情報保護學會論文誌
第 22 卷 第 5 號, 2012. 10

대한민국 전자정부 소프트웨어 개발보안 가이드 개선 방안 연구*

한 경 숙,[1†] 김 태 환,[2] 한 기 영,[2] 임 재 명,[3] 표 창 우[2‡]
[1]한국산업기술대학교, [2]홍익대학교, [3]한국인터넷진흥원

An Improvement of the Guideline of Secure Software Development for Korea E-Government*

Kyung Sook Han,[1†] Taehwan Kim,[2] Ki Young Han,[2]
Jae Myung Lim,[3] Changwoo Pyo[2‡]
[1]Korea Polytechnic University, [2]Hongik University,
[3]Korea Internet & Security Agency

요 약

본 연구에서는 행정안전부의 전자정부 소프트웨어 개발·운영자를 위하여 2012년에 발표한 소프트웨어 개발보안 가이드를 개선하기 위한 방안을 제안하였다. 개선 방안은 취약점 관점이 아닌 코딩 규칙 관점으로 개발보안 가이드를 구성하는 것이다. 이를 위해 보안약점과 코딩 규칙, 이를 진단하기 위한 진단도구의 상관관계를 연구하였다. 제안된 개발보안 가이드를 사용하게 되면 코딩 규칙을 준수함으로써 보안약점 감소 효과를 거둘 수 있을 것이다. 기존의 개발보안 가이드가 개발자에게 보안약점이 없는 프로그램을 개발하도록 하는 달성하기 어려운 책임을 지우는 것에 반해, 본 논문에서 제안하는 개발보안 가이드는 프로그래머의 책임을 코딩 규칙을 준수하도록 하는 것으로 제한할 것이다.

ABSTRACT

We propose an improvement on the Guideline of Secure Software Development for Korea e-Government that is under revision by the Ministry of Public Administration and Security in 2012. We adopted a rule-oriented organization instead shifting from the current weakness-oriented one. The correspondence between the weakness and coding rules is identified.

Also, added is the coverage of diagnostic tools over the rules to facilitate the usage by programmers during coding period When the proposed guideline is applied to secure software development, the weakness would be controlled indirectly by enforcing coding rules. Programmers responsibility would be limited to the compliance of the rules, while the current version implies that it is programmers responsibility to guarantee being free from the weakness, which is hard to achieve at reasonable cost.

Keywords: weakness, secure coding, coding rule, coding guide

접수일(2012년 9월 11일), 수정일(1차: 2012년 10월 19일), 게재확정일(2012년 10월 19일)

* 본 논문은 2012년 한국인터넷진흥원 '시큐어코딩 기반 SW 개발보안 기반기술 연구' 위탁과제의 연구결과로 수행되었음(KISA-2012-024)

* 본 논문은 2010학년도 홍익대학교 학술연구진흥비에 의하여 지원되었음

† 주저자, khan@kpu.ac.kr
‡ 교신저자, pyo@hongik.ac.kr

I. 서 론

본 연구에서는 행정안전부의 전자정부 소프트웨어 개발·운영자를 위하여 금년 5월에 발표한 소프트웨어 개발보안 가이드[1]를 개선하기 위한 방안을 제안하였다. 행정안전부의 소프트웨어 개발보안 가이드는 금년 9월에 고시한 정보시스템 구축운영 지침[2]에 따라 소프트웨어 개발 단계에서 보안약점을 감소시키기 위한 지침을 제공하고 있다.

개발자를 위한 개발보안 지침은 보안상의 취약점 측면과 안전한 코딩을 위한 규칙 측면에서 기술할 수 있다. 보안상의 취약점 측면으로 지침을 기술하는 경우, 보안상의 취약점에 대한 전반적인 이해를 하고 있는 개발자가 아닌 경우에는 개발 단계의 코드와 연관시키기가 어렵다. 즉, 개발 단계에서 어떤 보안 취약점이 발생할 지 예측할 수 없는 개발자는 이런 관점으로 작성된 지침서를 활용하기 어렵다. 이런 경우 개발자는 예제로 제공된 코드에 따라 문제를 이해하게 되어 취약점의 전체적인 의미를 이해할 수 없게 된다. 이에 반해 언어 별 프로그램 구조에 따른 코딩 규칙 관점으로 기술하게 되면 개발자가 필요에 따라 작성하고 있는 프로그램 구조에서 주의해야 할 점을 쉽게 참고할 수 있게 된다. 따라서 전체적인 취약성을 이해하지 못하는 개발자도 필요한 부분의 가이드를 참고하여 코딩 규칙을 준수하는 프로그램을 작성할 수 있다.

정보시스템 구축운영 지침[2]에서는 보안약점이 없는 것을 보장하도록 요구하고 있으며 기존 개발보안 가이드 역시 제거되어야 할 보안약점 중심으로 기술되어 있다. 보안약점의 발생 여부는 개발 과정에서 즉시 확인되기 어려우며 완전히 제거되는 것을 보장하는 것은 현재 기술로는 불가능한 일이다. 따라서 개발자 가이드는 개발자들이 준수해야 하는 코딩 규칙을 제공하고 준수 여부를 확인할 수 있도록 규칙 중심으로 제시되어야 한다.

이를 위해 보안약점과 코딩 규칙, 이를 진단하기 위한 진단도구의 상관관계를 연구하였다. 제안된 개발보안 가이드를 사용하게 되면 코딩 규칙을 준수함으로써 보안약점 감소 효과를 거둘 수 있을 것이다. 기존의 개발보안 가이드가 개발자에게 보안약점이 없는 프로그램을 개발하도록 하는 달성하기 어려운 책임을 지우는 것에 반해, 본 논문에서 제안하는 개발보안 가이드는 프로그래머의 책임을 코딩 규칙을 준수하도록 하는 것으로 제한할 것이다.

또한 이러한 접근 관점의 차이는 진단 도구의 적용에도 차이를 가져올 수 있다. 취약점 관점에서 진단도구를 적용하는 것은 코딩 규칙의 준수 여부를 진단하는 것에 비하여 어려운 일이다. 그러나 코딩 규칙의 준수 여부는 프로그램의 구조로 판단하거나 입력 값을 검사하는 등의 방법으로 명확하게 진단할 수 있는 가능성이 높다. 정적 분석 도구를 사용하여 취약점 발생 여부를 완전히 진단하는 것은 물론 효과적인 일이지만 현실적으로 구현하는 것이 어렵고 여러 가지 오탐 문제를 발생시키게 된다[3]. 코딩 규칙을 기반으로 하여 규칙 준수 여부를 진단할 수 있는 진단도구 정보를 제공하면 개발자가 프로그램 개발 과정에서 자신이 규칙을 준수하였는지 자가진단 하는데 도움이 될 것이다.

이를 위하여 자바와 C 언어에 대하여 보안약점에 대응되는 코딩 규칙을 검토하였다. 또한 코딩 규칙을 준수하는지 여부를 진단할 수 있는 진단도구에 대한 정보를 함께 제공하기 위하여 코딩 규칙과 관련된 진단 도구에 대하여 자료 조사와 테스트를 수행하였다. 이러한 자료를 기반으로 하여 취약점 관점이 아닌 코딩 규칙 관점에서 개발보안 지침을 구성하는 방법을 제안하였다.

본 논문은 다음과 같이 구성된다. 2장에서는 기존의 소프트웨어 개발보안 가이드와 보안 약점, 코딩 규칙에 대한 연구에 대해 간략히 설명하였다. 3장에서는 보안개발 지침을 구성하기 위하여 보안 취약성과 코딩 규칙, 진단 도구의 연관 관계에 대하여 기술하였고, 4장에서는 이러한 정보를 토대로 한 보안개발 지침서를 제안하였다.

II. 관련 연구

관련연구에서는 행정안전부의 소프트웨어 개발보안 가이드[1]의 구성을 설명하고 본 연구에서 제안한 방법과의 차이점을 설명한다. 또한 취약점과 코딩 규칙을 정리한 CWE[4], CERT[5] 사이트에 대하여 간략하게 소개한다.

2.1 행정안전부 소프트웨어 개발보안 가이드

행정안전부의 전자정부 소프트웨어 개발·운영자를 위한 소프트웨어 개발보안 가이드[1]는 분야 별 보안상의 취약점을 기준으로 각 보안약점에 대해 보안 문제점 개요와 보안 대책, 코드 예, 참고 문헌으로 구성되어 있다. 보안 문제점 개요는 보안약점에 대한 간략한 설명과 공격자의 공격 방법과 같은 현상 관점의 설

명으로 이루어져 있으며, 보안대책은 보안약점을 방지하기 위한 개략적인 방법을 설명한다. 코드 예제는 안전하지 않은 코드와 이에 해당하는 안전한 코드의 예를 들고 그 예제에서 안전하지 않은 부분에 대한 설명과 그 문제를 어떻게 수정하여 방지하는지 설명한다. 참고문헌은 해당 보안약점에 대하여 기술하고 있는 CWE, OWASP 등 관련 사이트 링크로 이루어져 있다.

이러한 소프트웨어 개발보안 가이드는 개발자들이 보안약점에 대하여 이해할 수 있게 해주며, 안전하지 않은 코드와 안전한 코드의 예를 제시함으로써 개발 과정에서 이러한 보안약점을 방지하는데 도움을 줄 수 있다. 그러나 보안약점과 코딩 시 유의해야 할 점을 직관적으로 이해하기 어려운 경우, 개발자는 예제를 기반으로 하여 방지 기법을 이해할 수밖에 없게 된다.

2.2 CWE(Common Weakness Enumeration)

CWE[4]는 보안 약점을 구조화하여 정보를 제공한다. CWE ID로 식별되는 각 보안약점은 식별 번호와 이름, 약점에 대한 설명과 관련 개발 단계, 플랫폼, 진단 방법, 예제 프로그램, 완화 정책, 연관된 보안약점, 관련 규칙, 오류 분류, 공격 방법들에 대한 정보를 제공하며, 여러 사용자 그룹에 의하여 매년 25가지의 가장 위험한 보안약점 리스트를 결정하여 제공한다. 또한 보안 약점 사이의 연관 관계나 취약점과의 관계, 웹 보안 표준 기구인 OWASP나 공격 유형인 CAPEC과의 관계 정보를 제공하며 현재 900개 이상의 보안약점으로 분류하여 관리하고 있다.

2.3 CERT 안전한 코딩 표준

CERT[5] 안전한 코딩 표준은 카네기 멜론 대학의 SEI(Software Engineering Institute)에서 정리한 안전한 코딩[6]을 위한 표준으로, 현재 C와 C++, 자바, Perl 언어에 대하여 코딩 표준을 제공하고 있다. 프로그램 언어 별, 프로그램 구조에 따라 안전한 코딩을 위한 표준을 제공함으로써 보안상의 문제를 감소시킬 수 있는 코딩 규칙을 쉽게 접근할 수 있게 한다. 언어 별로 적절한 분류에 따라 규칙과 권고로 구분된 코딩 표준을 제시하며, 각 표준은 중요성, 발생 가능성, 개선비용을 점수로 환산하여 위험도를 평가한다. C와 C++의 경우 프로그램 구조를 중심으로 코딩 표준을 분류하고[7], 자바 언어는 프로그램 구조 관점의 분류에 시스템 특성이나 보안 관점의 분류가 추가되어 있다[8]. 코딩 표준은 간략한 설명과 코드 예제, 위험도, 진단 도구, 관련 자료로 구성되어 있다. 코드 예제는 안전하지 않은 코드의 예와 설명, 안전한 코드의 예와 설명으로 구성되며 진단도구 정보는 자동 진단도구에 대한 정보를 제공한다. 관련 자료는 다른 언어에서 관련되는 표준 정보나 보안약점, 다른 표준 분류 정보를 포함한다.

III. 보안약점과 코딩 규칙, 진단도구의 관계

개발자가 쉽게 참조할 수 있는 개발보안 지침을 작성하기 위하여 행정안전부의 소프트웨어 개발보안 가이드[1]에서 기술한 보안약점을 대상으로 하였다. 각 보안약점에 대하여 약점을 감소시킬 수 있는 코딩 규칙과의 연관관계를 구축하고, 각 코딩 규칙에 대하여 규칙을 위반하는지 여부를 검사할 수 있는 진단도구를 대응시킴으로써 개발보안 지침을 구성하기 위한 자료를 구성하였다.

3.1 보안약점과 코딩 규칙의 관계

프로그램의 문장이나 표현에는 많은 보안약점이 포함되어 있을 수 있다. 프로그램 중의 보안약점은 특정한 환경이 갖추어졌을 때, 공격자의 공격에 취약한 취약점으로 나타날 수 있다. 이러한 보안약점에 대하여 이해하는 프로그램 개발자는 동일한 기능을 수행하는 많은 함수나 수식, 프로그램 구조 중에 좀 더 안전한 방법을 선택할 수 있다. 이를 위해서는 취약점과 안전한 코딩을 위한 코딩 규칙을 이해하는 것이 도움이 되며, 이는 특정 취약점을 방지하고자 하는 경우 개발자가 코딩 시 유의할 점을 인식할 수 있도록 해 준다.

예를 들어, CWE-259(하드코드 된 패스워드)와 같은 보안약점은 패스워드와 같은 주요정보는 하드코드 되지 않도록 해야 한다는 것을 직관적으로 이해하고 개발 과정에서 방지하는 코드를 작성할 수 있다. 그러나 CWE-367(경쟁 조건: 검사시점과 사용시점, TOCTOU)과 같은 보안약점은 발생 원인이 다양하므로 개발자 입장에서는 주어진 예제를 기반으로 이해하게 된다. 따라서 이에 해당하는 코딩 규칙으로 설명하면 개발자들은 개발 중인 프로그램의 각 구조에서 준수해야 할 규칙을 좀 더 쉽게 이해할 수 있을 것이다. 검사시점과 사용시점 관련 경쟁조건에 대응되는

C 언어와 자바 언어 코딩 규칙의 예는 다음과 같다.

"경쟁 조건: 검사시점과 사용시점" 보안약점에 대응되는 코딩 규칙의 예

- FIO01-C: 파일 명을 파일 구분에 사용하는 함수에 주의
- MSC34-C: 나쁘다고 판명되었거나 폐기된 함수 사용 금지
- POS01-C: 파일 사용 시 링크 존재 여부 확인
- VNA03-J: 독립적인 여러 개의 단일 메소드 호출이 한 번에 수행될 것이라는 가정 금지
- LCK10-J: 잠금에 대한 중복 검사 시 정확한 표현 사용

코딩 규칙과 보안약점은 일반적으로 일대일로 관련되어지지는 않는다. 또한 관련된 코딩 규칙을 준수하는 것이 해당 보안약점을 완벽하게 방지한다는 것을 의미하지도 않는다. 그러나 이러한 대응 관계를 좀 더 견고하게 구성함으로써 개발 시점의 보안 문제에 좀 더 강한 시스템을 구축하기 위한 지침을 마련할 수 있을 것이다.

3.2 보안약점과 관련 코딩 규칙

본 절에서 기술한 상관관계는 행정안전부의 2012년 개발보안 가이드[1]를 기반으로 선정하였으며, CWE[4] 사이트와 CERT[5] 사이트의 정보, 예제의 상관관계를 기반으로 작성하였다. 표 1.은 이렇게 작성된 보안약점과 관련 코딩 규칙의 일부이다. 대상으로 한 보안약점은 43개이며, [표 1]의 CWE-113과 같이 관련 코딩 규칙을 연결하지 못한 보안약점은 13개이다. 43개의 보안약점 중 C 언어 코딩 규칙과 관련된 것은 20개이며, 22개의 코딩 규칙이 연결되었다. 자바 언어의 경우 27개 보안약점에 25개 코딩 규칙이 연결되었다. 이 중 C 코딩 규칙만 연결된 보안약점은 3개이며, 자바만 연결된 보안약점은 10개이고, 두 언어에서 관련 코딩 규칙을 찾은 보안약점은 17개이다.

보안약점에 대하여 코딩 규칙이 연결되지 않은 부분은 아직 적당한 규칙을 찾지 못한 것이다. 이러한 보안약점들은 이미 존재하는 규칙 중에 적당한 것을 연결하거나 새로운 코딩 규칙을 생성하여 보완해 나갈 수 있을 것이다. 이 중 웹 관련 보안약점이 4개이고, 응용 개발 관련 보안약점 7개, 설계 단계와 관련된 보안약점 1개, 데이터베이스 서버 관련 보안약점 1개로 분류하였다. 이 외에도 시스템 관련 문제나 네트워크 프로토콜 관련 문제 등으로 분류해 볼 수 있다. [표 1]에서 나타난 CWE-113은 웹 관련 보안약점으로 분류하였다. 웹 관련 보안 약점은 크로스 사이트 스크립팅, SQL 삽입[9] 등 많은 보안약점을 포함하고 있다. 아직 코딩 규칙을 대응하지 못한 보안약점 중 응용 개발 관련 보안약점은 코딩 규칙을 생성하거나 적절한 것을 찾아서 연결해야 하는 것이고, 나머지 보안약점도 개발 단계와 관련된 것은 적절한 코딩 규칙을 연결해 나가야 할 것이다.

프로그램 개발자가 보안상의 취약점을 이해하는 것은 어떻게 코딩하는 것이 이러한 취약점을 줄일 수 있는 지 이해하는 것과는 다른 문제이다. 프로그램 개발자는 코딩 규칙과 취약점의 상관관계를 이해함으로써, 보안상의 취약점을 줄이기 위하여 코딩 시 유의해야 할 점에 대하여 이해하고 프로그램을 작성할 수 있다.

[표 1]에서 볼 수 있듯이, 보안 취약점과 코딩 규칙은 일대일로 대응 되지는 않으며, 관련 코딩 규칙을 준수하는 것이 해당 보안 취약점을 완전히 방지한다고 보장할 수도 없다. 그러나 이러한 대응 관계를 좀 더 견고하게 구성함으로써 개발 시점의 보안 문제에 좀 더 강한 시스템을 구축하기 위한 지침을 마련할 수 있을 것이다.

3.3 코딩 규칙과 진단도구

준수해야 할 코딩 규칙에 대하여 개발 과정에서 준수 여부를 검사하기 위한 정보를 제공하는 것 역시 필요한 일이므로, 각 코딩 규칙에 대하여 진단할 수 있는 도구에 대한 대응 관계를 조사하였다. 코딩 규칙과 진단도구의 대응 정보는 프로그램 개발자가 개발 과정에서 코딩 규칙의 준수 여부를 점검하기 위해 사용할 수 있다. 진단 도구는 코딩 규칙에 따라 전반적으로 검사 가능한 경우도 있고, 일부 특정 조건에 대해서만 검사 가능한 경우가 있다. 또한 도구에 따라 특정 규칙을 검사하기 위한 옵션 지정을 해야 하는 경우가 있다. 모든 코딩 규칙에 대하여 진단 도구가 지원되는 것은 아니며, 진단 가능하다는 것이 코딩 규칙을 완벽하게 지원한다는 것을 보장하지는 않는다는 문제가 있으나, 부분적으로 진단하는 경우에도 일단 진단도구에 포함시켰다. 부분적으로 진단하는 도구의 경우, 진단 가능한 부분에 대한 정보를 포함하여 일단 지원되는 도구를 활용할 수 있도록 개발보안 가이드에 정보를

〔표 1〕 보안약점과 관련 코딩 규칙

보안약점 (CWE ID)	코딩 규칙 (C)	코딩 규칙 (자바)
CWE-99: 자원 삽입	STR02-C: 복합 시스템의 입력 데이터는 확인하여 사용한다.	
CWE-22: 경로 조작	FIO02-C: 신뢰할 수 없는 경로명은 정규화하여 사용	IDS02-J: 신뢰할 수 없는 경로명은 검증 전에 정규화 하여 사용
CWE-78: 운영체제 명령어 삽입	STR02-C: 복합 시스템의 입력 데이터는 확인하여 사용한다.	IDS07-J: Runtime.exec() 메소드에 신뢰할 수 없는 데이터 전달 금지
CWE-113: HTTP 응답 분할		
CWE-190: 정수 오버플로우	INT30-C: unsigned int 형에서 오버플로우 되지 않는지 확인 필요	NUM00-J: 정수 오버플로우 검사 또는 방지
	INT32-C: int 형에서 오버플로우 되지 않는지 확인 필요	
	INT35-C: 연산 이전에 큰 크기의 자료형에서 미리 계산하여 검사	
	MEM07-C: calloc()의 인수 중 크기는 size_t 형으로 사용	NUM12-J: 수치 자료형에서 작은 자료형으로의 변환 시 값을 잃거나 다른 값이 되지 않는지 확인
	MEM35-C: 충분한 크기의 메모리 할당이 필요	
CWE-256: 패스워드 평문 저장	MSC18-C: 프로그램에서 패스워드와 같은 민감한 데이터 관리에 주의	SER05-J: 암호화되지 않은 주요 데이터 직렬화 금지
CWE-614: 보안속성 미적용으로 인한 쿠키 노출		MSC00-J: Socket 클래스 대신 SSLSocket 클래스를 사용하여 데이터 전달
CWE-327: 취약한 암호화 알고리즘의 사용	MSC30-C: 의사 난수 생성에 rand() 함수 사용 금지	MSC00-J: Socket 클래스 대신 SSLSocket 클래스를 사용하여 데이터 전달
	MSC32-C: 난수 생성에 적합한 시드 값 사용 확인	MSC02-J: 강력한 난수 사용
CWE-367: 경쟁 조건: 검사시점과 사용시점	FIO01-C: 파일 명을 파일 구분에 사용하는 함수에 주의	VNA03-J: 독립적인 여러 개의 단일 메소드 호출이 한 번에 수행될 것이라는 가정 금지
	MSC34-C: 나쁘다고 판명되었거나 폐기된 함수 사용 금지	
	POS01-C: 파일 사용 시 링크 존재 여부 확인	LCK10-J: 잠금에 대한 중복 검사 시 정확한 표현 사용
CWE-754: 비정상적 혹은 예외적 조건의 부적절한 검사	EXP12-C: 함수 반환 값을 무시하지 않음	EXP00-J: 메소드의 리턴 값을 무시하지 말 것
	MEM32-C: 메모리 할당 오류를 검사하여 처리해야 함	ERR06-J: 선언되지 않은 예외를 호출 메소드로 전달하지 말 것
	API04-C: 일관되고 유용한 오류 검사 방법 사용	
CWE-404: 자원의 부적절한 반환	FIO42-C: 더 이상 사용하지 않는 파일은 반드시 닫음	FIO04-J: 더 이상 사용하지 않는 자원은 해제함
CWE-476: 널 포인터 역 참조	EXP34-C: 널 포인터에 대한 역 참조 금지	EXP00-J: 메소드의 리턴 값을 무시하지 말 것
	MEM32-C: 메모리 할당 오류를 검사하여 처리해야 함	

제공할 것이다.

코딩 규칙과 진단도구의 대응 정보를 구축하기 위해 C 언어와 자바 언어에 대해 서로 다른 방법을 적용하였다. C 코딩 규칙의 경우, CERT〔5〕 사이트에서 진단 가능한 도구 정보를 제공하므로 상용 도구에 대해서도 문서를 통한 자료 수집이 가능했으나 자바 코딩 규칙의 경우 대응되는 도구 정보를 제공하지 않아 공개된 소프트웨어를 테스트하는 방식으로 연관관계를 조사하였다.

3.3.1 C 코딩 규칙에 대한 진단도구 연구

C 코딩 규칙의 준수 여부를 진단하기 위한 진단도구 연구는 실제 도구 사용이 불가능하여 문서에 기초

〔표 2〕 C 코딩 규칙과 진단도구

코딩 규칙 (CERT 룰)	진단 도구(상용)	진단 도구(공개)
EXP12-C: 함수 반환 값을 무시하지 않음	Coverity Prevent, Klocwork, LDRA tool suite,	Splint, Compass/ROSE
EXP34-C: 널 포인터에 대한 역 참조 금지	LDRA tool suite, Fortify SCA, Klocwork, Coverity Prevent	Splint, Compass/ROSE, CppCheck
INT30-C: unsigned int 형에서 오버플로우되지 않는지 확인 필요	Fortify SCA	Compass/ROSE
INT32-C: int 형에서 오버플로우되지 않는지 확인 필요	LDRA tool suite, Fortify SCA	
INT35-C: 연산 이전에 큰 크기의 자료형에서 미리 계산하여 검사	Fortify SCA	Compass/ROSE
STR02-C: complex subsystem의 입력 데이터는 확인하여 사용한다.	Fortify SCA, Klocwork	
MEM07-C: calloc()의 인수 중 크기는 size_t 형으로 사용		Compass/ROSE
MEM32-C: 메모리 할당 오류를 검사하여 처리해야 함	Coverity Prevent, Fortify SCA	Compass/ROSE, CppCheck
MEM35-C: 충분한 크기의 메모리 할당이 필요	LDRA tool suite, Fortify SCA, Coverity Prevent	Compass/ROSE
FIO01-C: 파일 명을 파일 구분에 사용하는 함수에 주의	LDRA tool suite, Klocwork	Compass/ROSE
FIO02-C: 신뢰할 수 없는 경로명은 정규화하여 사용	Klocwork, LDRA tool suite	Compass/ROSE
FIO06-C: 적절한 접근 권한으로 파일 생성		
FIO42-C: 더 이상 사용하지 않는 파일은 반드시 닫음	LDRA tool suite, Fortify SCA, Klocwork,	Compass/ROSE
ENV02-C: 동일한 효과를 가진 여러 개의 환경변수 사용 시 주의		
ERR33-C: 오류를 검사하여 처리해야 함		Splint
API04-C: 일관되고 유용한 오류 검사 방법 사용		
MSC16-C: 함수 포인터 암호화 고려		Compass/ROSE
MSC18-C: 프로그램에서 패스워드와 같은 민감한 데이터 관리에 주의		
MSC30-C: 의사 난수 생성에 rand() 함수 사용 금지	LDRA tool suite, Fortify SCA,	Compass/ROSE
MSC32-C: 난수 생성에 적합한 시드 값 사용 확인		Compass/ROSE
MSC34-C: 불필요한 함수 사용 금지		
POS01-C: 파일 사용 시 링크 존재 여부 확인		Compass/ROSE

하여 평가한 상용 진단도구와 실제 테스트를 수행한 공개 진단도구를 구분하여 평가하였다. 상용 진단도구로는 Coverity Prevent〔10〕, Fortify SCA〔11〕, Klocwork〔12〕, LDRA tool suite〔13〕를 대상으로 했고, 공개 진단도구는 Compass/ROSE〔14〕와 Splint〔15〕, CppCheck〔16〕를 사용하였다. CppCheck는 CERT에서는 언급하지 않는 도구로, C/C++ 관련 진단 기능이 있는 공개 소프트웨어이다. 코딩 규칙에 대한 진단 도구 정보는 CERT 사이트의 정보를 기반으로 하여 테스트 결과를 추가하여 정리하였다. 테스트는 Compass/ROSE와 Splint, CppCheck를 대상으로 수행하였다. 공개 소프트웨어로 실제 테스트가 가능한 도구에 대해서는 예제를 기반으로 코딩 규칙을 위반한 사례에 대한 검출이 가능한 지 여부를 검사하였다. 검출이 가능한 경우, 이를 위한 옵션 지정이 가능한 도구에 대해서는 옵션 정보도 명시해 나가야 할 것이다. 대부분의 진단 도구는 상용 도구로 실제 테스트가 불가능하여 문서를 기반으로 하여 검사 가능성 여부를 조사하였다.

〔표 2〕는 C 코딩 규칙과 진단도구의 연관관계를 나타낸 것이다. 〔표 1〕에서 43개 보안약점과 관련된 22개의 C 코딩 규칙을 대상으로 하였다. 상용 진단도구

평가 결과, Coverity Prevent는 4개, Klocwork은 6개, LDRA tool suite는 8개, Fortify SCA는 9개의 코딩 규칙 위반에 대해 진단하는 것을 볼 수 있었다. 공개 진단도구의 경우, Compass/ROSE가 14개, Splint는 3개, CppCheck는 2개의 코딩 규칙에 대해 위반 여부를 진단하였고 진단도구가 연결되지 않는 코딩규칙이 5개였다.

평가 대상 코딩 규칙을 기준으로 하여 진단 도구에 대하여 평가한 결과 Compass/ROSE가 가장 많은 코딩규칙 위반에 대하여 검출하는 것을 볼 수 있다. 이는 Compass/ROSE가 CERT 코딩 표준을 기반으로 진단 규칙을 구성해 나가기 때문이며, 허가된 사용자들에 대해 규칙을 추가하는 것도 허용하고 있다. Coverity Prevent의 경우, 문자열과 널 포인터, 메모리 관련 일부 보안약점을 검출하였으며, Klockwork은 C/C++, 자바, C#에 대한 진단도구로 파일 입출력 관련 규칙 위반에 대한 검출 비중이 높았다. Fortify SCA는 다양한 언어에 대하여 다양한 보안 약점에 대한 진단 규칙을 제공하나, 자료상으로는 검출 비율이 높지는 않았다. LDRA tool suite는 여러 가지 정적/동적 도구를 제공하며 그 중 TBSecure가 C 언어에 대한 코딩 규칙 준수 여부를 검사하는 도구이다. 이 두 도구의 경우 전체 영역에서 일부 보안 문제점들을 검출했다. Splint의 경우 널 포인터 접근과 문자열 관련 크기 문제, 함수 리턴 값 검사 문제와 같은 일부 문제에 대하여 진단하였다. CppCheck는 문자열과 메모리 관련 문제, 널 포인터 참조와 같은 부분에서 보안 문제를 검출하는 것을 확인할 수 있었다.

이러한 진단 도구가 어떤 코딩 규약에 대하여 검출한다는 것이 그 규약을 준수하지 않는 모든 보안 문제점을 검출한다는 것을 의미하지는 않는다. 일부 특정한 경우에 대해서만 진단 가능하기도 하고, 특정 함수에 대해서만 진단하기도 한다. 이러한 진단도구에 대한 정보는 개발자나 진단원이 명확하게 범위를 이해하고 사용할 수 있도록 제공되어야 한다.

3.3.2 자바 코딩 규칙에 대한 진단도구 연구

자바 코딩 규칙의 경우, 진단도구에 대한 정보가 연결되어 있지 않으므로 상용 진단도구에 대한 평가는 수행하지 못하였고 공개 진단도구 중 PMD[17]와 Findbugs[18][19]에 대하여 코드 테스트를 수행하였다. C 코딩 규칙에 대한 진단도구로 사용한 Compass/ROSE의 경우 자바에 대해서도 진단이 가능하다고 명시되어 있으나, 본 연구에서는 사용하지 않았다. PMD는 윈도우 환경이나 리눅스 환경에서 모두 사용 가능하며, 이클립스에서 플러그인으로 작동할 수 있는 GUI 기반 도구이다. Findbugs는 바이트코드 상태에서 분석하는 정적 분석도구이며 역시 윈도우 환경이나 리눅스 환경에서 사용 가능한 GUI 기반 도구이다.

자바 관련 공개 진단도구의 경우 코딩 규칙 위반에 대하여 검출하는 경우가 아주 적게 나타났다. 25개의 코딩 규칙 가운데 PMD에서 진단 가능한 코딩 규칙은 LCK10-J(잠금에 대한 중복 검사 시 정확한 표현 사용)와 MSC03-J(민감한 정보가 하드코드 되지 않도록 함) 2개였다. Findbugs는 EXP00-J(메소드의 리턴 값을 무시하지 말 것)와 ERR06-J(선언되지 않은 예외를 호출 메소드로 전달하지 말 것), SER05-J(암호화되지 않은 주요 데이터 직렬화 금지) 3개 코딩 규칙에 대해서만 진단이 가능했다.

그러나 이 두 도구는 진단 규칙을 생성하여 추가할 수 있다는 특성을 가지고 있다. 진단 규칙을 기술하기 위한 문법을 이해하고 자바 코딩 규칙을 적절히 표현할 수 있게 되면 이러한 도구를 활용하여 좀 더 많은 코딩 규칙에 대한 진단이 가능한 도구를 얻을 수 있을 것이다.

IV. 개발보안 가이드의 구성

보안약점과 코딩 규칙의 대응 관계, 코딩 규칙과 진단도구의 연계를 통해 보안약점 기준으로 기술된 기존의 소프트웨어 개발보안 가이드를 코딩 규칙 관점으로 기반으로 구성하기 위한 자료를 구축하였다. 보안약점을 기반으로 한 가이드는 프로그램 개발 단계에서 개발자가 참고하기에 어려움을 느낄 수 있는 구조이므로, 프로그래밍 언어 별, 언어 구조에 따라 보안약점을 방지하거나 완화할 수 있는 코딩 규칙을 기술하는 방향으로 개선하기 위한 방안을 제시하였다.

4.1 개발보안 가이드의 구성 요소

행정안전부의 기존 개발보안 가이드[1]는 입력 데이터 검증 및 표현, 보안기능, 시간 및 상태, 에러 처리, 코드 오류, 캡슐화, API 오용과 같이 현상으로 나타나는 7가지 분류에 대하여 해당되는 취약점을 기술하는 형태로 되어있다. 프로그램 개발 결과로서 나타

나는 이러한 취약점 분류는 일부 개발 과정에서 바로 적용할 수 있는 부분도 존재하나 개발자가 바로 적용하기에 어려움을 느낄 수 있다. 따라서 취약점의 발생 요인에 따라 네트워크 영역, 운영체제나 환경설정과 같은 시스템 영역, 응용 프로그램과 같이 분류하고 응용 프로그램에서의 취약점에 대해서는 프로그램 언어 특성에 따라 분류할 수 있다. 언어 특성에 따라 프로그램 구성 요소를 기반으로 분류하게 되면 개발자가

널 포인터를 역참조하면 안 된다.

개요
정의되지 않은 동작 중 널 포인터 결과 값에 역참조를 시도하면, 일반적으로 비정상적으로 프로그램이 종료된다. 이와 관련된 보안약점으로는 널 포인터 역참조가 있다.

코드 예제
■ 안전하지 않은 코드 예
input_str은 str에 의해 참조되는 동적 할당 메모리로 복사되었다. 만약 malloc() 호출에 실패한다면, 널 포인터가 반환되어 str에 저장된다. str에 대한 역참조는 프로그램을 예측할 수 없는 방향으로 행동하게 만든다.

```
1:      size_t size = strlen(input_str)+1;
2:      str = (char *)malloc(size);
3:      memcpy(str, input_str, size);
4:      /* ... */
5:      free(str);
6:      str = NULL;
```

■ 안전한 코드 예
널 포인터에 대한 역참조를 하지 않으려면 malloc()에 의해 반환되는 값이 널이 아닌지 검사하고, 널인 경우 예외 상황에 대하여 처리하는 코드를 포함하여야 한다.

```
1:      size_t size = strlen(input_str)+1;
2:      str = (char *)malloc(size);
3:      if (str == NULL) {
4:          /* Handle Allocation Error */
5:      }
6:      memcpy(str, input_str, size);
7:      /* ... */
8:      free(str);
9:      str = NULL;
```

진단 도구
- LDRA tool suite
  - 45 D
- Fortify SCA
- Splint
- Compass/ROSE
- CppCheck
- Coverity Prevent
  - CHECKED_RETURN, NULL_RETURNS,
  - REVERSE_INULL, FORWARD_NULL
- Klockwork
  - NPD.* *RNPD.*

참고 문헌
[1] MITRE CWE: CWE-476, "NULL Pointer dereference"
[2] CERT C Secure Coding: EXP34-C, "Do not dereference null pointers"

[그림 1] 개발보안 가이드의 예

좀 더 쉽게 이해하고 참고할 수 있을 것이다.

프로그래밍 언어에 따라 선정한 코딩 규칙에 대하여 규칙 설명과 코드 예제, 진단도구 정보, 참고 문헌으로 구성한다. 규칙 설명은 각 코딩 규칙에 대한 간략한 설명과 관련 보안 약점을 설명한다. 관련 보안 약점은 〔표 1〕에서 예시한 보안약점과 코딩 규칙의 대응 관계를 활용할 수 있다. 코드 예제는 코딩 규칙을 구체적으로 설명할 수 있는 예제로, 안전하지 않은 코드의 예와 그 예에서 안전하지 않은 부분에 대한 설명을 포함한다. 또한 이에 대하여 코딩 규칙을 준수한 안전한 코드의 예와 코딩 규칙이 적용된 부분에 대한 설명을 포함한다.

진단도구 정보는 해당 코딩 규칙의 준수 여부를 검사할 수 있는 도구 리스트를 포함한다. 이를 위해 〔표 2〕에서 정리한 정보를 기반으로 하며, 진단 규칙을 설정할 수 있는 도구의 경우 해당 코딩 규칙을 검사하기 위한 옵션 값을 포함한다. 참고 문헌은 해당 보안약점에 대하여 기술하고 있는 CERT, CWE, OWASP 등 관련 사이트 링크를 제공하여 개발자가 필요한 정보를 참고할 수 있도록 한다.

이렇게 개발보안 가이드를 구성하고 방지하고자 하는 보안약점에 대하여 대응되는 코딩 규칙을 완성하게 되면, 개발자들은 보안약점을 방지하기 위하여 개발 과정에서 유의해야 할 점에 대하여 프로그래밍 언어와 구조 기반의 구체적인 정보를 쉽게 접근할 수 있게 될 것이다.

4.2 개발보안 가이드의 예

〔그림 1〕은 개선된 개발보안 가이드의 예를 나타낸다. 예제는 EXP34-C(널 포인터 역 참조 금지)에 대하여 작성하였다. 개요 부분에서 널 포인터에 대한 역참조를 할 경우 발생할 수 있는 문제점에 대하여 설명하고 관련 보안약점에 대하여 설명하였다. 코드 예제에서는 널 포인터를 반환할 가능성이 있는 함수인 malloc()을 호출한 결과에 대해 함수 반환 값을 검사하지 않는 경우 발생 가능한 문제점을 기술하고 이에 대한 안전한 코딩의 예를 기술하였다.

진단도구 항목에서는 LDRA tool suite와 Coverity Prevent, Klockwork의 경우 진단 규칙이 명시되어 나타나는 것을 볼 수 있다. 참고문헌은 C 코딩 규칙 EXP34-C에 대한 명세와 대응되는 보안약점인 CWE-476의 정보를 찾아볼 수 있도록 하였다.

기존 개발보안 가이드〔1〕의 경우 "널 포인터 역 참조"라는 프로그램에서 발생할 수 있는 문제에 초점을 맞추었다면 개선된 개발보안 가이드에서는 개발 과정에서 "널 포인터 역 참조 금지"를 위한 방법을 설명하고, "메모리 할당 오류를 검사하여 처리" 한다는 개발자 중심의 관점을 제공한다는 점에서 그 차이를 볼 수 있다. 즉 현상으로 나타나는 취약점 중심의 가이드에서 관련 코딩 규칙 중심의 가이드로 변환하게 되는 것이다. 이해를 돕기 위하여 코드 예제를 제공하는 것은 동일하나 진단도구에 대한 정보를 추가적으로 제공함으로써 프로그램 개발 과정에서 개발자가 진단도구를 활용하여 스스로 검사해볼 수 있도록 정보를 제공한다.

V. 결론

본 연구에서는 보안약점 관점에서 기술된 기존의 개발보안 가이드 구성의 문제점을 기술하고 그 개선 방안을 제시하였다. 보안약점이 없는 프로그램을 요구하는 정보시스템 구축운영 지침과 보안약점 중심으로 기술하고 있는 기존의 개발보안 가이드는 개발자에게 개발 과정에서 달성하기 어려운 부담을 부과한다. 이를 개선하여 개발자들이 준수해야 하는 코딩 규칙 관점에서 가이드를 제공하고 그 규칙의 준수 여부를 검사할 수 있도록 규칙 중심의 개발보안 가이드를 제시하였다.

이를 위해 기존의 개발보안 가이드에서 언급한 취약점에 해당하는 코딩 표준을 우선적으로 선정하였고, 코딩 규칙 준수 여부를 점검할 수 있도록 진단도구에 대한 정보를 제공하였다. 프로그래밍 언어와 구조에 따라 기술된 코딩 규칙을 기반으로 개발보안 가이드를 기술함으로써, 보안약점에 대한 전반적인 이해를 하지 못하는 개발자도 쉽게 보안개발에 접근할 수 있고, 규칙의 준수 여부도 상대적으로 쉽게 검사할 수 있을 것으로 기대한다. 개발자는 개발 과정에서 보안약점의 발생 가능성에 대하여 고민하는 대신 작성하고 있는 프로그램 구조에 대한 코딩 규칙만 준수함으로써 보안약점을 감소시키는 안전한 프로그램을 개발할 수 있다. 또한 적당한 진단도구를 활용하여 개발자 스스로 프로그램 개발 과정에서 코딩 규칙의 준수 여부를 확인할 수 있을 것이다.

준수해야 할 코딩 규칙 리스트는 두 가지 측면에서 확장할 수 있다. 한 가지 측면은 보안약점에 대하여 아직 대응되는 코딩 규칙을 찾지 못하였거나 일부 문제에 대한 해결 방안만 제시한 경우이다. 이러한 보안

약점에 대해서는 적절한 코딩 규칙을 찾아서 대응시키거나 새로운 코딩 규칙을 생성하여 나갈 수 있을 것이다. 기존의 코딩 규칙이 구체적으로 제시되지 않은 경우 좀 더 코딩 규칙을 구체화시켜 나가는 방법도 고려할 수 있다. 또 다른 측면은 대상이 되는 보안약점의 확장이다. 매 년 수정되는 주요 보안약점에 대하여 대응할 수 있는 코딩 규칙을 추가하거나, 발생 빈도가 높아지는 보안약점에 대한 확장이 가능할 것이다. 이를 위하여 매 년 갱신되는 발생 빈도수가 높은 취약점에 대한 정보를 활용하거나 우리나라에서 빈번하게 발생하는 문제점에 대하여 적용하는 방법도 고려해 볼 수 있다.

프로그램 개발 단계에서 코딩 규칙 준수 여부를 점검할 수 있도록 제공한 진단도구 정보 역시 확장해 나갈 수 있다. 좀 더 다양한 도구에 대한 정보를 추가하고, 사용 권한이 공개된 도구에 대해서는 검사를 통하여 정확한 정보를 제공하도록 할 것이다. 또한 자바 코딩 규칙을 대상으로 하는 공개 진단도구에 대해서는 코딩 규칙을 진단 규칙으로 표현할 수 있는 방법을 연구해 나감으로써 도구의 진단 능력을 향상시킬 수 있을 것이다.

참고문헌

〔1〕 행정안전부, 전자정부 소프트웨어 개발·운영자를 위한 소프트웨어 개발보안 가이드, *행정안전부*, 2012. 5

〔2〕 행정안전부, 정보시스템 구축 운영 지침(행정안전부고시 제2011-36호), *행정안전부*, 2012. 9

〔3〕 Nuno Antunes, Marco Vieira. "Defending against Web Application Vulnerabilities," *IEEE Computer*, 45(2), pp. 66-72, 2012.

〔4〕 "Common Weakness Enumeration," http://cwe.mitre.org/

〔5〕 "CERT," http://www.cert.org/

〔6〕 Robert C. Seacord. Secure Coding in C and C++, *Addison-Wesley*, 2005

〔7〕 Robert C. Seacord. The CERT C Secure Coding Standard, *Addison- Wesley*, 2008

〔8〕 Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean Sutherland, and David Svoboda. The CERT Oracle Secure Coding Standard for Java, *Addison-Wesley*, 2012

〔9〕 I.A. Elia, J. Fonseca, and M. Vieira, "Comparing SQL Injection Detection Tools Using Attack Injection: An Experimental Study," *Proc. 21st IEEE Int'l Symp. Software Reliability Eng. (ISSRE 10)*, pp. 289-298, 2010.

〔10〕 "Coverity Prevent," http://www.coverity.com/

〔11〕 "HP Fortify Static Code Analyzer," http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-fortify-static-code-analyzer/

〔12〕 "Klockwork," http://www.klockwork.com/

〔13〕 "LDRA Software Technology," http://www.ldra.com/

〔14〕 "ROSE compiler infrastructure," http://rosecompiler.org/

〔15〕 "Splint-Secure Programming Lint," http://www.splint.org/

〔16〕 "CppCheck," http://cppcheck.sourceforge.net/

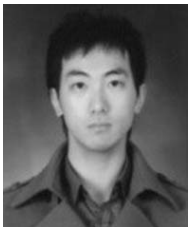
〔17〕 "PMD," http://pmd.sourceforge.net/

〔18〕 "Findbugs," http://findbugs.sourceforge.net/

〔19〕 Nathaniel Ayewah and William Pugh. "A report on a survey and study of static analysis users," *In Proceedings of the 2008 workshop on Defects in large software systems (DEFECTS '08)*, pp. 1-5, 2008.

〈著者紹介〉

한 경 숙(Kyung Sook Han) 정회원
1993년 2월: 홍익대학교 컴퓨터공학과 학사
1995년 2월: 홍익대학교 대학원 전자계산학과 석사
2002년 2월: 홍익대학교 대학원 전자계산학과 박사
2003년 3월~현재: 한국산업기술대학교 컴퓨터공학과 부교수
〈관심분야〉 소프트웨어 보안, 컴파일러, 웹 어플리케이션



김 태 환 (Tae-hwan Kim) 학생회원
2012년 2월: 홍익대학교 컴퓨터공학과 졸업
2012년 3월~현재: 홍익대학교 컴퓨터공학과 석사과정
〈관심분야〉 정보보호, 컴퓨터공학

한 기 영 (Ki-young Han) 학생회원
2002년 2월: 홍익대학교 컴퓨터공학과 졸업
2012년 3월~현재: 홍익대학교 컴퓨터공학과 석사과정
〈관심분야〉 정보보호, 컴퓨터공학

임 재 명(Jae-Myung Lim) 종신회원
1981년 2월: 한양대학교 전자공학과 학사
1983년 9월: 한양대학교 전자공학과 석사
2008년 2월: 항공대학교 통신정보공학과 박사
2000년 11월~현재 : 한국인터넷진흥원 공공정보보호단장
〈관심분야〉 SW보안약점, 정보화역기능(해킹, 웜, 스팸), CIP보호, 정보보호평가

표 창 우 (Changwoo Pyo) 정회원
1980년 2월: 서울대학교 공과대학 전자공학과 학사
1982년 2월: 서울대학교 대학원 컴퓨터공학과 석사
1989년 12월: University of Illinois at Urbana-Champaign, Computer Science 박사
1991년 1월: US ARMY Corpse of Engineers, 연구원
1991년 4월: 홍익대학교 교수
〈관심분야〉 프로그램 분석, 소프트웨어 보안, 프로그래밍 언어