Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.24, NO.1, Feb. 2014 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2014.24.1.171

CC에서의 소프트웨어 개발보안 활용에 대한 연구*

박 진 석,[1†] 강 희 수,[1] 김 승 주[2‡]
[1]고려대학교 정보보호대학원, [2]고려대학교 사이버국방학과/정보보호대학원

How to Combine Secure Software Development Lifecycle into Common Criteria*

Jinseok Park,[1†] Heesoo Kang,[1] Seungjoo Kim[2‡]
[1]Center for Information Security Technologies(CIST), Korea University
[2]Department of Cyber Defense/Center for Information Security Technologies(CIST), Korea University

요 약

CC는 평가보증등급에 따라 정보보호제품의 보안취약점을 최소화할 수 있도록 지원하는 제도이다. 소프트웨어 개발보안은 소프트웨어의 개발 생명주기에서 보안취약점을 발생시킬 수 있는 보안약점을 제거하는 방법이다. 하지만 CC는 정보보호제품이 인증된 시점 이전의 보안취약점에 대해선 고려하지만 인증된 시점 이후에 발생할 수 있는 새로운 보안취약점에 대해서 고려하지 않기 때문에 정보보호제품의 안전성과 신뢰성에 대한 문제가 발생할 수 있다. 또한, 국가 및 공공기관의 정보화사업에 도입되는 정보보호제품은 CC와 소프트웨어 개발보안을 모두 만족시켜야 되기 때문에 개발자, 평가자에게 부담이 된다. 따라서 본 논문은 CC에서 소프트웨어 개발보안을 활용해야하는 당위성을 검증하기 위해 CC와 소프트웨어 개발보안이 제거할 수 있는 보안약점 및 보안취약점의 상관관계를 비교하였다. 또한, CC에서 소프트웨어 개발보안을 활용하기 위한 평가방법을 제안하여 정보보호제품의 안전성과 신뢰성을 극대화하고 개발자와 평가자의 부담을 최소화하였다.

ABSTRACT

Common Criteria is a scheme that minimize IT products's vulnerabilities in accordance with the evaluation assurance level. SSDLC(Secure Software Development Lifecycle) is a methodology that reduce the weakness that can be used to generate vulnerabilities of software development life cycle. However, Common Criteria does not consider certificated IT products's vulnerabilities after certificated it. So, it can make a problem the safety and reliability of IT products. In addition, the developer and the evaluator have the burden of duplicating evaluations of IT products that introduce into the government business due to satisfy both Common Criteria and SSDLC. Thus, we researched the relationship among the Common Criteria, the static code analysis tools, and the SSDLC. And then, we proposed how to combine SSDLC into Common Criteria.

Keywords: Common Criteria, Secure Software Development Lifecycle, weakness, vulnerability

접수일(2013년 7월 31일), 수정일(2013년 10월 17일), 게재확정일(2013년 12월 3일)
* 본 연구는 지식경제부 및 정보통신산업진흥원의 대학IT연구센터육성 지원사업의 연구결과로 수행되었음 (NIPA-2013-H0301-13-3007)
* 본 논문은 2013년 한국인터넷진흥원 '시큐어코딩 기반 SW개발보안 기반기술 연구(2차년도)' 위탁과제의 연구결과로 수행되었음(Q1203812)
† 주저자, ilysoon7@korea.ac.kr
‡ 교신저자, skim71@korea.ac.kr (Corresponding author)

I. 서 론

NIST(the National Institute of Standards and Technology)의 보고서에 따르면 소프트웨어의 결함으로 발생하는 사고를 처리하기 위한 유지보수 비용은 미국 GDP(Gross Domestic Product)의 0.6%인 약 590억 달러로 나타났다. 그리고 소프트웨어의 보안취약점(vulnerability)을 제거하는 비용은 소프트웨어 설계단계보다 유지보수단계에서 최대 30배나 더 많이 요구된다[1].

또한, 최근에 발생하고 있는 사이버 공격을 살펴보면 보안 패치가 발표되기 이전의 보안취약점을 악용하는 제로-데이(zero-day) 공격이 대부분을 차지하고 있으며 보안취약점의 발생빈도가 꾸준히 증가하고 있는 추세이다. 이에 반해 중·소 소프트웨어 벤더들의 47%는 소프트웨어의 보안취약점을 제거하기 위한 조치가 미흡한 실정이다[2][3].

공격자는 보안취약점을 악용하여 소프트웨어의 무결성, 기밀성, 가용성을 손상시키기 때문에 개발자나 사용자는 보안취약점을 패치(patch)하거나 안티-바이러스(anti-virus)를 사용하여 소프트웨어를 보호하고 있다[4]. 하지만, 앞서 말한 방법들은 제로-데이 공격으로부터 소프트웨어를 보호하기 어려우며 지속적으로 추가적인 비용과 시간이 소모되기 때문에 효율적이지 않다. 따라서 위와 같은 문제점을 해결하기 위해 전 세계적으로 소프트웨어의 안전성을 강화하는 연구가 활발히 진행되고 있다[5].

이러한 연구의 중심에는 소프트웨어 보증(software assurance)이 있다. 소프트웨어 보증이란 '소프트웨어와 시스템은 요구사항에 맞게 정확하게 동작되어야 하며 우연히 또는 의도적으로 포함될 수 있는 보안취약점으로부터 소프트웨어가 안전하다.'는 확신과 신뢰 수준을 제공하는 개념이다[6]. Fig.1.을 살펴보면 소프트웨어 보증은 시스템 공학, 소프트웨어 공학, 정보보증, 프로젝트 관리 등의 다양한 분야에 사용된다[7].

소프트웨어 개발 생명주기에서 제거하지 못한 보안약점(weakness)은 소프트웨어 출시 이후에 보안취약점이 되어 사고를 발생시킬 수 있다. 따라서 소프트웨어 개발 생명주기(software development lifecycle)의 각 개발 단계에 소프트웨어 보증의 개념을 도입하여 보안약점을 제거할 수 있는 소프트웨어 개발보안(SSDLC, Secure Software Development LifeCycle)이 제안되었다[8].

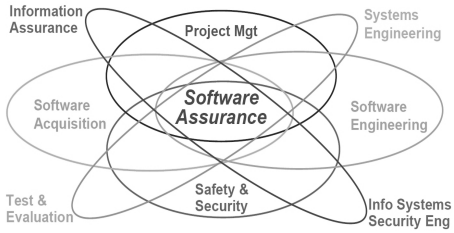

Fig.1. The scope of software assurance

Common Criteria(이하 CC)는 민간업체가 개발한 정보보호제품(보안기능이 포함된 IT제품)의 안전성과 신뢰성을 보증하여 사용자들이 안심하고 제품을 사용할 수 있도록 지원하는 제도이다[9]. 하지만 CC는 정보보호제품이 인증된 시점 이전의 보안취약점에 대해선 고려하지만 인증된 시점 이후에 발생할 수 있는 새로운 보안취약점에 대해서 고려하지 않기 때문에 정보보호제품의 안전성과 신뢰성에 대한 문제가 발생할 수 있다. 실제로 CVE[1](Common Vulnerabilities and Exposures)에 등록된 보안취약점을 살펴보면 정보보호제품이 인증된 시점 이후에 발견되는 새로운 보안취약점이 존재한다[10].

소프트웨어의 보안약점은 인증된 시점 이후로 공격자에게 발견되어 보안취약점이 될 수도 있기 때문에 보안약점을 제거한다면 인증된 시점 이후로 발생하는 보안취약점을 제거할 수 있다[11]. 따라서 CC에 소프트웨어 개발보안을 활용하여 소프트웨어에 존재하는 보안약점을 최소화할 수 있다면 인증 시점 이후에 발생되는 정보보호제품의 보안취약점을 최소화할 수 있을 것이다.

또한, 국가 및 공공기관에 도입하는 정보보호제품은 '국가정보화 기본법' 제38조 및 동법 시행령 제 35조에 근거하여 CC인증을 받아야 한다. 그리고 국가·공공기관의 정보화사업은 안전행정부의 '정보시스템 구축·운영 지침' 고시에 근거하여 소프트웨어 개발보안을 의무적으로 수행해야 한다. 따라서 CC에 소프트웨어 개발보안을 활용한다면 국가 및 공공기관의 정보화사업에 도입되는 정보보호제품에 대해 두 가지 평가방법을 모두 만족시킬 수 있어 개발자, 평가자의 부담을 최소화 할 수 있다.

본 논문은 CC에서 소프트웨어 개발보안을 활용해야하는 당위성을 검증하기 위해 CC와 소프트웨어 개발보안이 제거할 수 있는 보안약점과 보안취약점의 상

1) 소프트웨어 보안취약점이 발생된 사고사례 목록

관관계를 상세히 비교한다. 그리고 CC에서 소프트웨어 개발보안을 활용하기 위한 평가방법을 제안함으로써 정보보호제품의 안전성과 신뢰성을 극대화하고 개발자, 평가자의 부담을 줄일 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 보안약점과 보안취약점의 상관관계를 살펴보고 기존에 연구되었던 CC와 그 외 다른 평가방법들과의 상관관계를 살펴본다. 3장에서는 CC, 소프트웨어 개발보안, 정적 코드 분석 도구가 각 영역(설계, 구현, 보안기능 , 비보안기능 영역)에서 제거할 수 보안약점과 보안취약점을 상세히 비교 분석하여 CC에서 소프트웨어 개발보안을 활용해야하는 당위성을 증명한다. 4장에서는 CC에서 소프트웨어 개발보안을 활용하기 위한 평가방법을 제안한다. 마지막으로 5장에서는 결론을 맺는다.

II. 관련 연구

2.1 보안약점과 보안취약점의 상관관계

본 절은 Fig.2.와 같이 소프트웨어의 보안약점과 보안취약점의 상관관계를 살펴본다[11].

보안약점이란 보안취약점을 발생시킬 수 있는 가능성을 지닌 소프트웨어의 오류, 결함, 실수 등을 말한다. 보안취약점이란 공격자에게 실질적으로 악용된 보안약점을 말한다. 소프트웨어는 보안약점이 존재하며 보안약점은 보안취약점이 될 가능성이 있다. 다시 말해 소프트웨어에 보안약점이 있다면 보안취약점이 될 수 있지만 보안약점이 없다면 보안취약점도 존재하지 않는다. 소프트웨어가 가지고 있는 전체 보안약점 중 발견된 일부분은 CWE[2](Common Weakness Enumeration)에 등록된다. 마찬가지로 소프트웨어가 가지고 있는 전체 보안취약점 중 발견된 일부분은 CVE에 등록된다.

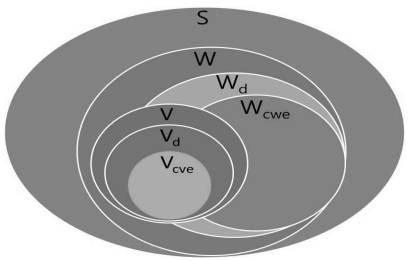

Fig.2. Correlation between weaknesses and vulnerabilities

- S : 소프트웨어
- W : 전체 보안약점
- W_d : 발견된 보안약점
- W_{cwe} : CWE에 등록된 보안약점
- V : 전체 보안취약점
- V_d : 발견된 보안취약점
- V_{cve} : CVE에 등록된 보안취약점

2.2 CC와 그 외 다른 평가방법과의 상관관계

본 절은 기존의 연구결과를 통해 CC와 그 외 다른 평가방법과의 상관관계를 살펴본다. 가장 먼저 Fig.3.을 살펴보면 CC의 평가제출물과 정적 코드 분석 도구를 사용하여 제거할 수 있는 보안취약점을 나타내었다. CC의 평가제출물이 제거할 수 있는 보안취약점들 중에 동적 분석 도구가 제거할 수 있는 보안취약점이 포함되어 있기 때문에 별도로 동적 분석 도구가 제거할 수 있는 보안취약점을 나타내지 않았다. CC의 평가제출물은 정보보호제품을 인증받기 위해 개발자가 평가자에게 제출해야하는 문서이다. 정적 코드 분석 도구는 소프트웨어를 실행하지 않은 상태에서 소스코드를 입력받아 존재하는 보안약점을 식별하고 제거할 수 있는 자동화된 도구이다.

CC의 평가제출물은 정보보호제품의 설계-보안 메커니즘 영역에서 보안취약점을 제거할 수 있고 정적 코드 분석 도구는 구현-기타기능 영역에서 발생하는 보안약점을 제거할 수 있다. 따라서 설계-보안 메커니즘 영역과 구현-기타기능 영역의 모든 보안취약점을

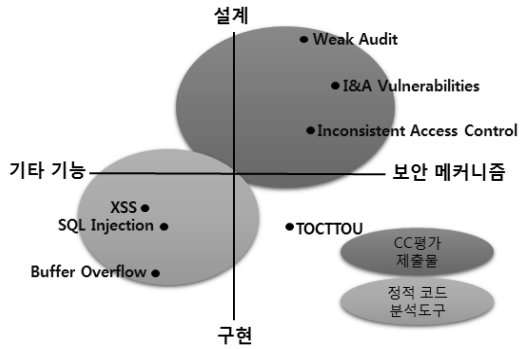

Fig.3. Removing possible vulnerabilities in TOE and static code analysis tools

2) 소프트웨어 보안취약점을 표준화한 보안약점 목록

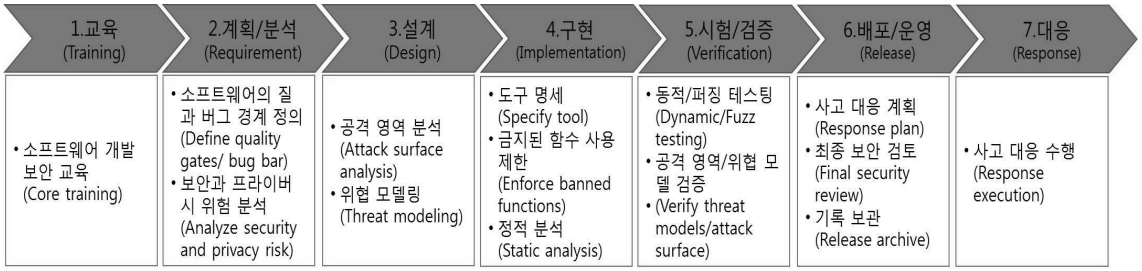

Fig.4. Secure software development lifecycle of the MS-SDL evaluation process

제거하기 위해서 CC 평가를 수행할 때 정적 코드 분석 도구를 사용해야한다[12].

MS-SDL은 Microsoft에서 2004년부터 개발되는 모든 소프트웨어에 적용하고 있는 대표적인 소프트웨어 개발보안 방법이다. Microsoft의 소프트웨어는 MS-SDL을 적용한 이후에 보안취약점이 상당히 감소했다[13]. 그리고 소프트웨어 개발보안을 적용하지 않은 소프트웨어와 비교했을 때 보안취약점의 수가 상대적으로 적다[14]. MS-SDL이 적용되지 않은 Windows XP와 Server 2000은 각각 119개, 34개의 보안취약점이 발생하였으며 MS-SDL이 적용된 Windows Vista와 Server 2005는 같은 기간 동안에 각각 45%와 91%가 감소한 66개, 3개의 보안취약점이 발생하였다. 그리고 MS-SDL이 적용되지 않은 타 업체의 운영체제는 각각 400개, 242개, 157개의 보안취약점이 발생하였다. 위 결과를 보았을 때 MS-SDL은 보안취약점을 제거하기 위한 효과적인 소프트웨어 개발보안 방법이다.

Fig.4.는 MS-SDL의 평가절차 및 평가방법을 나타냈다. MS-SDL은 소프트웨어의 모든 개발 단계마다 소프트웨어 개발보안을 위한 평가방법이 제시되어 있다. 평가방법은 소프트웨어 개발 보안에 대한 교육 수행, 보안요구사항 분석, 보안설계, 시큐어 코딩(secure coding), 정적·동적 분석, 사고대응계획 수립 등이 있다.

MS-SDL의 평가방법과 CC를 비교한 기존의 연구를 살펴보면 CC는 소프트웨어 개발보안을 위한 교육 수행, 보안 설계 검토, 시큐어 코딩, 보안 테스트가 미흡하다[15]. CC는 그 외 다른 평가방법들[3]과 비교해 보았을 때 소프트웨어 개발보안을 위한 교육 수행, 소스코드 검토, 보안취약점 분석, 안전한 소프트웨어를 위한 개발 기술 적용이 미흡하다[16].

위 연구결과들을 살펴보면 현재의 CC는 소프트웨어 개발보안을 위한 평가방법들이 부족하다는 것을 알 수 있다. 따라서 본 연구는 CC의 안전성과 신뢰성을 향상시키기 위해 보증요구사항에 소프트웨어 개발보안을 위한 평가방법을 제안하여 소프트웨어의 보안취약점을 최소화하였다.

III. CC, 소프트웨어 개발보안, 정적 코드 분석 도구 비교

3.1 비교 방법

앞서 언급한 2.2절의 연구결과들은 단순히 각 평가방법의 항목을 비교하였기 때문에 실질적으로 어떤 보안약점과 보안취약점이 제거되는지 식별하기 어렵다. 따라서 본 논문은 각 평가방법이 제거할 수 있는 보안약점과 보안취약점을 명확하게 식별하기 위한 새로운 방법론을 사용하였다.

본 논문의 방법론은 Fig.5.와 같이 CWE목록을 기반으로 하여 CC, MS-SDL, 정적 코드 분석 도구가 제거할 수 있는 보안약점과 보안취약점을 비교 분석하고 매핑(mapping)하였다. 매핑 결과를 통해 각 평가 방법이 제거할 수 있는 보안약점과 보안취약점의 공통점과 차이점을 알 수 있다. 동적 분석 도구는 제거할 수 있는 보안약점이 제한적이기 때문에 동적 분석 도구에 대해서 별도로 언급하지 않는다[17].

Table 1.은 각 평가방법에서 제시하고 있는 보안약점 및 보안취약점의 관련 내용이 CWE의 항목과 일치한다면 체크리스트 형태로 표시한 테이블이다. 본 논문에서는 매핑 테이블을 모두 나열할 수 없으므로 CWE/SANS에서 가장 위험한 소프트웨어 오류로 발표한 보안약점의 상위 25개만을 분석 결과로 제시한

3) MS-SDL(Microsoft-Security Development Life cycle), SSE-CMM(System Security Engineering Capability Maturity Model), OpenSAMM(Open Software Assurance Maturity Model)

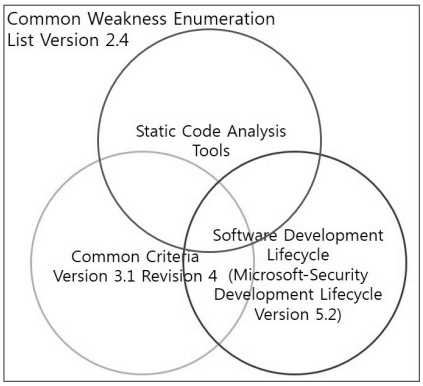

Fig.5. How to compare CC and SSDLC

다[18]. Table 1.의 가로축은 CWE/SANS에서 발표한 보안약점 25개를 나타냈다. 세로축은 각 보안약점의 특징으로 보안약점의 분류, 번호, 항목 명, 보안약점이 발생하는 개발 단계, 보안약점이 소프트웨어의 보안기능에 발생하는 경우, 보안약점이 발생시킨 보안취약점의 수를 나타냈다. 그리고 CC, MS-SDL, 정적 코드 분석 도구가 제거할 수 있는 보안약점을 매핑하였다.

본 논문에서는 가장 최근에 공개된 각 평가방법들의 자료를 바탕으로 분석하였다. 소프트웨어 개발보안은 가장 널리 사용되는 MS-SDL을 사용하였다. 그리고 정적 코드 분석 도구는 CWE의 호환성이 인정된 총 5가지(CodeSonar, Covertiy Quality Advisor, Security Advisor, HP Fortify Static Code Analyzer, Klocwork Insight)를 사용하였다[19].

3.2 비교 분석

각 평가방법의 비교 분석은 Table 1.과 같이 상세한 매핑 테이블을 통해 수행 되었다. Fig.6.을 살펴보면 CWE는 총 920개의 보안약점이 존재하며 7가지로 구성되어 있다. 각 구성별 보안약점의 수는 view 29개, category 176개, class 88개, base 330개, variant 276개, composite 6개, name chain 3개, deprecated 12개이다.

view는 사용자나 개발자 등의 특정 관점에 따라 분류한 보안약점이며 category는 보안약점의 공통적인 특징으로 분류한 보안약점이다. weakness는 class, base, variant로 각각 분류되는데 class는 가장 추상적인 보안약점이고 그 다음 base는 특정 기술, 언어 등에 종속적이지 않은 보안약점이다. 그리고 variant는 특정 기술, 언어 등에 종속적이고 가장 구체적인 보안약점이다. composite는 한 가지 보안약점이 아닌 다수의 보안약점으로 인해 발생하는 보안약점이며 named chain은 특정 보안약점에 의해 연쇄적으로 발생할 수 있는 보안약점이다. 마지막으로 deprecated는 중요도가 감소하여 현재 파기된 보안약점이다.

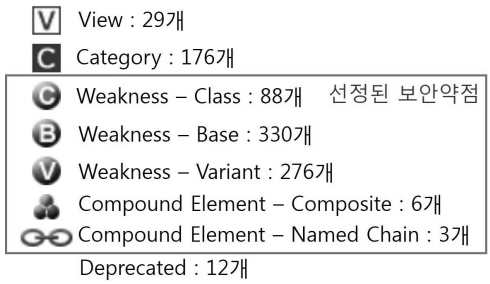

Fig.6. Selected vulnerability and weakness entries

본 논문은 소스코드와 직접적으로 관련 있는 class, base, variant, composite, named chain 구성만을 선정하여 총 703개의 보안약점을 분석하였다. 그리고 보안약점과 보안취약점을 앞서 연구된 [11]의 설계, 구현, 보안기능, 비보안기능 영역으로 분류하였다. [11]의 분류 방법은 정보보호제품의 보안기능 또는 보안기능과 관련이 없는 비보안기능을 기준으로 분류하였다. 그리고 대부분의 보안취약점이 발생하고 있는 소프트웨어의 설계단계 또는 구현단계를 기준으로 분류하였다.

이에 따라 본 논문은 CWE 목록을 참고하여 설계 영역은 CWE-701 : weaknesses introduced during design, 구현 영역은 CWE-702 : weaknesses introduced during implementation, 보안기능 영역은 CWE-254 : security features, 마지막으로 비보안기능 영역은 보안기능 영역이외의 보안약점과 보안취약점으로 분류하였다.

3.3 CC, MS-SDL 비교 결과

본 논문은 Fig.7.와 같이 703개의 보안약점을 각 영역에 중복되는 경우를 고려하여 968개의 보안약점과 이로 발생할 수 있는 3041개의 보안취약점을 분석하였다. 그리고 각 영역의 보안약점과 보안취약점의 분포율을 원 넓이의 비례하여 나타내었다.

Table 1. Result of mapping weakness, weakness's features, CC, MS-SDL, static code analysis tools

순위	CWE 분류	CWE-번호:항목명	보안약점의 특징				CC		MS-SDL			정적코드분석도구
			설계	구현	보안기능	보안취약점 수	보안기능요구사항	보증요구사항	설계	구현	검증	
1	Base	CWE-89:SQL Injection	○	○		7		○	○	○	○	○
2	Base	CWE-78:OS Command Injection	○	○		10		○	○	○	○	○
3	Base	CWE-120:Classic Buffer Overflow		○		5		○	○	○	○	○
4	Base	CWE-79:Cross-site Scripting	○	○		11		○	○	○	○	○
5	Variant	CWE-306:Missing Authentication for Critical Function	○		○	3	○	○	○	○	○	
6	Class	CWE-862:Missing Authorization	○	○		19	○	○	○	○	○	
7	Base	CWE-798:Use of Hard-coded Credentials	○		○	10	○	○	○			
8	Base	CWE-311:Missing Encryption of Sensitive Data	○	○	○	20	○	○	○		○	
9	Base	CWE-434:Unrestricted Upload of File with Dangerous Type	○	○		10	○	○		○		
10	Base	CWE-807:Reliance on Untrusted Inputs in a Security Decision	○	○	○	5		○	○	○	○	
11	Class	CWE-250:Execution with Unnecessary Privileges	○		○	7	○	○	○	○	○	○
12	Composite	CWE-352:Cross-Site Request Forgery(CSRF)	○			10	○	○	○	○		
13	Class	CWE-22:Path Traversal	○	○		11		○	○	○	○	○
14	Base	CWE-494:Download of Code Without Integrity Check	○	○		4	○	○	○	○		
15	Class	CWE-863:Incorrect Authorization	○	○		9	○	○	○	○	○	
16	Class	CWE-829:Inclusion of Functionality from Untrusted Control Sphere				20	○	○	○	○	○	
17	Class	CWE-732:Incorrect Permission Assignment for Critical Resource	○	○		17	○	○	○	○	○	○
18	Base	CWE-676:Use of Potentially Dangerous Function	○	○		6		○		○		○
19	Base	CWE-327:Use of a Broken or Risky Cryptographic Algorithm	○		○	8	○	○	○			
20	Base	CWE-131:Incorrect Calculation of Buffer Size		○		14		○	○	○	○	○
21	Base	CWE-307:Improper Restriction of Excessive Authentication Attempts	○		○	6	○	○	○	○	○	
22	Variant	CWE-601:Open Redirect	○	○		3	○	○	○	○	○	
23	Base	CWE-134:Uncontrolled Format String		○		6		○	○	○	○	○
24	Base	CWE-190:Integer Overflow or Wraparound		○		6		○	○	○	○	○
25	Base	CWE-759:Use of a One-Way Hash without a Salt				2	○	○	○			

Fig.8.은 각 영역에서 CC와 MS-SDL이 제거할 수 있는 보안약점과 보안취약점을 나타낸 것이다.

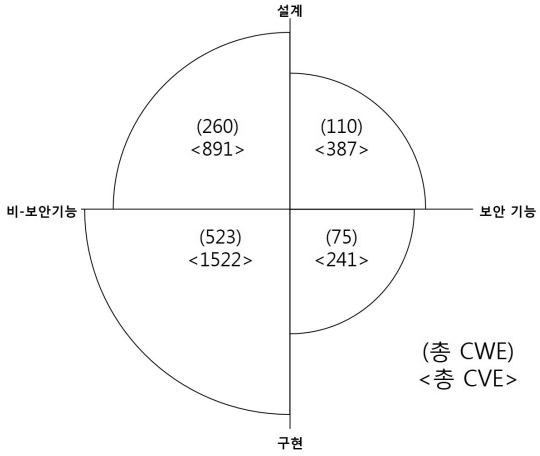

Fig.7. Distribution of weaknesses and vulnerabilities in each area

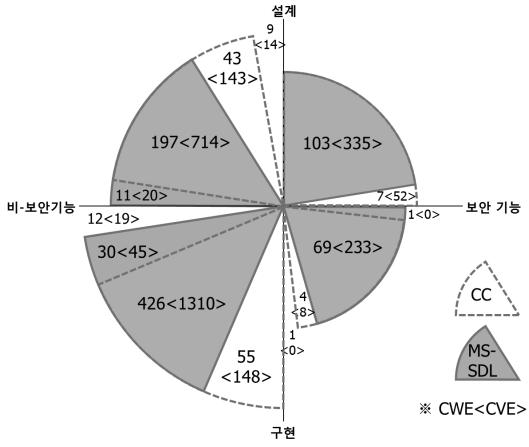

Fig.8. Analysis of weaknesses and vulnerabilities in each area (CC and MS-SDL)

3.3.1 설계-보안기능 영역

설계-보안기능 영역에서는 총 110개의 보안약점이 존재하며 그 중 63개의 보안약점이 387개의 보안취약점과 매핑되어 있다. 그리고 CC와 MS-SDL이 공통적으로 제거 가능한 보안약점은 103개, 보안취약점은 335개이며 CC가 독립적[4]으로 제거할 수 있는 보안약점은 7개, 보안취약점은 52개이다. MS-SDL이 독립적으로 제거할 수 있는 보안약점과 보안취약점은 없다. 그리고 두 평가방법이 제거하지 못하는 보안약점과 보안취약점은 없다. 따라서 이 영역에서는 CC가 모든 보안약점과 보안취약점을 제거할 수 있고 MS-SDL은 모든 보안약점과 보안취약점을 제거할 수 없기 때문에 CC는 MS-SDL을 활용하지 않고 독립적으로 사용 가능하다.

3.3.2 설계-비보안기능 영역

설계-비보안기능 영역에서는 총 260개의 보안약점이 존재하며 그 중 121개의 보안약점이 891개의 보안취약점과 매핑되어 있다. CC와 MS-SDL이 공통적으로 제거 가능한 보안약점은 197개, 보안취약점은 714개이며 CC가 독립적으로 제거할 수 있는 보안약점은 43개, 보안취약점은 143개이다. MS-SDL이 독립적으로 제거할 수 있는 보안약점은 11개, 보안취약점은 20개 이다. 두 방법이 제거하지 못하는 보안약점은 9개, 보안취약점은 14개이다. 따라서 이 영역에서는 CC와 MS-SDL이 제거할 수 있는 보안취약점이 다르기 때문에 CC에서 MS-SDL을 활용[5]해서 보안약점의 99.96%, 보안취약점의 99.98%를 제거할 수 있다.

3.3.3 구현-보안기능 영역

구현-보안기능 영역에서는 총 75개의 보안약점이 존재하며 그 중 42개의 보안약점이 241개의 보안취약점과 매핑되어 있다. 그리고 CC와 MS-SDL이 공통적으로 제거 가능한 보안약점은 69개, 보안취약점은 233개이며 CC가 독립적으로 제거할 수 있는 보안약점은 4개, 보안취약점은 8개이다. MS-SDL이 독립적으로 제거할 수 있는 보안약점은 1개, 보안취약점은 0개이다. 두 방법이 제거하지 못하는 보안약점은 1개, 보안취약점은 없다. 따라서 이 영역에서는 CC와 MS-SDL이 제거할 수 있는 보안취약점이 다르기 때문에 CC에서 MS-SDL을 활용해서 보안약점의 99.98%, 보안취약점의 100%를 제거할 수 있다.

3.3.4 구현-비보안기능 영역

구현-비보안기능 영역에서는 총 523개의 보안약점

4) 보안약점과 보안취약점을 해당 평가방법만이 제거할 수 있고 다른 평가 방법이 제거할 수 없는 경우를 말한다.

5) 소프트웨어 개발보안의 평가방법을 CC의 평가방법에 추가하여 평가 및 인증을 수행하는 것을 말한다.

이 존재하며 그 중 229개의 보안약점이 1522개의 보안취약점과 매핑되어 있다. 그리고 CC와 MS-SDL이 공통적으로 제거 가능한 보안약점은 426개, 보안취약점은 1310개이며 CC가 독립적으로 제거할 수 있는 보안약점은 55개, 보안취약점은 148개이다. MS-SDL이 독립적으로 제거할 수 있는 보안약점은 30개, 보안취약점은 45개 이다. 두 방법이 제거하지 못하는 보안약점은 12개, 보안취약점은 19개이다. 특히, 이 영역에서는 CC와 MS-SDL이 제거할 수 있는 보안취약점이 다르기 때문에 CC와 MS-SDL을 활용해서 보안약점의 99.98%, 보안취약점의 99.98%를 제거할 수 있다.

3.4 정적 코드 분석 도구, MS-SDL 비교 결과

소프트웨어 개발보안의 다양한 평가방법 중에 정적 코드 분석 도구는 모든 평가방법 중에 가장 쉽게 소프트웨어의 보안약점을 제거할 수 있는 방법이다. 따라서 여러 가지 문제로 인해 소프트웨어 개발보안을 적용하기 어려운 경우에는 단순히 정적 코드 분석 도구만 사용할 수 있을 것이다. CC에서는 평가보증등급이 4이상일 경우에만 소스코드를 제출한다. 따라서 CC에서는 평가보증등급 4이상일 경우에만 정적 코드 분석 도구를 사용할 수 있다.

본 절에서는 정적 코드 분석 도구만을 사용하였을 때 제거할 수 있는 보안약점을 비교하여 소프트웨어 개발보안의 모든 평가방법을 적용한 것에 비해 정적 코드 분석 도구만을 사용했을 경우 어느 정도의 보안약점과 보안취약점을 제거할 수 있는지 분석하였다.

본 논문은 Fig.9.와 같이 201개의 보안약점을 각 영역에 중복되는 경우를 고려하여 264개의 보안약점과 이로 발생할 수 있는 814개의 보안취약점을 분석하였다. 설계-보안기능 영역에서 정적 코드 분석 도구는 MS-SDL이 제거할 수 있는 보안약점 103개, 보안취약점 335개 중에 보안약점 19개, 보안취약점 40개를 제거할 수 있으며 설계-비보안기능 영역에서 MS-SDL이 제거할 수 있는 보안약점 208개, 보안취약점 734개 중에 보안약점 69개, 보안취약점 287개를 제거할 수 있다. 그리고 구현-보안기능 영역에서 MS-SDL이 제거할 수 있는 보안약점 70개, 보안취약점 233개 중에 보안약점 14개, 보안취약점 26개를 제거할 수 있으며 구현-비보안기능 영역에서 MS-SDL이 제거할 수 있는 보안약점 456개, 보안취약점 1355개 중에 정적 코드 분석 도구는 보안약점

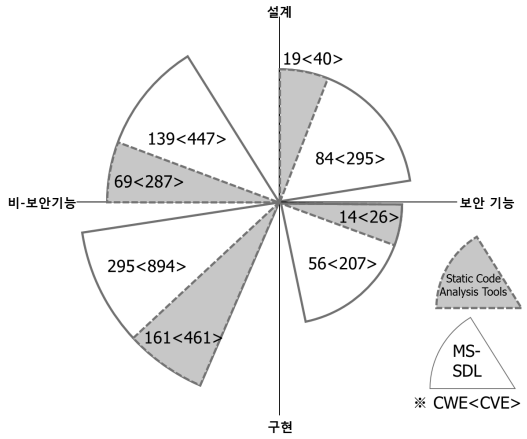

Fig.9. Analysis of weaknesses and vulnerabilities in each area (static code analysis tools)

161개, 보안취약점 461개를 제거할 수 있다.

IV. CC에서의 소프트웨어 개발보안을 활용하기 위한 방법

앞서 2장의 기존연구를 살펴보았듯이 CC는 소프트웨어 개발보안에 대한 평가방법이 미흡하다. 그리고 3장의 비교 결과를 살펴보면 설계-보안기능, 설계-비보안기능, 구현-보안기능, 구현-비보안기능 영역에서 CC와 소프트웨어 개발보안이 각각 제거할 수 있는 보안약점과 보안취약점의 차이점이 존재한다.

CC와 소프트웨어 개발보안의 상관관계를 정리하면 설계-보안기능 영역에서는 CC를 사용하여 모든 보안약점과 보안취약점을 제거할 수 있기 때문에 두 평가방법을 모두 사용할 필요가 없이 CC만 독립적으로 사용해서 보안약점과 보안취약점을 제거해도 된다. 그리고 설계-보안기능 영역을 제외한 나머지 영역들에서는 CC와 소프트웨어 개발보안이 제거할 수 있는 보안약점, 보안취약점이 서로 다르기 때문에 각 평가방법이 제거할 수 있는 모든 보안약점과 보안취약점을 제거하기 위한 방법이 필요하다. 따라서 CC에서 소프트웨어 개발보안을 활용하기 위한 평가방법을 보증요구사항[6]에 추가해서 소프트웨어 개발보안에 대한 평가와 인증을 수행해야 한다.

Table 2.는 CC에서 소프트웨어 개발보안을 활용

6) 보증요구사항은 개발자가 정보보호제품을 보증하기 위한 방법을 결정하기 위해 사용되고 평가자가 정보보호제품의 보증 수준을 결정하는 평가 기준으로 사용된다.

Table 2. Combine secure software development lifecycle with common criteria

소프트웨어 개발보안 절차	소프트웨어 개발보안의 평가방법	CC보증 요구사항
1.교육	소프트웨어 개발보안 교육 -소프트웨어 개발보안에 대한 교육 수행	ALC_DVS
2.계획/분석	보안요구사항 분석 -보안전문가 임명, -버그, 업무, 도구 명시, -최소한의 보안 기준 설정, -위험평가 수행	ASE
		ALC_TAT
		AVA
3.설계	보안 구조 및 설계 검토 -보안 구조 및 설계 검토, -위협 모델링, -공격노출면 분석	ADV
		AVA
4.구현	시큐어 코딩 및 주기적인 소스코드 검토 -안전한 컴파일러, 도구, 함수 사용, -정적 코드 분석 수행, 안전한 코딩 가이드라인 검토	ATE
		ADV_IMP
5.시험/검증	블랙, 화이트박스 테스트 -동적 분석, 퍼징(fuzzing), -공격노출면 검토	ATE
6.배포/운영	사고대응계획 -사고대응계획 작성, 최종 보안 검토	AGD
		ALC_CMC
		AVA

하기 위해서 CC의 보증요구사항에 소프트웨어 개발보안의 평가방법을 추가하여 나타냈다. 본 논문은 기본적으로 보증요구사항의 클래스 단위로 평가방법을 제안하지만 클래스 단위보다 상세한 내용이 필요한 경우 패밀리 단위로 제안하였다.

먼저 소프트웨어 개발보안의 교육에 대한 내용이 보증요구사항에 존재하지 않는다. 따라서 ALC_DVS (개발보안) 패밀리의 제안내용에 물리적, 절차적, 인적으로 발생할 수 있는 정보보호제품의 위협을 최소화할 수 있도록 개발자와 평가자의 소프트웨어 개발보안에 대한 교육을 제안하였다.

ASE(보안목표명세서) 클래스는 정보보호제품의 위험을 최소화시키기 위해서 자산과 운영환경을 식별하고 위협 시나리오를 통해 보안대책과 요구사항을 도출한다. 그리고 ALC_TAT(도구와 기법) 패밀리는 정보보호제품을 개발, 분석, 구현하는데 사용된 도구와 기법이 적절하게 사용되었는지 평가한다. 따라서 이러한 보증요구사항들에 소프트웨어 개발보안을 위한 보안담당자 임명, 발생된 버그, 할당된 업무, 사용된 도구 등을 명시 및 추적하도록 했으며 최소한의 보안 기준을 설정하고 위험 관리에 대한 내용을 수행하도록 제안하였다.

ADV(개발) 클래스는 보안기능요구사항이 정확하고 완전한 구현되었는지 보증하기 위한 설계 방법을 인터페이스, 서브시스템, 모듈로 나타낸다. 따라서 이 보증요구사항은 소프트 개발보안을 위한 보안요구사항과 설계 방법을 제시 및 검토, 공격 노출면 분석, 위협 모델링에 대한 내용을 제안하였다.

ATE(시험) 클래스는 정보보호제품의 보안기능이 설계대로 정확히 동작하는지 확인하기 위해 테스트한다. 따라서 소프트웨어 개발보안을 위한 블랙박스 테스트와 화이트박스 테스트에 대한 내용을 제안하였다.

ADV_IMP(구현의 표현) 패밀리는 개발자가 평가자에게 소스코드의 제출을 결정하는 보증요구사항이다. 평가보증등급이 3이하일 경우 소스코드의 제출이 의무화 되어 있지 않기 때문에 평가자는 소스코드를 활용하여 정적 코드 분석 도구를 사용하거나 시큐어 코딩을 확인할 수 없다. 반대로 평가보증등급이 4이상일 경우에는 소스코드 제출이 의무화되어 있기 때문에 정적 코드 분석 도구를 사용할 수 있으며 시큐어 코딩을 하였는지 확인 가능하다. 그리고 동적 분석, 소프트웨어의 설계와 구현된 부분을 비교하여 공격 노출면을 검토하는 내용을 제안하였다.

AGD(설명서) 클래스는 정보보호제품을 사용하기 위한 운영설명서와 설치하기 위한 준비설명서를 제공한다. 그리고 ALC_CMC(형상관리 능력) 패밀리는 소비자에게 배포되기 전까지 정보보호제품의 무결성을 지키기 위한 보증요구사항이다. 따라서 이 보증요구사항들은 소프트웨어 개발보안을 위한 사고대응계획을 수립해야 되며 알려지지 않은 보안취약점을 해결하기 위한 의사결정과 최종적으로 보안 검토를 수행하는 내용을 제안하였다.

마지막으로 AVA(취약성 분석) 클래스는 정보보호제품의 계획/분석, 설계, 시험/검증 단계에서 보안취약성 분석과 침투 시험을 수행하는 내용을 제안하였다.

위와 같이 소프트웨어 개발보안에 대한 평가방법을 CC의 보증요구사항에 추가한다면 개발자가 정보보호제품의 소프트웨어 개발보안을 보증할 수 있는 방법을 결정할 수 있으며 평가자가 정보보호제품의 소프트웨어 개발보안에 대한 보증수준을 결정하는 평가 기준으로 사용될 수 있을 것이다.

V. 결 론

본 논문에서는 설계, 구현, 보안기능, 비보안기능 영역에서 CWE 목록을 기반으로 CC와 MS-SDL을 비교하여 두 평가방법이 제거할 수 있는 보안약점과 보안취약점의 상관관계를 분석하였다. 각 영역의 CC와 소프트웨어 개발보안의 상관관계는 설계-보안기능 영역에서 CC가 단독으로 사용될 수 있으며 그 외 영역에서는 CC와 소프트웨어 개발보안이 상호보완적인 관계이기 때문에 CC에서 소프트웨어 개발보안을 활용하여 보안취약점을 제거해야 한다. 그리고 정적 코드 분석 도구는 CC 평가 시 반드시 사용해야 CC에서 제거하지 못하는 보안약점과 보안취약점을 제거할 수 있다. 즉, CC와 소프트웨어 개발보안은 제거할 수 있는 보안약점과 보안취약점의 같은 부분과 다른 부분이 존재하기 때문에 CC와 소프트웨어 개발보안을 활용하여 모든 보안약점과 보안취약점을 제거해야 한다.

본 논문은 CC에서 소프트웨어 개발보안을 활용하기 위해서 CC의 보증요구사항에 소프트웨어 개발보안의 평가방법을 제안하였다. 제안된 방법과 같이 소프트웨어 개발보안의 평가방법을 CC의 보증요구사항에 추가한다면 개발자는 소프트웨어 개발보안을 활용하기 위한 평가방법을 수행할 수 있으며 평가자는 소프트웨어의 보증 수준을 결정하는 평가기준으로 사용될 수 있을 것이다.

또한, CC에 소프트웨어 개발보안을 활용한다면 국가 및 공공기관의 정보화사업에 도입되는 정보보호제품을 두 가지 평가방법에 대해서 모두 만족시킬 수 있기 때문에 개발자, 평가자의 부담을 최소화 할 수 있으며 정보보호제품의 안전성과 신뢰성을 극대화할 수 있을 것으로 기대된다.

향후 연구계획은 ISO 15026와 ISO 27034 등과 같은 안전한 시스템과 소프트웨어를 개발하기 위한 국내·외 표준들과 CC의 상관관계를 비교 분석한다.

References

[1] Gregory Tassey, "The Economic Impact of Inadequate Infrastructure for Software Testing Planning Report," NIST, May 2002, pp. 169-170.

[2] Paul Wood, "Closing The Window of Vulnerability: Exploits And Zero-Day Attacks," Internet Security Threat Report, vol. 17, Symantec, Apr. 2012.

[3] Brian McGee et al, "Vulnerabilities in Enterprise Software," IBM X-Force 2012 Mid-year Trend and Risk Report, Sep. 2012, pp. 66-68.

[4] Gerhard Eschelbeck, "Systems and Software Threats," Security Threat Report, Sophos, Jan. 2012. pp. 14-16.

[5] Theresa Lanowitz, "Now Is the Time for Security at the Application Level," Gartner, Dec. 2005, pp. 2-8.

[6] Joe Jarzombek, "Software Assurance: Enabling Security and Resilience throughout the Software Lifecycle," MITRE, Nov. 2012, pp. 3.

[7] SwA, Capability Benchmarking, Software & Supply Chain Assurance-Community Resources and Information Clearinghouse, Accessed Jan. 17, 2014, https://buildsecurityin.us-cert.gov/swa/forums-and-working-groups/processes-and-practices/swa-capability-benchmarking

[8] Standard Life Cycle Processes View, Software & Supply Chain Assurance-Community Resources and Information Clearinghouse, Accessed Jan. 17, 2014, https://buildsecurityin.us-cert.gov/swa/process-view/overview

[9] ISO std. 15408, Common Criteria for Information Technology Security Evaluation Version 3.1 Revision 4, ISO, Sep. 2012.

[10] CVE List, Common Vulnerabilities and Exposures (CVE), Accessed Jan. 17, 2014, http://cve.mitre.org/

[11] Richard Struse, "Software Assurance-Making the Software Ecosystem Rugged," U.S Department Homeland Security National Protection &Programs Directorate, Oct. 2011.
[12] Adam O'Brien, "Common Criteria and Source Code Analysis Tools: Competitors or Complement," International Common Criteria Conferences 9th. Conf., Seoul, Rep of Korea, Sep.23-25,2008.
[13] Jeff Jones, "Measurable Improvements at Microsoft," Introduction to the Microsoft Security Development Lifecycle (SDL), Microsoft, Jan. 2008.
[14] "Basics of Secure Design Development and Test: Secure Software Made Easier," Microsoft, 2008.
[15] Ray Potter, "Setting Expectations Common Criteria and the SDLC," International Common Criteria Conferences 9th. Conf., Seoul, Rep of Korea, Sep.23-25,2008.
[16] Mehmet Kara, "Review on Common Criteria as a Secure Software Development Model," International Journal of Computer Science & Information Technology (IJCSIT) vol. 4, No 2, April. 2012. pp. 83-94
[17] B. Chess and C. McGraw, "Static analysis for security," IEEE Security & Privacy, vol. 2, no. 6, pp. 76~79, Nov. 2004.
[18] Bob Martin et al, "2011 CWE/SANS Top 25 Most Dangerous Software Errors," Common Weakness Enumeration (CWE), Sep. 2011.
[19] Assessment and Remediation Tool, Common Weakness Enumeration (CWE), Accessed Jan. 17, 2014, http://cwe.mitre.org/compatible/category.html

〈저 자 소 개〉

박 진 석 (Jinseok Park) 학생회원
2010년 2월: 숭실대학교 컴퓨터학부 졸업
2014년 2월: 고려대학교 정보보호대학원 정보보호학과 석사
〈관심분야〉 정보보증, 정보보호제품 보안성 평가, 정보보호관리체계

강 희 수 (Heesoo Kang) 학생회원
2013년 2월: 중앙대학교 컴퓨터공학부 졸업
2014년 2월: 고려대학교 정보보호대학원 금융보안학과 석사과정

〈관심분야〉 정보보증, 정보보호관리체계, 모바일 보안

김 승 주 (Seungjoo Kim) 종신회원
1994년~1999년: 성균관대학교 정보공학과 (학사, 석사, 박사)
1998년 12월~2004년 2월: KISA(舊한국정보보호진흥원) 팀장
2002년~현재: 한국정보통신기술협회(TTA) IT 국제표준화전문가
2004년 3월~2011년 2월: 성균관대학교 정보통신공학부 조교수, 부교수
2011년 3월~현재: 고려대학교 사이버국방학과/정보보호대학원 정교수
2004년~현재: 한국정보보호학회 이사
2005년~2006년: 교육인적자원부 유해정보 차단 자문위원
2007년: 국가정보원장 국가사이버안전업무 유공자 표창
2007년~2009년: 전자 정부 서비스 보안 위원회 사이버 침해사고대응 실무위원회 위원
2010년: 방송통신위원회 정보통신망 침해사고 민관합동조사단 위원
2012년 3월~2012년 6월: 선관위 디도스 특별검사팀 자문위원
〈관심분야〉 보안공학, 암호이론, 정보보증, 정보보호제품 보안성 평가, Usable Security