Secure Software Development with Coding Conventions and Frameworks

Takao Okubo, Hidehiko Tanaka

***Abstract*—It is difficult to apply existing software development methods to security concerns. Especially security testing is hard. We have been concerned with the fact that the restriction of implementation affects the easiness of testing. In this paper we propose a decision process of the coding conventions for security, regardful for testing security. Then, we apply our method to preventing injection attacks on Web application programs, and produce coding convention set against injection attacks. However, these conventions are not enough to prevent all kinds of attacks. So we also discuss the security frameworks which complement the conventions.**

***Index Terms*—security, programming, coding conventions, frameworks**

I. Introduction

EXISTING software development methods, such as waterfall model [1], UML [2] and various testing methods, have succeeded at maintaining the quality of software products, except security. Almost everyday SecurityFocus [3] reports vulnerability of some software products/systems. Incidents like information disclosure often make headline news because of software vulnerability. Now, we know *what vulnerability is* well. It is a buffer overflow, SQL injection or cross-site scripting (XSS). But we don't know *how to* make software with no vulnerability. The reason of these failures is that the uniqueness of *security* inhibits the direct application of existing development methods. With regard to the design phase, most of the security specifications are not functional, so it is not easy to describe them with usual design model like UML [1]. Same applies to the testing. Test planners have to care about the side-effect of the vulnerability, unlike usual software bugs [5], [6].

The most effective security testing method is the black box testing [7], [8]. Equivalence partitioning and boundary value analysis [9], [10] are usual software black box testing methods for enough coverage of testing data. But they are not useful for most of security testing because it is difficult to prepare testing data with enough coverage.

Some programming tips for avoiding vulnerability are known [11]. *Sanitizing* is one of the tips for preventing injection attacks such as XSS. Almost of these tips are ad hoc and ad hoc implementation often makes some omission. Additionally, they do not provide the way of testing. We need exhaustive and easily testable implementation.

ISO/IEC 15408, also known as Common Criteria (CC) [12] is a standard for building secure software. It provides the evaluation process of implementation and testing of software products, as the process of specification. However, CC does not give practical implementation and testing methods for each software product.

Existing security technologies and researches are not enough to control software security through the development lifecycle. Our goal is let the security requirement be achieved and verified like other software requirements, at the each phase of software development. We adopt *secure software engineering* approach for our goal. We start this approach by trying to apply using current software engineering methods to security concerns. Current methods might turn out to be insufficient. Then we would consider something new methods to complement them.

Focus of this paper is on the implementation phase of development lifecycle. We propose a coding convention decision process for security, considering testability. Next we apply our process to decide the proper coding conventions to prevent three types of injection attacks –SQL injection, OS command injection and XSS–, known as famous security threats at Web applications. Actually, even with the coding conventions and existing frameworks are not enough to avoid threats completely. We discuss what these frameworks should be, and propose some new security frameworks.

II. Deciding Coding Conventions for Security

A. Effect of Coding Conventions on Security

Coding conventions are some rules for writing program source codes [13]. They have been used at the programming phase for years. Most of currently used conventions are about coding style (indentation, naming, etc.) and mainly aim at readability and maintainability. With regard to security, coding conventions have hardly been adopted in the software development fields. We would like to insist that coding conventions are also useful for security, but they should play

Manuscript received December 17, 2006.

Takao Okubo is with Fujitsu Laboratories ltd., Kawasaki, Japan (corresponding author to provide phone: +81-44-754-2683; fax: +81-44-754-2666; e-mail: okubo@jp.fujitsu.com). He is also with Institute of Information Security, Yokohama, Japan.

Hidehiko Tanaka is with Institute of Information Security, Yokohama, Japan (e-mail: tanaka@iisec.ac.jp).

[1] UMLsec tries to describe security with UML [4].

different role from existing conventions.

We have pointed out the importance of restriction of implementation, because it affects the easiness of testing security requirements. We have proposed the early security cost estimate method using limited implementation options and testing methods related with the implementation [14]. Fig. 1. shows the architecture of the proposed method. Our research illustrates that coding conventions are useful for building secure software. Coding conventions for security have following two characteristics:

--They aim at security quality directly, rather than maintainability or readability.

--Therefore, they require strong restriction, which might limit not only flexible coding but also some functional availability.

The latter are supposed to conflict the feasibility of other software requirements. So we need care to decide conventions for security. Next we propose the decision process of coding conventions for each security requirement.

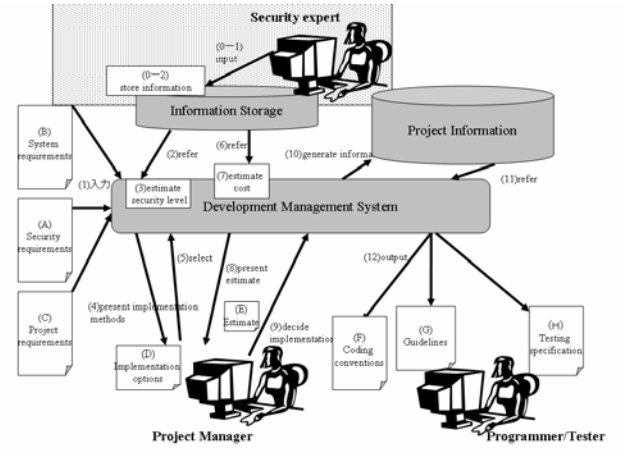

Fig. 1. Architecture of the proposed method.

B. Coding Convention Decision Process

Since we regard the verification of security as important, we prioritize the coding conventions which can be tested more easily. The decision process of conventions for every security requirement consists of two parts: system-independent decision and system-dependent one.

1) System-independent decision: This part is to decide general set of convention options so as to enable at least one of them adopted in various application systems. This process is supposed to be executed once, by persons with sufficient security literacy. The following is its detailed procedure.

a) *Definition of the Target Security Requirement:* The target Security requirement is defined here. For example, "The program must prevent SQL injection.".

b) *Definition of the Security Specification:* Security specification which fulfills the requirement is defined. This part is important for making conventions with no omission, so the persons in charge of this must be carefully. It still requires some security and programming knowledge.

c) *Extraction of Implementation Patterns:* Implementations of security specification, which are the candidates of conventions, are extracted. As various as possible implementation patterns are preferable. At first the direct implementation of the specification should be extracted. Next, indirect implementations, which do not aim at the original specifications directly, but achieve the specification consequentially, should be chosen.

d) *Selection/ Making order of Precedence:* Convention options are finally fixed here. Extracted implementation patterns should be selected, and ordered with the following valuation basis.

--Feasibility of testing observance of the convention

--Accuracy of testing

--less conflict with other software functions

2) System-dependent decision: To select convention(s) from 1) options for each development project/ system. This process should be available to persons with poor security literacy. The persons should examine each convention option with the following viewpoints:

--less conflict with other functions of the target system

--lower cost for testing

III. Coding Conventions for Injection Attacks

We applied the proposed decision process to some actual security requirements, and tried to decide the generalized coding convention options. This paper presents the application to the prevention of Injection attacks such as SQL injection, OS command injection and XSS.

A. Definition of the Requirement

An Injection attack is executed with injecting unanticipated data as input of the target program by an attacker. The target program sends a command to another module, as a database management system, OS or a Web browser. The command includes user input data usually as its parameters. If the command is changed by the input data to the unanticipated and harmful which do bad thing like information disclosure or tampering, the program has vulnerability against injection attacks. Therefore the security requirement can be defined as:

"The program is required not to generate the unanticipated command even with any user input data."

B. Definition of the Specification

Injection attacks can be classified into two types.

Type A): Attacks using the input data which change the command syntax. They aim at security quality directly, rather

than maintainability or readability. Fig. 2. shows the typical example of this type. SQL syntax can be changed by the input "' OR A=A", so the attacker can avoid user authentication without obtaining the password

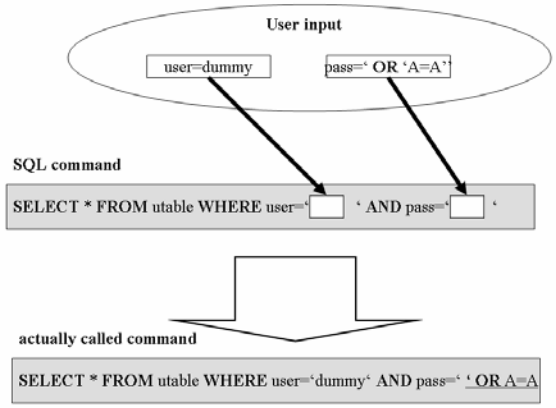

Fig. 2. The Example of Type A) injection. This is a typical SQL injection pattern. The input value(password) changes the syntactical structure of the SQL command.

parameters, and do not change the command syntax. Input of unanticipated database table name is an example of this type.

In this paper, we mainly discuss the injection of *Type A)*. Because most of the known injection attacks belong to *Type A)*, and *Type B)* problem can be treated as the current software specification.

Then, the specification can be defined as the following:

"The syntactical structure of the command should not be changed by the user input."

C. Extraction of Implementation Patterns

We have examined the specification, and devised the following patterns of restriction policy:

1) Distinguish dynamic elements of the command such as parameters, variables, from static elements like reserved words. When constructing the command string, be sure that the static elements do not involve data originally from user input. And before the command is constructed, dynamic elements of the command must be sanitized, in other words, even if they include a reserved word, the string should be escaped or eliminated so that it does not work as the reserved word.
2) Be sure that the command string must be composed of only the fixed values like constants.
3) Prohibit the functions/methods which invoke other module by forwarding the command without sanitizing.

Concrete Implementation patterns based on above policies vary depending on the programming language. Let us consider the coding conventions with Java. JDBC provides two types of class invoking SQL databases. One is java.sql.PrepaedStatement (and its subclass, CallableStatement), and the other is java.sql.Statement. PreparedStatement class [15] distinguishes parameters from others, and sanitizes the parameter string. So with regard to PreparedStatement, we only have to take care that the strings except parameters.

Table I shows the Java coding convention candidates (implementation patterns) to achieve the specification.

TABLE I
CONVENTION CANDIDATES AGAINST INJECTION ATTACKS

No.	Convention Candidates
(a)	Prohibit using the value originally from user input for parameters of the methods setting SQL statements (*).
(b)	Be sure to sanitize the parameters of (*) methods, if the methods do not sanitize them.
(c)	Be sure that the parameters of (*) methods use the value originally from constants or literal strings only.
(d)	Be sure that the parameters of (*) methods use the constants or literal strings only.
(e)	Prohibit the use of (*) methods

(*) Methods setting SQL statements are:
java.sql.Statement#executeXXX(): first argument,
java.sql.Statement#addBatch(): first argument,
java.sql.Connection#prepareStatement(): first argument,
java.sql.Connection#prepareCall(): first argument.

D. Selection/ Making order of Precedence:

The extracted candidates are examined by the feasibility/ accuracy of testing and less conflict with other functions.

Policy 1), 2) and Java convention (a), (b), (c) requires dataflow analysis for testing. Policy 3) and (d), (e) requires syntax analysis. Generally dataflow analysis is feasible, but its accuracy is lower than syntax analysis. The order by the probability of conflict is 3) > 2) > 1) and (e) > (d) > (c) > (b) > (a). Therefore we recommend the Java convention in the order of (d) > (e) > (a) > (c). Recommendation order of (b) cannot be determined because its accuracy depends on the validity of the sanitizing code.

IV. EVALUATION AND DISCUSSION

In this section we evaluate the coding convention options decided in section III. First, we verify the conventions are feasible for practical application programs. We clarify a limit of conventions, and consider another approach with *frameworks*.

A. SQL Injection:

1) Evaluation of Conventions

We have verified the feasibility of the proposed conventions with the open source program codes found by Bugle [16]. Bugle can find the source files with the suspicion of various bugs, including SQL injection. We have examined 185 files. In About 84% of the files, Invocation of the SQL can be written with the fixed String, or with the parameterized PreparedStatement. So they can adopt the Java convention (d) (Table I.). 8% files are the programs like SQL client. These programs permit arbitrary SQL invocation, which cannot exclude the SQL injection

by nature. In the rest of the files, the SQL command changes dynamically by the number of loop iteration or conditional branch. Fig. 3. shows the example.

This kind of coding is appeared in the programs which have to build complicated search conditions. In this case, PreparedStatement with fixed String cannot be used. So Project Managers of such system have to adopt convention (b) or (c).

2) Discussion of Frameworks

.Net, Perl and some programming languages provide the prepared statement mechanism like PreparedStatement. A distinct advantage of this mechanism is that programmers do not have to take care of sanitizing: what are the dangerous characters and how to process them. However some application programs cannot use the prepared statement if the SQL command strings have to be changed dynamically. The programmers have to code sanitizing routines by themselves, and testers have to use more complicated and inaccurate testing.

```
Statement stmnt;
Srting query = "SELECT *
                FROM table=xxtbl
                WHERE id="
                 + request.getParameter("ID");
for int (i=1; i< keys.length; i++) {
   query+= " AND "keys[i] + "=" + values[i];
}
   stmnt.executeQuery(query);
}
```

Fig. 3. The code example in which the SQL command changes dynamically.

We consider that the secure framework should provide all the sanitizing methods for all invocation, so that the sanitizing algorithm is hidden from the programmers. So we propose the classes shown in Fig. 4. for the parts of the secure framework. Although the classes are written with Java 5, they can be migrated to other languages.

PreparedStatement is usually a fixed string, but SecureStatement (Fig. 4. (a)) constructs the SQL query string of PreparedStatement at each query execution. Programmers add the string using the add() method, but the method distinguishes parameters from the fixed values internally. Even if the attackers input the data like "' OR A=A", the string is not defined as the fixed value, so it is treated as a parameter, and then sanitized inside PreparedStatement class. ReservedSQL class (Fig. 4. (b)) is used to identify the reserved words. Programmers may define the fixed parameter such as table name as enum, like ReservedSQL. It is also useful for preventing *Type B)* injection.

If these classes are included into the framework, the coding conventions will become simpler. The coding convention (d) will be enough for all programs.

```
import java.sql.*
import java.util.*;

public class SecureStatement {
  static final PLACE_HOLDER "?";
  private StringBuffer stmnt;
  private ArrayList params;
  private Connector conn;

  SecureStatement(Connector conn) {
    stmnt = new StringBuffer();
    params = new ArrayList;
  }

  public void add(Object arg) {
    if(arg instanceof enum) {
      stmnt.append(arg.toString());
    } else if(arg instanceof String) {
      params.add(arg);
      stmnt.append(PLACE_HOLDER);
    } else {
      throw new IllegalTypeException();
    }
  }

  public void execute() {
    PreparedStatement pstmnt =
conn.prepareStatement(stmnt);
    for (int i = 0; i < params.size(); i++) {
      setObject(params.get(i));
    }
    pstmnt.executeQuery();
  }
};
```

Fig. 4. (a) SecureStatement class. It constructs the temporal prepared statement internally.

B. OS Command Injection

1) Evaluation of Conventions

A method for execution of external OS command, Runtime#exec() [17] does not invoke shell. So the injection attacks adding the shell scripts to the command do not work.[2] Furthermore, java provides Runtime#exec() which uses the string array as its argument. So attackers cannot change the number of syntax element. The mechanism makes the program considerably safe, but not completely. Attackers may input the attack command which consists of the same number of elements. Runtime#exec() with string array cannot prevent this kind of attack.

2) Discussion of Frameworks

In order to make the program secure against OS command injection, a mechanism which distinguish parameters from static elements is necessary. The ideal framework is supposed to analyze the syntax of input data, and identify parameters like SecureStatement. However, analyzing all the syntax of OS and user command is not realistic. If the program can limit the kinds of commands, the reserved words which are allowed to use, can be defined as constants like ReservedSQL.

[2] If you invoke "/bin/sh", "-exec" you can execute the shell script including pipe and redirection.

TABLE II
HTML OUTPUT METHODS WITH JAVA (PART)

Method	function	Argument	Sanitizing rule
outputText()	output text	String text	escape < > &
outputURL	output url attribute	String tagname	tagname should be a fixed value
		String url	url should be a fixed value
outputAttribute()	output attribute	String tagname	tagname should be a fixed value
		String attribute	attribute should be a fixed value
outputEvent()	output event attribute	String tagname	tagname should be a fixed value
		String attribute	attribute should be a fixed value
outputScript()	output javascript	String script/ parameter	script should be a fixed value parameter should be a fixed value

Above approach is also useful for other languages which invoke OS shell directly.

C. XSS

1) Evaluation of Conventions

XSS have the same characteristics with other injection attacks. All you need to do is consider the HTML structure of the response, in spite of SQL command/ OS command.

Basic idea of the proposed conventions is also useful for XSS, but we have to be careful for sanitizing, since the sanitizing manners must vary depending on the context. If the target data are output in the text area like body text, you should escape three characters. <, >, &. If they are output as the tag attribute value of HTML, you should escape <, >, &, “, ‘. If URL attribute, the data must be URL-formed, and so on.

The right output method/ function must be used according to the context.

```
public enum ReservedSQL {
  SELECT("SELECT"),
  INSERT("INSERT"),
  UPDATE("UPDATE"),
  DELETE("DELETE"),
  CREATE("CREARE"),
  DROP("DROP"),
  WHERE("WHERE"),
  AND("AND"),
  OR("OR"),
  BLANK(" ");
  QUOTE("\'");
.....

   private String name;
  private ReservedSQL (String name) {
   this.name = name;
  }

  public String toString() {
   return name;
  }
}
```

Fig. 4. (b) ReservedSQL class. It is used to identify the reserved words.

2) Discussion of Frameworks

If there is no framework, the solution is rather simpler, because programmers can manipulate all the construction of response HTML. Secure Framework is desired to prepare a method / function for each output context. Table. II. shows the example of the Java output methods.

Some frameworks make the situation a little more complicated. Frameworks like Struts [18] prepares the response JavaServer Pages (JSP) [19] file, so the programmer cannot know directly on which part of the JSP file they are trying to output data. Syntax Analysis of JSP is necessary to know the context of the writing data.

JSP has some other problems:

--Expression Language directly outputs the value. You’d better prohibit the use of EL.

--You’d better prohibit the use of the scriptlet , because of the same reason as EL.

--May be you should not allow the use of custom tags until its safety is guaranteed.

V. CONCLUSION

The importance of coding conventions for security has hardly insisted ever. In this paper we showed it is useful for secure software development, and proposed the convention decision process considering the testability and functional conflict. Then, we developed general coding convention options for injection attacks based on the proposed decision process. Next, we evaluated the feasibility of the coding conventions. Because of their common characteristics, the proposed convention options are basically useful, but they must be customized a little. We also proposed the desired security function of framework which complements the flaw of the conventions. Please notice that our *secure framework proposal* is not only the programming tips like others. Framework has close connection with the coding conventions. They complement each other. So when you have to get the program secure you have to consider both of them. Furthermore, framework and conventions are useful not only for implementing the security, but also for easy testing.

We have noticed that the existing frameworks with security function do not always contribute to the security quality. The

reason is the framework design lacks the concept of testability.

We hope the designers of the secure framework take this concept into consideration.

As the future work, we are going to develop another conventions set for other software requirements. And we are also planning to do the empirical evaluation of the proposed coding conventions and frameworks.

REFERENCES

[1] W. Royce, “Managing the Development of Large Software Systems”, Proceedings of the IEEE WESCON, IEEE Press & Proceedings of the Ninth International Conference on Software Engineering, IEEE Press, 1970.
[2] Unified Modeling Language, Object Management Group, http://www.uml.org/.
[3] SecurityFocus, http://securityfocus.org/.
[4] J. Jürjens, Secure Systems Development with UML, Springer, 2004.
[5] B. Potter and G. McGraw, "Software security testing", Security & Privacy Magazine, IEEE Volume: 2 Issue: 5 Sept.-Oct, 2004, pp. 81- 85.
[6] H. H. Thompson, "Why security testing is hard", Security & Privacy Magazine, IEEE Volume: 1 Issue: 4 July-Aug. 2003, pp. 83- 86.
[7] G. McGraw, "Testing for security during development: Why we should scrap penetrate-and-patch", IEEE Aerospace and Electronic Systems, 1998.
[8] C. Weissman, "Penetration Testing, Information security: an integrated collection of essays, IEEE Computer Society Press, Silver Springs, MD, 1995.
[9] C. Karner, J. Falk and H. Q. Nguyen, Testing Computer Software Second Edition, International Thomson Computer Press, 1993.
[10] B. Beizer, Software Testing Techniques, 2nd Edition, Van Nostrand Reinhold, 1990.
[11] Secure Programming.com, http://secureprogramming.com/
[12] Common Criteria for Information Technology Security Evaluation v2.3, http://www.commoncriteriaportal.org/public/developer/index.php?menu=2, 2005.
[13] Code Conventions for the Java Programming Language, Sun Microsystems, http://java.sun.com/docs/codeconv/.
[14] T. Okubo, N. Nakayama, Y. Wataguchi and H. Tanaka, "A Study on Software Development Method which Fulfills Specified Security Requirements" Computer Security Symposium 2006, pp.387-392.
[15] java.sql.PreparedStatement, Sun Microsystems, http://java.sun.com/j2se/1.4.2/docs/api/java/sql/PreparedStatement.html.
[16] Bugle , http://www.cipher.org.uk/index.php?p=projects/bugle.project.
[17] java.lang.Runtime, Sun Microsystems, http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Runtime.html.
[18] Struts, the Apache Software Foundation, http://jakarta.jp/struts/.
[19] JavaServer Pages, Sun Microsystems, http://java.sun.com/products/jsp/.