情 報 保 護 學 會 誌
第 16 卷 第 4 號, 2006. 8

소프트웨어 생명주기에서의 설계문서에 대한 보안성 체크리스트*

손 경 호†, 김 승 주, 원 동 호‡

요 약

본 논문에서는 소프트웨어 개발 프로세스에서 보안성을 향상시키기 위해, 소프트웨어 개발단계에서 산출되는 문서를 바탕으로 내재되어 있는 취약성을 찾기 위한 체크리스트를 제안한다. 현재 소프트웨어 생명주기내에서 보안성을 지키기 위해서는 설계단계에서의 위험분석이 요구되며, 이를 확인하기 위한 검증단계가 필수적이다. 따라서, 본고에서는 취약성을 찾는 구체적인 방법으로 ISO/IEC 15408(Common Criteria, 이하 CC)[1]기준의 보안성 평가방법론인 CEM[2]에 기반한 취약성검색을 통해 소프트웨어 설계단계에서 산출되는 개발문서에 대해 검증해야 할 항목을 제시한다.

Ⅰ. 서 론

최근 정보보호 문제 발생의 근본적인 해결을 위해 소프트웨어 개발 초기에서부터 안전한 소프트웨어개발 및 소프트웨어 개발 생명주기에서의 보안성을 향상시키기 위한 많은 노력이 이뤄지고 있으나, 현재 시점에서 안전한 소프트웨어 개발을 위한 프로세서와 프랙티스(practice)가 존재하지 않는다. 소프트웨어의 취약성은 대부분이 잘못된 명세, 설계 와 구현에 의해 야기되고 있으며, 일반적인 소프트웨어 개발 방법론에서는 이점을 간과하고 있는 실정이다.

최근 기사 중 "왜 소프트웨어가 그렇게 나쁠까?"[3]에서는 나쁜 습관과 불충분한 소프트웨어 개발 생명주기 공정이 질 낮은 소프트웨어 개발하게 만들게 하고 있으며, "소프트웨어 보안결함은 개발자가 책임져야 한다."[4] 에서도 소프트웨어 개발자는 자신이 작성한 코드 보안을 책임져야 한다고 주장하고 있다.

또한 국내에서도 "IT839전략의 안전한 실현을 위한 소프트웨어 보안표준"[5] 에서 소프트웨어 보안 취약점이 발생하는 주요 원인은 개발과정에서의 명세, 설계 또는 구현단계에서 의도하지 않는 오류에 있으며, 이를 해결하기 위해 소프트웨어 자가 방어를 위한 실행시간 침입탐지 표준, 소프트웨어 보안강도 기준과 측정 절차 표준에 대한 필요성을 제시하고 있다.

본고에서는 소프트웨어 생명주기에서 산출되는 설계문서에 대해 CC기반 평가 방법론(CEM)을 적용해 소프트웨어에 내재되어 있는 오류 및 결함을 찾아내는 방법과 각 제출문서에 대한 보안성을 검증할 수 있는 체크리스를 제공하고자 한다.

이 체크리스트를 통해 각 개발조직은 자신들의 소프트웨어 개발방법론에 이를 적용시킴으로써, 소프트웨어 보안 위험성을 경감시킬 수 있을 것이다.

Ⅱ. 소프트웨어 보안에 대한 현재 상황

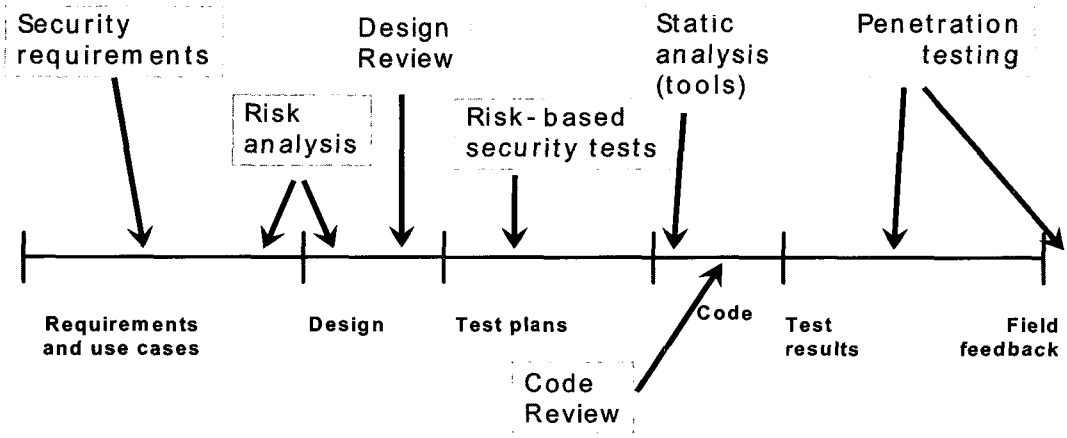
많은 이유로, 안전한 소프트웨어 개발은 복잡하다. 소프트웨어 제품과 그것의 코드는 너무나 크고 복잡하게 이뤄지고 있으며, 그 개발도 다양한 역할과 개발자에 의해 작성되고 있어, 전체 제품에서 요구하는 보안과 관련된 특성이 정확히 구현되기는 쉽지 않다. 또한, 소프트웨어가 갖춰야 할 보안특성과 기능성이 양립하기는 쉽지 않으며, 전체 시스템에서 요구하는 보안 기능성이 최초 계획대로 동작하는지에 대한 확

* 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성·지원사업의 연구결과로 수행되었음.
† 주저자, 한국정보보호진흥원(KISA) 연구원 (khson@security.re.kr)
‡ 교신저자, 성균관대학교 정보통신공학부 교수 (dhwon@security.re.kr)

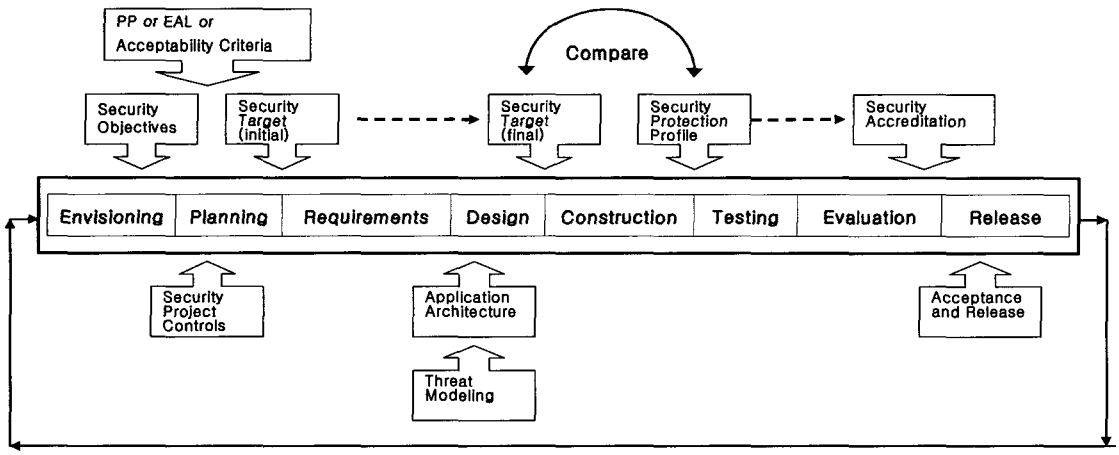
인과정은 쉽지 않다.[6]

또한, 안전한 소프트웨어 개발을 위해 현존하는 다양한 방법들(SSE-CMM, iCMM, CMMI-SE/SW/IPPD, CMMI-A, ISO 9001:2000, EIA/IS 731, Malcolm Baldrige National Quality Award, Total Quality Management, Six Sigma, President's Quality Award criteria, ISO/IEC TR 15504, ISO/IEC 12207, and ISO/IEC CD 15288)[7][8][9]은 대부분의 조직에서 안전한 소프트웨어 개발을 위한 프로세스로 도입을 꺼리고 있거나 지각하고 있지 않으며, 각 프로세서들은 높은 신뢰성과 안전성을 상위개념에서 요구하기 때문에 조직에서 이를 적용하기는 쉽지 않다. 그리고, 이 방법들은 소프트웨어 보안성을 위해 개인, 팀, 프로젝트의 모든 레벨에서의 요구되는 사항에 대해 언급하고 있지 않으며, 안전한 소프트웨어 개발을 위해 드는 비용과 조직의 역량문제, 법적인 강제사항 등이 이유로 많은 조직에서 적용하고 있지 않은 실정이다.

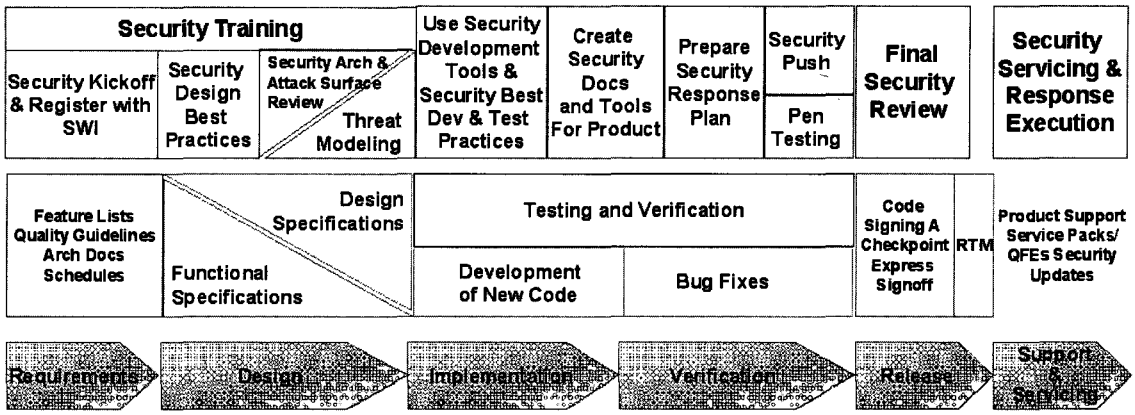
US CERT®의 분석에 따르면, 소프트웨어에서 발견되는 취약성의 90% 이상이 잘 알려진 공격 형태이며, 그 중 10개의 취약성이 전체 취약성 중 75% 이상을 차지한다고 말하고 있으며[10][11], 새로운 코드나 변경된 코드는 1000라인 중에 1-1.5개의 결함을 지닌다고 말하고 있다.[12] 따라서, 초기 소프트웨어 설계 시 이런 결함 및 취약성들을 회피 할 수 있는 설계를 통해 대부분의 취약성이 경감 될 수 있을 것이다. 그러나, 이를 해결하기 위한 프랙티스가



[그림 1] 다양한 소프트웨어 단계에서의 소프트웨어 보안 베스트 프랙티스 적용



[그림 2] 생명주기 보안을 위한 프래임웍



〔그림 3〕 Traditional Microsoft Software Product Development Lifecycle Tasks and Processes

부족해 실제 소프트웨어 개발 단계에서 이를 적용하기는 쉽지 않다.

McGraw가 제안한 안전한 생명주기 프랙티스에 따르면, 생명주기 내에서 개발과 구조적인 설계레벨에서의 위험분석 및 설계 검토, 소스코드 검토 등을 언급하고 있다. 설계단계에서 위험분석을 통해 보안 분석가가 위험 순위를 정하고 이를 경감시키기 위한 활동을 해야 한다고 말하고 있으며, 구현단계인 코드 레벨에서 소스코드 정적 분석 툴 등을 이용해 취약성 검색 등이 필요함을 말하고 있다.〔13〕(그림 1 참조)

IEEE P1074〔14〕에서는 ISO/IEC 12207〔15〕인 소프트웨어 생명주기 프로세스에 기반 해 안전한 코딩 프랙티스 및 생명주기 보안 프레임웍 및 ISO 17799와 ISO/IEC 15408과의 연계를 연구 중에 있으며 특히, 소프트웨어 개발 생명주기 내에서 CC 평가체계를 통한 안전한 소프트웨어 개발방법론을 제안하고 있다.(그림 2 참조)

마이크로소프트에서는 안전한 소프트웨어 개발 프로세스인 보안성 개발 생명주기(Security Development Lifecycle : SDL)를 개발 해 자사의 주요제품 개발 등에 적용하고 있다. SDL은 보안성에 중점을 둔 일련의 개발 활동과 그 성과물이 추가되고, 이 개발 활동과 그 성과물에는 소프트웨어 설계시의 위협 모델 개발, 구현 시의 정적분석 코드 스캔 툴 사용 및 코드 검토와 보안성 시험 등이 있다.〔16〕

SDL을 통해 제품 개발 후 발견된 보안권고 사항이 Window 2000개발에는 62개였으나, Window 2003 개발에는 24개로 87%의 효과를 보았다고 한다.

그러나, Microsoft 사의 SDL은 상업적인 소프트웨어를 개발하는 관점에서 시도되었기 때문에 소프트웨어 개발 프로젝트의 시작과 끝이 기술적인 관심사항에만 집중되어 있어, 식별된 위협들에 대해 기술적인 대응수단을 강구하기 때문에 이를 실제 다양한 분야에 적용하기는 어렵다고 한다.〔17〕(그림 3 참조)

IEEE, Microsoft사 등의 다양한 움직임에도 불구하고, 실제적인 소프트웨어 프로세스와 프랙티스가 부족해, 최근 미국의 국토안보국(DHS)과 민간 IT 업계가 주축이 되어 National Cyber Security Partnership(NCSP)을 발족하고 소프트웨어 보안을 위한 프랙티스 제공 등을 개발 중에 있다.

구체적으로 NCSP의 Security Across the Software Development Life Cycle 그룹에서는 안전한 소프트웨어 개발 향상을 위해 교육/연구 기관과의 유대강화, 소프트웨어 스펙/설계/사용상의 결함을 삭감시키는 소프트웨어 개발 프로세스 개발, 보안 패치 및 안전한 소프트웨어 개발을 위한 기술 프랙티스와 관리 프랙티스를 개발 중에 있다.〔6〕

Ⅲ. 개발관련 보증문서

1. 개발프로세스와 개발문서

전통적인 소프트웨어 개발 프로세스(Waterfall 모델)에서 산출되는 개발문서들과 CC 평가 시 요구되는 보증문서에 대한 차이점은 아래 표와 같으며, 본고에서는 개발 프로세스 중 설계와 관련된 CC기반 보증문서인 기능명세서, 기본설계서, 상세 설계서, 소스코드를 대상으로 한다.

개발 프로세스	개발 문서	CC 보증 문서	CC요구 사항
–	–	*보안목표명세서(ST)*	보안 목표를 설정한 문서(보안 요구사항 설계서)
개발 준비	생명주기 모델서	생명주기 지원 증거 자료	개발 및 유지보수의 생명주기 모델 정의
	개발 환경 규정서	개발 보안 증거 자료	개발 환경의 물리적, 절차적, 인적, 및 그 외의 보안 대응방법에 대해 규정
		개발 도구 증거 자료	개발에 사용하는 개발 도구, 기술을 식별해, 도구·기법의 사용법을 정의
	개발 프로세스 수행 계획서	형상 관리 규정서	형상관리의 절차·방법·순서를 규정한 사내 기준·규정·가이드라인. 관리자 부주의 등에 의한 인위적인 오류를 경감하기 위한 툴 사용 방법 등을 포함
	형상항목(목록)	형상 목록	TOE를 생성하기 위해 필요한 개발 대상(소스 코드, 매뉴얼, 시방서 등)의 목록
시스템 설계	시스템/소프트웨어 설계서	기능명세서	보안기능을 기술하고, TOE의 보안기능 요구사항을 구체화
내부 설계	기본 설계서	상위레벨 설계서	기능 사양을 보안기능의 주요한 구성 부분(하부조직)을 상세화
상세 설계	상세 설계서	하위레벨 설계서	상위 레벨 설계를 모듈 및/또는 하드웨어를 작성하기 위한 기초로서 사용할 수 있는 레벨에서의 상세화
프로그래밍	소스/오브젝트 코드	구현표현서	소스코드, 하드웨어 도면 등에서 보안기능의 상세한 내부 동작을 나타내는 것
–	–	*기능대응*	TOE 요약 명세로부터 구현표현까지의, 모든 인접하는 개발 문서간의 보안기능 대응
	–	*보안정책 모델*	보안 정책의 구조화 표현이며, 기능 명세가 보안정책과 일치해, 최종적으로 TOE 보안기능 요구사항에 일치하고 있는 것을 제시
테스트	시험 품질 보증서	시험범위 분석	시험을 실시한 범위(어떤 보안기능을 시험했는가)를 제시
		시험깊이 분석	시험을 실시한 레벨(하부조직/내부 인터페이스까지 의식해 테스트하고 있는지)을 제시
	시험계획 및 시험결과	시험 증거 자료	시험계획, 순서, 기대하는 테스트 결과 및 실제 테스트 결과
매뉴얼 작성	관리자 설명서	관리자 설명서	올바른 방법으로 TOE를 구성, 보수, 관리하는 것에 책임이 있는 사람에게 사용되는 것을 목적으로 한 문서
	사용자 설명서	사용자 설명서	관리자 이외의 사용자에게 사용되는 것을 목적으로 한 문서
출하	배포 수속서	배포 증거 자료	TOE가 수정되는 일 없이 배포되는 것을 기술한 문서
도입	설치 가이드, 설치 순서서	설치, 생성, 시동	개발자가 의도한 대로 안전한 설치, 생성 및 시동하도록 기술한 문서
–	–	*오용 분석*	사용자나 관리자가 인식해야 할 취약성에의 대응(설명서의 기술 등)이 충분한 것을 분석
	–	*기능 강도 분석*	확률/통계적 메커니즘과 관계되는 취약성에 대한 강도를 분석
	–	*취약성 분석*	알려진 공격자(저위, 중위, 고위)에 대해서, 취약성이 발생하지 않는 것을 분석

이탤릭 밑줄 글자는 CC에서 신규로 작성하는 문서

2. 개발관련 문서의 세부작성 내용

CC의 개발(Development, ADV) 보증관련 문서는 아래와 같은 상호연관관계를 가지며, 보안목표명세서(ST, Security Target)에서는 CC의 세부보안요구사항이(SFR, Security Function Requirement)이 제품의 보안기능(SF, Security Function)으로 매핑되고 기능명세서는 이 기능들에 대한 인터페이스를 기술한다. 기본설계서는 제품의 평가범위인 TOE(Target of Evaluation)를 서브시스템(Sub-System) 단위로 나누고, 상세설계서는 모듈(Module) 단위로 나눈다. 그리고, 마지막으로 구현표현(Implementation)은 모듈을 소스코드 등으로 구현 및 실체화한다.

각각의 제출물이 담고 있어야 하는 중요사항 및 그들 간의 상호관계는 아래와 같다.(그림 4 참조)

▸ 기능명세서(Functional Specification, FSP)
- 제품의 모든 외부인터페이스 식별(예, 사용자와 관리자에 의해 보이거나 접근이 가능한 인터페이스들) 예, APIs, configuration files
- 보안목표명세서(ST)에 정의된 보안기능에 대한 인터페이스 식별(TSFI - TOE security function interfaces)
- 모든 보안기능 인터페이스 완전한 서술(예, 각 인터페이스의 보안관련 입력 파라미터, 인터페이스에 의한 사건발생, 필요한 인터페이스 접근 권한, 응답메시지와 관련된 인터페이스 출력, 에러코드 등)

▸ 기본설계서(High-level Design, HLD)
- 구조적인 "서브시스템" 식별(예, 제품 영역에서의 기본적인 보안 기능영역의 모듈화 표현)
- 제품의 하부 하드웨어, 펌웨어, 소프트웨어에 구현된 보조적인 보호 메커니즘에 의해 제공되는 기능 서술
- 하나의 서브시스템과 그들 간의 상호관계에 의해 구현된 보안기능성 서술
- 서브시스템의 모든 인터페이스 식별(외부에서 보여지는 접입점(Entry point)을 기능명세와 같이 서술

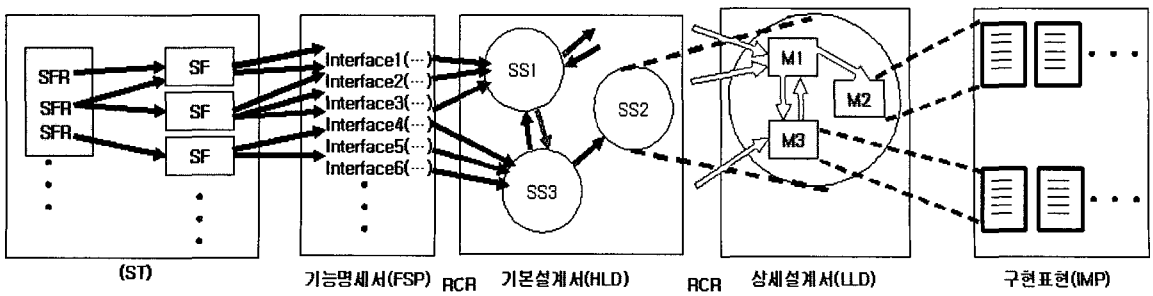
▸ 상세설계서(Low-level Design, LLD)
- 보안기능이 구현된 하위 수준의 모듈에 대해 식별하고 그들의 인터페이스 서술
- TSF 모듈에 대한 모든 인터페이스의 사용 목적, 사용 방법, 효과, 예외사항, 오류 메시지 등에 대한 세부사항을 제공

▸ 구현표현명세서(Implementation and Source Code, IMP)
- 더 이상의 설계과정 없이 TSF가 생성될 수 있는 상세수준으로 TSF를 모호하지 않게 정의
- TOE 보안기능요구사항을 정확하고 완전하게 실체화하고 있는지 여부

Ⅳ. CEM 기반 취약성 검색

CEM에서는 "우회(Bypassing)", "침해(Tampering)", "직접공격(Direct Attack)", "오용(Misuse)" 이라는 4가지 일반적인 공격방법에 기초해 TOE에 대한 취약성들을 식별하는 방법론을 제공하고 있다.

침해와 오용은 공격자가 보안기능 동작에 대해 공격자가 악용하기 위해 변경하는 것이다. 이 둘의 차이점은 "누가"(또는 무엇이) 행동의 변화를 야기시키냐는 것이다. 침해의 경우, 공격자가 능동적으로 행동을 변경시키고, 오용은 사람 또는 다른 운영상의 오류에 의해 행동의 변경이 발생한다. 이 공격들을



〔그림 4〕 각 개발보증 문서들간의 관계

식별하기 위해서는 SF의 행동 변화를 발견하는 것이 필요하다.

대조적으로 우회는 보안기능 변경을 통한 행동변화에 의존하지 않는다. 공격자는 개발자가 예상치 못한 경로를 통해 자산에 접근하기 위한 경로를 찾는다. 만약 우회공격이 성공한다면, 이는 변경된 경로 상에서 보안기능이 호출되지 않았음을 의미하거나 보안기능이 자산을 보호하기 위해 효과적으로 호출이 되지 않았음을 의미한다. 이 공격을 식별하기 위해서는 보호가 필요한 자산에 변경된 경로를 찾는 것이다.

직접공격은 보안기능의 동작을 변경하거나 변경된 경로를 찾는 시도가 아닌, 보안기능 자체를 무력화 시키는 것을 의미한다. 이 공격을 식별하기 위해서는 SF 구현에 사용된 하부 메커니즘의 약점을 찾는 것이다.

예를 들어, 아래와 같이 주체와 객체 속성에 따라 Client의 접근 요청을 중재하는 접근통제(Access Control) 메커니즘이 있다고 하자.(그림 5 참조) 만약, 접근이 허용되면, 객체 서버(Object Server)에게 알리고, 클라이언트는 객체서버와 동작해 자산인 Object에 접근에 접근한다.

여기서 우회 공격은 객체에 직접적인 접근 시도(객체서버를 우회)하거나 객체서버 인터페이스를 통해 접근(AC 메커니즘 우회) - 'notify' 메시지를 속이는 시도를 포함한다.

또한 침해공격은 주체 또는 객체의 속성을 변경시켜 요청이 허용되게 하거나, AC 메커니즘 프로세서에 대해 메커니즘이 동작되지 않게 하는 것을 말한다.

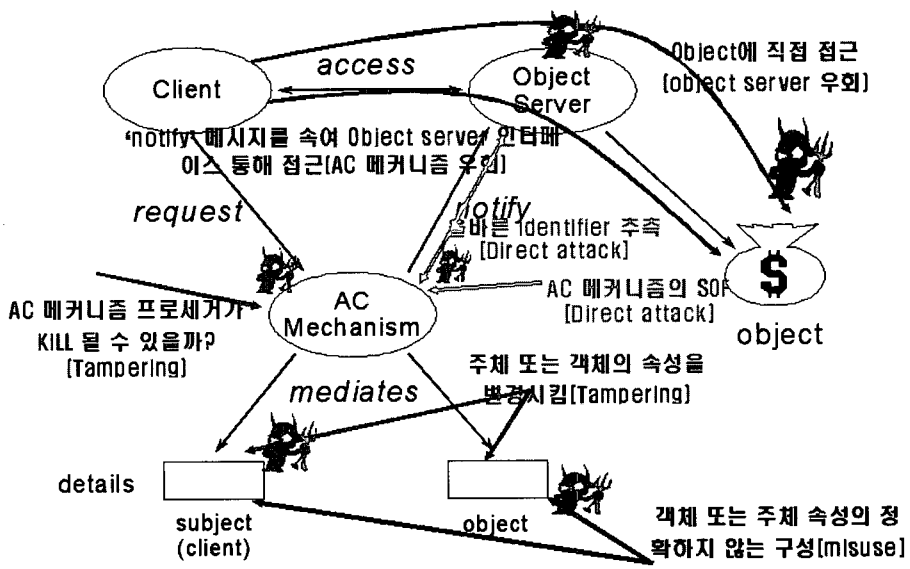
직접공격은 위 메커니즘에서는 잠재적인 공격방법이 되지 못할 것이다. 그러나, 객체서버에게 알림(notify)을 하는 대신에 클라이언트에게 객체에게 식별자를 전달하도록 메커니즘이 구현된다면, 클라이언트는 객체서버에게 접근 권한을 증명하기 위해 식별자를 제출할 것이다. 공격자가 객체서버와 상호 동작 할 때 정확한 식별자를 추측하는 시도를 통해 직접공격이 가능할 것이다.

오용은 객체 또는 주체의 속성의 잘못된 구성(Configuration)을 통해 발생할 수 있다.

V. 개발문서에 대한 취약성 검색 체크리스트

1. 우 회

우회는 공격자가 "TOE의 인터페이스 기능 또는 TOE와 상호작용할 수 있는 유틸리티의 기능 악용", "거부되어야 하는 권한 또는 기타 능력 상속", "(비밀성이 문제가 되는 경우) 부적절하게 보호된 영역에 저장되거나 복사된 중요 데이터 읽기"에 의한 보안 수행을 피할 수 있는 모든 수단을 포함한다.



[그림 5] 주체-객체 시스템 예

공격 방법	제출물	활동
인터페이스 기능 악용		
순서 변경	FSP	인터페이스 호출 순서에 의해 어떤 보안에 관련된 동작이 구현되어 있는가?
		어떤 인터페이스 호출이 순서에 의해 적절한 SF가 호출되지 않는가?
		TOE와 IT환경 사이에 상호 동작하는 정의된 순서가 있어 변경이 의심되는 것이 있는가?(예, man-in-the middle 공격?)
	HLD	TSF 서브시스템 행동 순서에 의해 구현된 SF가 있는가?
		그런 순서에 의해 TSF 서브시스템 도는 그들 사이에 통신이 변경될 수 있는가?
예상치 못한 인터페이스 사용 -내용 또는 목적	FSP	어떤 일반적인 목적의 TSF 인터페이스가 예상치 못하게 보안을 해치는 목적으로 사용될 수 있는가?
	LLD IMP	잠재적으로 인터페이스를 예상치 못한 방법으로 사용 될 수 있게 끔 LLD 또는 구현표현에 구현에 대한 상세가 소개되어 있는가?
구현 상세 이용	HLD LLD IMP	TSF 표현에서 불법의 데이터 흐름을 노출시킬 수 있는 어떤 공유된 자원이 명백한가?
시간 지연 이용	FSP HLD	위에서 식별된 순서들에서 시간 지연 주입이 의심되는 것이 있는가? 중간 또는 잠시 동안 이전의 수행을 무효화시킬 수 있는 어떤일이 발생할 수 있는가?
권한 상속		
예상치 못한 실행	FSP HLD	TOE에 권한 프로그램 또는 응용이 무엇인가? 그들의 외부 인터페이스로부터 직/간접으로 데이터가 입력되는 잠재적인 실행이 있는가?
예상치 못한 입력 [B2.2]	FSP	권한 프로그램과 응용의 외부 인터페이스로 예상치 못한 입력으로 불능을 야기시키거나 잘못된 권한상속이 발생할 수 있는 어떤 잠재사항이 존재하는가?
무효 가정사항/특성	HLD	주어진 계층 또는 레벨에서 SF가 수행될 때 더 낮은 계층/레벨에서 자산에 접근 할 수 없는 가정사항에서 접근을 제공하는가? 그런 가정사항들이 유효하거나 그들이 어떤 방식으로 훼손될 수 있는가?
중요 데이터 읽기		
보호되지 않는 영역	HLD FSP	만약 TOE가 중요 데이터를 가지고 있고 데이터에 대한 복사본(임시 또는 영구)을 관리하는가? 만약 그렇다면, 그것이 어디에 저장되고, 누가 그 영역에 접근이 가능한가?
공유자원 접근 이용	HLD FSP	중요 사용자 데이터의 복사본을 포함하는 공유된 자원이 무엇인가? 어떻게 그런 자원에 대한 접근 통제가 이뤄지나?
오류복구 이용	FSP	만약 오류복구가 호출되면, 어떻게 TOE가 중요한 사용자 데이터를 처리하는가?(예, 복구 또는 임시 파일로?) 어디에 저장되며 누가 그들에 접근이 가능한가?

2. 침 해

침해는 "보안기능이나 메커니즘이 의존하는 비밀성 또는 무결성 데이터 접근", "TOE가 일반적이지 않거나 예상하지 못한 상황에 대처하도록 함", "보안 시행을 무력화 또는 지연"과 같이 보안기능이나 메커니즘

공격 방법	Source	활동
데이터 접근		
내부 데이터 접근	FSP HLD LLD IMP	SF들에 사용되는 내부 TSF 데이터가 무엇이며, 어디에 저장되는가(각각의 구현된 증거에 따라)? 그런 TSF 데이터에 어떤 개체가 접근 또는 접근권한을 가질 수 있나?
예상치 못한 상황 또는 목적	FSP HLD LLD IMP	SF에 의해 사용되는 내부 TSF데이터가 무엇이며, 그것에 대해 어떻게 접근통제가 이뤄지나? 이 통제를 우회하는데 사용될 수 있는 어떤 TSF데이터가 있는가(FSP와 LLD의 특별한 정보에 기반해)?
간섭 악용	FSP HLD LLD IMP	어떤 내부 TSF 데이터가 공유된 자원에 저장되는가 (각각의 표현에서 보여지는)? 어떤 개체들이 이 자원을 공유하는가? 어떤 자원들이 수정되기 쉬운가?
예상치 못한 입력/상황 강제		
예상치 못한 입력 생성	FSP HLD	SF의 어떤 종속물이 설계로부터 식별될 수 있는가? 이 종속물들이 TSF 인터페이스를 경유한 예상치 못한 입력에 의해 훼손될 수 있나?(예, 예상치 못한 형태의 데이터 또는 예상치 못한 크기 또는 값)
유효하지 않은 가정사항	LLD IMP	SF를 구현한 모듈(LLD 또는 코드)들이 명백하거나 암시적으로 그들의 데이터가 취급되는 가정사항들이 있는가? 그런 가정사항들이 훼손될 수 있나?
보안정책 수행을 무력화 또는 지연		
순서 붕괴	HLD FSP	어떤 SF들이 이 형태의 침해가 발생될지 의심되는가? 스케쥴링 또는 종료의 의해 수정될 수 있는 과정에 의해 구현된 SF가 있는가?
동시발생 붕괴	HLD LLD	어떤 SF들이 이 형태의 침해가 발생될지 의심되는가? SF에 의해 사용되어 잠겨지는 공유된 자원에 의존하는 그런 SF들이 있는가?
간섭 악용	HLD LLD IMP	어떤 SF들이 이 형태의 침해가 발생될지 의심되는가? 어떤 SF들이 지연에 의해 노출될 수 있는 종속물들이 있는가? 이들이 다른 내부 개체들에 의해 작용되거나 조작될 수 있나?

의 행동에 영향을 주기 위한 공격자의 시도(즉. 변조 또는 비활성화)에 기반한 모든 공격을 포함한다.

3. 직접공격

직접공격은 CEM에서 확률 및 순열 메커니즘과 다른 메커니즘들의 강도를 시험하기 위하여 필요한 모든 침투 시험들을 식별하고 그들이 직접공격에 견디는 것을 보증함을 포함한다.

TOE에 구현된 메커니즘의 보안기능 강도는 공격자의 공격 잠재력(Attack Potential) 계산을 통하여 평가결과를 얻는다. 공격 잠재력을 계산하기 위한 요소에는 경과시간(Elapsed Time), 전문성(Expertise), TOE에 대한 지식(Knowledge of TOE), TOE에 대한 접근(Access to TOE), 장비(Equipment)의 5가지 항목이 있다. 경과시간은 공격자가 공격을 시작해서부터 성공할 때까지의 시간을 말하고 전문성은 공격자의 전문지식의 소유정도를 말한다. TOE에 대한 지식은 TOE 관련 지식의 소유정도를 말하고 TOE에 대한 접근은 공격자가 공격을 시작하기 전에 TOE에 접근 하는데 필요한 시간을 말하며 장비는 공격자가 공격할 때 사용하는 장비의 수준을 말한다.

따라서, 직접공격은 개발관련 문서들에서 보안관련 메커니즘을 식별하고, 각 메커니즘들에 대한 보안기능 강도 계산을 통해 확인 및 검증될 수 있다.

4. 오 용

오용은 "미비한 설명서", "이치에 맞지 않는 지침서", "TOE의 의도하지 않은 잘못된 구성", "TOE의 무리한 예외 행동" 등에 의해 발생한다.

TOE가 안전하지 않은 상태로 들어갈 수 있는 잠재력은 ST, 기능 명세서와 TOE 설계에 관련된 구성 요소가 포함되어 있는 명세서등의 제출물을 확인한다. 오용에 대한 확인사항으로는 "시동 시, 종료 시, 오류 복구 활성화 시 TOE의 행동", "극단적인 상황 하에서 TOE의 행동(때때로 과부하 또는 점근적 행동으로 불림), 특히 보안 수행 기능 또는 메커니즘을 비활성화 또는 무력화시킬 수 있는 경우의 극단적인 상황과 침해(tampering) 부분에서 언급한 공격으로부터 발생되는 의도하지 않은 잘못된 환경구성 또는 안전하지 않은 사용에 대한 가능성에 대한 항목이 검증되어야 한다.

V. 결 론

본 논문에서는 소프트웨어 생명주기 모델에서 개발단계에서 산출되는 설계관련 문서를 CEM에 기반해, 각 개발문서들에 대해 보안성을 검증할 수 있는 체크리스를 제안하였다. 이를 통해 조직에서 소프트웨어 개발 시에 설계문서를 비롯한 코드 상의 취약성을 미연에 방지해 향후 발생할 수 있는 위험을 경감시키고 안전한 소프트웨어 개발을 가능케 할 것이다.

또한, CC평가를 받고자 하는 조직에서는 CC 기반 제출물 작성에 참고할 수 있을 것이며, 향후 개발단계에서 조건문 오류, 구성 오류, 부적절한 인터페이스 등으로 발생하는 소프트웨어 보안 취약점을 감소시키기 위한 안전한 소프트웨어 프로세서와 프랙티스 개발에도 활용될 수 있을 것이다.

참 고 문 헌

[1] "Common Criteria for Information Technology Security Evaluation Version 2.3," Aug. 2005, http://www.commoncriteriaportal.org/public/expert/index.php?menu=2

[2] International IT Security Evaluation Community, "Common Evaluation Methodology 2.3", Aug. 2005

[3] C. Mann, "Why Software Is so Bad," Technology Review (July/August 2002).

[4] "Hold developers liable for flaws" By Tom Espiner, ZDNet (UK)

[5] "IT839전략의 안전한 실현을 위한 소프트웨어 보안표준", 김홍근, 정보통신표준화 논문, TTA

[6] Improving Security Across The Software Development Life cycle, Task force Report, April 2004, (http://www.cyberpartnership.org)

[7] Herbsleb, J. et al. "Benefits of CMM-Based Software Process Improvement: Initial Results." CMU/SEI-94-TR-013, Software Engineering Institute, Carnegie Mellon University, 1994.

[8] Goldenson, Dennis R. and Gibson, Diane L. "Demonstrating the Impact and Benefits of CMMI", Special Report CMU/SEI-2003-SR-009, The Software Engineering Institute, Carnegie Mellon University, 2003

[9] Jones, Capers. Software Assessments, Benchmarks, and Best Practices, Reading, MA: Addison-Wesley, 2000.

[10] Hayes, W. and J. W. Over, "The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers." CMU/SEI-97-TR-001, ADA335543. Pittsburgh, PA: The Software Engineering Institute, Carnegie Mellon University, 1997.

[11] Davis, Noopur, and Mullaney, Julia, "The Team Software Process in Practice: A Summary of Recent Results," Technical Report CMU/SEI-2003-TR-014, September 2003.

[12] Jones, Capers. Software Assessments, Benchmarks, and Best Practices, Reading, MA: Addison-Wesley, 2000.

[13] [McGraw and Morrisett] Gary McGraw and Greg Morrisett, "Attacking Malicious Code: A report to the Infosec Research Council", submitted to IEEE Software and presented to the Infosec Research Council. http://www.cigital.com/~gem/malcode.pdf [McGraw 2004] McGraw, Gary, "Software Security", IEEE Security and Privacy, to appear March 2004

[14] IEEE P1074-2005:Roadmap for Optimizing Security in the System and Software Life Cycle © Bar Biszick-Lockwood/ QualityIT Redmond, WA 2005
[15] ISO/IEC 12207 Software Life Cycle Processes http://www.12207.com/
[16] Howard, M., and S. Lipner, "Inside the Windows Security Push," IEEE Security & Privacy, vol.1, no. 1, 2003, pp. 57-61. and MicroSoft page, http://blogs.msdn.com/michael_howard/
[17] Bar Biszick-Lockwood, IT Quality and Security Assurance, "Framework Solution for Life Cycle Security"
[18] D. Gilliam, J. Kelly, M. Bishop, "Reducing Software Security Risk Through an Integrated Approach," Proc. of the Ninth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (June, 2000), Gaithersburg, MD, pp.141-146.
[19] Hall, Anthony, and Roderick Chapman, Correctness by Construction: Developing a Commercial Secure System, IEEE Software, January/ February 2002, pp.18-25.
[20] Neumann, Peter, Principles Assuredly Trustworthy Composable Architectures: (Emerging Draft of the) Final Report, December 2003

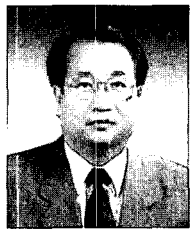
〈著 者 紹 介〉

손 경 호 (Kyungho Son)
회 원

2001년 2월 : 성균관대학교 전기전자컴퓨터공학과(공학사)
2004년-현재 : 성균관대학교 컴퓨터공학과 석·박사과정 재학중
2001년-현재 한국정보보호진흥원(KISA) 연구원
관심분야: 정보보호, 정보보호제품 및 시스템 보안성평가, 스마트카드 취약성

김 승 주 (Seungjoo Kim)
종신회원

1994년 2월 : 성균관대학교 정보공학과(공학사)
1996년 2월 : 성균관대학교 대학원 정보공학과(공학석사)
1999년 2월 : 성균관대학교 대학원 정보공학과(공학박사)
1998년 12월-2004년 2월 : 한국정보보호진흥원(KISA) 팀장
2001년 1월-현재 : 한국정보보호학회 논문지편집위원
2002년 4월-현재 : 한국정보통신기술협회(TTA) IT 국제표준화 전문가
2004년 3월-현재 : 성균관대학교 정보통신공학부 교수
관심분야 : 암호이론, 정보보호표준, 정보보호제품 및 스마트카드 보안성 평가, PET



원 동 호 (Dongho Won)
종신회원

1976년-1988년 : 성균관대학교 전자공학과(학사, 석사, 박사)
1978년-1980년 : 한국전자통신연구원 전임연구원
1985년-1986년 : 일본 동경공업대 객원연구원
1988년-2003년 : 성균관대학교 교학처장, 전기전자 및 컴퓨터공학부장, 정보통신대학원장, 정보통신기술연구소장, 연구처장.
1996년-1998년 : 국무총리실 정보화추진위원회 자문위원
2002년-2003년 : 한국정보보호학회장
현재 : 성균관대학교 정보통신공학부 교수, 한국정보보호학회 명예회장, 정보통신부지정 정보보호인증기술연구센터 센터장
관심분야 : 암호이론, 정보이론, 정보보호