Design and Implementation of a Compiler with Secure Coding Rules for Secure Mobile Applications

Yunsik Son and Seman Oh

Dept. of Computer Science and Engineering, Dongguk University
26 3-Ga Phil-Dong, Jung-Gu, Seoul 100-715, KOREA
sonbug@dongguk.edu, smoh@dongguk.edu

Abstract

The dissemination and use of mobile applications have been rapidly expanding these days. And in such a situation, the security of mobile applications has emerged as a new issue. Especially, the software including mobile applications will always exist the possibility of malicious attacks by hackers, because it exchanging data in the internet environment. These security weaknesses are the direct cause of software breaches causing serious economic loss. In recent years, the awareness that developing secure software is intrinsically the most effective way to eliminate the software vulnerability than strengthening the security system for the external environment has increased. Therefore, Methodology to eliminate the vulnerability using secure coding rules and checking tools is getting attention to prevent software breaches in the coding stage. However, the existing coding rules do not reflects the characteristics of the mobile environments and the applications.

In this paper, we will define the secure coding rules that reflect the characteristics of the mobile environments and applications by the analysis of the existing secure coding rules. And, we will design and implement the compiler to inspect vulnerabilities of the mobile applications using defined secure coding rules in the coding stage.

Keywords: *Secure Software, Secure Coding Rules, Compiler Construction, Mobile Application Vulnerability*

1. Introduction

Recently, computer security incidents have become social hot issues and led to enormous economic losses. It has also triggered damages due to individuals' personal information being leaked. Security systems available at this point to prevent such security violations are merely network firewalls and user authentication systems but according to Gartner's report, 75% of software security violations were occurred by weakness-containing applications. A majority of such security incidents have been directly linked to software weaknesses. Especially, the programs of today exchange data in the internet environment making it difficult to secure data reliability for the input and output data [1, 2].

For this reason a new trend proposing coding guides to solve software weaknesses at the coding stage has risen. Consequently, if weaknesses are blocked from the software development stage, the significant costs invested in recognizing and adjusting the software at the execution stage can be saved. In addition, this can contribute greatly to developing software which is safe from hackers.

Based on the previously researched analysis methodology and tools, this study will propose a tool which combines a compiler and analysis tool so that programmers can develop safe mobile applications from the beginning stages of development.

2. Related Studies

2.1. Secure Coding

The software of today exchanges data in the internet environment making it difficult to secure validity of the data input and output. There exists the possibility of being maliciously attacked by random invader [1, 2]. This weakness has been the direct cause of software security incidents which generate significant economic losses.

Security systems installed to prevent security incidents from occurring, mostly consist of firewalls, user authentication system and etc. However, according Gartner's report 75% of software security incidents occur due to application programs including weaknesses. Therefore rather than making security systems for the external environment more firm, programmers creating software codes more firm is the more fundamental and effective method of increasing the security levels. However, efforts to reduce the weaknesses of a computer system are still mainly biased to network servers.

Recently, there has been recognition of this problem and therefore research on secure coding, writing secure codes from the development stage, is being carried out actively. Especially, CWE(Common Weakness Enumeration) [3], an organization which analyzes the weaknesses that can arise from programming language, has analyzed and specified the various weaknesses that can occur in the source code creation stage by the different languages. Also, CERT(Computer Emergency Response Team) defines secure coding rules [4] to ensure secure source code creation.

2.2. Source Code Weakness Analysis Tools

Fortify SCA[5] is a weakness detection tool developed by Fortify. Fortify SCA detects weakness in source codes by using dynamic and static analysis methods and it supports 12 languages including C/C++ and Java. Detected weakness information is notified to the users along with statistical data. Coverity's Coverity Prevent[6] is a static analysis tool for source codes. Coverity Prevent displays weaknesses detected through an entire source code with lists. Each list contains the location of a weakness's occurrence and its cause. MOPS[7] is a model inspector developed in Berkeley. MOPS defined security vulnerability elements as properties and formalized them using finite automaton. Thus, all modeled vulnerabilities can be detected at low analysis costs. However, there is a ceiling to the vulnerability analysis because no analysis is performed on data flows. Findbugs [8], developed at the University of Maryland as an open source static analysis tool to analysis weakness in Java source code which used by Eclipse plug-in. Findbugs using the predefined 401 bug patterns and it has the characteristic that do not require the source because it perform the target Java program analyzing on bytecode level. And the bug patterns can be expanding easily when the new weaknesses are added. PMD[9] is the weakness analysis tool developed by open source project and it can be used on Eclipse, JBuilder and Netbeans by plug-in. PMD has 271 Rules, base on it, perform the source code level analysis to check the weaknesses like possible bugs, dead code, suboptimal code, overcomplicated expressions and duplicate code.

3. Designed and Implementation of the Compiler

3.1. Compiler Implementation Model

Our compiler is consisted of a module that builds traditional compilers, a static analysis module and a weakness analysis module. Figure 1 shows the proposed compiler.

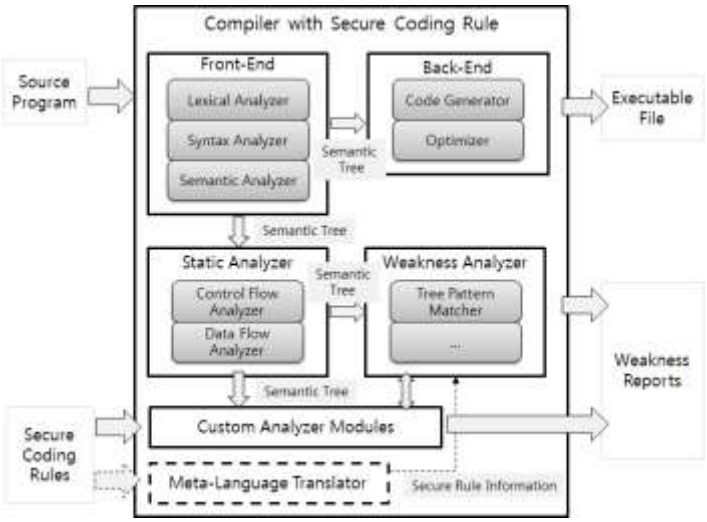

Figure 1. Proposed Compiler Configuration

In order to design the compiler with secure coding rules, a Meta language needs to be defined and a programming language's standard policy safe from the Meta language must be documented. The policy must include in addition the semantic tree [10], control flow, and data flow of the source code. The documented information is reprocessed by Meta language converter and is used as a very important piece of analysis information for weakness analysis. The static analysis module analyzes the control flow and data flow of a source program by using the symbol information and abstract syntax tree generated by the front end of the compiler. Information for analyzing weaknesses can be added to the abstract syntax tree as needed in the weakness analysis stage and this information is added to the semantic tree. Also, proposed compiler is designed to add check modules that analyze weakness point directly, if the specific weakness analysis is too difficult by meta-language method.

3.2. Secure Coding Rules for Mobile Applications

The secure coding rules for mobile applications are defined and categorized by mobile application specific weakness groups defined at previous researches [11]. Especially, the secure coding rules are defined for the android applications, and each weakness major rule is shown in Table 1.

Table 1. Secure Coding Rules for Mobile Applications

Category	Weakness Type (No. sub category)	Major Secure Coding Rule
Language Independent	Code Quality(3)	Do not reuse public identifiers
	Input Validation(1)	Sanitize untrusted data passed across a trust boundary.
	Security Features(1)	Do not allow privileged blocks to leak sensitive information across a trust boundary
Language Dependent	Class(3)	Defensively copy private mutable class members before returning their references
	Time & States(2)	Avoid deadlock by requesting and releasing locks in the same order
	Error Handling(2)	Prevent exceptions while logging data
	Java Libraries(2)	Do not use Thread.stop() to terminate threads
	Security Features(1)	Call the superclass's getPermissions() method when writing a custom class loader

Platform Dependent	Android Libraries(2)	Do not use deprecated methods
	Resource Usage(7)	Do not allow unprivileged resources
	System Events(2)	Must write the event handlers
	Runtime Environments(1)	Consider multiple vendor's device characteristics
	Security Features(3)	Do not leak personal information

4. Experimental Results and Analysis

In this section, we will experiment with the diagnosis of weaknesses that may occur in mobile applications using the developed compiler. We selected Findbugs and PMD to compare the performance of the implemented compiler. Two selected open source software testing tools can inspect the programs written in Java. And the secure coding rules for mobile applications that used in implemented compiler were defined at previous researches [11] and section 3.2.

Firstly, we determine the range the weakness check for implemented compiler and selected tools. Table 2 shows the checkable weaknesses in 43 weaknesses released by Korea Internet & Security Agency and Ministry of Public Administration and Security using 3 tools; implemented compiler, Findbugs and PMD. Findbugs and PMD are general software analysis tool, so we select 43 rules in order to compare reasonable. Also, selected 43 weaknesses can be applied to mobile environment.

Table 2. Weakness Check Result for 43 Items

ID	Find bugs	PMD	Proposed Compiler	ID	Find bugs	PMD	Proposed Compiler	ID	Find bugs	PMD	Proposed Compiler
1	○		○	16				31			
2			○	17				32	○		○
3	○		○	18			○	33			
4			○	19				34		○	○
5			○	20	○		○	35	○	○	○
6			○	21				36	○		○
7			○	22				37			
8		○	○	23			○	38		○	
9				24			○	39			
10				25			○	40			
11	○			26				41		○	○
12				27				42		○	○
13				28		○	○	43			
14				29				Total			
15				30					7	7	20

Findbugs has 7 items and PMD has 7 items, as shown in Table 1, and the proposed compiler checks 20 items from 43 items. The implemented compiler can cover the all kind of weakness which checked by two tools except ID 11 and ID 38.

Also, for the tool that checks the weakness, the false positive is also a very important performance metrics. For 22 weaknesses which checked by tools used in the experiment, we use the test source programs that selected from SAMATE, and the each false positive result is

shown in Table 2. In Figure 2, the 0 value item means the case of cannot be examined. The experimental results show that the proposed compiler showed that the false positive is less compared to other tools, except ID 36. For the checkable weakness by each tools, Findbugs 56.8%, PMD 63.7%, proposed compiler 49.3% of the average false positives, respectively. A result of the analysis of the experimental, the proposed compiler in this paper is considered to be superior compared to the tools with which to compare the performance.

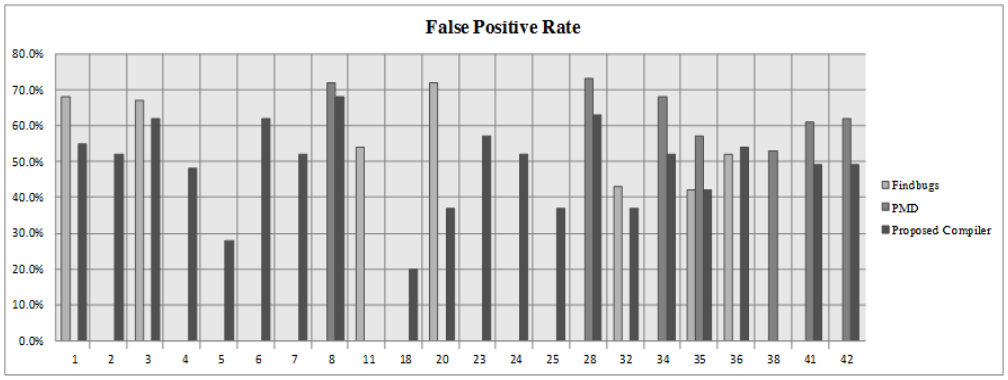

Figure 2. False Positive Rate

5. Conclusions and Further Researches

Detection of bugs for smart device contents rely mostly on classic software test methodology and classic test automation tools. This methodology separates the development process and the test process, serving as a factor that makes it difficult to analyze problems and change errors in the beginning of the development process. The expanded compiler for weakness analysis proposed in this study examines the weaknesses that can exist within programs at the beginning of contents development. It also enables safe contents development and a differentiated function from existing developing/testing tools.

In the future, research on automating the addition of analysis modules to compilers will be carried out. For this, the rules for secure coding must be standardized and research on automatic reading and analyzing of rules written in Meta language will be carried out. In addition, there is a need to review the execution speed, precision of analysis results and the correlation between the two for the proposed expanded compiler.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2011-0025582).

References

[1] G. McGraw, “Software Security: Building Security In”, Addison-Wesley, **(2006)**.
[2] J. Viega and G. MaGraw, “Software Security: How to Avoid Security Problems the Right Way”, Addison-Wesley, **(2006)**.
[3] Common Weakness Enumeration (CWE): A community-Developed Dictionary of Software Weakness Types, http://cwe.mitre.org/.
[4] J. McManus and D. Mohindra, “The CERT Sun Microsystems Secure Coding Standard for Java”, CERT, **(2009)**.

[5] Fortify SCA, https://www.fortify.com/products/hpfssc/.
[6] Coverity, Inc., "Coverity Static Analysis", http://www.coverity.com/products/ static-analysis.html, **(2009)**.
[7] H. Chen and D. Wagner, "MOPS: an infrastructure for examining security properties of software", Proceedings of the 9th ACM Conference on Computer and Communications Security, **(2002)**, pp. 235-244.
[8] FindBugs, http://findbugs.sourceforge.net/.
[9] PMD, http://pmd.sourceforge.net/pmd-5.0.0/.
[10] Y. Son and Y. S. Lee, "The Semantic Analysis Using Tree Transformation on the Objective-C Compiler", Multimedia, Computer Graphics and Broadcasting, CCIS, vol. 262, Springer, **(2011)**, pp. 60.
[11] Y. Son, I. Mun, S. Ko and S. Oh, "A Study on the Weakness Categorization for Mobile Applications", KCC2012, vol. 39, no. 1(A), **(2012)**, pp. 434-436.

Authors

Yunsik Son

He received the B.S. degree from the Dept. of Computer Science, Dongguk University, Seoul, Korea, in 2004, and M.S. and Ph.D. degrees from the Dept. of Computer Engineering, Dongguk University, Seoul, Korea in 2006 and 2009, respectively. Currently, he is a Researcher of the Dept. of Computer Science and Engineering, Dongguk University, Seoul, Korea. His research areas include smart system solutions, secure software, programming languages, compiler construction, and mobile/embedded systems.

Seman Oh

He received the B.S. degree from the Seoul National University, Seoul, Korea, in 1977, and M.S. and Ph.D. degrees from the Dept. of Computer Science, Korea Advanced Institute of Science and Technology, Seoul, Korea in 1979 and 1985, respectively. He was a Dean of the Dept. of Computer Science and Engineering, Graduate School, Dongguk University from 1993-1999, a Director of SIGPL in Korea Institute of Information Scientists and Engineers from 2001-2003, a Director of SIGGAME in Korea Information Processing Society from 2004-2005. Currently, he is a Professor of the Dept. of Computer Science and Engineering, Dongguk University, Seoul, Korea. His research areas include smart system solutions, programming languages, and embedded systems.