ANALYSIS OF SECURITY VULNERABILITIES USING MISUSE PATTERN TESTING APPROACH

Mr. Yifan Yuan

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Computer Science and Information Technology
Department of Mathematics and Computer Science
Faculty of Science
Chulalongkorn University
Academic Year 2014

การวิเคราะห์จุดอ่อนด้านความปลอดภัยโดยใช้วิธีการทดสอบแบบรูปที่ใช้ผิด

นายยี่ฟาน หยวน

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2557

Thesis Title	ANALYSIS OF SECURITY VULNERABILITIES USING MISUSE PATTERN TESTING APPROACH
By	Mr. Yifan Yuan
Field of Study	Computer Science and Information Technology
Thesis Advisor	Somjai Boonsiri

Accepted by the Faculty of Science, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

....Dean of the Faculty of Science

(Professor Supot Hannongbua, PhD)

THESIS COMMITTEE

....Chairman

(Associate Professor Peraphon Sophatsathit, PhD)

....Thesis Advisor

(Assistant Professor Somjai Boonsiri, PhD)

....External Examiner

(Assistant Professor Kriengkrai Porkaew, PhD)

ยี่ฟาน หยวน :
การวิเคราะห์จุดอ่อนด้านความปลอดภัยโดยใช้วิธีการทดสอบแบบรูปที่ใช้ผิด (ANALYSIS OF SECURITY VULNERABILITIES USING MISUSE PATTERN TESTING APPROACH)
อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ. ดร. สมใจ บุญศิริ.

วิธีการตรวจจับจุดอ่อนโดยส่วนใหญ่ในปัจจุบันนั้นมักจะกระทำในขั้นตอนการทดสอบของการพัฒนาซอฟต์แวร์ ทำให้วิธีการเหล่านี้ไม่สามารถที่จะตรวจจับจุดอ่อนด้านความปลอดภัยของระบบหรือซอฟต์แวร์ ที่อาจเกิดจากการโจมตีในช่วงต้นของการพัฒนาซอฟต์แวร์ได้ การโจมตีนี้อาจเกิดขึ้นนอกเหนือจากความคาดหมายในขั้นตอนการออกแบบซอฟต์แวร์ ดังนั้นงานวิจัยนี้ จึงนำเสนอวิธีการตรวจจับจุดอ่อนด้านความปลอดภัยที่อาจเกิดขึ้นในขั้นตอนการออกแบบของการพัฒนาซอฟต์แวร์ วิธีการนี้ได้จำลองการโจมตีที่มีพื้นฐานบนแบบรูปที่ใช้ผิดโดยใช้วิธีการทดสอบตัวแบบ ซึ่งวิธีการนี้สามารถวิเคราะห์จุดอ่อนด้านความปลอดภัยในขั้นตอนการออกแบบของการพัฒนาซอฟต์แวร์ได้ จากผลการทดลองได้แสดงให้เห็นว่า วิธีการที่เสนอนี้มีประสิทธิภาพในการวิเคราะห์จุดอ่อนด้านความปลอดภัยได้อย่างเหมาะสม

ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์	ลายมือชื่อนิสิต	....
		ลายมือชื่อ อ.ที่ปรึกษาหลัก	....
สาขาวิชา	วิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ		
ปีการศึกษา	2557		

5672640423 : MAJOR COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

KEYWORDS:

VULNERABILITY; SECURITY PATTERNS; MISUSE PATTERNS; UML; USE; MODEL TESTING.

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 were anticipated as well as the ones unknown during the design phase. This research proposes 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, one is able to analyze system security vulnerabilities during the design stage of the system development. The practical examples provide evidences to the feasibility of the proposed method.

Department:	Mathematics and Computer Science	Student's Signature	....
		Advisor's Signature	....
Field of Study:	Computer Science and Information Technology		
Academic Year:	2014		

ACKNOWLEDGEMENTS

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.

I would like to thank my advisor Dr. Somjai Boonsiri, who directed my researching field, offered great guidance and helped me not only on the technical problems but also on every details in the process of researching and writing this thesis.

Additionally, I would like to thank my committee members Dr. Peraphon Sophatsathit and Dr. Kriengkrai Porkaew for participating and examining my dissertation. And also thanks for their excellent and valuable suggestions.

I also sincerely appreciate my friends for their enthusiastic assistances and my parents' positively support.

I would never have been able to make this without any of your helps. Thanks all again.

CONTENT

CHAPTER 1

INTRODUCTION

1.1 Introduction

Most modern applications are distributed and connected by networks. Mostly the Internet, and also LANs and other types of networks. A growing size of important and secret information is stored on the networks since their platforms support web interfaces, web services and web agents. The complexity of the distributed systems brings vulnerabilities without doubt. To reduce the risk of the important information disclosure, it is crucial to develop secure systems and applications. And this needs security techniques.

There are several approaches to build secure systems such as Microsoft's Security Development Lifecycle (SDL) [1], QWASP's CLASP [2] etc. However, in SDL and CLASP, most of those traditional vulnerability detection methods 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 traditional security approaches 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 are useful packages with the knowledge of security experts. Software engineers can develop secure software or system by choosing and following the guide of specific security patterns. Each security pattern guarantees that the protected system has ability to resist a specific type of threat. However, security patterns have some limitations [3]. One of the limitations of security patterns is that they focus on threats instead of vulnerabilities. Usually a single vulnerabilities could result in multiple attacks.

The misuse pattern has been proposed in order to help identifying which security pattern should be used to stop or mitigate a specific type of attacks and understand the underlying principle of attacks. Merely using the misuse pattern still does not address the focus on vulnerabilities. Additionally, sometimes even the identification of vulnerabilities in the existing design model of a software or a system can be very difficult.

Therefore, this research proposes a systematic method to detect the system security design vulnerabilities through the model testing of misuse patterns. This method helps uncovering existing vulnerabilities that expose the system to potential threats. This method also indicates what security countermeasures should be used to mitigate such risks. Most importantly, drawing benefits from the contributions of model testing on security patterns [4], this method is able to implement the analysis process of analysis during the design stage of system development. From a cost-benefit perspective, this is valuable because improving the design itself often has a much lower cost than debugging and fixing the security vulnerabilities in the system during runtime.

1.2 Problem formulation

This research focuses on the following problems:

1. What is the method to simulate an attack on a design model with misuse pattern?

2. How can one find out the vulnerabilities and corresponding countermeasures from the simulated attack during the design stage?

3. What is the process to validate the offered countermeasures?

1.3 Contributions

1. A method using modeling misuse patterns to simulate attacks.

2. Preliminarily identify the security vulnerabilities to a type of attacks during design stage.

1.4 Scope of the work

1. The focus of this work is at the design stage of software development life cycle.

2. The models and patterns in this research are described in UML.

3. The UML models and their OCL constraints are tested in USE.

4. In order to ensure the correctness of the attack models, the misuse patterns used in this research are chosen from existing misuse patterns.

5. In principal spoofing, the way of stealing credential by social engineering is not involved in this research since human behaviors cannot be controlled by computers.

1.5 Document organization

This document is organized as follows. Chapter 2 contains the related prior work. Chapter 3 concretely demonstrates the proposed method of vulnerability analysis process. Chapter 4 shows two practical experiments in order to introduce the proposed method with deeper understanding. The conclusion and future works are given in Chapter 5.

CHAPTER 2

LITERATURE REVIEW

This chapter introduces the related background knowledge about this thesis. The content includes basic introduction of security and misuse patterns, testing tool as well as the attacks and countermeasures that will be used in the experiments. Having this background knowledge gives a good assistance on understanding the proposed method in this thesis.

2.1 Security pattern

Security patterns, a term initially been introduced in 1998[5], 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 [6].

A security pattern describes a solution to the problem of stopping or mitigating a set of specific threats through some security mechanism. Besides solving a set of forces, the solution also need to be able to describe using UML class, sequence, state and activity diagrams. The consequence indicates how well the forces were satisfied and how well the threats were mitigated. Security pattern focuses on threat, the vulnerability is not directly related to security pattern. A pattern may stop or mitigate a set of threats caused by one vulnerability, but it does not intend to repair the vulnerabilities.

Security pattern can be considered as any of the following ways. An architectural pattern, a design pattern, an analysis pattern and a special type of pattern. Here in this thesis, the pattern is only looked as a design pattern since the proposed the method is focusing on the design stage of the software development.

Each security pattern consist of several sections as shown in Figure 2-1, and their functions are shown as below:

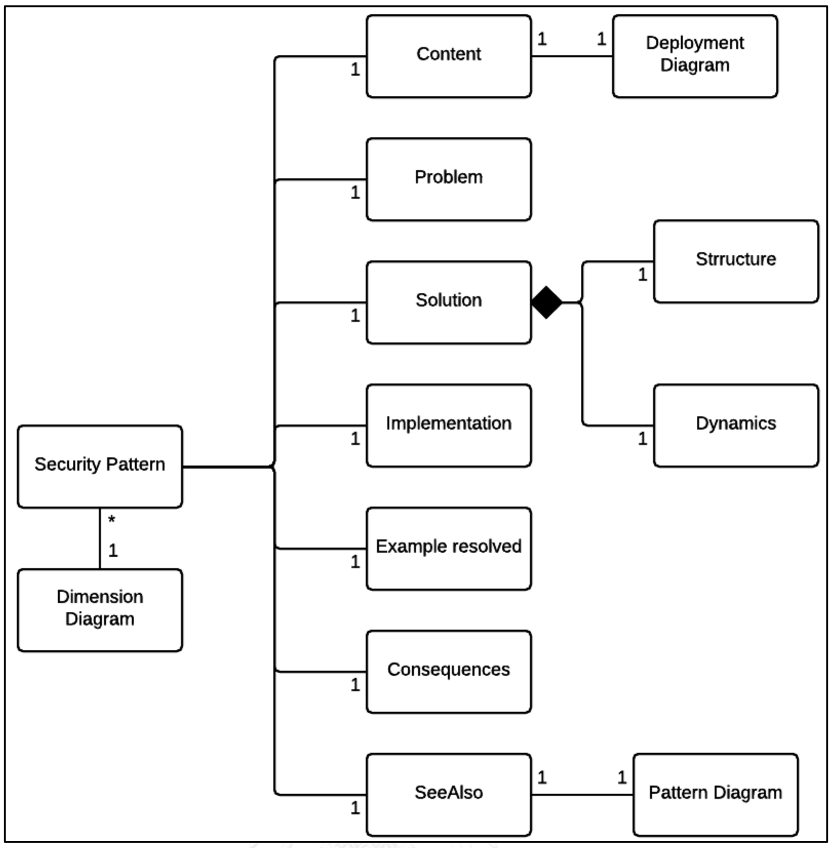


Figure. 2-1 The structure of security pattern

Context: To define the context in which the pattern solution is applicable.

Problem: To demonstrate what happens if developers do not have a good solution under the situation in context. Also indicate the forces that affect the possible solution.

Solution: To describe the idea of the pattern. This section includes the static view and dynamics of the solution which are described in UML model.

Implementation: To describe what should be considered when implementing the pattern.

Example Resolved: To tell the example results after implementing the solution.

Consequences: To indicate the benefits and liabilities of the solution in this pattern.

Known Uses: To accept this solution as a pattern, some examples of the use in real systems are required.

See also: To relate this pattern to other known patterns.

Apparently, the solution section is the core section of a pattern. However, it does not mean only the solution section matters. Patterns can be very valuable for building security systems because they emphasize not only the solution but also the problem.

2.2 Misuse pattern

As introduced in the previous section, security patterns focus on threats instead of vulnerabilities. It is useful to guide the security design of systems but it does not clearly tell the designer what pattern should be applied to stop a type of attacks especially those who are not expert in security.

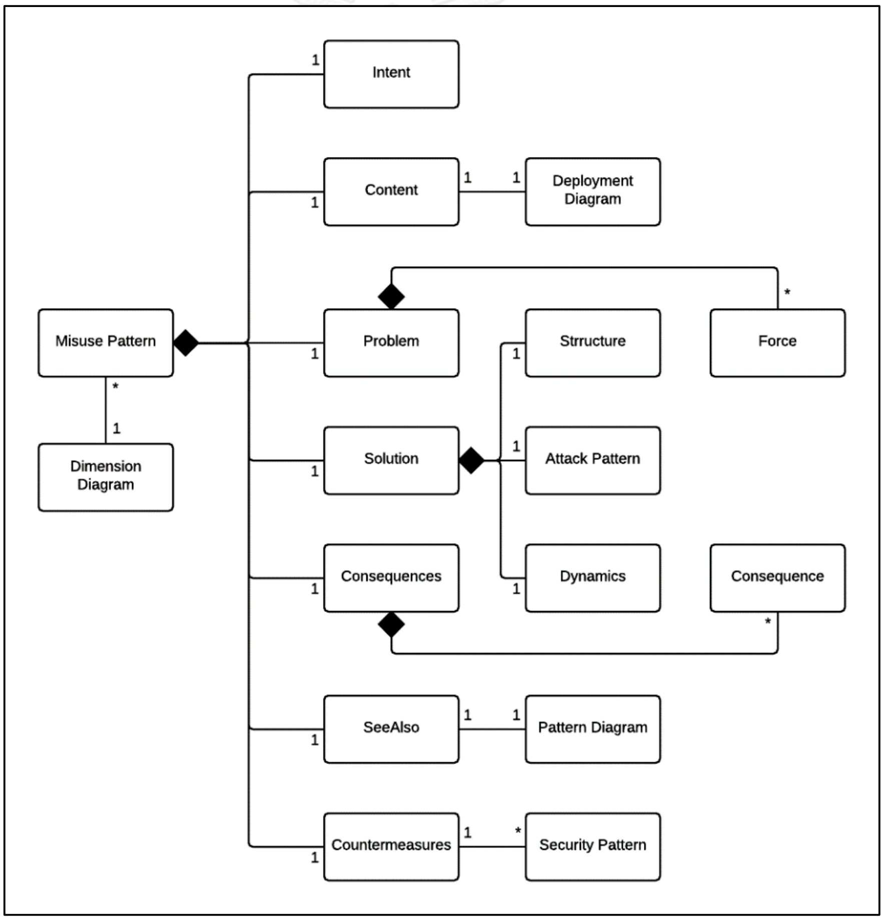


Figure. 2-2 Class model of misuse pattern [3]

The misuse patterns are proposed in order to complement security patterns. The structure of misuse pattern is similar to security pattern. The difference is that misuse pattern describe from the perspective of attacker. The main purpose of misuse pattern is to describe and show how a type of attack is performed, what components are utilized by the attack in the target model and also analyze the way to stop the attack.

A misuse is an unauthenticated use of assets in a system or software. E. B. Fernandez, et al., proposed the method of modeling misuse pattern in 2009 [3]. Figure 2-2 present the structure of misuse patterns. The structure of misuse patterns is similar to security patterns however the misuse pattern describes from the perspective of attacker. Most apparently, there is an attack model under the solution sections.

Some of the sections are different from their definition in security patterns as shown in Figure 2-1:

Problem: Differ from the problem section in security patterns, the problem here describes how to find a way to attack the system. The forces indicate what factors may be required in order to accomplish the attack and in what way.

Solution: In security patterns, solution section tells how and what security countermeasures should be applied to against the threat described in problem section. However the solution section in misuse pattern describes how the misuse can be accomplished and what is the expected results of the attack.

Additionally, the countermeasure section describes the security measures necessary in order to stop, mitigate, or trace this type of attack.

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 [7]. Many useful misuse patterns have been proposed such as the denial of service attack in VoIP [8] and worm [9]. 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. Subsequently, the method helps to analyze the security vulnerabilities and find means to counter the attack.

2.3 Security requirements

Security requirements are a class of Non-Functional Requirements (NFRs) that relate to system confidentiality, integrity and availability. Explicitly stating security requirements during project inception is the perfect complement to security testing. Clearly outlining potential security requirements at the project allows development teams to make trade-offs about the cost of applying security mechanisms into a project.

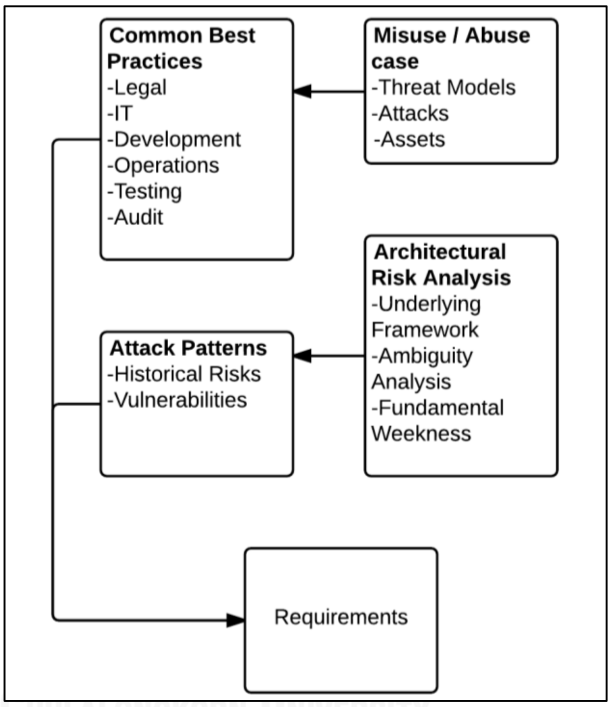


Figure 2-3: Four ways to create security requirements [10]

There are four ways to create security requirements as shown in Figure 2-3.

The security requirements of withdrawing cash from ATM is quoted here as an Example.

The functional requirements of withdrawing cash from ATM includes:

- Use a valid bank card.
- Require correct PIN code to login.
- Withdraw not exceed amount balance.

Suppose there are already some countermeasures exist in the ATM to satisfy these above security requirements. In order to test and judge whether these countermeasures are applied appropriately in the ATM, they need to be transferred into a more concrete and readily tested form such as the Table 2-1.

Table 2-1: Security requirements of withdraw cash from ATM

		1	2	3	4
Conditions	Regular user	Yes	Yes	No	No
	Valid transaction	Yes	No	Yes	No
Actions	Execute “withdraw cash” process	X			
	Not Execute “withdraw cash” process		X	X	X

Table 2-2: Security design requirements of withdraw cash from ATM

		1	2	3	4	5	6	7	8
Conditions	Valid bank card	Y	Y	Y	Y	N	N	N	N
	Correct PIN code	Y	Y	N	N	Y	Y	N	N
	Enough Balance	Y	N	Y	N	Y	N	Y	N
Actions	Considered as regular user	X	X						
	Considered as irregular user			X	X	X	X	X	X
	Considered as valid transaction	X		X		X		X	
	Considered as invalid transaction		X		X		X		X
	Execute “withdraw cash” process	X							
	Not Execute “withdraw cash” process		X	X	X	X	X	X	X

This table shows what the expected results of testing the countermeasures should be. However, it is still not practical enough for the tests. The problem is obvious. As human beings, it is easy for us to understand that we have to satisfy all of the conditions in order to execute the willing process. But for tests, it is necessary to tell the computer what exactly is a “regular user” and a “valid transaction”. Therefore, another table called the security design pattern are required as shown in Table 2-2.

All of the 8 cases have to be generated as test cases in the model testing process. The countermeasures are considered applied appropriately if the results are exactly fit the content in the Table 2-2.

2.4 Misuse requirements

The principle of misuse requirements is similar to the security requirements. They just change from the perspective of defender to the perspective of attacker. The method and process of testing are the same as security requirements. So a repeated example is not shown here. One thing need to be learned about misuse requirements is that it is not necessary to test all of the cases in misuse requirements since the misuse test simulates an attack, the cases without attacking intentions are no need to be concerned.

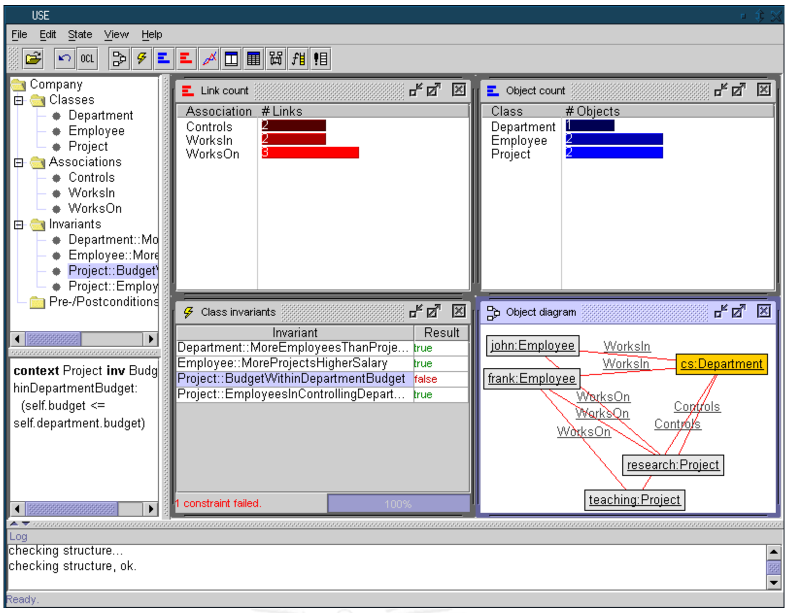
2.5 UML-based specification environment

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, there must be some methods 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. Figure 2-5 shows a sample interface in USE tool.

This is the major tool used for implementing experiments in this thesis.



Association	# Links
Controls	2
WorksIn	2
WorksOn	3

Class	# Objects
Department	1
Employee	2
Project	2

Invariant	Result
Department::MoreEmployeesThanProje...	true
Employee::MoreProjectsHigherSalary	true
Project::BudgetWithinDepartmentBudget	false
Project::EmployeesInControllingDepart...	true

1 constraint failed.

```
context Project inv Budg
hinDepartmentBudget:
  (self.budget <=
self.department.budget)
```

```
checking structure...
checking structure, ok.
```

Figure 2-5: Example of USE tool interface

2.6 Principal spoofing

Principal spoofing pattern is one of the misuse patterns that will be used in this thesis. It is necessary to have some basic knowledge about this type of misuse. A spoofing misuse is in a circumstance that a person tries to impersonate another.

Regardless of whether in the human world or the internet. There are always some situations that need to prove one's identity when doing something. For example in the airport, the staffs compare passengers' face with their passport pictures to validate their identities. Identically, every user has a digital "face" in the internet, most commonly the user's ID. For a regular authentication method, users are also required to provide something to be their digital "passport" such as a password. Unfortunately, both the digital "face" and "passport" can be stolen. The principal spoofing happens when a person uses the stolen "face" and "passport" to impersonate others. An integrate introduction of spoofing attack is introduced in [14].

Many different systems and platforms can be the targets of spoofing. This thesis scope the target in web services.

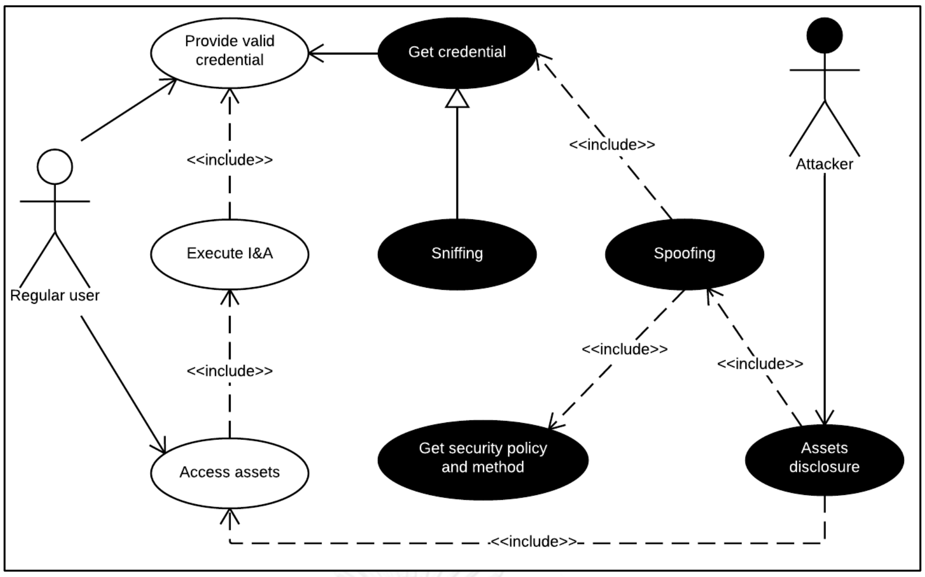


Figure 2-6: The misuse case of principal spoofing

Figure 2-6 shows principal spoofing misuse in use case diagram. It is easy to 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 by maliciously sniffing the network communication data. It can also 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 has a valid credential.

2.7 SQL injection

The other misuse in this thesis is SQL injection. SQL injection attacks represent a serious threat to any database-driven site and they are one of the most frequent types of attacks. This attack is effective especially on the active server pages (ASP) based sites.

A SQL injection misuse consists of insertion of arbitrary code into a SQL query by the client in order to alter its intended function, allowing the attacker to retrieve arbitrary amounts of unauthorized data from the database.

The SQL injection can be very flexible according to the website and the input page. Several major SQL injection are introduced in [15]. In the design stage, it is not vital to know what exactly the SQL injection code is since the webserver is not really exist. However it is important to demonstrate the idea of this attack using a typical example.

A simple example of the SQL injection is demonstrate here to help understanding this attack.

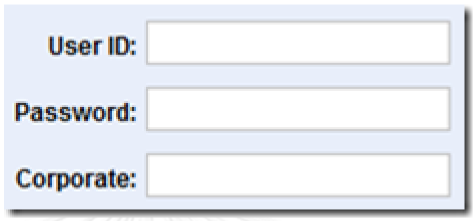

User ID:	
Password:	
Corporate:	

Figure 2-7 A common login window of company website

The login interface in Figure 2-7 is very common in company websites. The users enter the ID, password and the corporate as a credential to pass the identification and authentication.

It could be very easy for attackers to guess that the SQL statement construction of this login page is as below:

*SELECT * From Table WHERE Name='XX' and Password='YY' and Corp='ZZ'*

Since the SQL database just simply gets the command from the web and return the results to the web. The input boxes could be utilized to generate some arbitrary statement to the database.

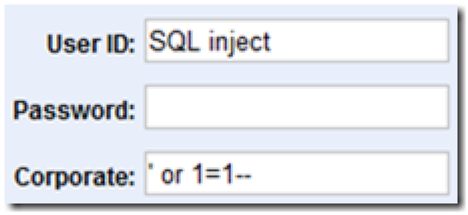

User ID:	SQL inject
Password:	
Corporate:	' or 1=1--

Figure 2-8 A sample SQL injection

Figure 2-8 shows a very simple example of the SQL injection on this website. Enter anything in the “user ID” and “Password” boxes and input the content as shown in the Figure in the “corporate” box.

Then the SQL statement submitted to the database would be:

*SELECT * From Table WHERE Name=‘SQL inject’ and Password=‘’ and Corp=‘’ or 1=1--’*

This statement could be separated into:

*SELECT * From Table WHERE Name=‘SQL inject’ and Password=‘’ and Corp=‘’*

Or

1=1

The first statement must not get a valid return because the password and corporate are missing. However, the “corporate” box is utilized to input “’or 1=1--”. The “’” is considered as the closing single quotation mark of the box and the “--” makes the embedded closing single quotation mark in the SQL statement as an annotation which will not be executed by the computer.

Therefore, in the condition of judging the whole statement, the first part will be returned a “false” while the second part “1=1” is always true. The attacker then skips the system I&A and logins into the system without any valid credential.

Skipping I&A is not the only way of using SQL infection on a database-driven system. Various more harmful SQL injections are truly exist. For the scope of this thesis, the more examples about this attack are not demonstrated here.

2.8 XML encryption

XML encryption is a security countermeasure that will be used in this thesis. It already has its pattern available in [16].

The idea of XML encryption is simple. It provides confidentiality by hiding selected sensitive information in a message using cryptography. Although it is called XML encryption, it actually can encrypt any kind of data.

XML encryption is very valuable for cyber communication. Especially nowadays internet is not simply a tool for entertainment. People intend to do more and more commercial things on internet like shopping, banking and so on.

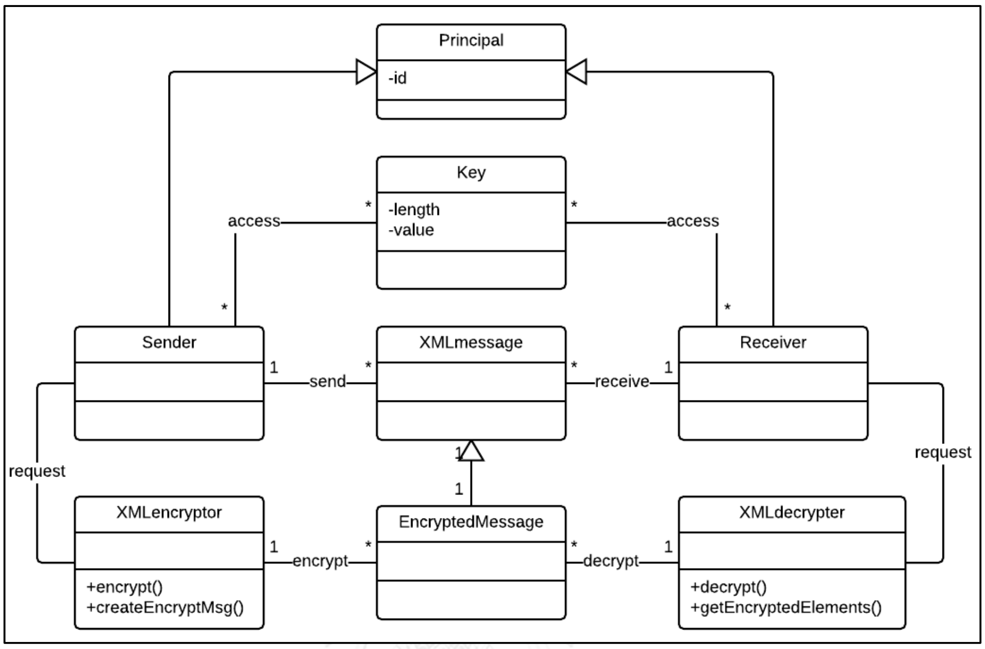


Figure 2-9: The class diagram of XML encryption

Figure 2-9 shows the class diagram of the XML encryption pattern. A principle as introduced in the principal misuse pattern, is commonly considered as a user. The user sends and receives XMLMessages and EncryptedXMLMessages. The Principal may has the roles of sender and receiver. The XMLEncryptor and the XMLDecryptor encipher a message and decipher an encrypted message, respectively.

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
   xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <CipherData>
      <CipherValue>A23B45C56</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

Figure 2-10: An example of XML encryption syntax [17]

Learning from the example in Figure 2-10, a user called John Smith is trying to conduct an online payment. Obviously, Smith's credit card number is sensitive information. If the application wishes to keep that information confidential, it can encrypt the “CreditCard” element. By encrypting the entire CreditCard element from its start to end tags, the identity of the element itself is hidden. The “CipherData” element contains the encrypted serialization of the “CreditCard” element.

Although current research shows that XML encryption is not invincible [18], XML encryption is still commonly considered as an efficient way to protect the cyber communication.

2.9 Input validation

Input validation is the other countermeasure which is been used in this thesis beside XML encryption.

It is always recommended to prevent attacks as early as possible during the process of attackers’ request. Input validation is able to detect malicious or unwanted inputs before they are passed to the system. The two main ways of input validation are introduced here. However, they are specified concretely in the design model test in thesis’s experiments.

The two ways are using black list and white list. These two strategies are like the access control in network firewalls.

Developers often use black list validation to detect some obvious malicious characters such as “’”, “=” or “<>”. The validation blocks the requests that contain those characters.

The other way, white list validation tells what exactly is allowed to be a valid input. It is useful especially for some types of data like dates, social security numbers, zip codes, e-mail addresses, etc.

The input validation is not some kind of miracle drug to a specific misuse. But it is valuable on preventing many of the common vulnerabilities being actively exploited by malicious users if all of the data received and processed by your application is sufficiently validated.

2.10 Code review

The code review, also known as “white box testing” is in the implementation phase of a Security Development Lifecycle. SDL is a software development process that helps developers build more secure software and address security compliance requirements while reducing development cost [1].

As what have been introduced in Chapter 1. In the system and software development field, it is undeniable that the code-based analysis is still the most popular way on vulnerability digging. When this technique is used with automated tools and manual penetration testing (introduced in the next section), code review can significantly increase the cost effectiveness of an application security verification effort [19].

Security code review is a process on scanning the source code of a software or a system in order to verify the proper security controls are present. In the other word, the code review tries to make sure that the target software is doing what developers intended to. It is known that there are many types of vulnerabilities could be found in a software, a system or a website. It could be an insecure design, code, system service. It also could be caused by insecure internet protocols, transitions and so on. The duty of code review is to guarantee that the code-based vulnerabilities can be dug out as much as possible.

Latterly the proper relationship between this technique, penetration test and the proposed method in this thesis will be demonstrated.

2.11 Penetration test

Penetration test is another extremely useful technique on finding system vulnerabilities especially cyber systems and platforms. A penetration test can actually be treated as an attack. Hackers and penetration testers have the same skills and knowledge. The key differences between penetration test and hacking are that the first, penetration test intends to find system vulnerabilities, potentially gaining access to it, its functionality and data [20], while hacking has malicious purposes. The second is, in the scope of the attack, penetration test is authorized by the owner of the target. The last, the penetration testers are required to submit the report of the test.

The process of penetration test includes probing for vulnerabilities as well as providing proof of concept attacks to demonstrate the vulnerabilities are real.

Identical as regular hacking, the penetration test also consist of four main phases (the names and numbers of each step might be different from different introducers, however the idea is the same).

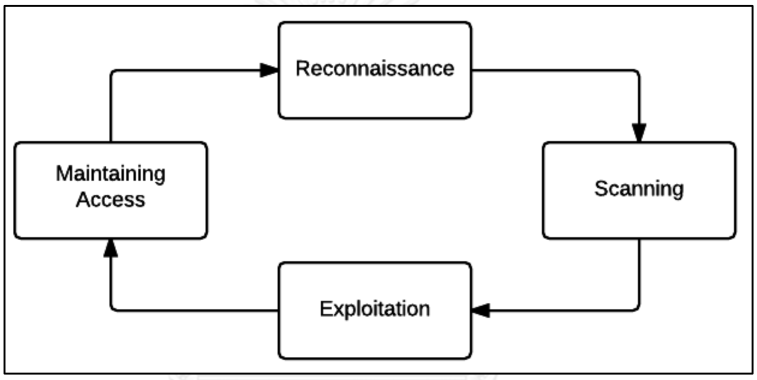


Figure 2-11: Four phases of penetration test

The four phases are shown in Figure 2-11. Reconnaissance is the information gathering phase. Subsequently the tester use tools or manually scan the system vulnerabilities. Once the vulnerabilities are found, testers are able to conduct an exploitation on the target system. The last thing is to leave a “back door” for maintaining the access to the target system.

If there is any vulnerability that can be proved. The penetration tester then need to report the result to the system owner.

CHAPTER 3

PROPOSED METHOD

The entire process of implementing the proposed method is introduced in this chapter. Once the testers have their system model and attack model ready, they are able to implement and accomplish the test by following the concrete process below. The test will end with telling the tester either no vulnerability found or the appropriate countermeasures to patch the vulnerabilities.

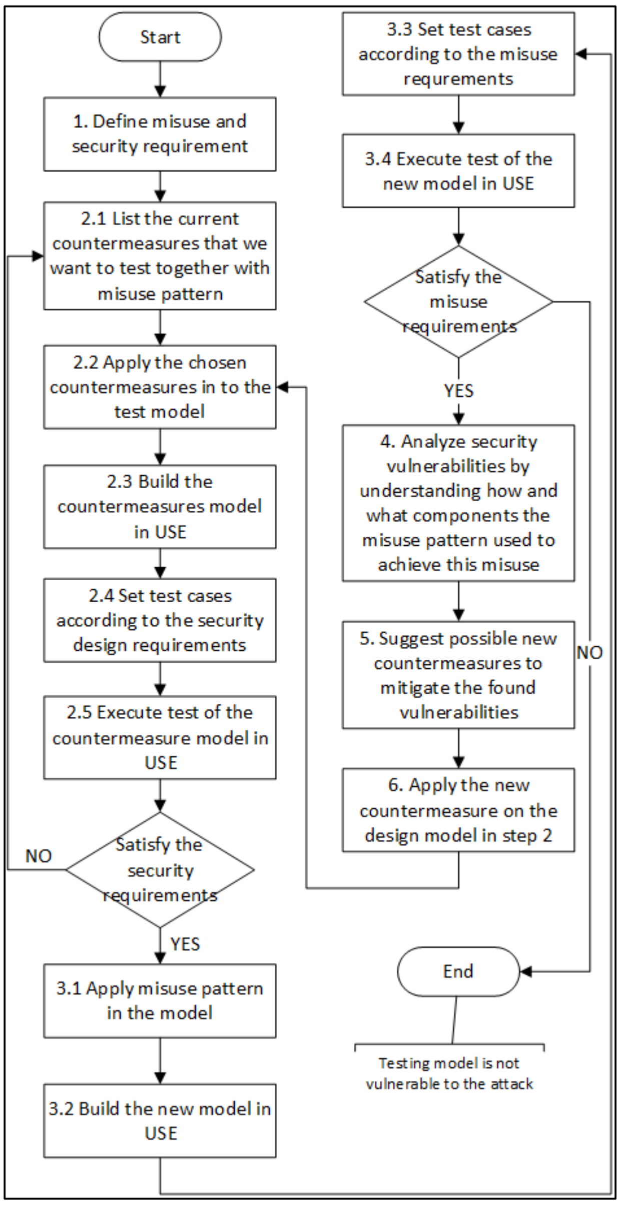

Start
1. Define misuse and security requirement
2.1 List the current countermeasures that we want to test together with misuse pattern
2.2 Apply the chosen countermeasures in to the test model
2.3 Build the countermeasures model in USE
2.4 Set test cases according to the security design requirements
2.5 Execute test of the countermeasure model in USE
Satisfy the security requirements
NO
YES
3.1 Apply misuse pattern in the model
3.2 Build the new model in USE
3.3 Set test cases according to the misuse requrements
3.4 Execute test of the new model in USE
Satisfy the misuse requirements
YES
NO
4. Analyze security vulnerabilities by understanding how and what components the misuse pattern used to achieve this misuse
5. Suggest possible new countermeasures to mitigate the found vulnerabilities
6. Apply the new countermeasure on the design model in step 2
End
Testing model is not vulnerable to the attack

Figure 3-1: The process of the proposed method

Figure 3-1 shows the process of the proposed method, which is partitioned into six steps:

1. Define misuse and security requirements

The purpose of this method is to test whether a simulated misuse can be implemented successfully on the target system. However, one cannot assume the test result subjectively. The results have to be judged by a strict standard.

To accomplish this task, the misuse requirements and security requirements are needed.

Security requirement lists the tasks that a countermeasure must satisfied. If a model does not satisfy the security requirement then the threats may still exist in the system. Which means the applied security pattern does not solve the security problem. In the other word, the security requirements are used to judge whether the security pattern is applied and work appropriately in a system design model.

Similar to the security requirement, misuse requirement has the same function that is to judge the result of applied misuse pattern. The chosen misuse pattern and the security pattern have to focus on the same security issue but expecting the opposite results. For example, if the security pattern is trying to protect an asset, then the misuse pattern should intend to steal or make it disclosed. Therefore, it has to be guaranteed that the security requirements and misuse requirements are not possible to be satisfied simultaneously.

2. Test countermeasures (security patterns)

It is necessary to know what security countermeasures are implemented in the target system model before validating the misuse pattern on it. Also, be ensured 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.

This task is divided into 5 substeps:

2.1 List the countermeasures and misuses that will be tested. The countermeasures could be chosen from available security patterns so that the design model of the countermeasure can be easily applied into the target system. Those countermeasures will be considered as defending measures of the system. Therefore ensuring that all of those applied countermeasures are working appropriately is a must.

2.2 Apply the countermeasure models into the system design model. By combining the countermeasure models with the system design model, testers can get the integrate target system model with its expected security defense measures.

2.3 Build the countermeasures model in USE. The testing process is implemented in USE tool which has been introduced in the precious chapter. To do this, the integrate target system model made in step 2.2 has to be built in the USE tool. Both class diagram and object diagram are needed in USE testing. Additionally, the countermeasure functions are described by OCL in classic diagram.

2.4 Set the test cases according to the security requirements. Test cases are created using object diagrams. Each diagram refers one test case. The examples are shown in the subsequent chapter. The security countermeasures are considered applied appropriately only if all of the test results from possible cases satisfy the security requirements.

2.5 Execute test of the countermeasure model in USE. The step 2.1-2.5 should be done repeatedly until the test results satisfy the security requirements.

3. Test attacks (misuse patterns)

The accomplishment of step 2 indicates that the preparations are completed. In this step, the selected misuse pattern is also applied and tested it in USE. This is the core step in validating misuse pattern.

3.1 Apply misuse pattern in the model. Firstly, the class diagram from the selected misuse pattern is needed. And combine this with the tested integrate target system model which was built in the previous step.

3.2 Build the new model in USE. Similar to step 2.3, the new model also needs to be built in USE tool in order to be tested.

3.3 Set test cases according to the misuse requirements. Different from step 2.4, the test case of simulating the misuse on this model does not need to cover all the possible cases since here what is being testing is not the operation of the misuse. Therefore the test case should be set as near as possible to the ideal case to accomplish the goal in the misuse requirement. However, it does not mean that the test case can be created arbitrarily. It has to be under the control of the OCL constraints from security countermeasures that were built in step 2. Otherwise the USE outputs error for the constraint violations.

3.4 Execute test of the new model in USE. The whole test process should be ended if the result does not satisfy the misuse requirements. This indicates the target system is not vulnerable to the kind of chosen attack. Otherwise if the result satisfy the misuse requirement then the process should go further to step 4.

4. Vulnerability analysis

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, this step is executed for analyzing 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.

5. Offer new countermeasures

New countermeasures against the applied misuse are offered according to the vulnerabilities found in step 4.

6. Apply the new countermeasures and retest

Add this countermeasure into the model in step 2 and redo step 2 and 3. The failure of step 3 indicates the new countermeasures offered by step 5 are able to stop this attack. Otherwise redo step 4, 5 and 6 until effective countermeasures are found to stop this attack.

CHAPTER 4

EXPERIMENTS

Two practical experiments are shown in this chapter in order to demonstrate the integrated process of the proposed method in Figure 3-1.

4.1 Experiment 1:

For the first experiment, a Web server in the design stage is chosen as a victim system, which is employing credential based identification and authentication (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 different security situations. For the explicit expression of the process in this example, this test uses the combination of username and password as credentials in this experiment.

For attacking model, this test uses the principal spoofing misuse pattern [21] to simulate the attack on the victim system which have been mentioned above.

Step 1: Define Misuse and security requirements

Table 4-1 and Table 4-2 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 4-3 where the spoofing attack requires the possibility of accessing protected assets as an irregular user.

Table 4-1: Security requirements of I&A

		1	2
Conditions	Regular user	Yes	No
Actions	Allow to access assets that need I&A	X	
	Reject to access assets that need I&A		X

Table 4-2: Misuse requirements of principal spoofing in web services

		1	2
Conditions	Regular user	Yes	No
Actions	Allow to access assets that need I&A	X	X
	Reject to access assets that need I&A		

Step 2: Validate system current countermeasures

2.1: List the currently existing countermeasures.

According to the context of this case. There is only one countermeasures in the victim system model that is the credential based identification and authentication. To implement this countermeasure, the credential pattern is been used in this test.

Table 4-3: Security design requirements of credential pattern

		1	2
Conditions	With valid credential	Yes	No
Action	Considered as regular user	X	
	Considered as irregular user		X
	Allow to access assets that need I&A	X	
	Reject to access assets that need I&A		X

Table 4-3 shows the security design requirements of the credential pattern which is implemented on the target system. A bit more explanations about the differences between Table 4-1 and Table 4-3 seem to be necessary here. Table 4-1 is the requirements of I&A. It is well known that there are many different strategies to implement an I&A such as using voice recognition, facial recognition, finger print and so on. Credential based I&A is obviously a number of this huge family. The common requirements of any kind of I&A is in Table 4-1. Since here a credential based I&A is chosen in this test, to test whether this countermeasure is working appropriately, a specific security design requirements for this credential based I&A is needed.

2.2: Apply the chosen countermeasures in to the test model.

A credential based I&A as a countermeasure and a purely basic web server as the victim system model are available now. The testing model can be generated by applying the countermeasure model into the system model. See Figure 4-1.

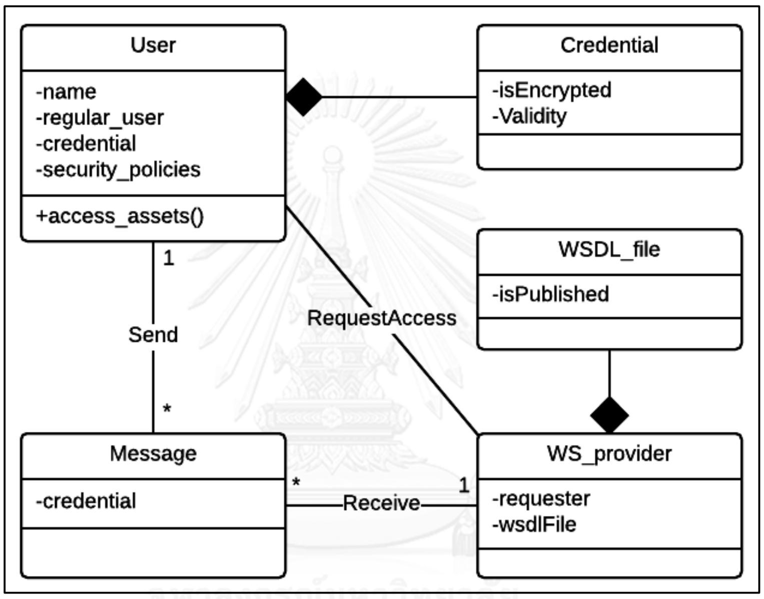


Figure 4-1: The test model with credential based I&A

2.3: Build the test model in USE

Building a model in USE is not simply drawing the UML in it. Coding is required for the tool to understand the model. The coding includes defining classes, their constraints in OCL and associations.

A part of the constructing code is shown in Figure 4-2. 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 WS_Provider
attributes
  requester:User
  wsdlFile:WSDL_File
constraints
  inv Authentication:
    if requester.credential.validity = true and
requester.securityPolicies = true then
      requester.regular_user = true
    else
      requester.regular_user = false
    endif
end
```

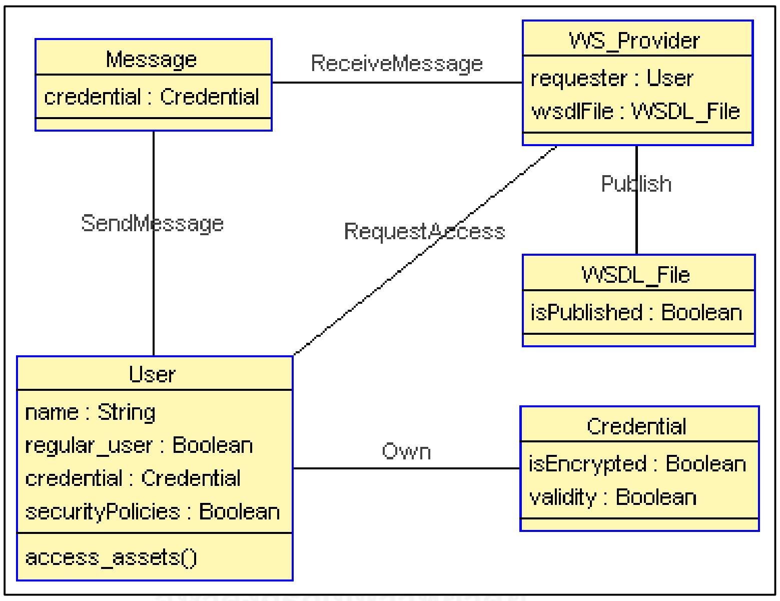
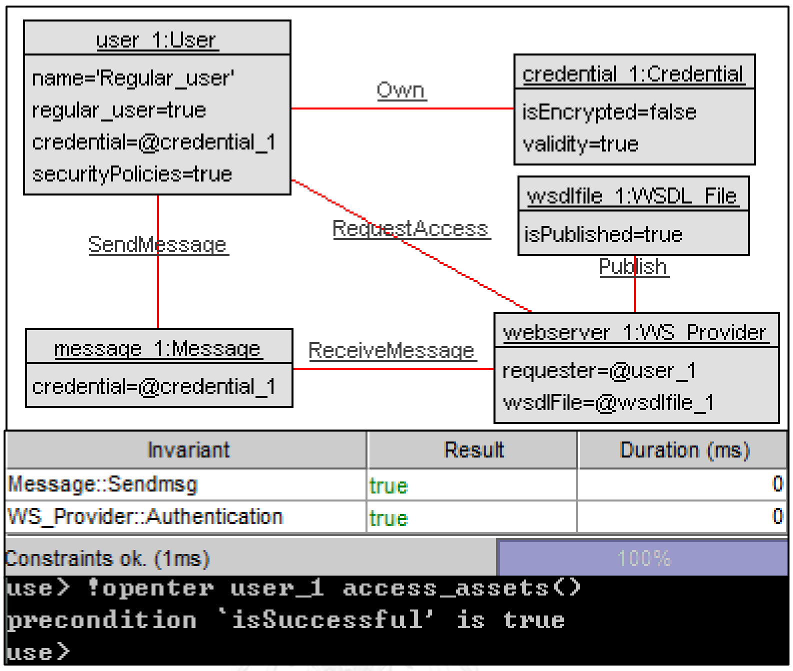


Figure 4-2: The credential structure in USE and its constraints

The class diagram of the model in USE which is shown in Figure 4-2. This is like the entire environment of a simulator is built up. The next step is to execute the tests in different cases according to the requirements in Table 4-3.

2.4 and 2.5: Set test cases according to the security design requirements and execute the tests.

In this environment, testers are able to create objects of the existing classes and can also assign values to the attributes in the objects. However, the values assigned to the objects have to satisfy the constraints in OCL, otherwise the OCL check will output errors. A case with errors means this case is not possibly exist. For example a case could be an irregular is rejected to login to the system or a regular user is allowed to login to the system but the case that an irregular is allowed to login to the system will result in errors because this case is not exist.



Invariant	Result	Duration (ms)
Message::Sendmsg	true	0
WS_Provider::Authentication	true	0

Constraints ok. (1ms) 100%

```
use> !openter user_1 access_assets()
precondition `isSuccessful' is true
use>
```

Figure 4-3: The example test results in USE interface

Figure 4-3 shows a test result of the case 1 in Table 4-3 as an example. All of the values are shown in the object diagram. The constraint checks are passed and the result in the command panel shows that the regular user is allowed to access the assets. This is an example of the test result shown in USE interface. To make the data and results more intuitional, the results will be performed in tables in the following tests.

Table 4-4: The USE test results of credential pattern

	case 1		case 2	
Attributes	**user_1**		**user_1**	
	name	Regular_user	name	Irregular_user
	regular_user	TRUE	regular_user	FALSE
	credential	credential_1	credential	credential_2
	securityPolicies	TRUE	securityPolicies	TRUE
	message_1		**message_1**	
	credential	credential_1	credential	credential_2
	credential_1		**credetial_2**	
	isEncrypted	FALSE	isEncrypted	FALSE
	validity	TRUE	validity	FALSE
	webserver_1		**webserver_1**	
	requester	user_1	requester	user_1
	wsdlFile	wsdlfile_1	wsdlFile	wsdlfile_1
	wsdlfle_1		**wsdlfle_1**	
	isPublished	TRUE	isPublished	TRUE
Allow to access assets that need I&A	Yes		No	

Table 4-4 shows the USE test results of the two conditions shown in Table 4-3. The USE test outputs true while the value of credential validity is “true”, and the test outputs false if the credential validity is “false.” The results satisfy the security design requirements in Table 4-3, hence, the pattern is applied appropriately. The satisfaction of security requirements in Table 4-1 indicates that this input model of countermeasure works appropriately and offers protection for sensitive assets. This means that only the users holding valid credentials can access the protected assets

Step 3: Apply and validate misuse pattern

In this step, testers need to change their perspective from a defender into an attacker. Table 4-2 has 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.

Table 4-5 shows the misuse design requirements of principal spoofing. The form looks like similar to the security design requirements in Table 4-3. However the desired result of this test is different from the countermeasure test. In countermeasure test, the result of each of the case has to be guaranteed that it is identical with the cases in security requirement. Use the previous test as an example. The results have to satisfy both of the cases, “Regular user can access” and “Irregular use cannot access” so that this countermeasure is considered working well. However, here the expected result is that the attacker successfully sniffing the regular user’s credential and using it to access assets as considered as a regular user. Not like in the security countermeasure tests, it is not necessary to care about the results of all of the cases. But only the cases which result in the success of accessing the assets that need I&A as shown in the red line in the table. If this case is satisfied, no matter what are the results of other three cases, the system assets is disclosed anyway.

Table 4-5: Misuse design requirements of principal spoofing

		1	2
Conditions	Successfully sniffed regular user’s credential	Yes	No
	Considered as regular user	X	
	Considered as irregular user		X
	Allow to access assets that need I&A	X	
	Reject to access assets that need I&A		X

3.1: Apply misuse pattern in the model

Similar to step 2.2, attacking model also need to be joined in to the test model. As a result, the principal spoofing misuse model is applied in to the test model.

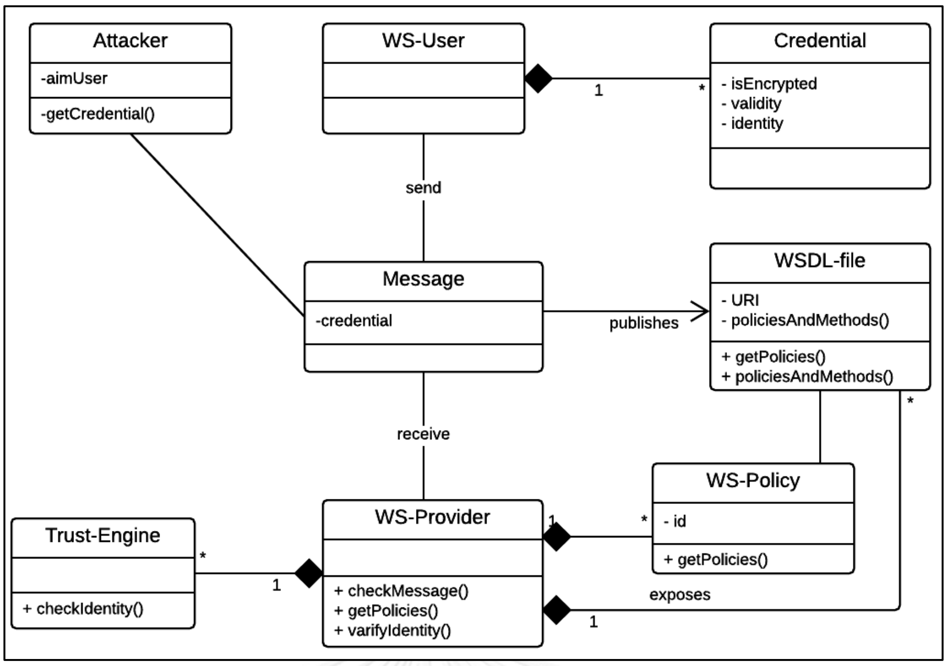


Figure 4-4: The test model with principal spoofing

Figure 4-4 shows the new test model which has been combined with principal spoofing attack model. The strategy of authentication 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 will be explained them in the subsequent step.

3.2: Build the new model with attack model in USE

In order to validate whether this misuse pattern satisfies the requirements in Table 4-2, testes need to build this new model into USE again. Some of the new constraints have to be introduced here because these constraints have the duty on judging the attributes of the new class – "Attacker". They are directly related to the test result.

```
constraints
  inv getCredential:
    if sniffingMessage.credential.isEncrypted =
false then
      credential = sniffingMessage.credential
    else
      credential.isUndefined
    endif
  inv getSecurityPolicies:
    if aimServer.wsdlFile.isPublished = true
then
      securityPolicies = true
    else
      securityPolicies = false
    endif

constraints
  inv Authentication:
    if requester.credential.validity = true and
requester.securityPolicies = true then
      requester.regular_user = true
    else
      requester.regular_user = false
    endif
```

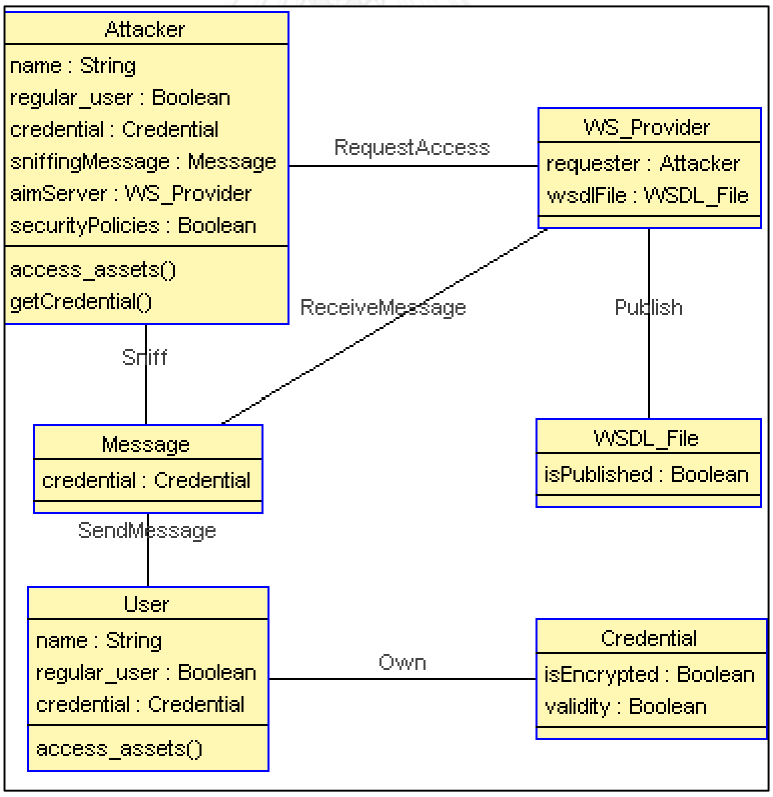


Figure 4-5: The principal spoofing model in USE and its constraints

The constraints are shown in Figure 4-5. The invariant “getCredential” and “getSecurityPolicies” 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 plus a valid credential.

3.3 and 3.4: Set the test case and execute the test in USE

The left part of table 4-6 contains a test case. This test case satisfies the misuse requirements in Table 4-5. 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, it is unknown that whether this ideal case satisfies the OCL described in Figure 4-5. Therefore, this case is applied in USE to check all the constraints. The “access_assets()” function then can be executed to see the result after the test case passed all the constraint checks.

Table 4-6: The USE test result of principal spoofing

Case 1		**Assets disclosure**
user_1		
name	Regular_user	
regular_user	TRUE	
credential	credential_1	
message_1		
credential	credential_1	
credential_1		
isEncrypted	FALSE	
validity	TRUE	
webserver_1		
requester	user_1	Yes
wsdlFile	wsdlfile_1	
wsdlfle_1		
isPublished	TRUE	
attacker_1		
name	Irregular_user	
regular_user	TRUE	
credential	credential_1	
victimUser	user_1	
aimServer	webserver_1	
securityPolicies	TRUE	

Table 4-6 also shows the test result from USE. This test has passed all the invariants check according to the constraint in Figure 4-5. 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 rights to access the protected assets. The protected assets are disclosed. Therefore, the test case finally satisfies the misuse requirements in Table 4-2. It indicates that the victim system in this example is vulnerable to the principal spoofing attack. The next step is to analyze the vulnerabilities.

Step 4: Analysis of the vulnerabilities

As 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 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.

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 it is not possible to 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 using the XML encryption pattern.

Step 6: Apply countermeasures and retest

XML encryption pattern has been introduced in Chapter 2.

To use the XML encryption pattern, firstly, the XML encryption pattern itself needs to be tested before being applied in the model with misuse. The function of XML encryption pattern is simple and clear. So the explicit test is not demonstrated here. But be noticed that the countermeasure security pattern test is indeed necessary and has been passed in this experiment. The security requirements of each countermeasures have to be satisfied before applying them with misuse pattern.

The system model is as shown in Figure 4-6 after applying the new countermeasure into the current system model.

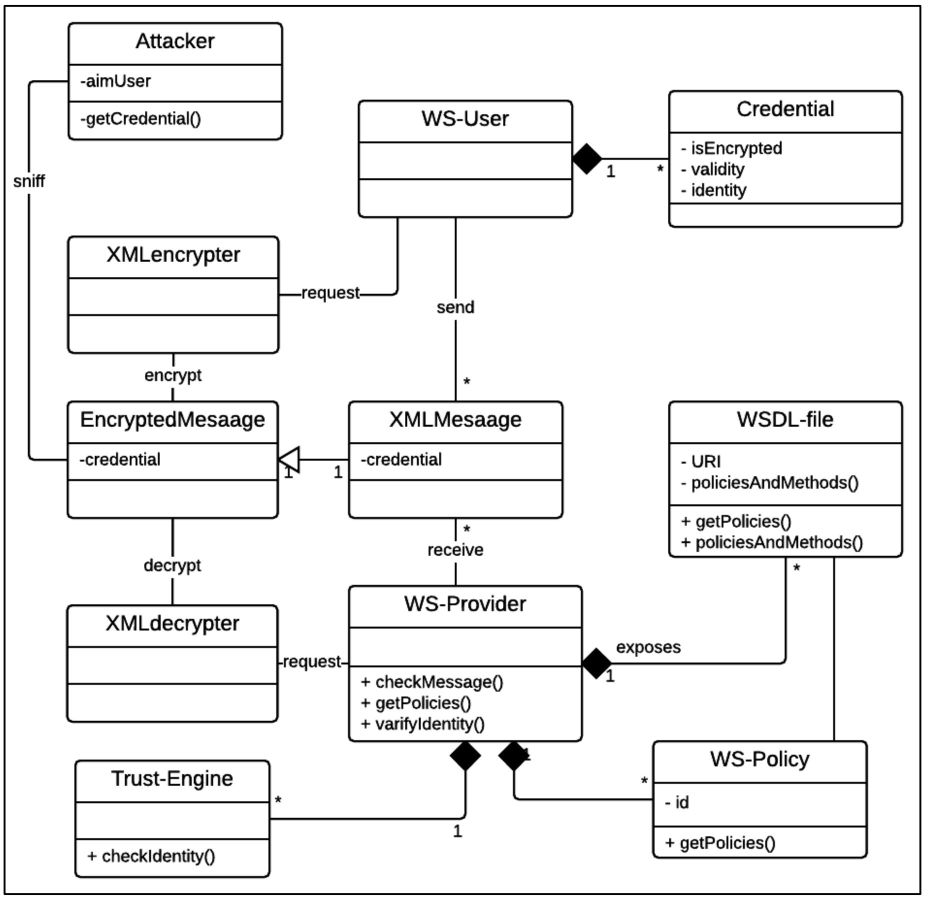


Figure 4-6: Entire system model with XML encryption countermeasure

Same as the tasks have been done in step 3. The class diagram of the entire system model with XML encryption and its constructing commands in USE tool are shown in Figure 4-7.

```
class Encryptor
attributes
  ecrMessage:Message
constraints
  inv encrypt:
    ecrMessage.isDefined implies
ecrMessage.credential.isEncrypted=true
end

class Message
attributes
  credential:Credential
constraints
    inv Sendmsg:
        sender.credential.isDefined implies
credential=sender.credential
end
```

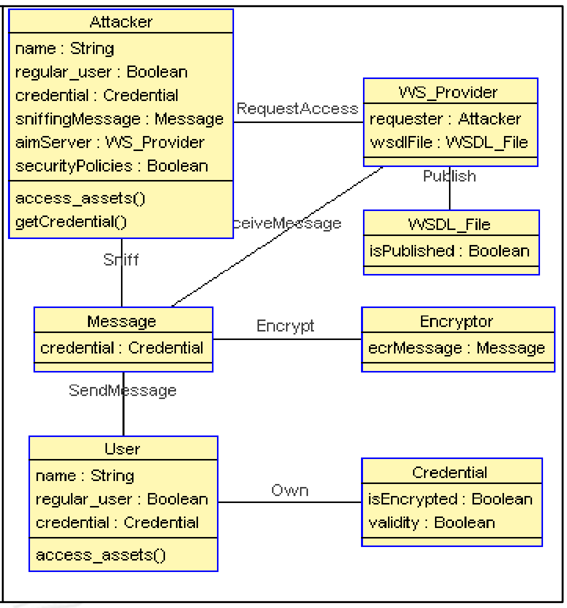
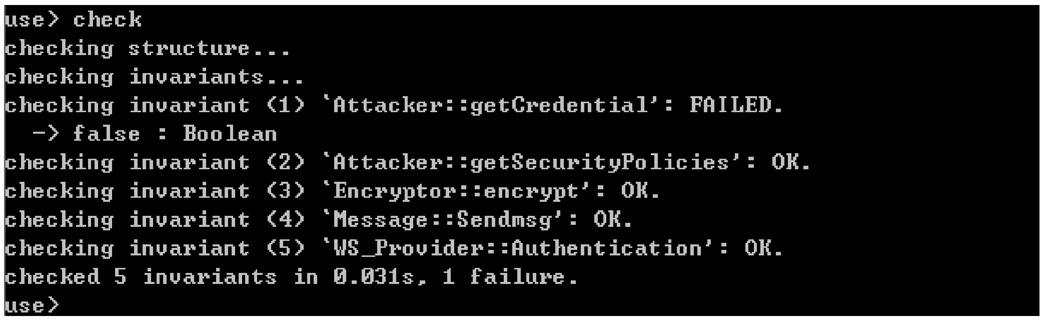


Figure 4-7: The entire system model with XML encryption and its constraints

Figure 4-7 shows the new model where the XML encryption pattern has been applied in the USE tool. The two new constraints in above take the responsibilities of the new forces brought by the countermeasure. Identically, the aim of the attacker here is still stealing the regular user's credential. As mentioned in step 5, analyzing the result does not consider the way of ordering the credential from the user personally. Therefore, the way that is considered here is sniffing from the communication between user and the system web server.

The new class message in the model carries the content and the sender's credential in order to communicate with the web server. And the encryptor encrypts the message before sending it.

The misuse requirements are the same as in Table 4-3 and Table 4-5 since the attacking method and pattern are not changed in this model. Now try to build the same test case in Table 4-6, which had made the misuse successful in step 3.

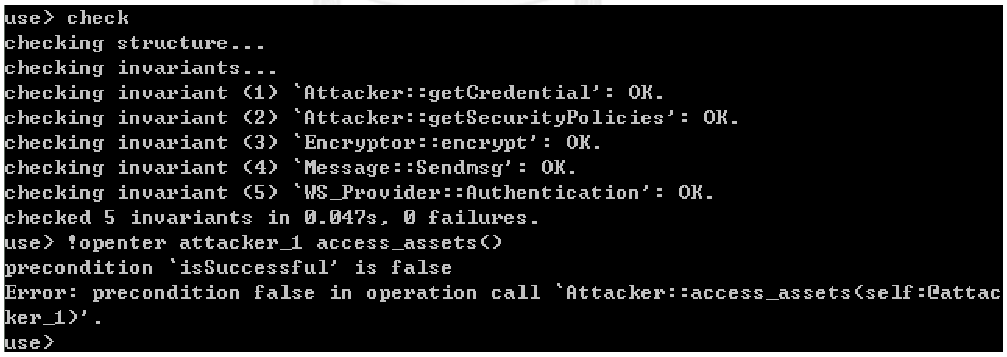


```
use> check
checking structure...
checking invariants...
checking invariant (1) `Attacker::getCredential': FAILED.
  -> false : Boolean
checking invariant (2) `Attacker::getSecurityPolicies': OK.
checking invariant (3) `Encryptor::encrypt': OK.
checking invariant (4) `Message::Sendmsg': OK.
checking invariant (5) `WS_Provider::Authentication': OK.
checked 5 invariants in 0.031s, 1 failure.
use>
```

Figure 4-8: Constraint check of principal spoofing test case 1

As seen in Figure 4-8, the test case has been set exactly the same as in Table 4-6 in step 3. However, the constraint check provides a different result. The constraint “getCredential” reports a “Failed” since the credential that the attacker intends to sniff is encrypted. This results indicates that the case 1 in Table 4-5 is no longer possible to occur. The reason is the credential of user_1 is encrypted so that the attacker now is not able to get it by sniffing. More accurately, the attacker still can sniff the message, but he cannot read the encrypted content in the message.

Therefore only one of the two cases showed in the misuse requirements in Table 4-3 is possible to be built and passed the constraint check. So then try the second test case.



```
use> check
checking structure...
checking invariants...
checking invariant (1) `Attacker::getCredential': OK.
checking invariant (2) `Attacker::getSecurityPolicies': OK.
checking invariant (3) `Encryptor::encrypt': OK.
checking invariant (4) `Message::Sendmsg': OK.
checking invariant (5) `WS_Provider::Authentication': OK.
checked 5 invariants in 0.047s, 0 failures.
use> !openter attacker_1 access_assets()
precondition `isSuccessful' is false
Error: precondition false in operation call `Attacker::access_assets(self:@attac
ker_1)'.
use>
```

Figure 4-9: Constraint check of principal spoofing test case 2

As shown in Figure 4-9, the second case of misuse requirements in Table 4-3 is successfully created and passed the constraint check. Which means this case may really occurs in the system.

However, the attacker is not able to access the protected assets under this case. Because it does not considered as a regular use to the system.

Hence, the misuse requirements cannot be satisfied under the protection from the new countermeasure XML encryption. Then it could be announced that the new model system is not vulnerable any more to the misuse “principal spoofing”. The risk from the threat has been mitigated and the vulnerability has been patched.

4.2 Experiment 2:

A vulnerability has been detected, analyzed and patched by implementing the entire processes of the proposed method in the previous experiment. The principal spoofing does not work on that system anymore. However, what if the system is in the circumstance of facing another kind of attack? Can those countermeasures handle the new type of attack too? The answer would be found in the second experiment. The patched system in the previous experiment is continually used as the victim system in this step.

The misuse pattern used in the second experiment is SQL injection misuse pattern [23].

Since the SQL injection misuse is intending to accomplish the same goal as the principal spoofing though a different way. The misuse requirements are the same as in Table 4-2. However the misuse design requirements are different which will be shown in later step.

All of the countermeasures in the current system model have been tested in the previous experiment so that the step 1 and 2 could be skipped. So this experiment starts from step3.

Step 3: Apply and validate misuse pattern

Table 4-6: Misuse design requirements of SQL injection

		1	2
Conditions	Enter and send malicious code to the web server	Yes	No
	Web server sends regular SQL statement to Database		X
	Web server sends injected SQL statement to Database	X	
	Database returns regular outputs		X
	Database discloses assets	X	

As usual, the first task is to list the misuse requirements of SQL injection attack. As what have been demonstrated in the previous experiment, the misuse model test considers only the case which contains attacking behaviors. Therefore, in Table 4-6, the only case need to be tested is case 1 since there is no attacking intention in case 2.

3.1 Apply misuse pattern in the model

The SQL injection has been concretely introduced in Chapter 2. Here is the result of the victim system model after applying SQL injection misuse model on it. Compare with the system model in Figure 4-6, since the encryption is not the key function in this test, this model has a simplified encryptor class which keeps the encryption function. The two new classes "Database" and "Query" are added in to the model.

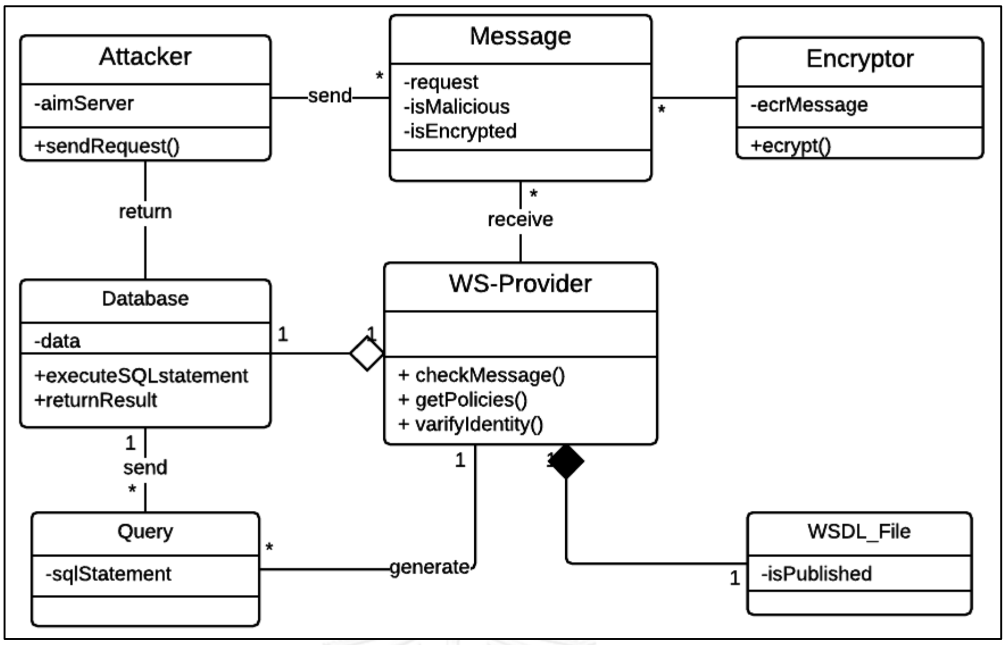


Figure 4-10: Entire system model with SQL injection

Similar to the first experiment, this step applies this misuse model to the victim system model. The entire combined model is shown in Figure 4-10. The query class could be generated by both the trust engine and the web-provider itself. This depends on the type of the query. I&A queries, for example, are from the trust engine. A regular search may be generated from the web-provider. The trust engine will be embedded into the web-provider.

3.2: Build the new model with SQL injection attack model in USE

There are obviously some new constrains are required to simulate the new situation of the SQL injection attack.

There are two new constraints need to be introduced.

The first is that if a request with malicious code is sent to the web server, then the web server will send an injected query to the database.

The second is that a data disclosure will occur if the database receives an injected query.

The constructing statements is not been shown here due to its length. However it can be found in the appendix.

The credential based I&A pattern and XML encryption pattern work the same as in the previous experiment. The request sent from attacker will be encrypted and the web server will check the requester's credential as normal. Since the web server needs to check the credential's validity by searching and matching it with its database. The web server has to send a query which is generated according to the content in request from requester (The requester is attacker here).

As a result, if the request contains malicious code, the query contains malicious SQL statement. The malicious SQL command causes the disclosures in SQL database. The operation "data_disclosure_check" is able to check the result of data disclosures.

Figure 4-11 shows the entire system model with SQL injection in USE.

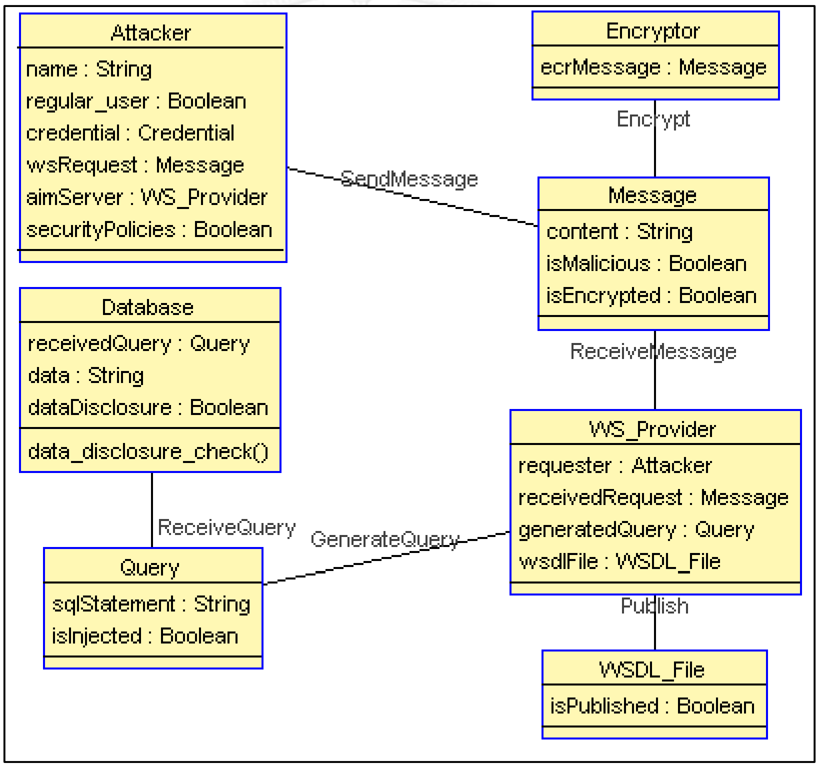


Figure 4-11: The entire system model with SQL injection in USE.

3.3 and 3.4: Set the test case and execute the test in USE

The test case in Table 4-7 is according to the case 1 in table 4-5. Obviously, this case contains attacking behaviors. This case has passed all the constraint checks in USE. Which means this case can really happen in the victim system. On the other hand, the test result in USE also shows that this case causes a data disclosure. Therefore, the attack contained by this case successfully made a misuse in the system.

Moving the focus on the attributes of each class in the object diagram. It is shown that the request from attacker is malicious so that the SQL query generated by the web server is injected. As a result, this case causes a data disclosure in the database. Therefore, this case satisfies the case 1 in Table 4-5.

This is the evidence to say that the SQL injection is implemented successfully in the victim system. In the other word, this system is vulnerable to the SQL injection misuse.

Table 4-7: SQL injection test case 1

	Case 1			
Attributes	**attacker_2**		**database_1**	
	name	SQL_injector	receivedQuery	query_1
	regular_user	FALSE	data	Sample_data
	credential	Undefined	dataDisclosure	TRUE
	wsRequest	request_1	**query_1**	
	aimServer	webserver_1	sqlStatement	SQL_sample
	securityPolicies	TRUE	injected	TRUE
	webserver_1		**Encryptor_1**	
	requester	attacker_2	ecrMessage	request_1
	receivedRequest	request_1	**request_1**	
	generatedQuery	query_1	content	malicious_code
	wsdlFile	wsdlfile_1	isMalicious	TRUE
	wsdlfile_1		isEncrypted	TRUE
	isPublished	TRUE		
data disclosure	Yes			

Step 4: Analysis of the vulnerabilities

It is impossible to prevent user entering and requesting SQL queries because input is necessary in various cases in a web sites such as the ID, password, searching key words and so on. The identification and authentication does not help on this situation since the process of I&A also requires SQL queries to check the user's credential. These queries could also be utilized by the attacker. For the XML encryption, it keeps the XML messages from sniffing and stealing but it does not provide any detection measures on the message's content.

Five classes are participated into the process of SQL injection. Attacker, Message, Web server, Query and Database. The attacker is obviously cannot be controlled. However, creating some strategy to control the requests sent from the attacker is feasible.

Step 5: Countermeasures

The inputs to the requests should be constrained in order to prevent malicious code in the requests which may cause injected SQL queries. All the input to your applications should be validated for type, length, format, and range. By constraining the input used in the data access queries [24]. For example, an input with an integer as its type must have an integer value passed through it. For the string type of input such as the name, address, need to be guaranteed that something like "'" should not be allowed in the input.

This methodology is called input validation. The next step is to add this countermeasure to the current model.

Step 6: Apply countermeasures and retest

The input validation countermeasure has been introduced in Chapter 2. This countermeasure will be applied in the victim system to mitigate the SQL injection attack.

Table 4-8: Security requirements of input validation

		1	2
Conditions	Authorized input in the request	Yes	No
Actions	Allow to generate query to the database	X	
	Reject to generate query to the database		X

Table 4-9: Security design requirements of input validation

		1	2
Conditions	No banned character is found in the input	Yes	No
Actions	Consider as authorized input	X	
	Consider as unauthorized input		X
	Allow to generate query to the database	X	
	Reject to generate query to the database		X

Table 4-8 and Table 4-9 illustrates the security requirements and security design requirements of input validation. The input from the user will be checked in the web server by the input validation model. Only the requests with authorized input are allowed to be translated into SQL query and further sent to the database. Otherwise the requests are rejected.

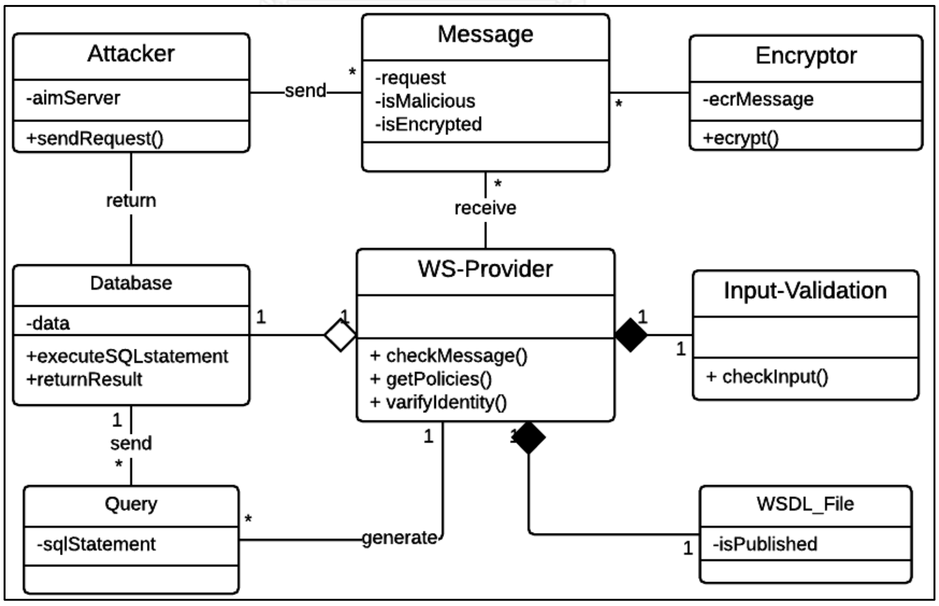


Figure 4-12: New system model with input validation

The input validation model is belong to a specific web server, which is commonly considered as a function however, here it is represented by an individual class in Figure 4-12 in order to make the structure clearer.

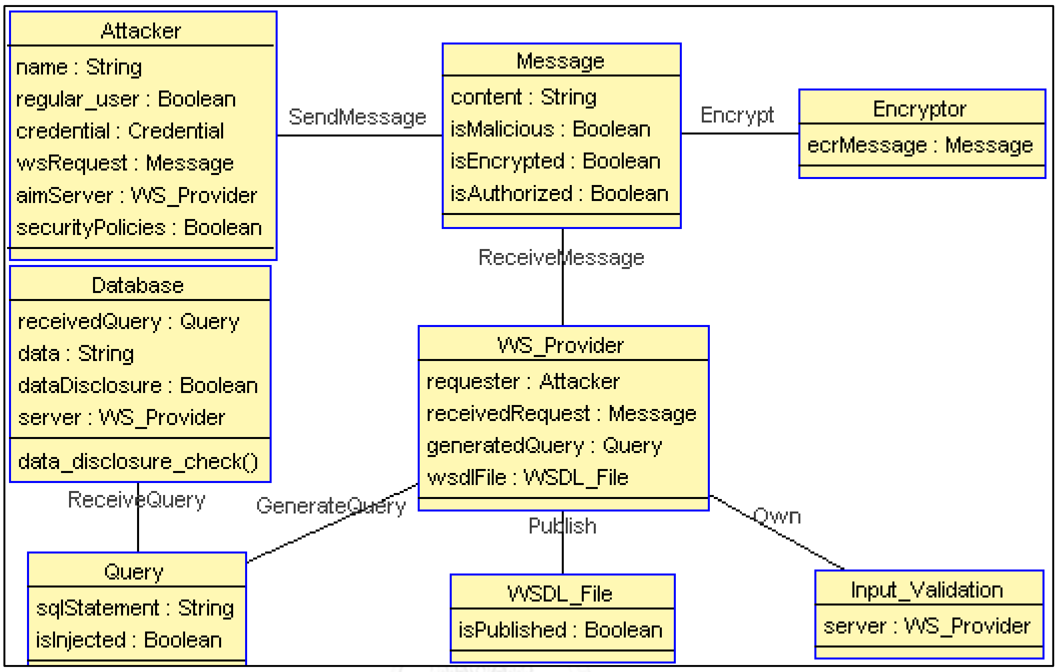


Figure 4-13: The entire system model with input validation in USE

Figure 4-13 shows the entire system model which has been built in USE. An additional constraint in this model is belong to the input validator. And a new attribute “isAuthorized” is added into the Message class. Only authorized messages will be translated to SQL queries and be sent to the database. The input validator determines the value of the attribute “isAuthorized” according to whether the requests contain irregular characters. If the request is not authorized, the web server will not create query and the database will not receive the query apparently. Also, the entire statements of building this model in USE could be found in appendix.

Since the input validation is a patched countermeasure. The SQL injection model is already exist in the testing system. Therefore the countermeasure and attack can be tested together.

The next step is to create the test cases in the USE. The case in Table 4-7 which successfully caused a data disclosure is no longer possible to generate here since it cannot pass the new constraint check from input validator.

Table 4-10: The USE test cases and results of input validation

	Case 1		Case 2	
	attacker_2		**auser_2**	
	name	SQL-injector	name	Regular
	regular_user	FALSE	regular_user	TRUE
	credential	Undefined	credential	credential_1
	wsRequest	request_1	wsRequest	request_1
	aimServer	webserver_1	aimServer	webserver_1
	securityPolicies	TRUE	securityPolicies	TRUE
	database_1		**database_1**	
	receivedQuery	Undefined	receivedQuery	query_1
	data	Sample_data	data	Sample_data
	dataDisclosure	FALSE	dataDisclosure	FALSE
	server	webserver_1	server	webserver_1
	query_1		**query_1**	
	sqlStatement	SQL_sample	sqlStatement	SQL_sample
	isinjected	TRUE	isinjected	FALSE
Attributes	**encrypter_1**		**encrypter_1**	
	ecrMessage	request_1	ecrMessage	request_1
	request_1		**request_1**	
	content	Malicious code	content	Malicious code
	isMalicious	TRUE	isMalicious	FALSE
	isEncrypted	TRUE	isEncrypted	TRUE
	isAuthorized	FALSE	isAuthorized	TRUE
	webserver_1		**webserver_1**	
	requester	attacker_2	requester	user_2
	receivedRequest	request_1	receivedRequest	request_1
	generatedQuery	Undefined	generatedQuery	query_1
	wsdlFile	wsdlfile_1	wsdlFile	wsdlfile_1
	wsdlfile_1		**wsdlfile_1**	
	isPublished	TRUE	isPublished	TRUE
	validator_1		**validator_1**	
	server	webserver_1	server	webserver_1
Allow generating query to the database	No		Yes	
Data disclosure	No			

The two possible cases are shown in Table 4-10. These two cases passed all of the constraint checks. The highlighted attributes in case 1 tells the differences from the case in Table 4-7 and it. The input validator detected that the request_1 contains malicious inputs. As a result, the web server did not generate a query, then apparently the database received nothing. Hence, the data disclosure did not happen in this case. This case satisfy the case 1 in the input validation requirements in Table 4-8.

The case 2 is completely a regular case. For the test on attacking purpose, the case 2 is not necessary since the case 1 and the failure of the case in Table 4-7 have shown that the system is no more vulnerable to SQL injection attack. However to fully satisfy the security requirements, case 2 is still need to be tested in order to make sure that the regular users can use this system normally under the protection of existing countermeasures. The result shows that the countermeasure is working appropriately in both of the cases. The security requirements are fully satisfied and the treats from SQL injection is eliminated.

CHAPTER 5

CONCLUSION

This chapter gives an introduction on the relationship among the three vulnerability detection techniques that have been mentioned in this thesis. Additionally, a comparison on these three techniques are also provided. The thesis ends up with the major contributions and future works about the proposed methods.

5.1 Model-based test, code-based test and penetration test

Before illustrating the relationship among those methods. Its better list the duties of each step in a SDL. Rather than the traditional software development life cycle (SDLC), it is more adequate to use SDL to describe the issues in this thesis. Using SDL does not mean the development process will conflict with SDLC. Each of the phase in SDL can be addressed in SDLC. It is just that SDL is more appropriate to demonstrate security related developments.

Table 5-1: Security development lifecycle

Training	1. Core Security Training		
Requirements	2. Establish Security Requirements	3. Create Quality Gates/Bug Bars	4. Perform Security and Privacy Risk Assessments
Design	5. Establish Design Requirements	6. Perform Attack Surface Analysis/ Reduction	7. Use Threat Modeling
Implementation	8. Use Approved Tools	9. Deprecate Unsafe Functions	10. Perform Static Analysis
Verification	11. Perform Dynamic Analysis	12. Perform Fuzz Testing	13. Conduct Attack Surface Review
Release	14. Create an Incident Response Plan	15. Conduct Final Security Review	16. Certify Release and Archive
Response	17. Execute Incident Response Plan		

It is much clearer to explain why it was mentioned in Chapter 1 that the code-based vulnerability checks are good complement of the model-based work with the detail of SDL in Table 5-1. It is because that the code review is located in the implementation phase while the model testing is in the design phase. Each of the phase has its own duty. It is inappropriate say one is more important than the other. It is just the design stage goes first so the implemental methods are considered a complement of model-based works. The penetration test is in the verification phase, so similarly, it could regard to be a complement of code-based tests.

Although the vulnerability analysis in different phases focuses on different types of vulnerabilities, they still have their own pros and cons that can be used to compare the proposed method with the others.

It is actually not adequate to compare the model testing method with code-based tests. Because they focus on different field. Quoting the example in Chapter 4, the model testing method tells the developer that your design is not secure and you should use input validation in your developing system. On the other hand, after maybe few months, the developers implemented the system into a real one. Then now they should use code-based test to test whether the input validation is working as they expected in the design stage.

Model testing a misuse pattern is more like a simulated penetration test. They both try to find the system vulnerabilities by attacking the system. The difference is that the model testing a misuse pattern uses a simulated attack in design stage while penetration test really implement real attacks on the target system. Theoretically, penetration test is able to find any kind of vulnerabilities in design, code and implementations while the proposed method is apparently only able to find design problems. It seems like penetration test is a lot more valuable than the model-based tests. But this is not the truth. A software or system development needs to go through all the phases in SDL in order to make a secure system. The developers cannot skip the design phase and leave all the vulnerability detections to the verification phase. Because fixing a vulnerability that is found in verification phase is very expensive especially the ones will cause a redesign. That is why both of those techniques are vital for secure development.

Compare with the other two techniques, there are some limitations on the proposed method. Since code review and penetration test are already widely used in the software development industry, so there are a lot of automated tools that can be used to implement those tests. Their databases are frequently updated so that they are able to detect even very new type of vulnerabilities. For our method, the limitation is that this method depends on the available misuse patterns. The category of misuse pattern is still limited. And this effect the usability of our method.

5.2 Contributions of proposed method

This thesis has proposed a method to analyze system security vulnerabilities using misuse pattern model testing approach in UML simulation environment. This method is able to preliminarily detect the system vulnerabilities under specific attacks during the design stage of the software development.

Apparently, it is impossible to uncover all of the vulnerabilities with this method since code-based test and penetration test are not possibly implemented during the design stage. However, the proposed method offers a new way of detecting the specific security vulnerabilities which are corresponded to a certain attack.

The major advantage of our method is the ability to detect security vulnerabilities in the design stage of software or system development. This allows for reduced the cost of fixing the vulnerabilities in the design stage rather than in the verification or maintenance stages in the development of a system.

The following questions that were given in Chapter 1 have been solved:

1. What is the method to simulate an attack on a design model with misuse pattern?

2. How to find out the vulnerabilities and corresponding countermeasures from the simulated attack during the design stage?

3. What is the process to validate the offered countermeasures?

The experiments concretely introduced the whole validating process. Which includes simulating an attack in a design model by modeling and testing the misuse pattern in USE tool. And also demonstrated the way of analyzing and patching the system vulnerabilities by using the constraints checks. Hence, the proposed problems are solved in this thesis. Subsequently, there are some tasks for us as the future works of this research.

5.3 Future works

The case study in this thesis simulated a simple model of a web server with two different types of misuses. It is sure that a lot more case studies are required to evaluate the usability and performance of a method. And also, as illustrated in the previous section, the number of available misuse patterns are limited. Therefore, In the future, this research will continue on proposing a method that can create simplified misuse patterns which only focus on the attacking model of real-life attacks. This will help breaking the limitation from the number of available misuse patterns and makes it possible to generate a lot more simulations according to the various attacking on the internet.

Although the proposed model testing approach is scoped to be used only on the vulnerability detection in this research. Model testing is also a common and efficient way on other design phase testing. There are reasons to believe that this testing method supported by USE could be also used in such as software bug detection, design efficiency test and so on in the future.

At last, conducting an evaluation will be possible once the sample size of the case study is enough.

REFERENCES

[1] Microsoft. “The security development lifecycle.” Available: www.microsoft.com/en-us/sdl/, 2015.

[2] OWASP. “Comprehensive Lightweight Application Security Process.” Available: https://www.owasp.org/index.php/Category:OWASP_CLASP_Project, 2015.

[3] E. B. Fernandez, N. Yoshioka and H. Washizaki. “Modeling misuse patterns.” *Availability, Reliability and Security (ARES)*, March 16-19, 2009, pp. 566-571.

[4] T. Kobashi, N. Yoshioka, T. Okubo, H. Kaiya, H. Washizaki and Y.Fukazawa. “Validating Security Design Pattern Applications Using Model Testing.” *Availability, Reliability and Security (ARES)*, September 2-6, 2013, pp. 62-71.

[5] J. Yoder and J. Barcalow. “Architectural Patterns for Enabling Application Security.” In N. Harrison, B. Foote and H. Rohnert, editors, *Pattern Languages of Program Design - 4*. Addison Wesley, 1999.

[6] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Bushmann, and P. Sommerlad. *Security Patterns: Integrating Security and Systems Engineering*. US: John Wiley and Sons, 2006.

[7] S. Konrad, Betty H.C. Cheng, Laura A. Campbell, and Ronald Wassermann. “Using patterns to understand and compare web services security products and standards.” *Telecommunications, AICT-ICIW*, February 19-25, 2006, pp. 157.

[8] J. Peleaz, E. B. Fernandez and M. M. Larrondo-Petrie. “Misuse Patterns in VoIP.” *Security and communication Network Journal*, vol. 2, pp. 635-653, 2009.

[9] E. B. Fernandez, N. Yoshioka and H. Washizaki. “A Worm Misuse Pattern.” Asian *Conference on Pattern Languages of Programs (Asian PLoP)*, March 16-17, 2010, Article No.: 2.

[10] P. Hope and P. White. “Software security requirements.” Available: http://sqgne.org/presentations/2007-08/, Sep. 12, 2007.

[11] G. Booch, J. Rumbaugh and I. Jacobson. *Unified Modeling Language User Guide*. US: Addison-Wesley, pp. 496, 2005.

[12] J. Rumbaugh, G. Booch and I. Jacobson. *The Unified Modeling Language 2.0 Reference Manual*, US: Addison-Wesley, 2003, pp. 367.

[13] M. Gogollaa, F. Buttner and M. Richtersb. “USE: A UML-based specification environment for validating UML and OCL.” *Science of Computer Programming*, 2007, vol. 69, pp. 27–34.

[14] Edward W. Felten, Dirk Balfanz, Drew Dean, and Dan S. Wallach. “Web Spoofing: An Internet CON Game.” *Software World*, March, 1997, Vol. 28, pp.6.

[15] Chris A. “Advanced SQL Injection In SQL Server Applications.” *NGS Software Insight Security Research (NISR)*, 2002.

[16] Keiko K., Fernandez, E.B. “Symmetric encryption and XML encryption patterns.” *Conference on Pattern Languages of Programs (PLoP)*, August 28-30, 2009, Article No.: 3.

[17] W3C. “XML Encryption Syntax and Processing.” Available: www.w3.org/TR/2002/REC-xmlenc-core-20021210/Overview.html, 2002.

[18] Tibor J., Juraj S. “How to break XML encryption.” *ACM Conference on Computer and Communications Security (CCS)*, October 17-21, 2011, pp. 413-422.

[19] OWASP. “Code Review Introduction.” Available: www.owasp.org/index.php/Code_Review_Introduction, Sep. 9, 2010.

[20] Kevin M. Henry. *Penetration Testing: Protecting Networks and Systems*, UK: IT Governance, 2012.

[21] J. Muñoz-Arteaga, H. Caudel-García and E. B. Fernandez. “Misuse Pattern: Spoofing Web Services.” *Asian Conference on Pattern Languages of Programs (Asian PLoP)*, October 5-8, 2011, Article No.: 11.

[22] Fernandez, E.B. Alder, R. Bagley, S. Paghdar. “A Misuse Pattern for Retrieving Data from a Database Using SQL Injection.” *BioMedical Computing (BioMedCom)*, December 14-16, 2012, pp 127-131.

[23] QWASP. “Input Validation Cheat Sheet.” Available: www.owasp.org/index.php/Input_Validation_Cheat_Sheet, Apr. 7, 2014.

APPENDIX

Construction statements of building system model with SQL injection in USE:

```
model PrincipalSpoofing

class Database
attributes
  receivedQuery:Query
  data:String
  dataDisclosure:Boolean
operations
  data_disclosure_check()
    pre isSuccessful: dataDisclosure
constraints
  inv disclosure:
    if receivedQuery.isInjected = true then
      dataDisclosure = true
    else
      dataDisclosure = false
    endif
end

class Query
attributes
  sqlStatement:String
  isInjected:Boolean
end

class Encryptor
attributes
  ecrMessage:Message
constraints
  inv encrypt:
    ecrMessage.isDefined                            implies
ecrMessage.isEncrypted=true
end

class Message
```

```
attributes
  content:String
  isMalicious:Boolean
  isEncrypted:Boolean
end

class Attacker
attributes
  name:String
  regular_user:Boolean
  credential:Credential
  wsRequest:Message
  aimServer:WS_Provider
  securityPolicies:Boolean
constraints
  inv getSecurityPolicies:
    if aimServer.wsdlFile.isPublished = true then
      securityPolicies = true
    else
      securityPolicies = false
    endif
end

class WS_Provider
attributes
  requester:Attacker
  receivedRequest:Message
  generatedQuery:Query
  wsdlFile:WSDL_File
constraints
  inv Authentication:
    if    requester.credential.validity    =    true    and
requester.securityPolicies = true then
      requester.regular_user = true
    else
      requester.regular_user = false
    endif
  inv CreateQuery:
    if receivedRequest.isMalicious = true then
      generatedQuery.isInjected = true
    else
      generatedQuery.isInjected = false
    endif
end

class WSDL_File
attributes
  isPublished:Boolean
end

class Credential
attributes
  validity:Boolean
end

association GenerateQuery
between
  WS_Provider[1] role generator
  Query[*] role query
end

association ReceiveQuery
between
  Query[*] role query
  Database[1] role receiver
end

association Publish
between
  WS_Provider[1] role webserver
  WSDL_File[1] role Policies
end
```

```
    association SendMessage
    between
      Attacker[1] role sender
      Message[*] role message
    end

    association ReceiveMessage
    between
      Message[*] role message
      WS_Provider[1] role receiver
    end

    association Encrypt
    between
      Encryptor[1] role encryptor
      Message[*] role message
End
```

Construction statements of building the system model with input validation in USE:

```
    model PrincipalSpoofing

    class Input_Validation
    attributes
      server:WS_Provider
    constraints
      inv inputValidation:
        if server.receivedRequest.isMalicious = true then
          server.receivedRequest.isAuthorized = false
        else
          server.receivedRequest.isAuthorized = true
        endif
    end
```

```
class Database
attributes
  receivedQuery:Query
  data:String
  dataDisclosure:Boolean
  server:WS_Provider
operations
  data_disclosure_check()
    pre isSuccessful: dataDisclosure
constraints
  inv receiveQuery:
    server.generatedQuery.isUndefined implies
    receivedQuery.isUndefined
  inv disclosure:
    if receivedQuery.isDefined and receivedQuery.isInjected
= true then
      dataDisclosure = true
    else
      dataDisclosure = false
    endif
end

class Query
attributes
  sqlStatement:String
  isInjected:Boolean
end

class Encryptor
attributes
  ecrMessage:Message
constraints
  inv encrypt:
    ecrMessage.isDefined                            implies
ecrMessage.isEncrypted=true
end
```

```
class Message
attributes
  content:String
  isMalicious:Boolean
  isEncrypted:Boolean
  isAuthorized:Boolean
end

class Attacker
attributes
  name:String
  regular_user:Boolean
  credential:Credential
  wsRequest:Message
  aimServer:WS_Provider
  securityPolicies:Boolean
constraints
  inv getSecurityPolicies:
    if aimServer.wsdlFile.isPublished = true then
      securityPolicies = true
    else
      securityPolicies = false
    endif
end

class WS_Provider
attributes
  requester:Attacker
  receivedRequest:Message
  generatedQuery:Query
  wsdlFile:WSDL_File
constraints
  inv Authentication:
    if    requester.credential.validity    =    true    and
requester.securityPolicies = true then
```

```
      requester.regular_user = true
    else
      requester.regular_user = false
    endif
  inv CreateQuery:
     if receivedRequest.isAuthorized = true then
       generatedQuery.isInjected = false
      else
        generatedQuery.isUndefined
    endif
end

class WSDL_File
attributes
  isPublished:Boolean
end

class Credential
attributes
  validity:Boolean
end

association GenerateQuery
between
  WS_Provider[1] role generator
  Query[*] role query
end

association ReceiveQuery
between
  Query[*] role query
  Database[1] role receiver
end

association Publish
between
```

```
  WS_Provider[1] role webserver
  WSDL_File[1] role Policies
end

association SendMessage
between
  Attacker[1] role sender
  Message[*] role message
end

association ReceiveMessage
between
  Message[*] role message
  WS_Provider[1] role receiver
end

association Encrypt
between
  Encryptor[1] role encryptor
  Message[*] role message
end

association Own
between
  WS_Provider[1] role webserver
  Input_Validation[1] role validator
end
```

Yuan Yifan: He 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.

Publication:

"Analysis of security vulnerabilities using misuse pattern testing approach." International Conference on Information Technology, 2015.

Accepted and published in Journal of Software, ISSN: 1796-217X, vol. 10, pp. 650-658, 2015.