Specifying Security Aspects in UML Models *

Karine P. Peralta, Alex M. Orozco, Avelino F. Zorzo, Flávio M. Oliveira
{karine.peralta, alex.orozco}@cpph.pucrs.br
{avelino.zorzo, flavio.oliveira}@pucrs.br

Faculty of Informatics
Pontifical Catholic University of Rio Grande do Sul,
Av. Ipiranga, 6681, Building 32, Porto Alegre, Brazil

Abstract. Expansion of computer systems and the increasing number of services provided by Internet has lead software engineers to worry about security issues of their software. The reason is the short amount of time dedicated to test these characteristics, which leads to release insecure software to final users. To ease this problem, the use of model-based testing is becoming popular. Several works propose standards to model various elements, but a few related to security characteristics. This article presents a technique to specify UML security stereotypes, aiming to guide developers by annotating vulnerable model parts and to allow the automatic security test case generation.

1 Introduction

Due to the evolution of computer systems and the amount of functionalities they provide, software is becoming so complex that even to test them is a difficult task. Besides, a short period of time uses to be dedicated to the test phase, resulting in a deficient analysis of software correctness. Therefore, many software elements are not thoroughly tested, for instance system security.

In this context, model-based testing approach is becoming popular [1]. The goal of this technique is to generate test cases by extracting specific information from a model, accordingly to the characteristic that must be tested. The most common language used to support model-based testing is UML (Unified Modeling Language) [2]. UML is easy to use, to understand and most of software engineers had some training on it [3]. Other advantage is that UML can be extended using stereotypes. Several works propose expansions to represent various elements, but only a few of them are related to describing security characteristics. In addition, existent works have different goals, such as guaranteing a secure system behavior or modeling specific security criteria.

This work presents a set of UML stereotypes to specify some behaviors that may compromise software security. The use of these stereotypes allows the software engineer to annotate model parts that may contain vulnerabilities. It has

* This work was developed in collaboration with HP Brazil R&D

two main goals: to assist developers during the implementation process, emphasizing the functionalities that must be developed carefully; and, to provide security information to allow the test case generation.

This paper is organized as follows. Section 2, 3 and 4 give a brief overview of model-based testing, security taxonomies and relevant aspects of UML, respectively. Section 5 presents some related work about security modeling. Section 6 introduces the proposed model to describe security issues, while in Section 7 we illustrate a case study applying our proposal. Section 8 concludes the paper.

2 Model-based Testing

During the software testing phase, the test engineer must to analyze the system and to define what test cases must be performed to detect undesired behaviors. However, to define these cases manually requires time and effort from the engineers, who must understand the application in deep. Furthermore, the tests may not cover critical failures that may the system and that will be discovered only when running in a real environment.

In this sense, model-based testing approach is becoming popular. It consists of generating test cases based on the application model, which also includes the specification of elements that will be tested [4]. The advantage of this approach is the possibility to detect inconsistencies in the design phase, when to correct problems is cheaper and faster than after the implementation [5]. However, even with the expansion of model-based testing, this technique is not widely used to analyze security characteristics. This is due to the fact that it is difficult to specify security issues using models, since several details must be considered [6].

In order to propose a model that describes most part of security behaviors, one of the strategies is to evaluate security taxonomies [7]. The use of these taxonomies helps to identify similar behaviors in attacks or vulnerabilities, allowing the construction of precise models.

3 Security Taxonomies

Considering the lack of information about security metrics, the use of security taxonomies as guide is a way to identify vulnerabilities or attacks that can compromise a system. The most common criteria used to categorize the taxonomies are vulnerabilities and attacks, but some works also considerate security flaws. The use of taxonomies based on attacks and vulnerabilities is important for system administrators and testers, who want to know how their systems can be exploited. In the other hand, taxonomies based on flaws are useful for software engineers and developers, who must predict and avoid design and implementation problems. Due to the focus in model-based testing, this work will concentrate on flaw taxonomies.

Several security flaw taxonomies were analyzed [7–10], but most of them are incomplete or out-of-date. In this sense, the work of Sam Weber *et* al. [7] was

selected as a suitable guide. As classification criteria, it uses flaw types, helping to identify which attacks may derive from a specific flaw.

4 UML and OCL

UML [2] is a standard modeling language widely used to specify object-oriented systems. It defines notations to build several diagrams, each one representing a particular view of a specific artifact to be modeled. The flexibility provided by UML allows to extend its modeling capacity using profiles, which facilitates the performing of model-based testing. Besides, it is also possible to add constraints to the UML models by using OCL (Object Constraint Language) [11], which specifies pre and postconditions to formalize operation behaviors.

Although using OCL it is possible to represent different characteristics of a system, it is difficult to comprehend, leading software engineers to extend UML using stereotypes and tags. Considering the security taxonomies analyzed and the complexity associated to the OCL descriptions, this paper presents a set of stereotypes created to describe some security elements. The goal of these stereotypes is to easily annotate the parts of the model where a flaw may happen, aiming to guide developers and to generate security test cases.

5 Related Work

Even though being possible to describe security characteristics only with UML and OCL, several works propose profiles and frameworks to expand UML models. Jürjens proposes UMLsec [12], a UML profile for modeling and evaluating security characteristics in order to guarantee basic principles in the whole system, for instance to describe a fair exchange operation or to mark secure links. It is different from our work in that we do not focus on to guarantee a correct and secure system behavior, but to propose a strategy to emphasize parts of the model that may become vulnerable if developers are not careful enough. However, considering that some UMLsec stereotypes could also be used to represent a vulnerability, they were reused or adapted to our work. Other relevant work proposed by Jürjens is [13], that is a case study describing how UMLsec can be used to model-based security testing. However, even being about security testing, the focus of the work is still to assure security properties of the system.

Lodderstedt *et* al. propose SecureUML [14], an extension of the UML language to specify security policies for RBAC (Role-based Access Control). It combines the graphical notation for RBAC with the power of logical constraints on models. The problem is that SecureUML describes only RBAC policies, which consists of a specific type of vulnerabilities. The same restriction was found in the works presented by Ahn and Hu [15] and Ray *et* al. [3], which also refer to RBAC policies. The former proposes a framework to integrate security policies and access validation in the model, in order to guide developers. The latter employs visualization techniques to represent the violation of RBAC constraints instead of describing them in OCL.

We also expect to automatically generate security test cases based on model information, assisting the tester. Blackburn *et* al. [6] present a methodology to automate security functional testing. However, they are interested in to provide a way to perform security tests based on the model, while our purpose is to generate test cases based on it.

Percy *et* al. [16] proposed a work similar to ours, which also aims to support automatic test case generation. However, they use OCL statements to insert information in the model, while we use stereotypes, which are easier to understand. Besides, Percy's work demands to model the scenario of the attack before generating test cases, while in our we insert security stereotypes in the original model, without the need to describe other models.

6 Proposed Model for Security Specification

The analysis of the security taxonomies allows us to conclude that there are several types of vulnerabilities that may affect an application. In addiction, the study of related works showed that most of the authors are focused on some specific security characteristic, for example access control or secure communication. Hence, we propose some security stereotypes to emphasize parts of the model that may contain a flaw. The insertion of these stereotypes aims to guide developers to avoid flaws, as well as to provide information to allow the automated test case generation.

According to the type of application that will be analyzed, a different set of tests must be performed. This knowledge can be extracted from the security databases [17–19], that present the most critical situations that may compromise a software. The association between the flexibility supplied by UML stereotypes and the consistent knowledge provided by the security databases allows the definition and insertion of security stereotypes in the original UML model.

In order to support the definition of the stereotypes, we analyzed the database provided of OWASP (Open Web Application Security Project) [17]. This project maintains a list of the ten most critical web application vulnerabilities, named Top Ten (last version released in 2007). The list was used, in this work, to verify which vulnerabilities could be represented with stereotypes, and which ones would require a more complex structure.

Considering the categories presented by Weber's taxonomy and the set of vulnerabilities extracted from OWASP, the following stereotypes were defined:

1. **Case 1 - Buffer Overflow:** this vulnerability consists of assigning a value larger than the target variable may handle. The stereotype must be inserted in the diagram part that is responsible for catching the user data.
   Stereotype: ≪BufferOverflow≫
   Tag: {BOFlow = {<field>, <size>}}
   where <field> corresponds to the name of the field in which data will be inserted and <size> is the maximum number of characters allowed for that structure.

2. **Case 2 - Connection Flooding:** this vulnerability consists of starting, simultaneously, more connections than the service provider supports. The stereotype must be inserted in the diagram part that represents the successfully established connection.
   Sterotype: ≪Flooding≫
   Tag: {FDMaxConn = {<max_connection>}}
   where <max_connection> corresponds to the maximum number of simultaneous established connections.

3. **Case 3 - Encrypt (Connections):** this vulnerability consists of sending sensible data through the network without encryption. The stereotype must be inserted in the diagram part that is responsible for sending the data through the network.
   Stereotype: ≪Encrypt≫
   Tag: {CRField = {<field>}} where <field> corresponds to the name of the field that must be encrypted.

4. **Case 4 - Access Control (By Passing):** this vulnerability consists of asking for user's login information only in the first page of a session, not asking for it when the user access a directly internal link. In order to help the test case generation process, the software engineer must insert three types of tags, all of them derived from the ≪ByPassing≫ stereotype. The first step is to identify the system users by inserting the tag {BPRole} in the actors of the use case diagram. The second step is to inform the links of each page accessed by the user, using the tag {BPLink}. The last step is to mark the user roles who can access the system. The software engineer may choose between to represent the users who can access the system, through the tag {BPAllowed}, or the users who cannot access the system, using the tag {BPDenied}. The definition of each tag is as follows.

   Stereotype: ≪ByPassing≫
   Tag: {BPRole}
   without tag value because it is only to mark the system actors.

   Stereotype: ≪ ByPassing ≫
   Tag: {BPLink = {<link>}}
   where <link> is the path of the accessed page.

   Stereotype: ≪ ByPassing ≫
   Tag: {BPAllowed = {<role>}}
   where <role> means the actor that can access the system.

   Stereotype: ≪ ByPassing ≫
   Tag: {BPDenied = {<role>}}
   where <role> means the actor that cannot access the system.

5. **Case 5 - Access Control (Session Expiration):** this vulnerability consists in not to expire the user session after an amount of time. The stereotype must be used in the diagram part that represents the successful user's login.
   Stereotype: ≪Expiration≫
   Tag: {ExpTime = {<time>, <unit>}}
   where <time> represents the maximum time to expire the user's session and <unit> means the time measure unit, which may be 'ms' (milliseconds), 's' (seconds), 'm' (minute), 'h' (hour) or 'd' (day).

6. **Case 6 - SQL Injection:** this vulnerability does not validate the input data before assigning it to a SQL query. This way, a malicious user may insert another SQL query instead of his username or password, for example. This test tries to detect two vulnerabilities: the susceptibility of the system to malicious SQL codes and the existence of information leakage, which may expose sensible data. The stereotype must be inserted in the diagram parts that catch information provided by the user.
   Stereotype: ≪SqlInjection≫
   Tag: {SQLField = {<field>}}
   where <field> corresponds to the name of the field in which data will be inserted.

Considering the amount of existent vulnerabilities, we have created stereotypes to represent only a few of them. The used criteria were the easiness to represent the vulnerabilities and to generate test cases from the model. Besides, we have used only activity diagrams to insert the information, in order to make it more simple to understand (except in the By Passing case, which also needs the use case diagram). After inserting the stereotypes in the model, the next step is to generate the correspondent XMI (XML Metadata Interchange) file, which will be used as input for the test case generator that is still under development.

7 Case Study

In this section, we present a case study to evaluate the viability of our approach using a well-known application, named TPC-W [20]. TPC-W is a transactional web benchmark specially designed for evaluating the performance of e-commerce systems. It models an e-commerce site in the form of an online bookstore, providing operations such as searching, buying and adding new books. According to Sopitkamol and Menascè [21], it is very difficult, if not impossible, to conduct experiments on commercial e-commerce sites, which makes the testing of TPC-W a good choice. Besides, there is a lot of information about the software, that includes a complete specification and some UML models. Finally, the analyzed Top Ten list refers to web applications, what has contributed in choosing TPC-W as an experimental application to our proposal. Therefore, it is important to note that TPC-W will not be used here as "benchmark", but as a commercial web application in which some security elements will be evaluated.

7.1 System Modeling

The TPC-W application reproduces the main user's actions during the visit of a bookstore e-commerce site. Based on the information provided in the TPC-W specification, we have built the UML model of some features in that it could be possible to insert security information. Analyzing the TPC-W specification, four essential security requirements were extracted. They are as follows:

- Requirement 1: The user's first name field (FNAME), completed during the user's registration, must have at most 15 characters.
- Requirement 2: The page that contains the order information must encrypt the user's credit card number (CX_NUM) during the data sending process.
- Requirement 3: To update books prices, the user must be an administrator logged in the system.
- Requirement 4: The databases must be protected from unauthorized users.

Considering the stereotypes proposed in Section 6, we have modeled the TPC-W security requirements. Figure 1 presents the use of ≪BufferOverflow≫ and ≪Encrypt≫ stereotypes in the activity diagram. Figure 2 exhibits the ≪ByPassing≫ and ≪SqlInjection≫ stereotypes, also in the activity diagram. Finally, Figure 3 complements the description of the ≪ByPassing≫ stereotype by marking the actors in the use case diagram. Note that none of these requirements have demanded the use of ≪Flooding≫ or ≪Expiration≫. To complement the case study, suppose that the following requirements were also part of the TPC-W specification:

- Requirement 5: At most 30 users can be logged in the system.
- Requirement 6: The user's session must expire after 5 minutes of inaction.

Figure 4 represents the modeling of the new requirements in the activity diagram previously used to insert ≪BufferOverflow≫ and ≪Encrypt≫ stereotypes.

To evaluate the efficiency of the stereotypes inserted in the diagrams, some manual tests were applied to TPC-W. The performed cases were the ones extracted from the TPC-W specification. The obtained results are as described in Table 1.

Analyzing the obtained results lead us to conclude that TPC-W is not a secure application. If it was a real e-commerce system, user's personal information would be exchanged without encryption and a malicious user could crack the application. A good point about TPC-W is that it uses precompiled SQL statements, which when used correctly makes impossible to modify the actual SQL statement, avoiding the SQL Injection attack.

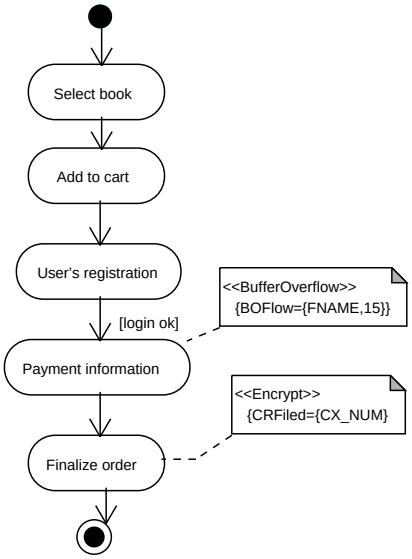

Fig. 1. *Application of ≪BufferOverflow≫ and ≪Encrypt≫ stereotypes*

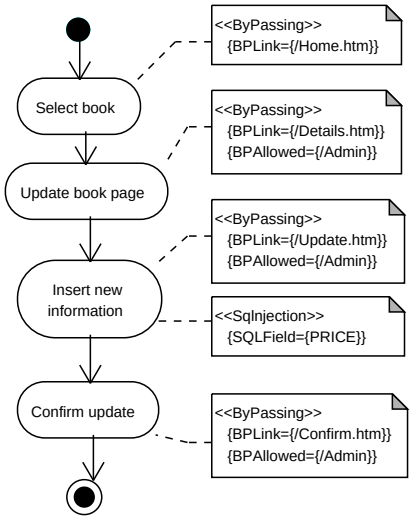

Fig. 2. *Application of ≪ByPassing≫ and ≪SqlInjection≫ stereotypes*

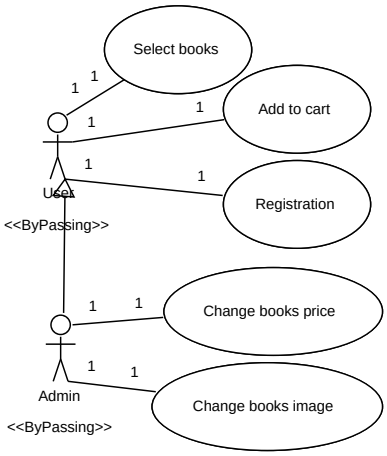

Fig. 3. *Application of ≪ByPassing≫ stereotype in use case diagram*

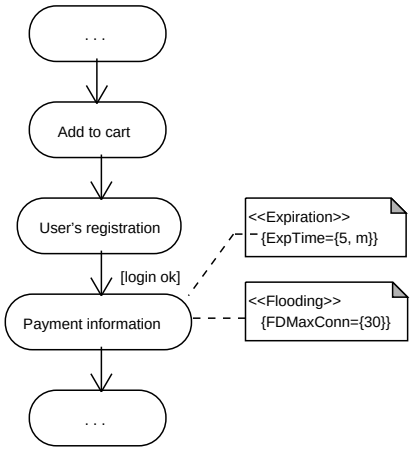

Fig. 4. *Application of ≪Expiration≫ and ≪Flooding≫ stereotypes*

8 Conclusions and Future Works

Based on the Weber's taxonomy and on the vulnerability list provided by OWASP project, we have defined some stereotypes to represent the most common security

Table 1. Test case results

Test Case	Results
≪BufferOverflow≫	TPC-W has accepted the registration with a user's first name larger than 15 characteres, but no abnormal situation was detected
≪Encrypt≫	TPC-W does not encrypted the user's credit card number, which could be stolen using a packet sniffer
≪ByPassing≫	TPC-W did not ask for the user's login before updating the books price, accepting a new price
≪SqlInjection≫	TPC-W was not vulnerable to SQL Injection

flaws. The insertion of these stereotypes allows representing security characteristics since the design phase, guiding developers to avoid vulnerabilities.

Furthermore, the stereotypes also allow the automatic generation of security test cases. This process will be performed by extracting information from the XMI file, which represents the system UML model. For each security information inserted in the model, one test case will be generated, describing the steps to verify the occurrence of the vulnerability.

The efficiency of the security stereotypes has been proved when it was possible to perform tests and to detect some security vulnerabilities in TPC-W, our case study. Although the presented cases may be considered basics, we are working to support the generation of more elaborated test cases, as well as to expand our model to describe more critical web application vulnerabilities. Therefore, it will be possible to perform tests that demand the use of other UML diagrams, such as XSS (Cross-site Scripting) and Malicious File Execution.

References

1. Marick, B.: *"New Models for Test Development"*. 12th International Software Quality Week Technical Program, 1999.
2. *"OMG'S Unified Modeling Language (UML)"*. Available in <http://www.omg.org/gettingstarted/what_is_uml.html>. July 2008.
3. I. Ray, N. Li, D. Kim and R. France. *"Using UML to Visualize Role-based Access Control Constraints"*. 9th ACM Symposium on Access control Models and Technologies (SACMAT '04), pages 115 - 124, 2004.
4. Popovic, M., Velikic, I.: *"A Generic Model-Based Test Case Generator"*. 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, pages 221 - 228, 2005.
5. Hailpern, B., Santhanam, P.: *"Software debugging, testing, and verification"*. IBM Systems Journal, vol. 41, n° 1, pages 4 - 12, 2002.

6. Blackburn, M., Chandramouli, R.: *"Model-based Approach to Security Test Automation"*. 13th International Symposium on Software Reliability Engineering, Industry Track, 2002.
7. Weber, S., Karger, P. A., Paradkar, A.: *"A Software Flaw Taxonomy: Aiming Tools at Security"*. Workshop on Software Engineering for Secure Systems, pages 1 - 7, 2005.
8. Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham, R. K., Zissman, M. A.: *"Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation "*. DARPA Information Survivability Conference and Exposition, 2000.
9. Aslam, T.: *"A Taxonomy of Security Faults in the Unix Operating Systems"*. Master Thesis, University of Purdue, 1995.
10. Bazaz, A., Arthur, J. D., Tront, J. G.: *"Modeling Security Vulnerabilities: A Constraints and Assumptions Perspective"*. 2nd IEEE International Sysmposium on Dependable, Autonomic and Secure Computing, pages 95 -102, 2006.
11. *"Object Constraint Language Specification (OCL)"*. In OMG Unified Modeling Language Specification, version 1.3, June 1999.
12. Jürjens, J.: *"UMLsec: Extending UML for Secure Systems Development"*. 5th International Conference on the Unified Modeling Language, pages 412 - 425, 2002.
13. Jürjens, J.: *"Model-based Security Testing using UMLsec: A case-study"*. 11th European Joint Conferences on Theory and Practice of Software, 2008.
14. Lodderstedt, T., Basin, D. A., Doser, J.: *"SecureUML: A UML-based Modeling Language for Model-Driven Security"*. 5th International Conference on the Unified Modeling Language, pages 426 - 441, 2002.
15. Ahn, G. J., Shin, M. E.: *"Role-based Authorization Constraints Specification using Object Constraint Language"*. 10th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, pages 157 - 162, 2001.
16. Salas, P. A. P., Krishnan, P., Ross, K. J.: *"Model-based Security Vulnerability Testing"*. Australian Software Engineering Conference, pages 284 - 296, 2007.
17. The Open Web Application Security Project. *"The Ten most Critical Web Application Security Vulnerabilities"*. Available in <http://www.owasp.org>. July 2008.
18. National Institute of Standards and Technologies. *"National Vulnerability Database"*. Available in <http://nvd.nist.gov>. July 2008.
19. Sans Institute. *"SANS Top 20 List"*. Avaliable in <http://www.sans.org/top20>. July 2008.
20. TPC-W Benchmark (TPC-W). Available in <http://www.tpc.org/tpcw>. July 2008.
21. Sopitkamol, M., Menascè, D.: *"A Method for Evaluating the Impact of Software Configuration Parameters on e-commerce Sites"*. 5th International Workshop on Software Performance, pages 53 - 64, 2005.