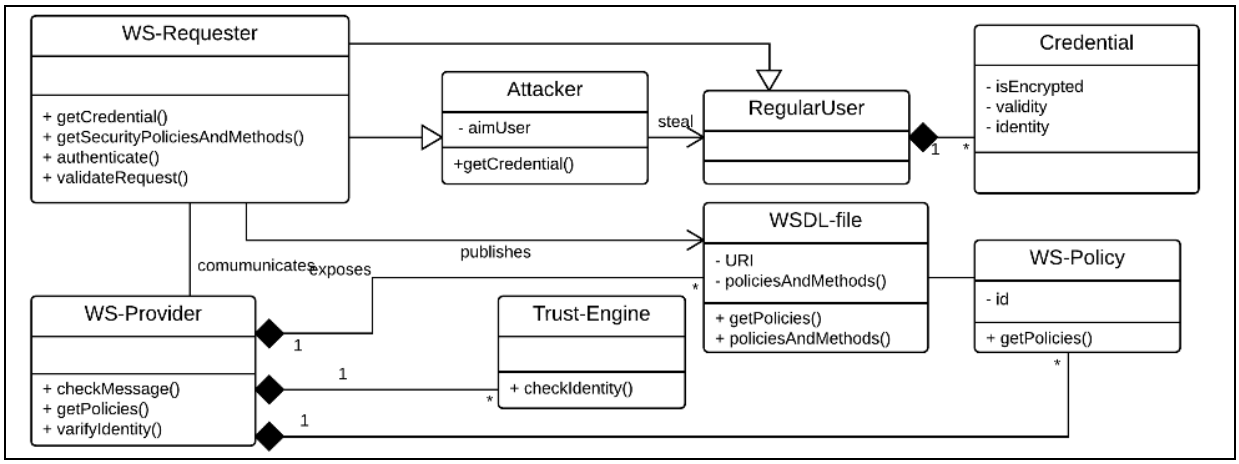

Fig. 5. Class diagram of principal spoofing [14].

Fig. 5 shows the class diagram of principal spoofing in UML. The principle of authentication part in this pattern is similar to the credential pattern. The additional units in this misuser pattern are presented to support the performance of principal spoofing. The WS-requester is normally a user. The WS-Provider have some policies which are stored in the WSDL file. Another addition to this pattern is the attacker class. The

attacker and the user are both considered as principal but the attacker has the "getCredential()" function. However, having this function does not mean that the attacker can definitely steal the credentials from regular users. There are some limitations, we explain them in the subsequent step.

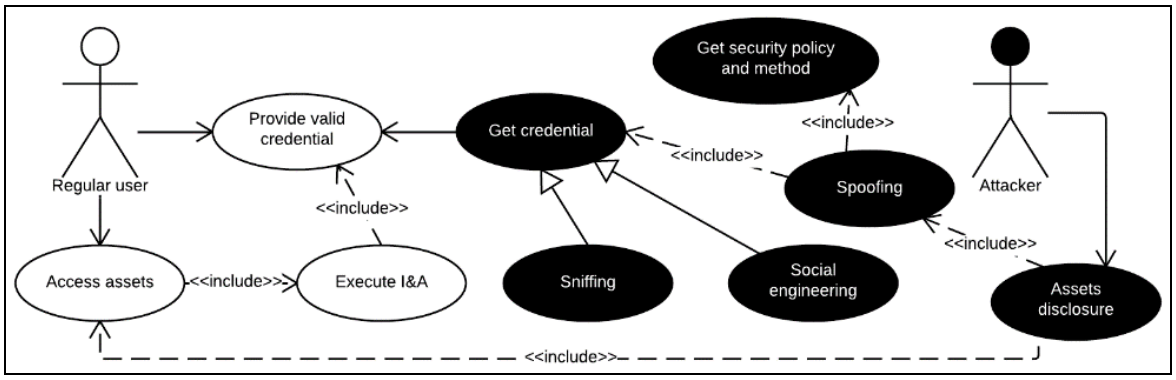

Fig. 6. The misuse case of principal spoofing.

Fig. 6 shows the misuse in use case diagram. We can easily understand that the key of the spoofing a web server is to obtain valid credentials and masquerading as a regular user. An attacker can obtain credentials from regular users as well as get security policies and methods of the web service from WSDL file. With valid credential and information from WSDL file, the attacker can create web services according to their intention. The attacker will pass the identification and the request will be authenticated as long as the attacker have a valid credential.

```
constraints
  inv getCredential:
    if self.victimUser.credential.isEncrypted = false then
      self.credential = victimUser.credential
    else
      self.credential.isUndefined
    endif
  inv getSecurityPolicies:
    if self.aimServer.wsdlFile.isPublished = true then
      self.securityPolicies = true
    else
      self.securityPolicies = false
    endif
constraints
  inv Authentication:
    if self.requester.credential.validity = true and
    self.requester.securityPolicies = true then
      self.requester.regular_user = true
    else
      self.requester.regular_user = false
    endif
```

Fig. 7. The constraints in principal spoofing pattern.

```
--Set regular user
!set user_1.name := 'Regular_user'
!set user_1.regular_user := true
!set user_1.credential := credential_1
--Set attacker
!set attacker_1.name := 'Irregular_user'
!set attacker_1.regular_user := true
!set attacker_1.credential := credential_1
!set attacker_1.victimUser :=user_1
!set attacker_1.aimServer :=webserver_1
!set attacker_1.securityPolicies := true
--Set credential
!set credential_1.isEncrypted := false
!set credential_1.validity := true
.
.
--Set WSDL-File
!set wsdlfile_1.isPublished := true
--Execute
--!openter attacker_1 access_assets()
```

Fig. 8. Principal spoofing pattern test case.

In order to validate whether this misuse pattern satisfies the requirements in Table 2, we need to build a test template and implement a test in USE again. The constraints are shown in Fig. 7. The invariant "get Credential" and "get Security Policies" are in the attacker class, while the variant "Authentication" is in the

WS_Provider class. Those constraints indicate that the attacker can get the credential from a regular user if the credential is not encrypted in the communication between the user and the web server. The attacker is also able to get the security policies form WSDL file if the WSDL file has been published by the web server. A principal (user/attacker) is considered as a regular user if it uses right security policy and has a valid credential.

Fig. 8 shows a part of the test case. This test case satisfies the misuse requirements in Table 2. The case setting is for testing whether an attacker can personate a regular user by holding a valid credential got from a regular user. However, we do not know whether this ideal case satisfies the OCL described in Fig.7. Therefore, we test this case in USE to check all the constraints. We then execute the "access_assets()" function to see the result after we pass all the constraint checks.

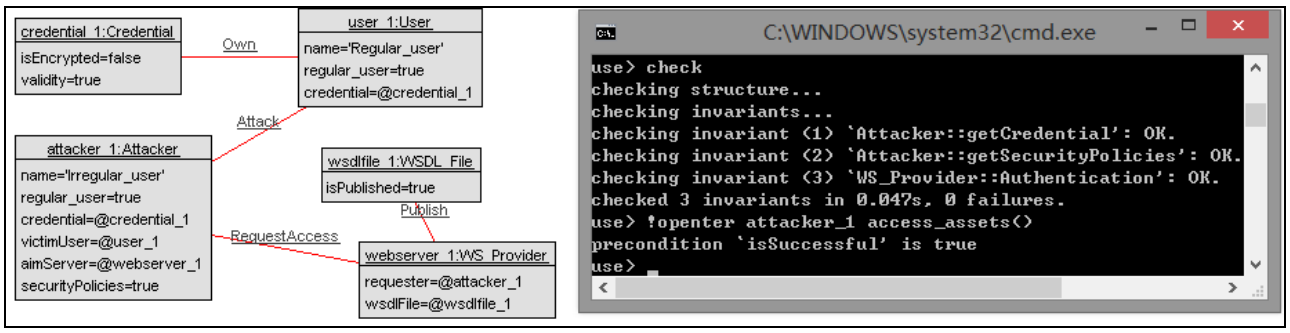

Fig. 9. The USE test result of principal spoofing.

Fig. 9 shows the test result in USE. From the commander window we can see that the test case model satisfies all the constraints in Fig. 7. The result of executing the "access_assets()" function is true. This means that the attacker in this test case is considered as a regular user and has the right to access the protected assets. Therefore, the test case finally satisfies the misuse requirements in Table 2. It indicates that the victim system in this example is vulnerable to the principal spoofing attack. We next then need to analyze the vulnerabilities.

4.4. Step 4: Analysis of the Vulnerabilities

As we observed from the model testing process, the success of this identity spoofing is attributed to the achievement of stealing valid credential and getting security policies from WSDL file. WSDL file is open for everyone since people need it to communicate with the web server. Accessing WSDL file is not considered as a disclosure. The misuse of credential is indeed the key of this misuse pattern. As we mentioned in the previous step, in the real world, stealing a credential from other user is not a simple work. However, in this testing model, the lack of protection on the credentials offers the attacker a possibility of stealing the credentials. This is the vulnerability of this example system model.

4.5. Step 5: Countermeasures

In this case, the countermeasure is a combination of user ID and password. The two main ways of getting credential are obtaining from the user personally and sniffing the communication between a regular user and the server.

It is difficult to control the first one since we cannot control human behavior with computers. So the countermeasures should mainly focus on reducing the risks arising from the second approach. Encrypting the credential or communications between the users and servers is an effective method of doing this, an example is the XML encryption pattern in [15].

5. Conclusion, Limitations and Future Word

In this paper, we have proposed a method to analyze system security vulnerabilities using misuse pattern model testing approach in UML simulation environment. Although it is impossible to uncover all of the vulnerabilities of a system with this method, our method offers a new efficient way of detecting the specific security vulnerabilities which are corresponded to a certain attack.

Another advantage of our method is the ability to detect security vulnerabilities in the early stage of software or system development. This allows for reducing the cost of fixing the vulnerabilities in the design stage rather than in the verification or maintenance stages in the development of a system. Need to be known that the security vulnerability detection during the design stage is not able to find out the coding vulnerabilities. The code-based security checking is a good complement to model-based work [16].

The number of available misuse pattern limits the practicality of our method since the model testing processes rely on misuse patterns. Therefore, modelling the attack beforehand is required if this attack is not yet described in the pattern.

The case study in this paper simulated a simple model of a part of a web server. In the future, we will build and use a model from real-life systems to evaluate the usability and performance of our method empirically by comparing the results offered by our method to the real vulnerabilities in the system.

Acknowledgment

Researching and discovering the new knowledge and methods in a specific field are tough tasks. The power, knowledge and experiments from a single person seemed to be insufficient to accomplish those tasks. I would not be able to finish this work without the helps and supports from my advisor, friends and my family. Here I would like to express the deepest appreciation to those who helped me on this research.

References

[1] Howard, M. & Lipner, S. (2009). *The Security Development Lifecycle*. Microsoft press.

[2] OWASP. Comprehensive Lightweight Application Security Process.

[3] Yoder, J., & Barcalow, J. (1998). Architectural patterns for enabling application security. *Urbana*, *51 (61801)*.

[4] Schumacher, M., Fernandez, B. E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2013). *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons.

[5] Fernandez, E. B., Yoshioka, N., & Washizaki, H. (2009, March). Modeling misuse patterns. *Proceedings of IEEE International Conference on Reliability and Security* (pp. 566-571).

[6] Livshits, V. B., & Lam, M. S. (2005, August). Finding security vulnerabilities in java applications with static analysis. *Proceeding of USENIX Security Symposium* (pp. 18-18).

[7] Kobashi, T., Yoshioka, N., Okubo, T., Kaiya, H., Washizaki, H., & Fukazawa, Y. (2013, September). Validating security design pattern applications using model testing, *Proceeding of IEEE Eighth International Conference on Availability, Reliability and Security* (pp. 62-71).

[8] Fernandez, E. B., & Delessy, N. (2006, February). Using patterns to understand and compare web services security products and standards. *Proceeding of IEEE Advanced International Conference on Internet and Web Applications and Services*. (pp. 157-157).

[9] Pelaez, J. C., Fernandez, E. B., & Larrondo, P. M. M. (2009). Misuse patterns in VoIP. *Security and Communication Network Journal*, *2(6)*, 635-653.

[10] Fernandez, E. B., Yoshioka, N., & Washizaki, H. (2010, March). A worm misuse pattern. *Proceedings of the 1st Asian Conference on Pattern Languages of Programs*.

[11] Rumbaugh, J., Booch, G., & Jacobson, I. (2003). *The Unified Modeling Language 2.0 Reference Manual*. Addison-Wesley, 2003.

[12] Warmer, J., and Kleppe, A., (2003). *The Object Constraint Language: Precise Modeling with UML*. 2nd edition, Addison-Wesley.
[13] Gogolla, M., Büttner, F., & Richters, M. (2007). USE: A UML-based specification environment for validating UML and OCL. *Science of Computer Programming, 69(1)*, 27-34.
[14] Muñoz, A. J., Caudel, G. H., & Fernandez, E. B., Misuse pattern: spoofing web services. *Proceedings of the 2nd Asian Conference on Pattern Languages of Programs*.
[15] Hashizume, K., & Fernandez, E. B. (2009, August). Symmetric encryption and XML encryption patterns. *Proceedings of the 16th Conference on Pattern Languages of Programs*.
[16] Fernandez., E. *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*. John Wiley & Sons.

Yuan Yifan received his bachelor's degree in information security from Beijing University of Technology and his second bachelor's degree in information technology from Mikkeli University of Applied Science in Finland in 2009. He is currently a graduate student in Chulalongkorn University, Bangkok, Thailand.

Somjai Boonsiri received her bachelor's degree in mathematics from Chiang Mai University in 1984, She received her master's degree in computer science in 1992 and She received her PhD degree in computer engineering in 2002 from Chulalongkorn University, Bangkok, Thailand. She is an associate professor of the Department of Mathematics and Computer Science, Faculty of Science in Chulalongkorn University.