Secure Software Development through Coding Conventions and Frameworks

Takao Okubo, Hidehiko Tanaka
Fujitsu Laboratories ltd., Institute of Information Security
okubo@jp.fujitsu.com, tanaka@iisec.ac.jp

Abstract

It is difficult to apply existing software development methods to security concerns. Using software for security testing purposes, in particular, is hard to do. The fact that there is a restriction on the implementation of software affects the ease with which security can be tested. In this paper we propose a decision process of coding conventions for security, mindful of testing security. Then, we apply our method to preventing injection attacks on Web application programs, and establish some coding conventions that can be used against injection attacks and cross site scripting. We also discuss security frameworks, which are also useful as conventions.

1. Introduction

Software engineering technology has been researched for decades, and is steadily improving. Large numbers of development methods, such as a waterfall model [1], UML [2] and various testing tools, have been developed, and some of these are already being applied to actual development fields. These methods may not be perfect, but at least in the development of large systems, there seems to be a consensus that software that has been built and tested using some methods or tools is deemed to be of sufficient quality. However, software engineering for security purposes has not yet reached such a level. Almost every day SecurityFocus [3] reports on vulnerabilities with some software or systems. And incidents such as information leaks that have occurred because of software vulnerabilities often make headline news. Of course one of the reasons why these problems occur is that not everyone is well aware of such vulnerabilities. But even if software developers notice such vulnerabilities, it is still not easy to eliminate the vulnerabilities completely. The developers may not know how to build software that is invulnerable. The reason for these failures is that the uniqueness of security makes it difficult to directly apply existing development methods. With regard to the design phase of software, most of the security specifications are not functional, so it is not easy to describe such specifications with conventional design models like UML[1]. The same applies to testing. Test planners have to care about the *side effects* [5] of vulnerabilities, unlike usual software bugs [6].

Black box testing is said to currently be the most effective security testing method [7], [8]. Equivalence partitioning and boundary value analysis [9], [10] are the usual software black box testing methods that are used provided there is a sufficient amount of testing data with enough coverage. But these methods are not useful for most types of security testing because it is difficult to prepare testing data with enough coverage.

There are some known programming tips for avoiding vulnerabilities [11]. “Sanitizing” is one way to prevent attacks such as XSS. Tips are useful as a guide for implementing some security functions. But special care needs to be taken to avoid assuming that such tips assure security, unless they are managed and have been properly tested. There is a need for collectively exhaustive and easily testable ways to ensure security.

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 a process of specification. However, CC does not give practical implementation and testing methods for each software product.

Existing security technologies and research are not enough to control software security through the software’s development life cycle. Our goal is to find such control methods. Our approach is to use a secure software engineering, and we started this approach by

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

trying to apply current software engineering to security concerns.

This paper focuses on the implementation phase of software development life cycle. We use two current software engineering methods to achieve our goal: coding conventions and software frameworks. At first, 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[2]—all well-known security threats to Web applications. Then, we discuss what kind of framework can be used to efficiently prevent such attacks, and propose some new security frameworks.

2. Coding Conventions and Frameworks for Security

2.1. Effect of Coding Conventions on Security

Coding conventions are some rules for writing program source codes [14]. They have been used in the programming phase of software development. Most of the currently used conventions are about coding style (indentation, naming, etc.) and they mainly aim to make software code readable and maintainable.

However, coding conventions have hardly been adopted in an effective way in the software development field, with regard to security. Coding conventions are also considered to be useful for security, but they should play a different role from the existing conventions.

We have proposed a security cost estimate method in the early phase of software development, using limitation of implementation and testing methods related to implementation [15]. As mentioned in the introduction, it is difficult to test security for two reasons:

1) With black box testing, the way the item works is hidden from the testers. So the test case should have sufficient variation considering the various kinds the implementation patterns. It makes testing complicated.

2) Black-box testing tools, in particular, have a high false positive rate, so human knowledge and input from security experts are required to remove false positives from the test results.

[2] XSS is not usually categorized as an injection attack[13], but this paper treats XSS as a kind of injection attack because it shares some common characteristics with that kind of attack. See 5.3.

We will focus on limiting the way testing is implemented, in an attempt to solve these problems.

1) By limiting the implementation, we only need to consider whether the testing implementation method is allowed.

2) If the rule for limiting testing is adopted, those implementation methods that are potentially false positive can be treated as a violation of the rule. This makes testing easier.

Therefore, it is just conceivable that deciding on the proper coding conventions makes testing easier, and makes it possible to assure a certain level of security. Coding conventions for security have the following two characteristics:

- They aim to achieve a good level of security, rather than to make software code maintainable or readable.
- Therefore, they require strong restrictions, which might interfere not only with flexible coding but also with some functional availability.

The latter are supposed to conflict with the feasibility of other software requirements. So we need to take care when deciding on conventions for security.

2.2. Effect of Frameworks on Security

A framework is a structure that supports the development of software [16]. Part of it is provided as a set of programs and libraries. It is a common technology as coding conventions. Numbers of framework for various platforms have been released [17]. We think frameworks are also useful for security:

- Frameworks offer developers with libraries of security functions so that the developers do not need to program such functions by themselves.
- Frameworks can put restrictions on the way software is developed, just like coding conventions.

Some of the existing frameworks provide partial security functions. For example, Struts [18], a Java framework for Web application, has a **<bean:write>** tag which outputs the JavaBean value and avoids using the dangerous characters that are factor of XSS. However, their effect on security is only partial. Struts assure secure output by means of this <bean:write> tag, but it does not prohibit other custom tags from being used, or EL expression of JSP. In the same way as with coding conventions, if the restrictions placed on software development are too strong and other functions are limited this will make the frameworks useless. A proper degree of restriction is needed for security frameworks.

3. A Coding Convention Decision Process

Since we regard the verification of security as important, we prioritize those coding conventions that can be tested more easily. The decision process of conventions for every security requirement consists of two parts: a system-independent decision and a system-dependent one. Figure 1. shows the flow of the process. In the first part 3.1., several coding convention option sets for a security requirement are decided by some people (possibly security experts). The option sets are general (system-independent) and are reused to give choices to each system development project. In the second part 3.2., a manager of each development project chooses one or more conventions that fit the target system.

3.1. Deciding General Convention Options

This part is to decide on a general set of convention options so as to enable at least one of them to be adopted in various application systems. Its detailed procedure is as follows:

(1) Defining the Security Requirement:

A Security requirement is defined here. For example, "The program must prevent SQL injection."

(2) Defining the Security Specification

A security specification that fulfills the requirement is defined. It is not necessary to worry too much about how the specification is implemented. Instead, the specification should be defined precisely so as to ensure the requirement is achieved.

(3) Extracting Implementation Patterns

Implementations of security specification, which are the candidate conventions, are extracted. It is preferable to have a large number of implementation patterns. At first the most faithful implementation of the specification is extracted. Next, other implementations, which do not aim at the original specifications directly, but achieve the specification consequentially, should be chosen.

(4) Selection/Making order of Precedence

Convention options are finally fixed here. Extracted implementation patterns should be selected, and ordered using the following valuation basis:

- If it is to enable to test compliance of the convention?
- How high is the Accuracy of its testing?
- How low is the possibility of conflict with other software functions?

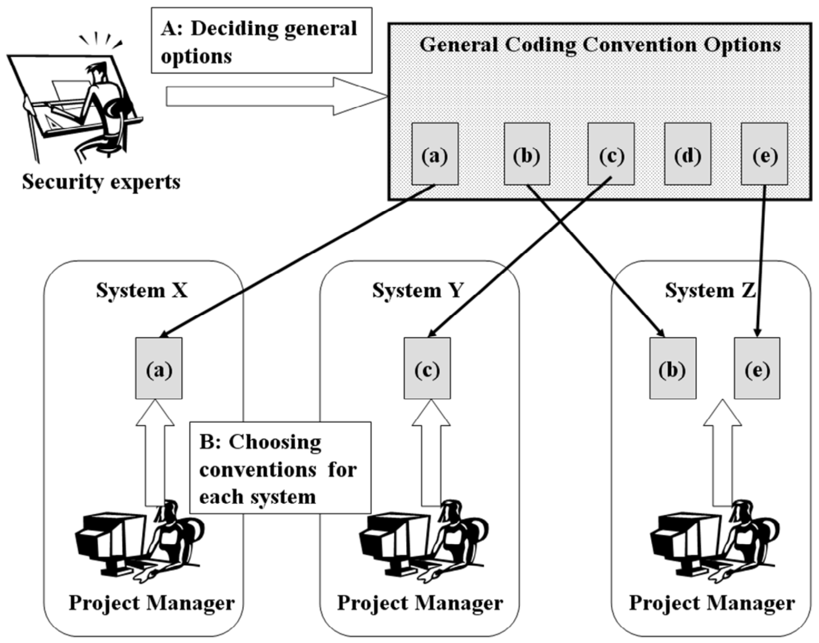

Figure 1. Process flow.

3.2. Choosing Conventions for each system

To select convention(s) from defined options for each development project/system. This process should be available to persons with insufficient security knowledge. The persons should examine each convention option with the following viewpoints:

- How low is the possibility of conflict with other software functions?
- How low is the cost for testing the convention?

4. Coding Conventions against Injection Attacks

We applied the proposed decision process to some actual security requirements, and tried to decide on the generalized coding convention options. This paper presents the application of coding conventions to the prevention of injection attacks such as SQL injection, OS command injection or XSS. We have assumed that there is a common coding convention set for all kinds of injection attacks, since they are based upon the same mechanism. The evaluation by applying the convention set to each kind of attack will be presented in section 5.

(1) Defining the Requirements

An injection attack is executed by injecting unanticipated data as an input to the target program. The target program sends a command to another external program, such as a database management system, OS or Web browser. The command includes user input data usually as its parameters. If the command is changed by the input data to an unanticipated or harmful one that causes some detrimental effect such as information leakage or tampering, the program is vulnerable to an injection attack. Therefore the security requirement can be defined as:

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

(2) Defining the Specification

Injection attacks can be classified into two types.

Type A): Attacks using the input data that change the command syntax. Figure 2. shows a typical example of this type. SQL syntax can be changed by the input "' OR A=A", so an attacker can bypass user authentication without obtaining the password

Type B): Attacks using the input only act as unanticipated parameter values, and do not change the command syntax. Input of unanticipated database table name is an example of this type.

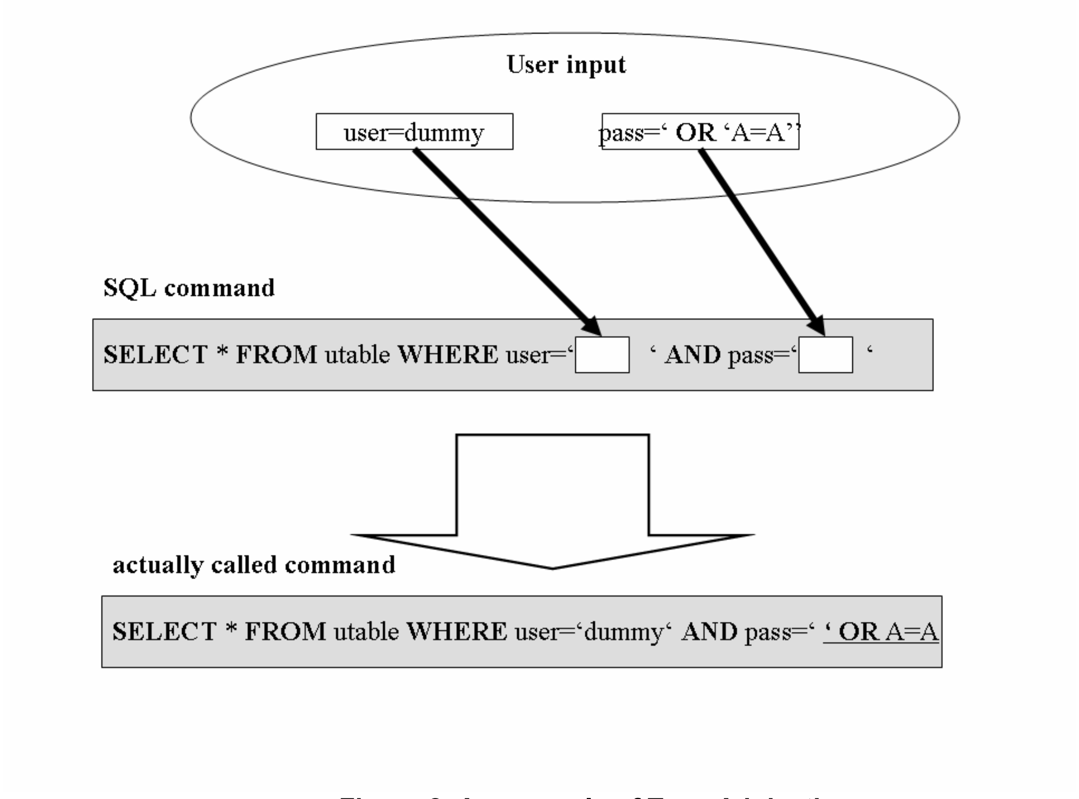
User input
user=dummy
pass=' OR 'A=A'
SQL command
SELECT * FROM utable WHERE user=' ' AND pass=' '
actually called command
SELECT * FROM utable WHERE user='dummy' AND pass=' ' OR A=A

Figure 2. An example of Type A injection.

In this paper, we mainly discuss the injection of *Type A)*. Because most of the known injection attacks belong to *Type A)*, and a solution of *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."

(3) Extracting Implementation Patterns

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

(I) Distinguish dynamic elements of the command such as parameters, variables, from static elements like reserved words. When constructing a command string, It must be ensured 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*. There are several methods to sanitize the input .

(II) Be sure that the command string must be composed of only the fixed values like constants.

(III) Prohibit the functions/methods that call other programs by forwarding the command without sanitizing.

**Table 1.
Convention Candidates against Injection Attacks**

No.	Convention Candidates
(a)	Prohibit using the value originally from user input for parameters of (*) methods that set the command string.
(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 values 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.

(*) for example, 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.

Concrete implementation patterns based on above policies vary depending on the programming language. We show the coding conventions with Java as an example. JDBC provides two types of classes calling SQL databases. One is java.sql.PrepaedStatement (and its subclass, CallableStatement) [19], and the other is java.sql.Statement. PreparedStatement class accepts SQL query strings such as the following:

```
SELECT * FROM utable WRERE id=? AND pass=?
```

PreparedStatement does not allow dynamic change of SQL queries, except the parameters that are assigned to the position where the character "?" is placed. It distinguishes parameters from others, and sanitizes the parameter strings. So, with regard to PreparedStatement, we only have to take care of the strings except parameters.

Table 1. shows the candidate Java coding conventions (implementation patterns) to achieve the specification.

(4) Selecting/Making order of Precedence:

The extracted candidate conventions are examined for the feasibility/accuracy of testing and less conflict with other functions.

Policy (I), (II) and Java convention (a), (b), (c) require dataflow analysis for testing. Policy 3) and (d), (e) require syntax analysis. Generally, dataflow analysis is feasible, but its accuracy is lower than syntax analysis. The order by probability of conflict is (III) > (II) > (I) and (e) > (d) > (c) > (b) > (a). Therefore we recommend the Java convention in the order of (d) > (e) > (a) > (c). We cannot determine the recommendation order of (b) because its accuracy depends on the validity of the sanitizing code.

5. Evaluation and Discussion of Frameworks

In this section we evaluate the coding convention options decided in section 3. First, we verify that the conventions are feasible for practical application programs. Then, we consider another approach in terms of *frameworks*.

5.1. SQL Injection:

5.1.1. Evaluation of Conventions

We have verified the feasibility of the proposed conventions with the open source program codes found by Bugle [20]. Bugle can find open source files that are suspected of containing various bugs, including

SQL injection. We have examined 185 files. In about 84% of the files, the vulnerable code calling the SQL with Statement class, can be written with a fixed string, or with a parameterized PreparedStatement. For these programs Java convention (d) (Table 1.) can be adopted. In addition, 8% of the files are programs that act as SQL Web client. These programs permit arbitrary SQL invocation, which cannot exclude the SQL injection by nature. Therefore we can disregard this type. In the rest of the files, the SQL command changes dynamically by the number of loop iterations or conditional branch. Figure 3. shows an example.

This kind of coding appears in programs that have to build complicated search conditions. In this case, PreparedStatement with fixed strings cannot be used. So project managers of such systems have to adopt convention (b) or (c), for which testing is less accurate than for (d).

5.1.2Discussion of Frameworks

PreparedStatement class in Java partly meets the requirement for a security framework. It offers the sanitizing library to programmers, and it also restricts the SQL query as the form of *parameterized prepared statement*. .Net, Perl and some other programming languages also have the same prepared statement mechanism as PreparedStatement. However, some application programs cannot use the fixed prepared statement if SQL command strings have to be changed dynamically (such as shown in Figure 3.). In order to make such programs secure, programmers have to code sanitizing routines by themselves, or use more complicated and inaccurate testing.

If a framework is able to provide automatic sanitizing for all kinds of commands that the specification requires, it will be an ideal *security framework*. So we propose the classes shown in Figure 4. as library classes for a security framework. The classes are written with Java 5, and they can be migrated to other languages.

```
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);
}
```

Figure 3. An example of code in which the SQL command changes dynamically.

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

With these classes, Figure 3. code can be rewritten as shown in Figure 5. If the user inputs three pairs of keys and values, the following prepared statement is created dynamically.

```
SELECT * FROM table=xxtbl WHERE id=aaa AND
key1=? AND key2=? AND key3=?
```

In this case SQL injection is protected by the

```
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.

```
public final 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.

prepared statement mechanism.

Next, consider the case in which a programmer adds a fixed element of SQL that is not defined as "enum". The added value is treated as the parameter inside SecureStatement class, so a SQL syntax error may occur when building a prepared statement. Therefore this case is not a cause for SQL injection vulnerability.

The coding convention (d) will be enough for all programs, if these classes are included into the framework, because even if a dynamic changing query is needed, programmers can construct the query as a fixed string with Figure 4. classes. All other SQL query calling classes can be replaced with SecureStatement class. So the coding conventions can be simpler, such as "*Use SecureStatement class only*", which also makes the testing easier.

In other programming languages that have a parameterized prepared statement mechanism, libraries such as Figure 4. classes are useful for security frameworks. In other languages without prepared statement mechanisms or sanitizing, such mechanisms must also be implemented in the security framework as Figure 4. classes.

5.1.3. Comparison with Other Measures

In this section we compare our convention and framework solution with other measures against SQL injection.

a) Advice or Convention to Use a Prepared Statement

```
SecureStatement stmnt;

// fixed part of the SQL query
stmnt.add(myConstant.FIXEDSQL);
stmnt.add(request.getParameter("ID"))

for int (i=1; i< keys.length; i++) {
stmnt.add(ReservedSQL.BLANK);
stmnt.add(ReservedSQL.AND);
stmnt.add(ReservedSQL.BLANK);
stmnt.add(keys[i]);
stmnt.add(ReservedSQL.EQUAL);
stmnt.add(values[i]);
}
stmnt.execute();
}
```

Figure 5. The rewritten code of Figure 3. with Figure 4. classes.

This advice is a popular measure for preventing SQL injection. As mentioned above, a prepared statement provides a parameterized structure and a sanitizing function, which are useful for preventing attacks. However, this measure contains the following two problems:

- A prepared statement cannot be applied to a program that requires dynamically changing SQL query (see 5.1.2.).
- If a programmer writes a program with improper usage of a prepared statement string, the program becomes vulnerable to SQL injection. An example of code containing such improper usage is shown in Figure 6. In Figure 6, the prepared statement string is composed of user input data that are not treated as parameters and thus not sanitized.

Our conventions and frameworks provide solutions for the problems above. For the first problem, Figure 4 classes enables programmers to write a dynamically changing SQL query execution program with PreparedStatement. For the second, coding convention (d) and Figure 4 classes limit each component part of the query string to a fixed value. So, coding like that shown in Figure 6. can be detected as an error with our frameworks.

b) Advice or Convention to Validate and Sanitize All the User Input Data

This advice is also popular as a). However with the viewpoint of testing, there are two problems.

- Dataflow analysis is needed to confirm the adherence of the convention.
- The safety-level of this measure depends not only

on making sure all user input data are validated and sanitized, but also on how they are validated and sanitized. It requires the programmers' precise knowledge of inappropriate characters.

For the first problem, the proposed convention set offers convention options that can be tested with the analysis easier than dataflow analysis (e.g. (d)). For the second, the Fig. 4 classes render the programmers' code of sanitizing unnecessary, since the framework does all of it. Even if neither the programming language nor the libraries have a sanitizing function, we propose that the security framework must provide the function.

5.2. OS Command Injection

5.2.1. Evaluation of Conventions

If there are command calling methods or functions that have parameterized mechanism such as a prepared statement, the convention (d) (Table 1.) can be adopted. Unfortunately, no such methods or functions are provided, so we have to adopt other conventions without any security frameworks.

Java is rather more secure than C, C++, PHP and Perl where some functions can execute the parameter string as a shell script. A Java method for execution of external OS command, Runtime#exec() [22] does not call a shell program. Therefore, in Java the typical OS command injection using shell script such as shown in Figure 7. does not occur[3] Furthermore, Java provides Runtime#exec() which uses the string array as its argument. Its mechanism prevents some kinds of injections to some degree, but does not prevent them completely because Runtime#exec() with string array does not distinguish between parameters and static elements.

5.2.2. Discussion of Frameworks

A mechanism that distinguishes parameters from static elements, such as a prepared statements, is needed for a security framework, if the program passes some dynamic data to an external program. The ideal security framework is supposed to analyze the syntax of input data, identify and sanitize parameters like SecureStatement. However, in order to achieve this, it needs to analyze not only the syntax of all the OS commands but also that of all the user commands, and this is not realistic.

If project managers can limit the kinds of commands

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

```
String query;
query = "SELECT * FROM utable WHERE user=";
query += request.getParameter("ID");
query += " AND password="
query+= request.getParameter("Password");

connector.prepareStatement(query);
query.execute();
```

Fig. 6. Improper usage of PreparedStatement.

that are supposed to be called by the program, the reserved words that are allowed to be used can be defined as constants like ReservedSQL.

5.2.3. Comparison with Other Measures

In the same way as with SQL injection, advice or conventions to validate and sanitize all the user input data is a popular measure for OS command injection. So if the security framework mentioned above is achieved, out conventions and frameworks have an advantage of testability and flexibility over the current advice.

5.3. XSS

5.3.1 Evaluation of Conventions

XSS has the same characteristics as other injection attacks. All we need to do is consider the HTML structure of the response, regardless of the SQL command/OS command.

The basic idea of the proposed conventions is also useful for XSS, but we have to be careful for sanitizing, since the manner of sanitizing varies depending on the context. If the target data are output in the text area like body text, you should avoid using the three characters. "<", ">", and "&". If they are output as tag

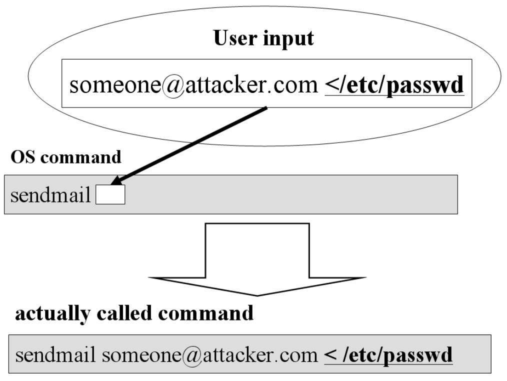
User input
someone@attacker.com </etc/passwd
OS command
sendmail
actually called command
sendmail someone@attacker.com < /etc/passwd

Figure 7. An example of OS command injection.

attribute values of HTML, you should avoid using "<", ">", "&", and ", ". If they are URL attributes, the data must be URL-formed, and so on.

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

5.3.2. Discussion of Frameworks

If there is no framework, the solution is rather simpler, because programmers can manipulate the entire construction of response HTML. A secure framework is required to prepare a method or function for each output context. Table 2. shows an example of the Java output methods.

5.3.3. Comparison with Other Measures

Some frameworks like Struts not only make Web application development easier, but also provide sanitizing functions. However such existing frameworks do not meet our requirements for *security framework*. Contrary to our expectations, they make the situation more complicated. Frameworks like Struts prepare the response JavaServer Pages (JSP) file, so a programmer cannot know directly which part of the JSP file they are trying to output as data. Syntax analysis of JSP is necessary to know the context of the writing data.

JSP has some other problems:

--Expression Language (EL) directly outputs the value. It is advisable to prohibit the use of EL.

--It is advisable to prohibit the use of *scriptlets*, for the same reason as EL.

--It is probably advisable to avoid using custom tags until it has been confirmed safe to do so..

6. Conclusion

It has been claimed that coding conventions are important for security, but this area has hardly been discussed. In this paper, we showed such conventions are useful for secure software development, and proposed a convention decision process considering testability and functional conflicts. Then, we clarified 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 to a certain extent for each type of attack. We also proposed the desired security functions of a framework that complements the flaws of the conventions. The proposed *secure frameworks* are not only programming tips like others. Security frameworks have a close connection with security coding conventions. They complement each other. So when project managers have to make programs secure they have to consider both of them. Furthermore, the security framework and security conventions are useful not only for implementing security functions, but also for making testing easier.

We have noticed that the existing frameworks, which provide *partial* security functions do not always contribute to a good level of security. The reason is that the existing framework design (which is not specialized for security) lacks the concept of testability.

Developing conventions set for other security requirements, and the empirical evaluation of the proposed coding conventions and frameworks remain as future tasks.

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] (2007 December 17).SecurityFocus [Online]. Available: http://securityfocus.org/

Table 2.
HTML Output Methods with Java (part)

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

[4] J. Jürjens, Secure Systems Development with UML, Springer, 2004.
[5] H. H. Thompson, "Why security testing is hard", Security & Privacy Magazine, IEEE Volume: 1 Issue: 4 July-Aug. 2003, pp. 83- 86.
[6] B. Potter and G. McGraw, "Software security testing", Security & Privacy Magazine, IEEE Volume: 2 Issue: 5 Sept.-Oct, 2004, pp. 81- 85.
[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] (2007 December 17). Secure Programming.com [Online]. Available: http://secureprogramming.com/
[12] (2007 December 17). Common Criteria for Information Technology Security Evaluation v2.3 [Online]. Available: http://www.commoncriteriaportal.org/public/developer/index.php?menu=2
[13] (2007 December 17). Category: OWASP Top Ten Project [Online]. Available: http://www.owasp.org/index.php/OWASP_Top_Ten_Project
[14] (2007 December 17). Code Conventions for the Java Programming Language [Online]. Sun Microsystems. Available: http://java.sun.com/docs/codeconv/
[15] T. Okubo, Y. Nakayama, Y. Wataguchi and H. Tanaka, "A Study on Software Development Method which Fulfills Specified Security Requirements" Computer Security Symposium 2006, pp.387-392 (in Japanese).
[16] R. E. Johnson and B. Foote, "Designing reusable classes." Journal of object-oriented programming 1(2), 1988, pp.22-35.
[17] (2007 December 17).Framework. Wikipedia the free encyclopedia [Online]. Available: http://en.wikipedia.org/wiki/Framework
[18] (2007 December 17). Struts [Online]. the Apache Software Foundation. Available: http://jakarta.jp/struts/
[19] (2007 December 17). JavaServer Pages [Online]. Sun Microsystems. Available: http://java.sun.com/products/jsp/
[20] (2007, December 17). java.sql.PreparedStatement [Online]. Sun Microsystems. Available: http://java.sun.com/j2se/1.4.2/docs/api/java/sql/PreparedStatement.html
[21] (2007, December 17). Bugle [Online]. Available: http://www.cipher.org.uk/index.php?p=projects/bugle.project
[22] (2007 December 17) java.lang.Runtime [Online]. Sun Microsystems. Available: http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Runtime.html