# Modeling Input Validation in UML

Pedram Hayati
Institute for Advanced Studies in Basic Sciences, Zanjan, Iran
pedram@iasbs.ac.ir

Nastaran Jafari
Institute for Advanced Studies in Basic Sciences, Zanjan, Iran
n_jafari@iasbs.ac.ir

S. Mohammad Rezaei
Institute for Advanced Studies in Basic Sciences, Zanjan, Iran
s_mohamadrezaei@iasbs.ac.ir

Saeed Sarenche
Institute for Advanced Studies in Basic Sciences, Zanjan, Iran
sarenche@iasbs.ac.ir

Vidyasagar Potdar
Digital Ecosystems and Business Intelligence Institute,
Curtin Business School, Curtin University of Technology, Australia
Vidyasagar.Potdar@cbs.curtin.edu.au

## Abstract

*Security is an integral part of most software systems but it is not considered as an explicit part in the development process yet. Input validation is the most critical part of software security that is not covered in the design phase of software development life-cycle resulting in many security vulnerabilities. Our objective is to extend UML to new integrated framework for model driven security engineering leading to ideal way to design more secure software. Input validation in UML has not been addressed previously, hence we incorporate input validation into UML diagrams such as use case, class, sequence and activity. This approach has some advantages such as preventing from common input tampering attacks, having both security and convenience in software at high level of abstraction and ability of solving the problem of weak security background for developers.*

## 1. Introduction

Every software application is deployed today to accomplish some goals. However every application can be misused and faces threats from Internet-aware client applications running on PCs, to complex telecommunications and power systems accessible over the Internet. The main source of vulnerability of systems has been recognized to be poor-quality software. So, software engineers must be aware of threats and engineer systems with credible defenses. Thus, security as a non-functional requirement plays a critical role in the development of many large-scale distributed software systems. In other words, while secure applications are also valid and robust ones, security is a specific *non-functional* requirement that has to be explicitly and carefully taken into account during analysis, design, implementation, testing, and deployment**.** The importance of this concept appears in the web applications because statistics show that 75% of security attacks occur on web applications [14]. We have all heard about web sites being hacked and private customer information being disclosed. This is only one example of the many potential security flaws in web applications. However, the breach of a company's website can cause significant revenue losses, large repair costs, legal consequences and loss of credibility with customers. In Early 2003 a public university's website was hacked into and thousands of social security numbers were released to the public [6]. Therefore, web applications must handle customer data and other electronic information as securely as possible [10, 6]. Security mechanisms and policies are generally added to the existing system as an afterthought (penetrate and patch approach), with all the problems of unsatisfied security requirements, integration difficulties, and mismatches between design models [11]. There are three main reasons for this [12]. First, security must cover every part of an application. Second, there are not enough tools to support security engineering. Finally developers do not have strong background on security and need guidelines for constructing secure applications. There is little work concerning the full integration of security and systems engineering from the earliest phases of software development. Although several approaches have been proposed for some integration of security, there is currently no complete methodology to assist developers of security sensitive systems. All this becomes a special concern when considering complex

1530-0803/08 $25.00 © 2008 IEEE
DOI 10.1109/ASWEC.2008.30

security requirements such as those associated with applications, e-commerce etc [4].

Software is designed to process a defined set of data. For example a word processor is designed to deal with document files not with audio or video files. When undefined set of data (audio/video files for word processor) sent to software, it might produce unpredictable results and many times an attacker by examining these undefined set of data which know as malicious data, try to reach his/her goals. Input validation means validating data flow in the software before using them. On the other hand, invalidated input is that set of data, which is directly used by software without any validating mechanisms.

Invalidated input is the most critical security flaw in applications, especially in web applications. Many security holes in applications are caused by invalidated inputs. Since entering invalid data, attackers take software into unpredictable conditions and exploit this condition for their own purpose. Common input tampering attacks include: forced browsing, remote command injection, cross site scripting, buffer overflows, format string attacks, SQL injection, cookie poisoning, and hidden field manipulation [16]. Consequently, before using data in the application, it must be completely and properly validated by security validating mechanisms. In addition, if validating mechanisms do no defined properly those data that come from believed–authoritative references might be reject, so, it annoys real application users [2]. In other words, some legitimate inputs will be incorrectly flagged as bad, leading to user frustration and some attacks will be incorrectly flagged as safe, leading to exploits. Input validation is a part of security engineering policies. The integration of security engineering into a model-driven software development approach has some advantages [12]

1. Security requirements can be formulated and integrated into system design at a high level of abstraction. So it becomes possible to develop security aware applications that are designed with the goal of preventing violations of a security policy.
2. The model information can be used to detect and correct design errors or to verify the correctness of the mapping between requirements and their realization in a design.
3. It saves more budgets.

Main problem of exciting literature is explained in section 2. In section 3, we provide some background of concepts which are used in the context of the article and we illustrate an overview of existing methodologies for designing secure software systems. In section 4, we represent our proposed model. With an example. Section 5 describes limitation in proposed model. And finally, we present a conclusion in section 6.

## 2. Problem Definition

According to our survey input validation has not been covered in any security approaches and existing literature does not sufficiently address it. We will provide an integrative approach supporting the integration of security and system engineering in order to model input validation in software. We discuss how input validation prevents from insecurity and how to present input validation in UML models.

## 3. Literature Review

This part introduces some background information and outlines the existing methodologies for input validation.

### 3.1. Object Constraint Language (OCL)

OCL is a formal language used to describe expressions in UML models. These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model. A UML diagram, such as a class diagram, is typically not refined enough to provide all the relevant aspects of a specifications. Thus, UML modeler can use OCL to specify application-specific constraints in their models [3]. OCL can be used for a number of different purposes:

- as a query language
- to specify invariants on classes and types in the class model
- to specify type invariant for stereotypes
- to describe pre and post conditions on operations and methods
- to describe guards
- to specify constraints on operations

### 3.2. Regular expression

A regular expression is a string that is used to describe or match a set of strings, according to certain syntax rules [15]. Much software such as word processors, applications, and programming languages use regular expression to search and manipulate text based on patterns. Using regular expressions are very useful since they give a concise description of a set, without having to list all elements. For example, the set containing two strings "gray" and "grey" can be by

regular expression "gr[ae]y". Sometimes regular expression is called *pattern*.

In our proposed model we employed regular expression in order to define security attributes.

## 3.3. SQL Injection Attack

One of the many web attacks used by hackers to steal data from organizations is SQL injection. Today, it is perhaps one of the most common application layer attack techniques [18]. This type of attack takes advantage invalidated inputs to web applications that allows hacker to inject SQL commands into say a login form to allow them to gain access to the data held within your database. In essence, SQL injection arises because the fields available for user input allow SQL statements to pass through and query the database directly. But in essence if input validation is in place, SQL statements would not be allowed to pass through the web interface.

### 3.3.1 Example of SQL Injection Attack

Below is HTML code of a simple login form that used to send user id and password:

```
<form method="post"
action="http://example.com/login.php">
<input name="username" type="text" id="userid">
<input name="password" type="password"
id="password">
</form>
```

User ID field is an identification number (e.g. Student number). Database query that used for user authentication is:

```
SELECT id
FROM logins
WHERE userid = '$userid'
AND password = '$password'
```

If the variables $userid and $password are requested directly from the user's input, this can easily be compromised. Suppose we provide the following string as user id: " ' OR '1'='1' " and "anything" as a password. So our database query changes to:

```
SELECT id
FROM logins
WHERE userid = '' OR '1' = '1'
AND password = 'anything'
```

As the inputs of the web application are not properly validated, the use of the single quotes has turned the WHERE SQL command into a two-component clause. The '1' = '1' part guarantees to be true regardless of what the second part contains. This will allow the attacker to bypass the login form without actually knowing a valid username / password combination.

In this paper, we use SQL injection as an example, which indicates that invalidated inputs to web applications are simply in front of dangerous SQL injection attacks.

## 3.4 Existing Methodologies

The Unified Modeling Language (UML) is the industry standard for designing software systems, but it only includes minimal capabilities for representing security aspects of a system. Therefore, methodologies like SecureUML, UMLpac, and UMLsec etc. present an extension to UML, which enables security attributes to be easily integrated into UML. We now review these methodologies in detail.

### 3.4.1 SecureUML

The main goal of SecureUML is to develop a complete model driven approach for developing secure e-commerce systems. SecureUML is an extended model for role based access control (RBAC). Access control infrastructures can be generated from SecureUML models and prevent errors during the realization of access control policies .As RBAC lacks, support for expressing access control conditions that refer to the state of a system, the concept of authorization constraints was introduced. In other words, SecureUML offers important design flexibility because it combines the simplicity of a graphical notation for RBAC with the power of logical constraints on models but the main drawback is that it just focuses on access control and lacks features to integrate all security aspects into the design of a security sensitive system [12].

### 3.4.2 UMLsec

UMLsec presents a way to implement secure-systems in UML [17]. In UMLsec, validation rules that evaluate a model against included security requirements are given.

### 3.4.3 UMLpac

UMLpac makes it possible for developers to layout security features onto UML class diagram of a system. By using security packages in UMLpac, a level of abstraction is created between the class diagram and the security features. It is important to say that SecureUML can be integrated into UMLpac through the use of a security package having a principle security descriptor for SecureUML and UMLsec can

be used effectively with UMLpac to layout security features in security tiles [19].
All in all, none of these methodologies incorporate input validation, and ignorance of this feature leads to almost all security attacks.

### 3.5 Validating Models and Strategies

There are three validation models or strategies for validating data [1]:
1) Rejecting bad data: creating a set of undesirable data and rejecting them. This model is also known as "black list" approach.

2) Accepting only known good data: data constrained by Five Primary Security Input Validation Attributes (FPSIVA) which are: *type, length, character set, format, reasonableness*. Data is rejected unless it matches for known good data. This model is also known as "white list" approach.

3) Sanitizing data: sanitizing a defined set of dangerous data so that it does not pose a threat to the software.

The first model has bottom-up approach and software developer should predict all malicious data that is dangerous for software. This model heavily depends on methods that attacks execute if attacker changes attack method, software would be vulnerable.

The third model faces the same disadvantage of the first model since a black list must be created. In this model software developer would create a list of sanitize patterns which would be employed to sanitize dangerous data. However at times a software developer does not exactly know all types of dangerous data, hence s/he is not able to define complete sanitized list. So software would be vulnerable.

The second model has top-down approach where the software developer makes a defined set of data attributes which allow eligible and safe data to flow into the software. Actually, this set makes limitation on collection of data that are expected to be processed by software. In this model each condition validates data for type, format, length, character set, and reasonableness and if data conform to the set of defined attributes, they would be used in software, otherwise they would be rejected.

We now explain the definition and importance of these FPSVIA from security perspective:

- Type: this attribute makes limitation on type of data. Common data types are Integer, Boolean, Strings, and Byte. This attribute can restrict attacker to input other data types into the software. For instance, according to section 3.3.1, if software developer validates user id field data type as an integer attacker can not inject his/her malicious command since it contains String data type.
- Format: this attribute defines data syntax which indicates how data should be represented in software. The security importance of this attribute is that it does not allow malicious data format are followed in software which can cause unpredictable software process. For example, email address should contain at sing '@' character. So, software developer defines a format for email address that check for this character. Otherwise attacker can input invalid email addresses.
- Length: this attribute limits counts of data characters. By itself, this attribute can restrict many attackers' malicious command to input into software. According to section 3.3.1, interestingly, this attribute can prevent attacker to inject malicious SQL queries. For instance, Attacker is no able to comprise user id field by malicious queries bigger than eight characters, if software developer limit user id field to eight characters long.
- Character set: this attribute defines characters types. Common characters types include Numbers, Alphabets, Symbols, and combination of them. Character set is the most important security attribute among other since it limits valid characters in each data type. Consequently, attacker is not able to input dangerous characters. For instance, according to section 3.3.1, attacker can not inject single or double quote in SQL user authentication query in order to bypass authentication if software developer defines a [0-9] domain as valid characters set for user id field.
- Reasonableness: the last attribute is the only attribute that directly deals with semantic part of each data. This attribute is employed for detecting which data are reasonable and which are not. The importance of this attribute backs to semantic part of software security which has been challenging issue. In addition, it can be used for preventing future malicious activities that are not known today. For example, according to section 3.3.1, by defining reasonable user id range, attacker can not submit negative value as user id.

Obviously, it is clear for software developer what kind of data should input into/output from the software and commonly, software developers store input data in variables which can be used for future usages in software. For instance, if a software gets users' age and stores it in a variable, it is obvious that a valid type of this variable is integer, a valid length is two or three

numbers long, a valid character set is combination of numbers from zero to nine, a valid format is two or three following numbers without any space among them and finally the valid reasonableness of this variable is that the age should be bigger than zero. According to what is discussed above, it is clear that defining set of known good data by using FPSVIA does not need very strong background on security and a software developer can set them without any profound security knowledge. Thus, by using this strategy we would solve the problem of weak background on security for developers. On the other hand, this strategy is a good choice for having both security and convenience in software, because users' legitimate data are included in white list and exactly is defined by FPSVIA. These kind of data would followed into the software but the other data (that commonly come from attackers such as single quote in our SQL injection example) is not valid according to defined FPSVIA (white list), so they would be rejected

## 4. Proposed Solution

Proposed security model uses OCL as it base for defining FPSVIA in software design phase. The expressiveness of OCL was then carefully examined in order to make sure that this notation could express security constraints [7]. The key issue was analyzed and it conformed that OCL is a sustainable and efficient technology to improve reliability and security when it can be combined by FPSIVA of the proposed security model. Until now, anything on OCL is about the functions of objects and none of them talks about what is entering to the object and what constraints it has. UML modelers can use OCL to specify application specific constraints in their models. By using OCL, In our proposed security model , we define five new constraints based on FPSIVA, each of these new constrains validate some part of input data, ultimately by employing all of them, according 3.2, we suppose to have secure input validated software. Constraints are as flow:

• var.type :< type>

This constraint validates the type of input data and if it conforms to the <*type*> it is acceptable, or else it should be rejected. For example the type of input data for a *phone number* should be *Integer*:

phone_number.type: Integer

• var.format :< pattern>

This constraint validates the format of input data if it conforms to the <*pattern*>, it is acceptable or else it should be rejected. According to 2.2, the format of input data can be defined by using regular expression. For example, the format of *a phone number* is like: *%d-%d%d%d-%d%d%d-%d%d%d%d*. Each "*%d*" is a representative of decimal number which means that only numbers are allowed. [17, 18].

phone_number.format: %d-%d%d%d-%d%d%d-%d%d%d%d

• var.length :< number>

This constraint validates the length of input by maximum length of characters include in input data. For example, phone number has 14 numbers long:

phone_number.length: 14

• var. charset:< pattern>

When data is entering the software, this part checks its characters domain with its <*pattern*>. We define character set of input data by regular expressions. For example, the domain of characters which are acceptable for phone number is from 0 to 9.

phone_number.charset: [0-9]

• var.value :< reasonableness>

This constraint presents which values of input data are reasonable. For example, only human's age bigger than zero are reasonable.

age. value>0

Below, we represent the use of these five constraints on some UML diagrams. Four UML diagrams demonstrated here are: Use Case diagram, Class diagram, Sequence diagram, and Activity diagram. Our proposed model constitute from these four UML diagrams in which each diagram represents unique view of input validation modeling.

### 4.1 Use Case Diagram

As documented in [5], there are numerous kinds of security requirements. Like any other type of quality requirement, security requirements should be based on an underlying quality model. Security signifies the degree to which valuable assets are protected from significant threats posed by malicious attackers .Thus, as a quality factor, we can add input validation because as mentioned before in section 3.5, each kind of security requirement typically has its own security use case that should be used to specify requirements that the application shall successfully protect itself from its relevant security threats [9].

Input validation use case is the extent to which a business enterprise, application, component, or center ensures that its data is validated before allowing request. Table 1, specifies the input validation requirements. In this table, a mis-user is the inverse of a user, someone who –intentionally or accidentally – initiates misuse cases and whom the system should not

support in doing so and a misuse case is the inverse of a use case, a function that the system should not allow .In more detail, it might be defined as a completed sequence of actions which results in loss for the organization or some specific stakeholders. We now verify the class diagram.

## 4.2. Class Diagram

For annotating input validation on class diagram UML-based models, first we must define a set of vocabulary to express different aspects of input validation. The mechanisms of extending UML such as stereotype and constraint are used in class diagram [17]. Also, we use *Metamodel* to define abstract syntax of the language. For each class that gets input from environment (user, other software), we define Input Validation Constraint (IVC). IVC is a graphical notation of input validation in class diagram, which is defining security constraint on variables of a class. As described in section 3.5, FPSIVA are needed for validating variables. This attributes constraint execute for variables which are submitted from environment. Figure 1 shows the class diagram Metamodel of input validation that is an extension of the UML Metamodel.

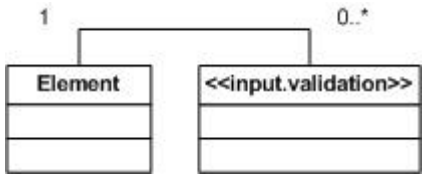


**Figure1, Class Diagram Metamodel of Input Validation**

The "Element" is the class that must be protected and <<*input. validation*>> is IVC.

Because IVC is derived from UML core type constraint, we use the standard UML association between it and "Element" to link them.

**Table 1, Input Validation Use-Case**

| Use-Case: Input Validation | | | |
|---|---|---|---|
| Security threat:<br>The system accepts the mis-user as if the mis-user were a valid user. | | | |
| Precondition:<br>The mis-user has an invalid data entry. | | | |
| *User interactions* | *Mis-user interaction* | *System requirement* | |
| | The mis-user input invalid data. | The system shall check the type, length, format, character set and reasonableness attribute of the entry data. | The system shall recognize that the input data is invalid. |
| The user input his/her valid data. | | The system shall reject the mis-user by canceling the transaction. | The system shall recognize that the input data is valid. |
| Post condition:<br>1) The system run all user input through validation before allowing requests to avoid insecurity.<br>2) The system shall ensure that the input is validated.<br>3) The system shall record the failure items. | | | |

## 4.3. Sequence Diagram

Sequence diagram is used to describe interaction between objects in term of sequence of messages [13, 8]. When this diagram is drawn, the objects which are valued from input can be recognized, thus we can distinguish the objects which need to be validated.

Figure 2 presents the sequence diagram of validating input data. At the top of the diagram, we see rectangle that presents object and the arrows show the messages that exchange between objects. The user sends specific procedure call messages and sends data to object. For validating the user's data, we use specific message to inform it for validating input data by FPSIVA. If this checking is successful, the software will check other attributes or else a message will be sent to user.

For example in Figure 2, the object takes data from the user then the software checks the type of input. If this attribute of input data equals with attribute of input which is defined for software, other attributes like length will be checked, like:"[input.type: true]: input . length". For other attributes the process is the same. If all of above constraint checking is true for input data then we can claim that the data is valid.

## 4.4 Activity Diagram

We use the activity diagram to show the internal activity of input validation process. In this diagram we suppose that what is entered to the system is invalid, thus the validation process validates it. First, it validates the type of input data, and if it conforms to the defined type constraints, it is accepted and continues, otherwise the input data will be rejected. This process is the same for length, format, character set and reasonableness attributes. After checking all of these attributes and making sure that all of them conform to the defined set of constraints, data gets privileged to be used in software. Figure 3, illustrates input validation activity diagram.

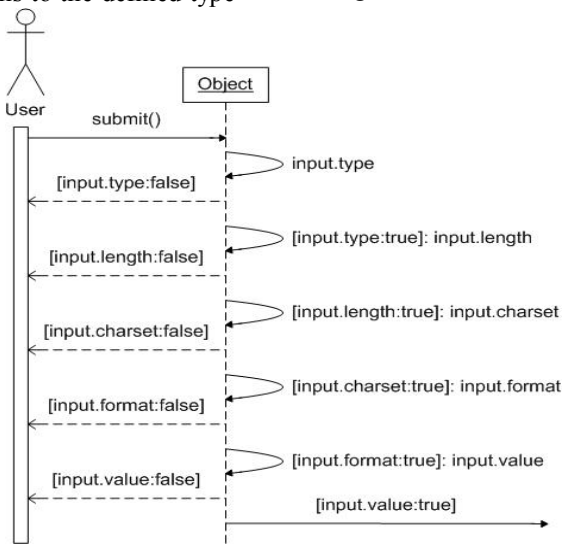


**Figure 2, Input Validation Sequence Diagram**

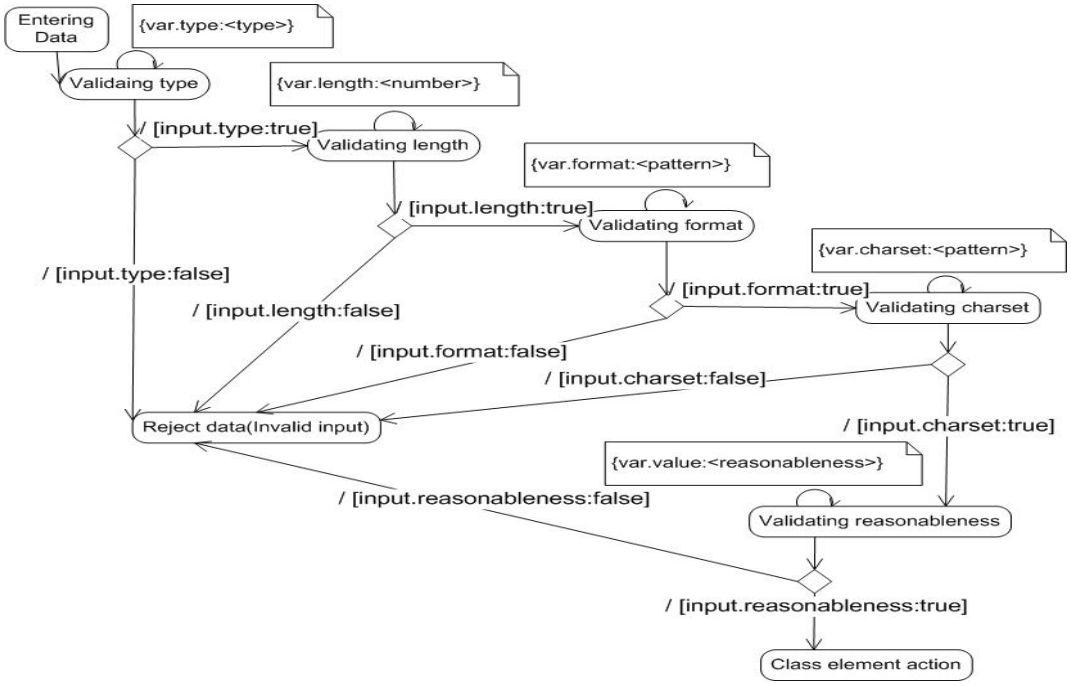


**Figure 3, Input Validation Activity Diagram**

## 4.5 Examples: Registration System

Our example is a simple registration system that is used for registering users on the web site. The registration form contains four fields: name, family name, age, phone number. It uses simple database to store data and has two components: Registration and DB. Registration component is boundary class that interacts with users and DB is entry class that is used for storing data.

### 4.5.1. Use Case Diagram

Unlike normal use cases that document interactions between an application and its users, misuse cases concentrate on interactions between the application and its mis-users who seek to violate its security. Because the success criteria for a misuse case is a successful attack against an application, using misuse cases for requirement modeling are highly adoptable ways of analyzing security threats but are inappropriate for the analysis and specification of security requirements [9]. Figure 3 will show the registration with input validation as a security requirement. Validation of input data includes five processes that check FPSIVA. Being invalid from the perspective of each of the mentioned attributes, it can result in misuse cases which are caused by mis-users. The traditional use case for registration is specializations of a general Manage Accounts use case.

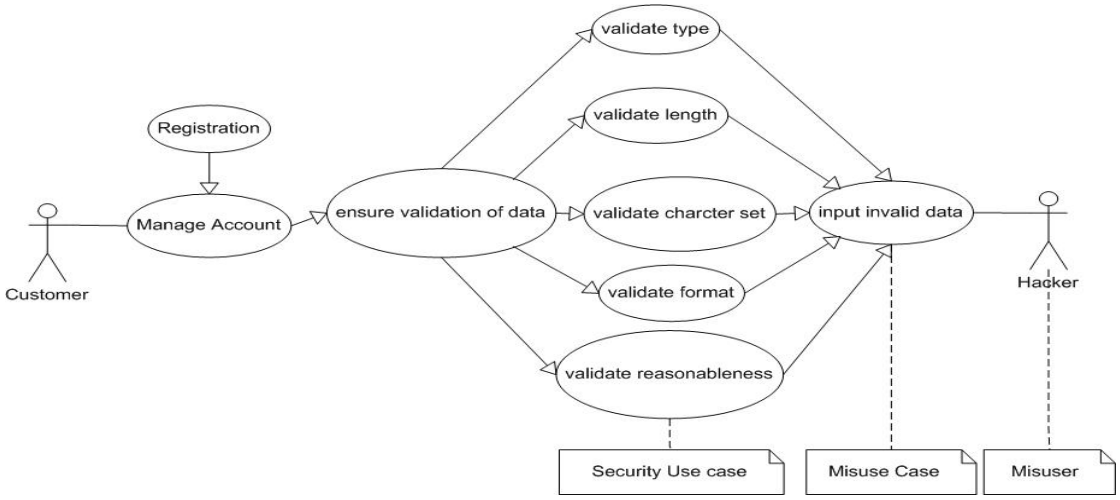


**Figure 4, Input Validated Registration Use Case**

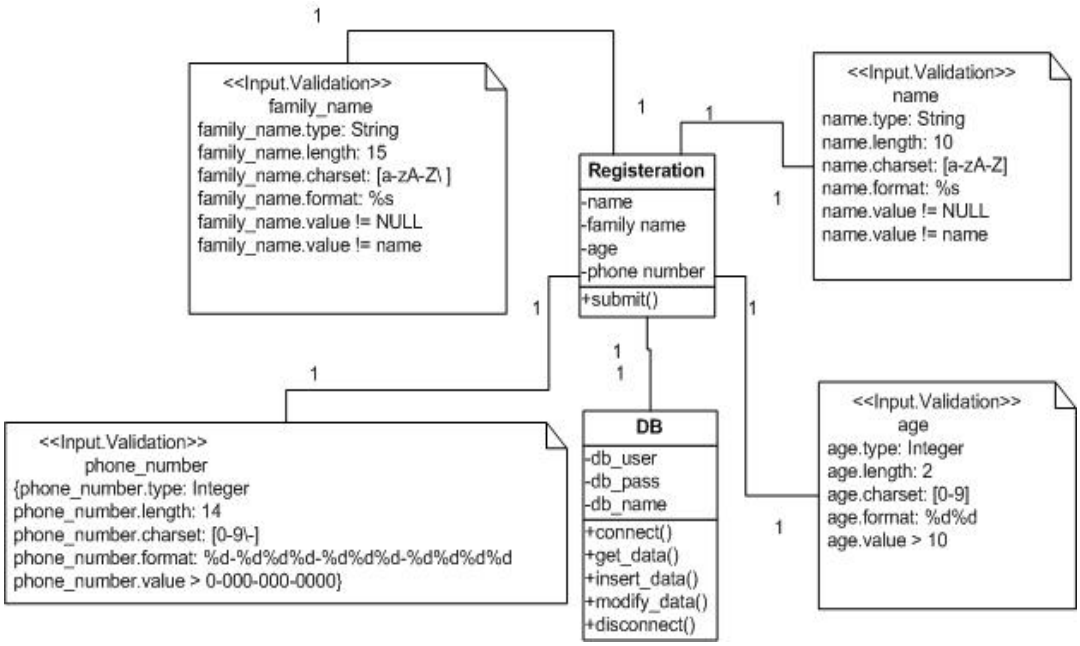


**Figure 5, Input Validated Registration Class Diagram**

### 4.5.2. Class Diagram

Registration class interacts with system users therefore it must be secure with "input validation constraint". DB class does not have any interaction with system users, so it does not need it. *Name, family name, age* and *phone number* are valued from outside. So these variables must

be constrained by FPSIVA. Figure 5 shows sample input validated registration class diagram.
For example we have following constraint for Name entity:
• Has string as type
• Has 10 characters long.
• Character set contains lower case and upper case Latin alphabet.
• Format is combination of character set without any restriction (like: John, Ali ...).
• Must not be empty and the same as family name.
And so on.

#### 4.5.3. Activity Diagram

As you see in Figure 6, at first for the name, input validation process checks all constraints which we show above the "*check name*", if all input data's attributes conform to constraints, input data is valid and, we show it by "*[valid]*" guard on arrow. By this valid data go to other part to check for family name and the other process is the same for age and phone number. But if it does not conform to all constraints it is invalid which we show by "*[invalid]*" guard on arrow, and the system rejects data to check it again. After all these checks are done, if it can pass all security constraints it means that input data is valid and the user will be registered.

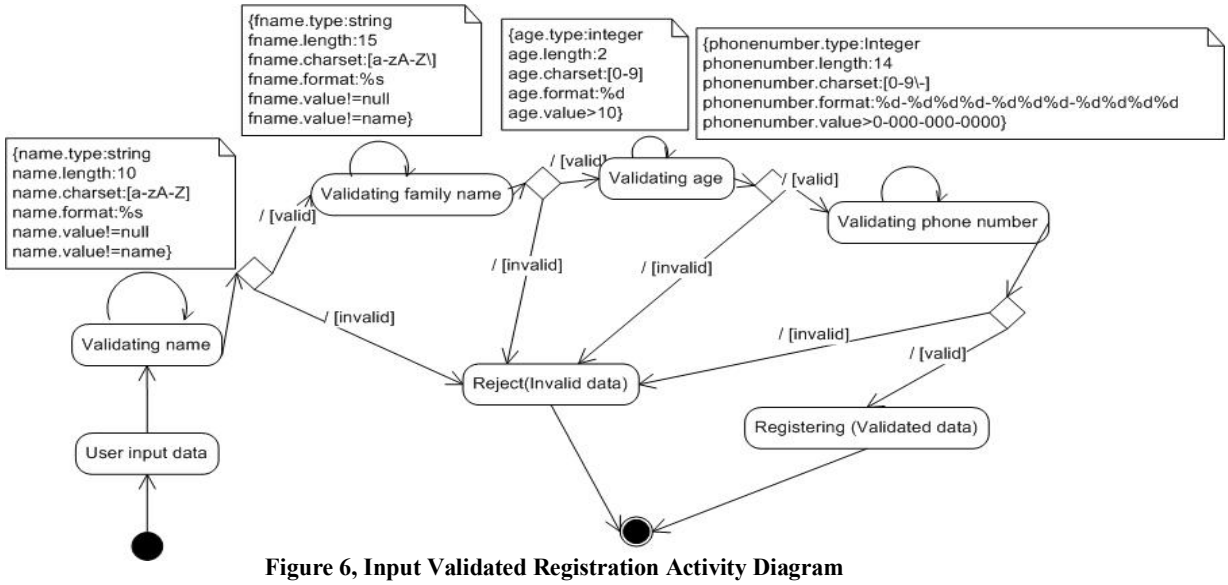


**Figure 6, Input Validated Registration Activity Diagram**

## 5. Discussion

As described in section 3.2, we used regular expression in order to define set of known legitimate data. Defining good regular expressions which make clear distinguish among malicious data and legitimate data are very important. Sometimes bad-defined regular expressions can allow malicious data flow in software that may cause dangerous security flaws or on the other hand, disallow legitimate data to flow on software that cause user frustration. Hence, in future study an improved model can be suggested in which decrease input validation dependencies on good-defined regular expression. Also, by doing numerical analysis, classifying more attacks and testing input validated software, these five primary security input validation attributes can be reduced or other attributes would be employed that are more robust and efficient.

## 6. Conclusion and Future Works

An overall aim of the work presented in this paper is to provide an integral guideline for software developers to create more secure software and we introduce an approach which is based on input validation to improve the integration of security details into UML diagrams using the OCL.
Input validation is part of security engineering policies. The integration of security engineering into a model driven software development approach has some advantages:

• We can address security requirements at a high level of abstraction.
• We can use the information of the model to recognize the design errors and correct them.
• We save more budget.
Indeed, modeling input validation in UML has some advantages such as:
• Preventing from common input tampering attacks include: forced browsing, remote command injection, cross site scripting, buffer overflows, format string attacks, SQL injection, cookie poisoning, and hidden field manipulation.

• Having both security and convenience in software at high level of abstraction.
• Ability of solving the problem of weak security background for developers.

Input validation in UML is not addressed previously. In this paper we incorporated input validation into UML diagrams such as use case, class, sequence and activity and illustrated a simple example to show how our methodology can be applied. Thus, the advantages mentioned make our approach an extremely useful approach and an ideal way to design more secure software in the future.

Regarding this study, in future research, one can decrease the number of input validating steps using numerical analysis methods and by so doing, the efficiency of this method will be improved.

Also, a researcher may try to discuss output validation or improve this method through combining it with output validation

## References


[1] DE VRIES, S. January 16, 2006. *A Modular Approach to Data Validation* [online]. [Accessed 1st September 2007]. Available from World Wide Web:<http://research.corsaire.com/whitepapers/060531-security-testing-web-applications-through-automated-software-tests.pdf >.

[2] HANSEN, R. J., PATTERSON, M. L. July, 2005. Guns and Butter: Towards Formal Axioms of Input Validation. In: *BlackHat USA 2005 Conference*, July 23-28 Las Vegas.

[3] Object Management Group (OMG). April 30, 2004. UML 2.0 OCL specification [online]. [Accessed 29 August 2006]. Available from World Wide Web < http://www.omg.org/docs/ptc/03-10-14.pdf>.

[4] RAY, D., MANA, A., YAGUE, M. 2004. Integration of Security Patterns in Software Models based on Semantic Description. In: *7th UML Conference, UML 2004*, October 10-15, 2004, Lisbon, Portugal.

[5] FIRESMITH, D. 2004. Specifying Reusable Security Requirements. In: *Journal of Object Technology*, vol. 3, no. 1, January-February 2004, pp. 61-75.

[6] WU, Y., OFFUTT, J., DU, X. 2004. Modeling and Testing Dynamic Aspects of Web Applications. In: *Computer Software and Applications Conference, 2004, COMPSAC 2004, Proceedings of the 28th Annual Internationa.* Vol. 2, 28-30 September 2004, pp. 106 – 109.

[7] CHARPENTIER, R., SALOIS, M. 2003. Security Modeling for C2IS in UML/OCL. In: *8th ICCRTS,* 17-19 June 2003, Washington DC.

[8] Object Management Group (OMG). March 2003. Unified Modeling Language specification [online]. [Accessed 29th May 2006]. Available from Word Wide Web: < http://www.omg.org/docs/formal/03-03-01.pdf>.

[9] FIRESMITH, D. 2003. Security Usecases. In: *Journal of Object Technology*, Vol. 2, No. 3, May - June 2003, pp. 53-64.

[10] OFFUTT, J. 2002. Quality attributes of web software applications. In: *IEEE Software: special Issue on software Engineering of Internet software*, March- April 2002. Published by: *IEEE Computer Society*, Vol. 19, Issue 2. pp. 25-32.

[11] BROSE, G., KOCH, M., LOHR, K. P. November 2001. Integerating Security Policies Design into the Software Development Process. Institut für Informatik, technical report, B-01-06. Available from Word Wide Web: <http://www.inf.fu-berlin.de/inst/ag-ss/papers/TR-B-01-06.ps >.

[12] LODDERSTEDT, T., BASIN, D., DOSER, J. 2001. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In: *Proceedings of the 5th International Conference on The Unified Modeling Language table of contents, Lecture Notes In Computer Science,* Vol. 2460, pp. 426 – 441.

[13] MARTIN, R. C. April 1998. UML Tutorial: Sequence Diagrams [online]. [Accessed 20 June 2006]. Available from World Wide Web: <http://www.objectmentor.com/resources/articles/UMLSequenceDiagrams.pdf >.

[14] GROSSMAN, J. WhiteHat Security industry. Statistical report [online]. [Accessed on 10th February 2007] Avaiable from Word Wide Web: <http://www.whitehatsec.com/press-releases/050905.shtml>.

[15] FRIEDL, J.E. F. 2006. Mastering Regular Expressions. Published by: *O'Reilly*, ISBN: 0596528124.

[16] The Open Web Application Security Project (OWASP). Top ten projects [online]. [Accessed on 17th April 2007]. Available from Word Wide Web: <http://www.owasp.org/index.php/OWASP_Top_Ten_Project>.

[17] JURJEN, J. 2002. UMLsec: presenting the profile. In: *6th Workshop on Distributed Objects and Components Security*. March 18-21, 2002.

[18] Web Application Security Consortium (WASC). SQL Injection WASC Threat Classification Entry [online]. [Accessed on 7 September 2007]. Available from World Wide Web: <http://www.webappsec.org/projects/threat/classes/sql_injection.shtml>.

[19] PETERSON, M. J., BOWLES, J. B., EASTMAN C. M. 2006. UMLpac: An Approach for Integrating Security into UML Class Design. In: *SoutheastCon, 2006, Proceedings of the IEEE,* March 31 - April 2, 2006 pp. 267 - 272