TESEM: A Tool for Verifying Security Design Pattern Applications by Model Testing

Abstract—**Because software developers are not necessarily security experts, identifying potential threats and vulnerabilities in the early stage of the development process (e.g., the requirement- or design-phase) is insufficient. Even if these issues are addressed at an early stage, it does not guarantee that the final software product actually satisfies security requirements. To realize secure designs, we propose extended security patterns, which include requirement- and design-level patterns as well as a new model testing process. Our approach is implemented in a tool called TESEM (Test Driven Secure Modeling Tool), which supports pattern applications by creating a script to execute model testing automatically. During an early development stage, the developer specifies threats and vulnerabilities in the target system, and then TESEM verifies whether the security patterns are properly applied and assesses whether these vulnerabilities are resolved.**

Keywords-Component; Security Patterns; Model Testing; Test-Driven Development; UML;

I. INTRODUCTION

Due to the increased number of business services on open networks and distributed platforms, security has become a critical issue. Software must be supported by security measures [1], which are addressed in every phase of development from requirements engineering to deployment. However, threats and vulnerabilities within a system cannot be sufficiently identified during the early development stage. Due to the vast number of security concerns and the fact that not all software engineers are security specialists, creating software with adequate security measures is extremely difficult.

Patterns, which are reusable packages that incorporate expert knowledge, represent frequently recurring structures, behaviors, activities, processes, or "things" during the software development process. Many security patterns have been proposed to resolve security issues [1]. For example, Bschmann et al. proposed 25 design-level security patterns [2]. By referring to these patterns, a developer can efficiently realize software with high security level.

Although UML-based models are widely used for design, especially for model-driven software development, whether patterns are applied correctly is often not verified [1]. It is possible to apply a security pattern inappropriately. Additionally, properly applying a security pattern does not guarantee that threats and vulnerabilities are resolved. These issues may result in security damage. Thus, we propose an application to verify security patterns using model testing.

Our method confirms that security patterns are properly applied and assesses whether vulnerabilities are resolved. Although we have already suggested a conceptual approach to verify security design pattern applications [3], this approach does not involve tool support that a developer can implement for automatic verification. Moreover, we previously did not evaluate our approach. Consequently, we suggest a new verification tool that supports pattern applications and evaluate our method via experiments.

Our research aims to answer the following four Research Questions (RQs):

- **RQ1**: Do developers inappropriately apply patterns?
- **RQ2**: Can our method detect incorrect applications of specific security design patterns in a design model?
- **RQ3**: Can our method detect the presence of vulnerabilities identified at the requirement stage before and after applying patterns?
- **RQ4**: Does our method help developers realize secure?

Because a security pattern alone does not provide systematic guidelines with respect to applications, herein we formally extend existing security patterns using OCL expressions. Then we propose a new testing process to verify correct pattern applications and a tool called TESEM[1] (Test Driven Secure Modeling Tool) to support model testing automatically. Our method provides three major contributions:

- New extended security patterns using Object Constraint Language (OCL) expressions, which include requirement- and design-level patterns
- A new model-testing process based on Test-Driven Development (TDD) to verify appropriate pattern applications and the existence of vulnerabilities using these extended patterns
- A tool called TESEM that supports pattern applications by creating a script to execute model testing automatically

This paper is organized as follows. Section II describes the background and problems with security software development. Section III details our verification method and the architecture of TESEM. Section IV shows an example of the verification process using TESEM. Section V evaluates the RQs by applying our method and discussing its impact. Section VI describes the threats to validity. Finally, Section VII summarizes this paper.

II. BACKGROUND AND PROBLEMS

In this section, we overview common existing techniques for secure design.

A. Security Requirement Patterns (SRPs)

A security requirement pattern (SRP) is an existing technique to identify assets, threats, and countermeasures [4].

[1] Source code of TESEM is open to the public at [https://github.com/takanorioo/TESEM]. Moreover its online demo targeting EMS is available at [http://www31168ue.sakura.ne.jp/uml].

A security pattern is reusable as a security package and includes security knowledge, allowing software developers to design secure systems like a security expert. Various types of security patterns exist. For example, SRPs are used at the requirement level, while security design patterns (SDPs), which are described in Section C, are applied at the design stage level.

The "Structure" of a SRP uses the Misuse case with the Assets and Security Goal (MASG) model [5], which is an extension of the misuse case [6] that provides the structure of assets, threats, and countermeasures at the requirement level. The MASG model can model attackers, attacks, and countermeasures as well as normal users and their requirements.

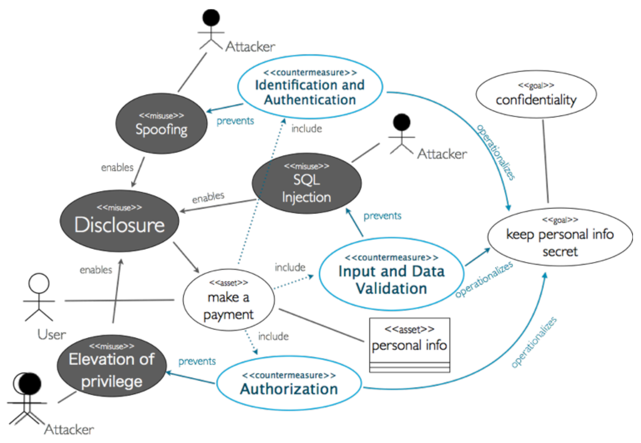


Fig. 1. Sample MASG model for a shopping website

Figure 1 shows a typical example of the MASG model for a partially modeled shopping website. The function "make a payment" has several assets, which could be threatened. In the model, "Disclosure" is a threat for "make a payment", while "personal information" is an asset. "Spoofing", "Elevation of privilege", and "SQL Injection" enable Disclosure. In addition, each countermeasure, such as "I&A (Identification and Authentication)", "Authorization", or "Input and Data Validation", effectively mitigates threats. Although the MASG model helps comprehensively detect security issues at the requirement level, it does not indicate whether the identified threats actually exist in the software system.

B. *Security Design Patterns (SDPs)*

SDPs are an established technique to satisfy security specifications. A SDP includes "Name", "Context", "Problem", "Solution", "Structure", "Dynamics", "Consequences", and "See Also". The pattern descriptions can be reused in multiple systems. As examples of SDPs, Bschmann et al. (2006) proposed 25 design-level security patterns [2].

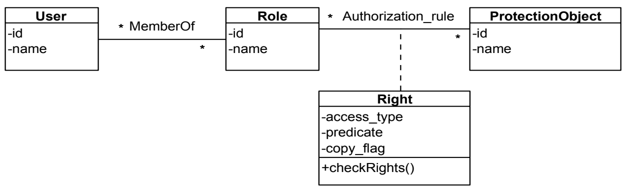


Fig. 2. Structure of RBAC

Figure 2 shows the structure of Role Based Access Control (RBAC) as an example of a SDP. The RBAC pattern, which is a representative pattern for access control, describes how to assign precise access rights to roles in an environment where access to computing resources must be controlled to preserve confidentiality as well as the availability requirements.

C. *Motivating example*

As an example of an applied pattern, Fig. 3 shows part of a UML class diagram that realizes a payment process on the Web.

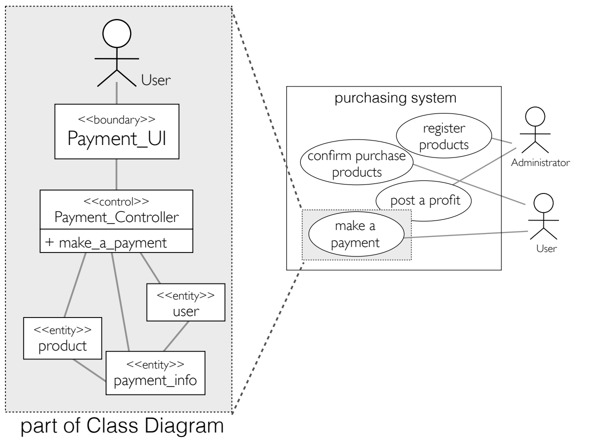


Fig. 3. Part of a class diagram for "make a payment"

A SDP alone cannot support the development lifecycle because it lacks systematic guidelines with respect to applications in the entire lifecycle [7]. Consequently, formally describing what rules must be verified is difficult [8][9]. In addition, most SDPs do not specifically mention systematic guidelines until the relations with Security Requirements are defined. Under these conditions, even if a developer intends to apply a SDP such as RBAC (Fig. 2) to the structural model (Fig. 3), it may be inappropriately applied to an identified threat. Additionally, the appropriateness of the applied pattern to the model and the pattern's ability to resolve vulnerabilities are often inadequately verified. These situations may cause incorrect pattern applications and unresolved vulnerabilities.

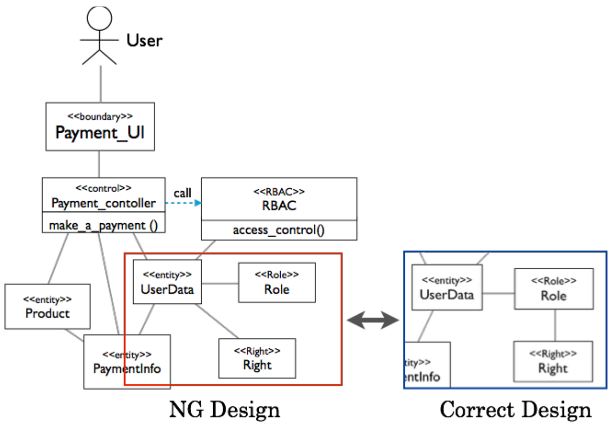


Fig. 4. Example of an inappropriate pattern application

Figure 4 shows an example of an inappropriate pattern application where RBAC is applied to the model shown in Fig. 3. Due to the lack of systematic guidelines with respect to pattern applications, a developer may apply the pattern inappropriately (e.g., like NG design in Fig. 4). The NG design implies that the access right depends on the user not on the role. Moreover, the appropriate functional behavior of the access control cannot be confirmed until the design model is tested. Thus, the applied measures may not mitigate or resolve the threats and vulnerabilities.

D. *Test-Driven Development (TDD)*

TDD is a software development technique that uses short development iterations based on prewritten test cases to define desired improvements or new functions. Here our testing process employs TDD, which requires that developers generate automated unit tests to define code requirements prior to writing the actual code [11]. The test case represents requirements that the program must satisfy [12].

Our method employs USE [13], which is a tool in the UML-based simulation environment that runs tests to specify and validate information systems based on subsets of UML and OCL [14]. OCL is a semiformal language that can express constraints for a variety of software artifacts as well as specify constraints and other expressions in modeling languages. USE was initially implemented in Java at Bremen University (Germany) to evaluate OCL expressions via simulations. To verify the OCL constraints, a developer creates an instance of a class in USE and then inputs a value as a test case.

Our method initially evaluates the OCL expressions that a design model should satisfy (Test First). TESEM generates these OCL expressions and a test script to verify whether these OCL expressions are satisfied. If the target model does not satisfy these OCLs, SDPs are applied, and the tests are re-executed to confirm that the vulnerabilities are resolved. The verification method also uses OCL expressions as the requirements.

III. VERIFICATION METHOD

This section explains our method. First, we show examples of new extended security patterns. Next, we explain the architecture and process of TESEM. Finally, as an example verification process, we apply our method to a purchasing system using these new extended security patterns.

A. *Extended SRPs and SDPs*

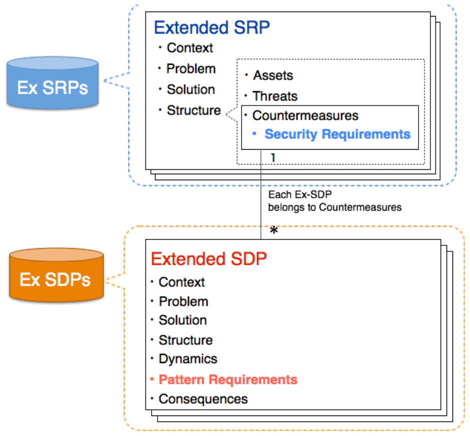


Fig. 5. Overview of Ex-SRPs and Ex-SDPs

The extended SRPs (Ex-SRPs) and extended SDPs (Ex-SDPs) are prepared beforehand. These new SRPs and SDPs are expansions of existing ones that can be used to verify whether the applied patterns are appropriate and to identify the presence of vulnerabilities in the target model. Figure 5 shows the overall structures of Ex-SRPs and Ex-SDPs. In addition to existing SRPs and SDPs, extended patterns contain **Security Requirements** and **Pattern Requirements**, respectively. These requirements are described using OCL expressions.

Security Requirements define the requirements that each countermeasure must satisfy. If a model does not satisfy the Security Requirements, then the measures do not remove the vulnerabilities and the system may contain threats. In TDD, these requirements represent test cases that must be satisfied. Herein we assume that there are nine types of countermeasures: “Input and Data Validation”, “Identification and Authentication”, “Authorization”, “Configuration Management”, ”Sensitive Data”, ”Session Management”, “Cryptography”, “Exception Management”, and “Auditing and Logging”. These countermeasures can be referenced in the Security Frame Category [15], which is Microsoft’s systematic categorization of threats and vulnerabilities. We assume that these nine categories are typical countermeasures at the requirement level because these categories represent the critical areas where security mistakes are most common.

Pattern Requirements describe the requirements that the applied pattern must satisfy. If a model does not satisfy the Pattern Requirements, it implies that the pattern is applied inappropriately.

B. *Architecture of TESEM*

Figure 6 overviews the system architecture of TESEM, which contains five major functional components. TESEM, which is Web service developed with PHP and JavaScript, is about 12-k lines of code. To manage data, we use MySQL. Below each major component is briefly described.

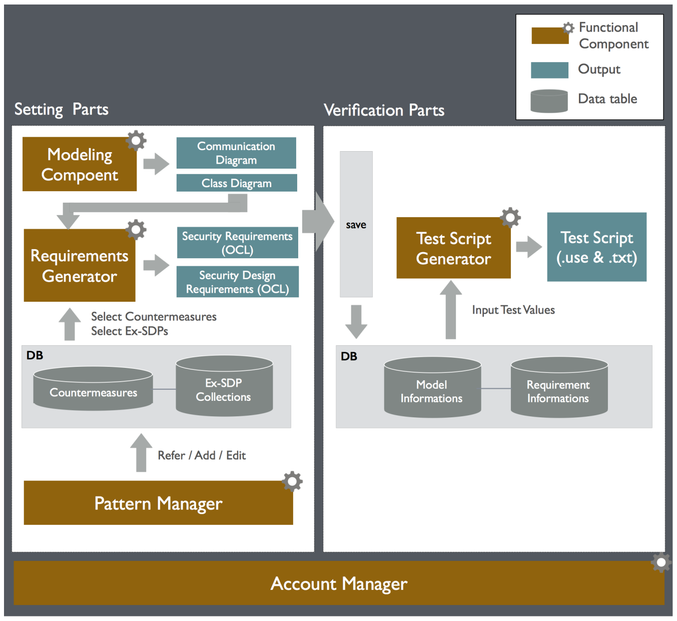


Fig. 6. System overview of TESEM

Modeling Component: TESEM has the function of a UML diagramming application. It supports class and communication diagrams. In the user interface, a user can add, edit, and delete class elements as well as describe relations between elements. Figures 7 and 8 show screenshots for a design target application.

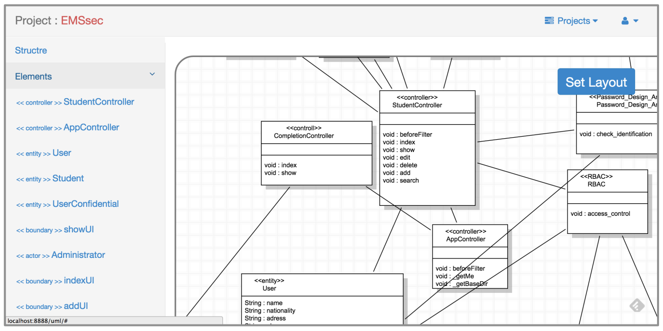


Fig. 7. Screenshot when creating a class diagram

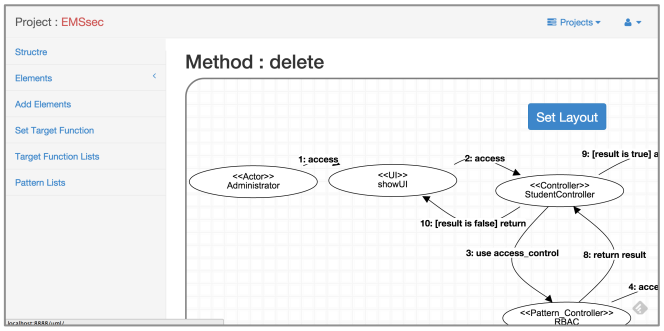
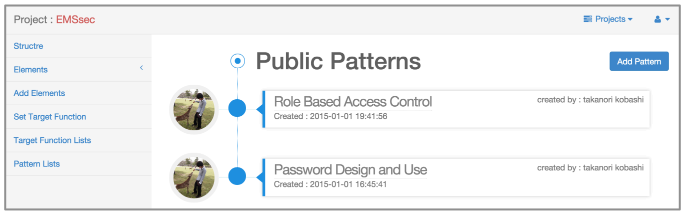


Fig. 8. Screenshot when creating a communication diagram

Requirements Generator: In this generator, the main outputs are Security Requirements and Security Design Requirements. Security Design Requirements are combinations of each Pattern Requirement. By selecting countermeasures for threats and the Ex-SDP related to the countermeasures, this component generates requirements that the target model must satisfy. TESEM generates these requirements as test cases.

Test Script Generator: To verify the test cases generated by the Requirements Generator, TESEM generates test script in a form that USE can execute test. To create this script, TESEM require concrete value as a test case.

Pattern Manager: This manages the countermeasure data and Ex-SDPs. Specifically, the structures and behaviors of patterns, the Security Requirements of each countermeasure, and the Patten Requirements of each Ex-SDPs are managed. Additionally, a user can submit new Ex-SDPs and share patterns. Figure 9 shows a screenshot of the Pattern Manager's index page where two Ex-SDPs ("Role Based Access Control" and "Password Design and Use") are submitted as public patterns.



Role Based Access Control
Created : 2015-01-01 19:41:56
created by : takanori kobashi

Password Design and Use
Created : 2015-01-01 16:45:41
created by : takanori kobashi

Fig. 9. Screenshot of the Pattern Manger's index page

Account Manager: TESEM manages user accounts and has sign-up and sign-in functions. Moreover, each user can create a new private or public project. Therefore, TESEM can create any number of models per user.

C. Process of our method using TESEM

Our method involves the following seven steps:

1. Create class and communication diagrams as a target system in the Modeling Component. TESEM uses a XML parser, so it also can import XMI files, including model information.
2. Identify threats and countermeasures in the system. Ex-SRPs identify the types of assets, threats, and countermeasures present in the developing software while considering the functional requirements and determining their associations at the requirement level.
3. Classify the type of countermeasures. These countermeasures involving Security Requirements are registered in advance. Then the Requirement Generator creates the Security Requirements that the target model must satisfy.
4. The Test Script Generator creates the test script to verify whether the input model satisfies the Security Requirements. Then the test script is used to evaluate these requirements in USE.
5. After confirming that the target model does not satisfy the Security Requirements, Ex-SDPs related to the "countermeasures" of Ex-SRP are selected. Then the Requirements Generator creates Security Design Requirements that the target model must satisfy. Security Design Requirements are combinations of the Pattern Requirements.
6. The structure and behavior of Ex-SDPs are applied to the input model by binding pattern elements based on stereotypes in the Modeling Component.
7. The Test Script Generator creates a test script to verify whether the model in which patterns are applied satisfies the Security Design Requirements. This test script is used to evaluate these requirements in USE.

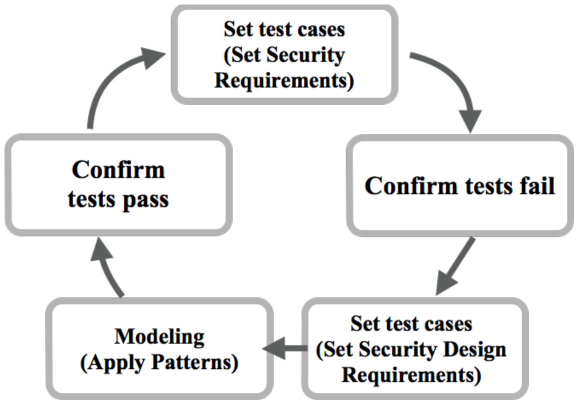

Set test cases (Set Security Requirements)
Confirm tests fail
Set test cases (Set Security Design Requirements)
Modeling (Apply Patterns)
Confirm tests pass

Fig. 10. Testing process of our method (conceptual)

Figure 10 shows the conceptual testing process of our method, which is based on TDD. Generally, TDD is used at the code level. However, our testing process employs TDD at the design level.

D. Example of the Verification Process using TESEM

To confirm that our method realizes a secure design, here we applied it to a purchasing system on the Web as an example verification process. Figure 1 shows the assumed assets, threats, and countermeasures in the MASG model.

STEP 1) Design a target application with UML notation.

As a case study, we designed a model that does not consider security (Fig. 11). Table 4 explains each element in this model. This system does not have a function to verify the condition to execute the "make a payment" process. In other words, even if the user is not a regular user, the process can be executed.

STEP 2) Identify threats and countermeasures in the system.

"I&A", "Input and Data Validation", and "Authorization" were selected countermeasures for "Spoofing", "Elevation of Privilege", and "SQL Injection" in the "make a payment" process, respectively (Fig. 1). For simplicity, each threat has one countermeasure.

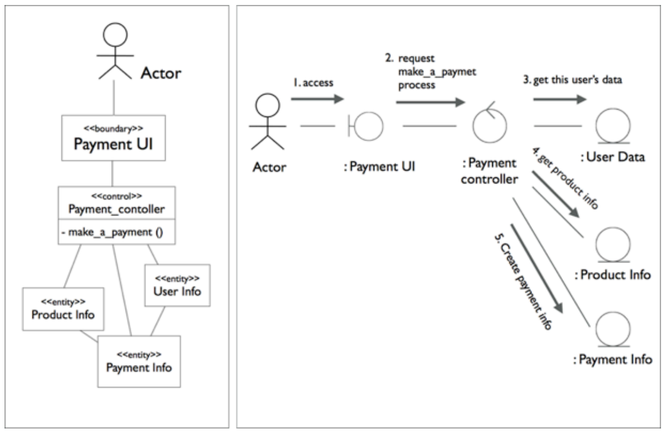


Fig. 11. Model that does not consider security

STEP 3) Select countermesures and generate Security Requirements.

Countermeasures are selected from the nine types. Here we selected the three identified in step 2. Then the Requirements Generator of TESEM creates the Security Requirements that the target model in Fig. 11 must satisfy. Table 1 and List 1 show the Security Requirements for the "make a payment" process, which include "actor is a regular user", "actor has access permission", and "valid data is inputted". If these requirements, which are a combination of "I&A", "Input and Data Validation", and "Authorization", are met, then the actor can execute the "make a payment process". These requirements represent the test cases in the TDD process.

TABLE I. Security Requirements for the "make a payment" process (conceptual)

		1	2	3	4	5	6	7	8
Conditions	<<Actor>> is a regular user	Yes	Yes	Yes	Yes	No	No	No	No
	<<Actor>> has access right	Yes	Yes	No	No	Yes	Yes	No	No
	valid data is inputted in <<UI>>	Yes	No	Yes	No	Yes	No	Yes	No
Actions	execute "make a payment" process	×							
	not execute "make a payment" process		×	×	×	×	×	×	×

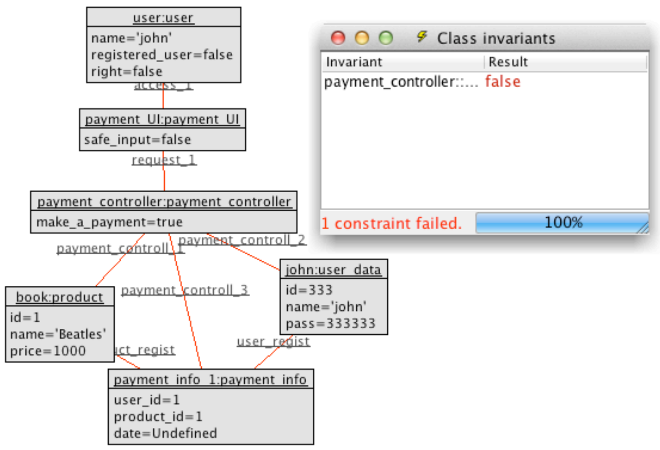
```
context payment_controller
  inv SecurityRequirements :
  if self.payment_UI.User.regular_user = true and
          self.payment_UI.User.right = true and
          self.payment_UI.valid_input_data = true
  then
          self.make_a_payment = true
  else
          self.make_a_payment = false
  endif
```

List. 1. Security Requirements for the "make a payment" process (OCL)

STEP 4) Execute a test to verify that the input model satisfies the Security Requirements.

Next we executed a model test to determine whether the input model that does not consider security satisfies the Security Requirements in List 1 (i.e., we verified whether each of test cases 1 – 8 behaves according to the expected action in Table 1). The Test Script Generator of TESEM creates test scripts to check if each test case is satisfied. These test scripts can be executed in USE.

Figure 12 shows a case where the "regular user", "has access permission", and "uses valid input data" are all "false" (Table 1, test case 8). Because the input model lacks object constraints, a false actor may carry out "make_a_payment = true" (i.e., an actor can execute the "make a payment" process without being a regular user or permission). Hence, the input model not considering security does not satisfy the Security Requirements of the "make a payment" process, and the OCL evaluation in USE becomes "false" in Fig. 12.



Invariant	Result
payment_controller::...	false

1 constraint failed.

Fig. 12. Conditions of the Security Test in USE

Table 2 shows the results of the eight test cases. Only case 1 satisfies the Security Requirements in Table 1, confirming the necessity of countermeasures "I&A", "Authorization", and "Input and Data Validation".

TABLE II. Results of the Security Test

		1	2	3	4	5	6	7	8
Conditions	<<Actor>> is regular user	Yes	Yes	Yes	Yes	No	No	No	No
	<<Actor>> has access right	Yes	Yes	No	No	Yes	Yes	No	No
	valid data is inputed in <<UI>>	Yes	No	Yes	No	Yes	No	Yes	No
Actions	execute "make a payment" process	×	×	×	×	×	×	×	×
	not execute "make a payment" process								

STEP 5) Select Ex-SDPs and generate Security Design Requirements.

We selected Ex-SDP related to the countermeasures of Ex-SRP, and added these to the structure to realize security capabilities. Specifically, Password design and Use, RBAC, and Prevent SQL Injection were employed for "I&A", "Authorization", and "Input and Data Validation", respectively. Table 3 and List 2 show the combinations of each Pattern Requirement necessary for the "make a payment" process, which is referred to as "Security Design Requirements".

STEP 6) Apply Ex-SDPs and generate Security Design Requirements.

We applied the above Ex-SDPs. During the pattern application, pattern elements are bound to a stereotype in TESEM. Figure 13 shows the structure after applying the patterns to the model. Hence, this model considers security. Compared to the model in Fig. 11, several conditions are necessary to execute the "make a payment" process (Table 3).

TABLE III. Security Design Requirements of the "make a payment" process (conceptual)

		1	2	3	4	5	6	7	8
Conditions	the same ID and Password that are inputted into <<Login_UI>> exist in <<User_Data>> respectively	Yes	Yes	Yes	Yes	No	No	No	No
	rights are given in <<Role>> which an <<User_Data>> belongs	Yes	Yes	No	No	Yes	Yes	No	No
	valid data is inputed in <<UI>>	Yes	No	Yes	No	Yes	No	Yes	No
Actions	<<Actor>> is considered regular user	×	×	×	×				
	<<Actor>> is consider non-regular user					×	×	×	×
	considers that <<Actor>> have access permission	×	×			×	×		
	consider that <<Actor>> does not have access permission			×	×			×	×
	consider that valid data is inputed	×		×		×		×	
	consider that invalid data is inputed		×		×		×		×
	execute "make a payment" process	×							
	not execute "make a payment" process		×	×	×	×	×	×	×

```
context payment_controller
  inv check_id_and_pass:
  if self.password_design_and_use.User_Data->exists(p |
      p.id = self.password_design_and_use.Login_UI.id and
      p.pass = self.password_design_and_use.Login_UI.pass)
  then
      self.Payment_UI.actor.regular_user = true
  else
      self.Payment_UI.actor.regular_user = false
  endif

context payment_controller
  inv access_control:
  if self.RBAC.Right->exists(p |
      p.right = true and
      p.role_id = p.Role.id and
      p.role_id = p.Role.User_Data.role_id )
  then
      self.Payment_UI.actor.right = true
  else
      self.Payment_UI.actor.right = false
  endif

context payment_controller
  inv sanitize_input_data_payment_UI:
  if self.Payment_UI.Prevent_SQL_Injection.sanitize_input_data =
true
  then
      self.Payment_UI.valid_input_data = true
  else
      self.Payment_UI.valid_input_data = false
  endif

context payment_controller
  inv sanitize_input_data_login_UI:
  if
self.password_design_and_use.Login_UI.Prevent_SQL_Injection.sani
tize_input_data = true
  then
       self.password_design_and_use.Login_UI.valid_input_data = true
  else
       self.password_design_and_use.Login_UI.valid_input_data = false
  endif

context payment_controller
  inv security design requirement:
  if self.Payment_UI.actor.regular_user = true and
```

```
        self.Payment_UI.actor.right = true and
        self.Payment_UI.valid_input_data = true and
        self.password_design_and_use.Login_UI.valid_input_data = true
    then
        self.make_a_payment = true
    else
        self.make_a_payment = false
    endif
```

List. 2 Security Design Requirements of the "make a payment" process (OCL)

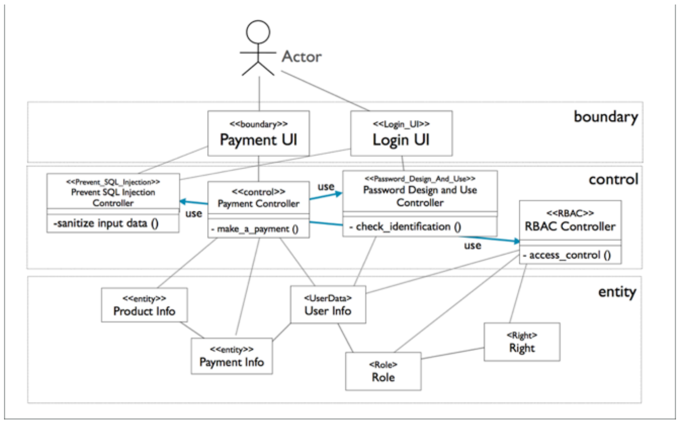
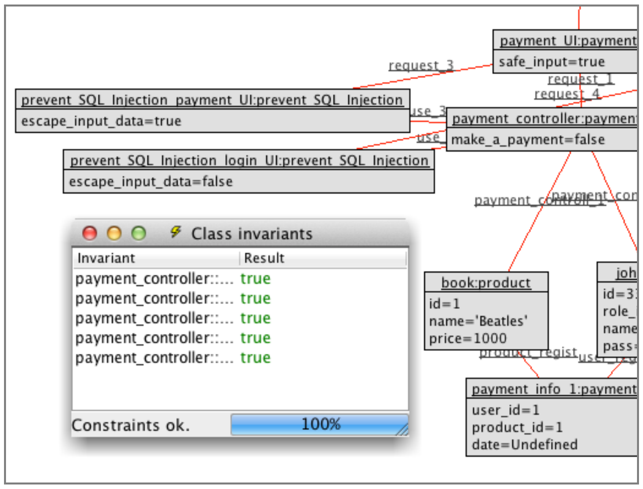


Fig. 13. Model-applied patterns (structure)

STEP 7) Execute a test to verify that the input model satisfies the Security Design Requirements.

To verify whether the patterns are applied appropriately to the "make a payment" process, we must confirm that the Security Design Requirements, which are combinations of each Pattern Requirement, are satisfied. We executed tests to confirm that the model in Fig. 13 satisfies the Security Design Requirements. Specifically, we confirmed that test cases 1 – 8 behave as expected (Table 3) using the Test Script Generator of TESEM to creates test scripts that can be executed in USE.



Invariant	Result
payment_controller::...	true
payment_controller::...	true
payment_controller::...	true
payment_controller::...	true
payment_controller::...	true

Fig. 14. Conditions of the Security Design Test in USE

Figure 14 shows the conditions of the Security Design Test in USE for a case where access permission is not given for the "Role" of the actor and the system does not sanitize the inputted data in "Login UI" (Table 3, test case 4). Prior to applying patterns, USE outputs "make_a_payment = true" (i.e., an actor without permission or inputting invalid data can execute the "make a payment" procces). However, after the patterns are applied, USE outputs "make a payment = false", and the actor cannot execute the "make_a_payment" process because access permission is not specified in the "Role" and the system assumes invalid data is used in "Login UI". By executing all the test cases, we confirmed that the output model-applied pattern satisfies the Security Design Requirements of the "make a payment" process.

To summarize, we applied Ex-SDPs for the "make_a_payment" process, which requires "I&A", "Input Data and Validation", and "Authorization", and executed a model test. Our verification process using TESEM confirms the appropriate application of security design patterns and the existence of vulnerabilities to threats identified at a requirements stage.

IV. Limitations

Our method has a few limitations. Because test cases are created based on threats and countermeasures identified in the requirement stage, the presence of threats not identified at this stage cannot be detected. In addition, the criterion for selecting Ex-SDP may be impractical because the range is influenced by the security policy, platform, and risk analysis.

V. Evaluation and Discussion

A. *Experimental Overview*

To evaluate the Research Questions, we conducted experiments involving ten students majoring in information sciences at Waseda University in Japan. So that the students were able to apply our method, we prepared a student information management system called EMS (Enrollment Management system) [16]. EMS is an actual web application used to evaluate security and privacy methodologies that was designed and analyzed in collaboration with IT companies and academic research institutes. This system involves typical software vulnerabilities such as SQL injection or XSS. The number of use case and class of this system are 24 and 31 respectively.

Students were given the use cases, model (class and communication diagram), and threats of this system. In the experiments, we considered the "delete function" of the "Student Controller" as a use case involving two threats ("Elevation of privilege" and "SQL Injection"). The experiment included the following:

- Exercise 1: Students realized a secure design to mitigate two threats ("Elevation of privilege" and "SQL Injection"), without referring to anything in particular.
- Exercise 2: Students realized a secure design to mitigate two threats while referring to a security design pattern. In this exercise, we instructed that two security design patterns ("RBAC" and "Prevent SQL Injection") be used.
- Exercise 3: Students realized a secure design to mitigate two threats using our method. Specifically, they set the Security Requirements and Security Design Requirements for the design model created in Exercise 2. Then they remodeled

while verifying these requirements were satisfied. In this exercise, we instructed that TESEM be employed initially.

B. *Experimental Results and Discussion*

1) *Exercise 1*

- In this exercise, seven of the ten students were unable to show a concrete design policy at all. Although the other three designed to mitigate threats, their design models differ, demonstrating that developers who are not security design specialists have difficulty realizing secure designs without referring to anything. Moreover, the design policy depends on the individual skills strongly, even if developers are familiar with security.

2) *Exercise 2*

- By referring a security design pattern, all students adopted the same design policy, and the attribute and method names are standardized. Additionally, compared to exercise 1, the average time decreased by 10 minutes. These results mean that referencing security patterns improves the design quality and affects the development time. However, eight of the students applied patterns inappropriately. Examples of incorrect pattern application include “lack of associations”, “insufficient class entities”, and “wrong coordination between patterns”. This result answers **RQ1** affirmatively; developers do inappropriately apply patterns.

3) *Exercise 3*

- In this exercise, the TESEM outputted “false” to eight incorrect models, which were created in Exercise 2 because the Security Requirements and Security Design Requirements are not satisfied. Exercise 3 confirms that our method can detect an incorrect application of each security design pattern by verifying each Pattern Requirements. Moreover, our method can detect the presence of vulnerabilities by verifying Security Requirements. Thus, this exercise answers **RQ2** and **RQ3** affirmatively; our method detects incorrect applications of specific security design patterns and the presence of vulnerabilities.

- In response to “false”, all students modified their models and two of the eight students realized “true” by themselves. Although measures that allow developers to realize correct model are necessary, detecting an incorrect application and the presence of vulnerabilities helps developers to improve their design models. Hence, **RQ4** is answered.

VI. THREATS TO VALIDITY

We did not verify whether our method is applicable to any type of system. Therefore, the case study results cannot be generalized. Additionally, the numbers of security patterns and testers were insufficient. Hence, it is possible that our method is not applicable to all security patterns. Although we used representative patterns and a typical model for software development to demonstrate the usefulness of our method, we need to examine more general patterns and employ large-scale examples.

VII. CONCLUSION AND FUTURE WORK

Non-expert software developers may inappropriately apply patterns, and even if the patterns are properly applied, threats and vulnerabilities may not be mitigated. Herein we propose a verification method for a security design pattern using a model test in the UML model simulation environment. Specifically, assets, threats, and countermeasures are identified in the target system during an early stage of development. We verified both the appropriateness of the applied patterns and the existence of vulnerabilities in the first stage of the design model.

This method offers three significant contributions. First, Ex-SRP and Ex-SDP, which are new extended security patterns using OCL expressions, include requirement- and design-level patterns. Second, a new model-testing process based on TDD verifies correct pattern applications and the existence of vulnerabilities. Finally, a tool called TESESM, which supports pattern applications, automatically generates script to test the model. In the future, we intend to conduct experiments using more general and large-scale examples as well as consider applications based on the dependencies among patterns, which should realize more practical uses.

REFERENCES

[1] Maruyama, k., Washizaki, H., & Yoshioka, N. (2008). A Survey on Security Patterns Progress in Informatics No.5 pp.35-47.

[2] Bschmann, F., Fernandez-Buglioni, E., Schumacher, M., Sommerlad, P., & Hybertson, D. (2006). SECURITY PATTERNS : Integrating Security and Systems Engineering (Wiley Software Patterns Series).

[3] Kobashi,T., Yoshioka, N., Kaiya, H., Washizaki, H., Okubo, T., & Fukazawa, Y. (2014). Validating Security Design Pattern Applications by Testing Design Models. International Journal of Secure Software Engineering. IJSSE volume 5 issue 4 pp1-30.

[4] Okubo, T., Kaiya, H., & Yoshioka, N. (2012). N. Effective Security Impact Analysis with Patterns for SoftwareEnhancement. IJSSE 3(1): 37-61 2012.

[5] Okubo, T., Taguch, K., & Yoshioka, N. (2009). Misuse Cases + Assets + Security Goals. International Conference on Computational Science and Engineering.

[6] Andreas L., & Sindre, G. (2000). Eliciting security requirements by misuse cases. IEEE Computer Society.

[7] Dong, J., Peng, T., & Zhao, Y. (2008). Verifying Behavioral Correctness of Design Pattern Implementation. SEKE, page 454-459.

[8] Abramov, J., Shoval, P., & Sturm, A. (2009). Validating and Implementing Security Patterns for Database Applications. SPAQu.

[9] Dong, J., Peng, T., & Zhao, Y. (2009). Automated verification of security pattern compositions. Information and Software Technology, vol 52, pages 274–295.

[10] Torsel, A.-M.A. (2013). Testing Tool for Web Applications Using a Domain-Specific Modelling Language and the NuSMV Model Checker. Software Testing, Verification and Validation (ICST), pages 383–390.

[11] Choi, B., Kim, H., & Yoon, S. (2009). Performance testing based on test-driven development for mobile applications. ICUIMC.

[12] Astels, D., Beck, K., Boehm, B., Fraser, S., McGregor, J., Newkirk, J., & Poole. C. (2003). Discipline and practices of TDD : (test driven development). OOPSLA.

[13] Büttnera, F., Gogollaa, M., & Richtersb,M. (2007). USE: a UML-based specification environment for validating UML and OCL. Science of Computer Programming (vol 69).

[14] Kleppe. A & Warmer, J. (1999). The Object Constraint Language: Precise Modeling with UML (Addison-Wesley Object Technology Series).

[15] Mackman, A., & Maher, P. (2007). Web Application Security Frame. Microsoft Patterns & Practices. http://msdn.microsoft.com/en-us/library/ms978518.

[16] Kaiya, H., Kobashi.T., Okubo, T. Washizaki, H., & Yochioka, N. (2013). SSR-Project.https://github.com/SSR-Project

Appendix

Here we describe our plan for a live demonstration, which has two objectives:

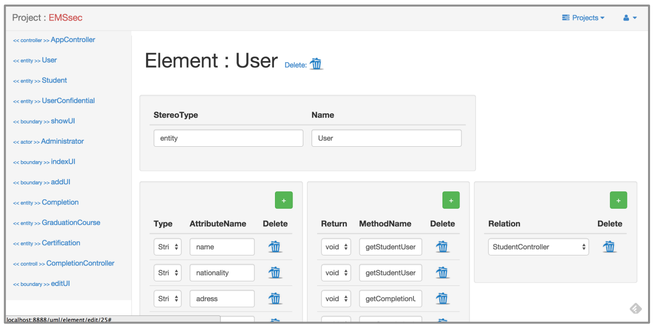
- To explain how to detect incorrect pattern applications using TESEM.
- To encourage developers who design models using UML to employ TESEM.

This demonstration of TESEM should help developers realize secure designs by applying security patterns. Below is an explanation of how we will achieve each objective.

A. *Example of the Verification Process using TESEM*

To demonstrate that TESEM helps detect incorrect pattern applications and helps realize secure designs, we intend to show an example of a verification process using TESEM where the target is EMS (Enrollment Management system) with typical software vulnerabilities.

First, we will explain how to model the target system and how to add or edit elements of the model. Figure 15 shows a screenshot when editing "User" element.


Project : EMSsec
Projects
<< controller >> AppController
<< entity >> User
<< entity >> Student
<< entity >> UserConfidential
<< boundary >> showUI
<< actor >> Administrator
<< boundary >> indexUI
<< boundary >> addUI
<< entity >> Completion
<< entity >> GraduationCourse
<< entity >> Certification
<< controll >> CompletionController
<< boundary >> editUI
Delete:
+
localhost:8888/uml/element/edit/25#

Element : User

StereoType	Name
entity	User

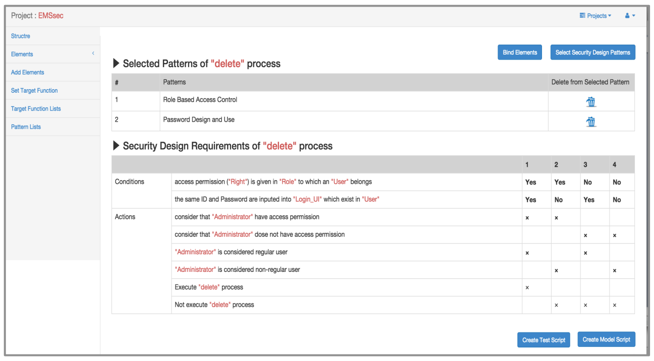
Type	AttributeName	Delete
Stri	name	
Stri	nationality	
Stri	adress	

Return	MethodName	Delete
void	getStudentUser	
void	getStudentUser	
void	getCompletionL	

Relation	Delete
StudentController	

Fig. 15. Screenshot when editing "User" element.

Then we will show how to apply security patterns. Additionally, we will demonstrate how to set up Security Design Requirements for the target model. Figure 16 shows a screenshot after selecting patterns and setting up Security Design Requirements ("RBAC" and "Password Design and Use").



Selected Patterns of "delete" process

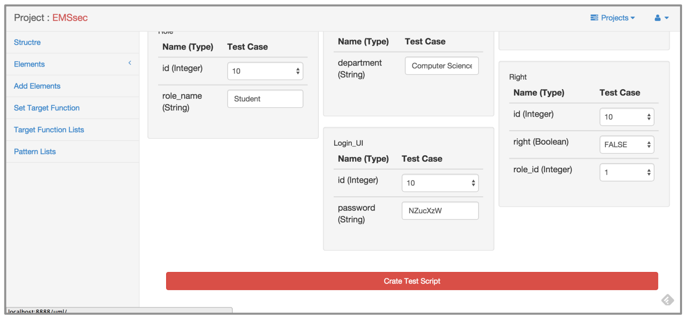
#	Patterns	Delete from Selected Pattern
1	Role Based Access Control	
2	Password Design and Use	

Security Design Requirements of "delete" process

		1	2	3	4
Conditions	access permission ("Right") is given in "Role" to which an "User" belongs	Yes	Yes	No	No
	the same ID and Password are inputed into "Login_UI" which exist in "User"	Yes	No	Yes	No
Actions	consider that "Administrator" have access permission	×	×		
	consider that "Administrator" dose not have access permission			×	×
	"Administrator" is considered regular user	×		×	
	"Administrator" is considered non-regular user		×		×
	Execute "delete" process	×			
	Not execute "delete" process		×	×	×

Fig. 16. Screenshot when editing User element.

Finally, we will explain how to generate a test script to verify whether the model in which patterns are applied satisfies the Security Design Requirements. In this demonstration, we will confirm whether specific test cases are satisfied. Figure 17 shows a screenshot when generating a specific test script. Moreover, we will show the results of a test using this test script.



Name (Type)	Test Case
id (Integer)	10
role_name (String)	Student

Name (Type)	Test Case
department (String)	Computer Science

Login_UI

Name (Type)	Test Case
id (Integer)	10
password (String)	NZucXzW

Right

Name (Type)	Test Case
id (Integer)	10
right (Boolean)	FALSE
role_id (Integer)	1

Fig. 17. Screenshot when generating test script

B. *Registration and creating new project*

TESEM manages user accounts and has sign-up and sign-in functions. Moreover, each user can create a new private or public project. We will show how the sign-up and create a new project.

First we will access TESEM's landing page and create new account (Fig. 18). TESEM can select password-login or Facebook-login.

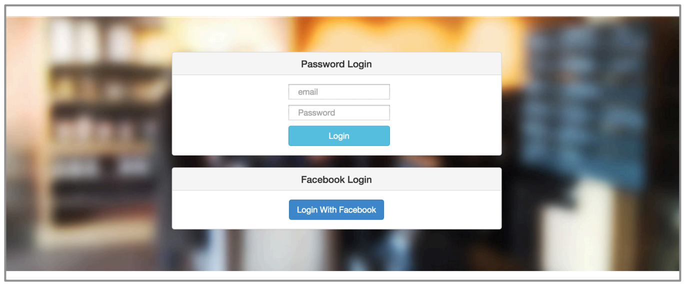


Fig. 18. Landing page of TESEM

Then we will create new project. The access level such as "private" or "public" allows developers to manage confidential information. Figure 19 shows screenshot when a new project is created.

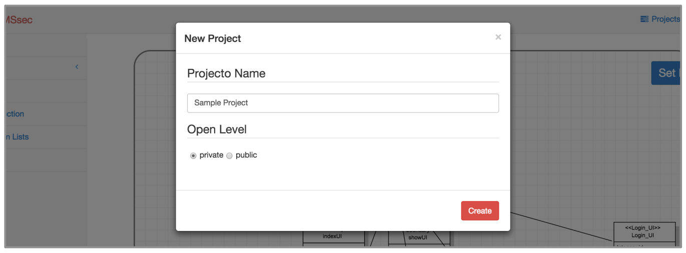


Fig. 19. Screenshot when creating a new project.