5

시큐어 소프트웨어 개발을 위한 소프트웨어 개발생명주기 동향

고려대학교 ▌이송희 · 최진영**
서울시립대학교 ▌강인혜**
성신여자대학교 ▌서동수**
한국인터넷진흥원 ▌안재영
행정안전부 ▌한근희*

1. 서 론

지금까지 대부분의 소프트웨어는 인터넷에 연결되지않은 형태로 소프트웨어 개발시 올바른 기능 동작 여부만을 점검하여 소프트웨어 내에 존재하는 오류를 효과적으로 예방하여 소프트웨어의 기능성 및 안전성(Safety)을 향상시키기 위해 노력하였다. 그러나 최근 인터넷에 연결된 소프트웨어에 대한 의존이 심화되면서 해킹, 바이러스, 프라이버시 침해, 사이버 테러등과 같은 보안문제가 날로 심각해지고 있다. 이에따라, 이제는 소프트웨어의 안전성뿐만 아니라 소프트웨어의 보안성도 중요시하게 되었다. 소프트웨어 내에 존재하는 오류(error) · 결함(flaw)과 같은 소프트웨어의 약점(weakness) 등을 효과적으로 제거하여 안전하게 자산을 보호하고 결함 없는 서비스의 제공을 보장하기 위해 많은 투자와 노력이 기울여지고 있다. 특히 행정, 국방, 교육 등 공공부문에서의 안전한 서비스 제공은 개인이나 기업 차원을 넘어선 국가 차원의 관심사가 되고 있는 실정이다.

시큐어 소프트웨어(Secure Software)는 보안관련 기능을 수행하는 소프트웨어가 아니라 신뢰성이 위협받는 상황에도 시스템이 신뢰할 수 있는 상태로 유지될 수 있도록 소프트웨어에서 약점을 줄이거나 제거하도록 하는 것으로 미 국토안보부의 한 보고서에서는 시큐어 소프트웨어는 다음의 세 가지 속성을 만족해야한다고 기술하고 있다[1].

첫째, 신뢰성(Dependability)을 제공해야한다. 신뢰성 있는 소프트웨어는 공격이 들어오거나, 악의적인 호스트에서 동작하는 상황을 포함한 모든 조건에서 예측 가능한 실행을 하고 올바른 작동을 해야 한다. 둘째, 신뢰가능성(Trustworthiness)을 제공해야한다. 신뢰 가능한 소프트웨어는 소프트웨어 내에 공격가능한 약점이나 악의적인 논리가 존재하지 않아야하고 내 · 외부적으로 삽입되지 않아야한다. 즉, 신뢰 가능성으로 간주되려면, 소프트웨어는 악의적인 논리를 반드시 전혀 포함하지 않아야 한다. 마지막으로 생존성 혹은 회복성(Survivability or Resilience)을 제공해야한다. 소프트웨어는 알려진 공격들과, 추가로 가능한 한 많은 새로운 공격들에 저항하고 견디기 충분해야하며, 저항하지 못하고 견디지 못한 공격들에 대해서는 가능한 한 빨리 회복하여 최대한 적은 손상을 얻을 정도의 회복력을 가진 소프트웨어이어야 한다.

이러한 시큐어 소프트웨어를 개발하기 위해서 가장 우선시 되어야하는 것은 소프트웨어를 만드는 개발자의 자각, 의지, 주의이다[1]. 보안을 신경 쓰고 자각하는 소프트웨어 전문가는 소프트웨어의 개념 혹은 구현 중간의 어느 시점에서라도 소프트웨어 취약성과 허점이 일어날 수 있다는 것을 알아차릴 것이다. 즉, 불충분한 요구사항으로부터, 부족한 설계와 구현, 부주의한 코딩 오류나 설정 실수까지 점검할 수 있다. 요구사항 분석가는 소프트웨어에 필요한 사항을 어떻게 동작 가능한 요구사항에 시큐어 하게 옮길 수 있는지 이해하고 있어야 하고, 설계자는 시큐어 설계 정책과 모순을 반드시 인지하고 선택해야 하며, 프로그래머는 시큐어 코딩 관습을 따르고 코딩 오류를 피하려고 조심하며 피할 수 없는 버그는 찾아서 삭제해야 한다. 소프트웨어 통합자는 취약한 컴포넌트(COTS, 오픈소스, 주문 제작)로부터 야기되는 보안 위협을 인식하고 줄이려고 노력해야하고, 그런 모듈들과 컴포넌트들 중 제거할 수 없는 취약성들이 최소한 알려지도

* 정회원
** 종신회원

록 통합하는 방법을 반드시 이해하고 있어야 한다.

또한 시큐어 소프트웨어를 개발하기 위해서 우선시 되어야할 점은 소프트웨어 개발 생명주기(SDLC : Software Development Life Cycle)전반에 걸쳐 보안 관습을 추가하는 것이다[1,2]. 즉, 소프트웨어 개발시 단계별 보안강화요소를 체계적으로 정리한 개발 프로세스를 제정하여 정리된 시큐어 소프트웨어 개발 방법론을 제공하는 것이다. 이러한 시큐어 소프트웨어를 위한 개발 방법론은 미국을 중심으로 연구가 활발히 진행중에 있으며, 국내에서는 SI업체를 중심으로 자체적인 개발방법론을 보유하고 있는 곳도 있으나 아직 그 체계는 미비한 실정이다.

이에, 본 연구에서는 시큐어 소프트웨어를 위한 개발방법론의 해외 모범사례의 최근 동향을 다루며, 각 개발방법론의 단계별 프로세스를 소개한다. 본 연구는 다음과 같이 구성된다. 2장에서는 마이크로소프트(MS : Microsoft) 회사에서 개발된 MS-SDL(Security Development Lifecycle)에 대해 소개하고, 3장에서는 CLASP 방법론, 4장에서는 Seven-Touchpoint 방법론, 5장에서는 TSP-Secure 방법론에 대해 소개하고, 6장에서 결론을 맺는다.

2. MS-SDL 방법론

마이크로소프트에서는 보안을 소프트웨어 개발 프로세스의 주요 요소로 간주하고 있다. 2002년 1월 "Trustworthy Computing(TwC)"이라는 구호 아래 마이크로소프트 내의 소프트웨어 개발 그룹에서는 코드 내의 보안성 개선을 위한 보안 강화를 강조해 왔다. 하지만 단순한 개발자들의 보안 작업만으로는 보안성 강화는 매우 어려운 일이었다. 마이크로소프트는 2달간의 시간을 내어 윈도우 기능 개발을 멈추고 윈도우서버 2003의 보안을 향상시키는데 전념하였다. 마이크로소프트에서 내세운 시큐어 소프트웨어를 개발하기 위한 기본 원리는 Secure by Design, Secure by Default, Secure in Deployment, Communications(SD3+C)이다[3,4].

- o Secure by Design : 아키텍처/설계/구조에서의 보안 고려, 설계 시 위협 모델링과 완화 방법 제공, 알려진 취약성 제거, 사용 중인 코드의 보안성 강화
- o Secure by Default : 최소 권한으로 실행, 하나의 위협에 대하여 두 개 이상의 위협완화 솔루션 제공, 보수적인 디폴트 셋팅, 위험을 야기하는 디폴트 변경 방지, 사용하지 않는 기능은 디폴트 세팅에서 제외시킴
- o Secure in Deployment : 배포 가이드 생성, 분석과 관리 도구 사용, 패치 배포 도구 제공
- o Communications : 보안 취약성 리포트 및 보안 업데이트 신속 제공, 사용자의 보안 관련 질문에 적극적 대응 보장

보안에 대한 이러한 움직임은 지속적으로 유지되어 보안 개발 생명주기(SDL : Security Development Lifecycle)을 도입하였다. SDL은 개발 생명주기 내에 단계별 주체의 보안 활동을 통하여 소프트웨어 보안을 강화한다. 예를 들면 소프트웨어 설계 단계에서는 위협 모델의 개발, 구현 단계에서는 정적 분석 도구 사용을 통한코드 스캐닝, 검증 단계에서는 코드 검증 및 보안성 테스팅 작업이 포함되어 있다. SDL에 따라 개발된 소프트웨어는 출시하기 전에 개발팀과는 별도의 팀에 의하여 최종 보안 검증(Final Security Review)을 거치도록 한다.

2009년 6월 발표된 SDL 4.1은 SD3+C의 모든 요소들을 개발 프로세스에 더함으로써 다음 그림 1과 같은 안전한 소프트웨어 개발 프로세스 모델을 제안한다.

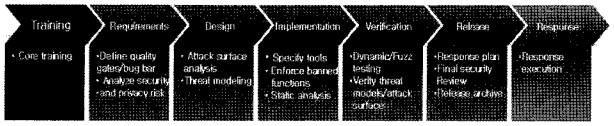


그림 1 MS-SDL

MS-SDL[3,4]은 Pre-SDL(교육) 단계, 요구사항 단계, 설계 단계, 구현 단계, 검증 단계, 릴리즈 단계, Post-SDL(대응) 단계로 구성되어 있다. 각 단계별 보안활동의 적용 수준을 필수 보안활동과 권고 보안활동의 두 단계로 구분하고 있다. 필수 보안활동의 경우는 반드시 수행해야하는 활동이고, 권고 보안활동은 상황에 따라 선택적 적용할 수 있는 활동이라 볼 수 있다. 각 단계별 세부 내용은 다음과 같다.

2.1 Pre-SDL 단계

보안 교육은 소프트웨어 개발팀의 구성원들이 보안의 기초와 최신 보안 동향에 대한 정보를 교육받을 수 있는 기회를 제공한다. 소프트웨어 프로그램을 개발하는 인력은 최소한 매 년 한 번의 보안 교육에 참가해야 한다. 소프트웨어가 보안성을 위배하지 않고 안전하게 동작하고 하는 것을 보장하기 위해서는 보안 교육이 기초가 되기 때문이다. 보안 교육은 시큐어 설계, 위협모델링, 시큐어 코딩, 보안 테스팅, 프라이버시 등의 내용이 포함된다.

2.2 요구사항 단계

프로젝트 개시와 함께 신뢰성 있는 소프트웨어를 구축하기 위하여 기본 보안 요구사항과 프라이버시 요구사항을 정의한다. 또한 비용 분석을 통하여 보안 위협의 영역을 정의하고 테스트 계획을 세운다. 요구사항 단계에서 필수 보안활동으로는 SDL 방법론 적용 여부 결정, 보안책임자(Security Advisor) 선정, 보안팀(Security Champion) 선정, 버그 리포팅 도구 정의, 보안 버그 경계(security bug bar) 정의, 보안 위험 평가 등이 있으며, 권고 보안활동으로는 보안 계획서 작성, 버그 추적 시스템 정의가 있다.

2.3 설계 단계

SDL의 구현에서부터 배포에 이르는 동안 수행해야 하는 작업 계획을 수립하는 단계로, 필수 보안활동으로는 보안 설계 검토, 방화벽 정책 준수, 위협 모델링, 위협 모델 품질 보증, 위협모델 검토 및 승인 등이 포함되며, 권고 보안활동으로는 "보안 설계서" 문서 작성, 보안 디폴트 인스톨 실행, 모든 샘플소스코드의 보안검토 수행, 안전하지 않은 함수와 코딩 패턴 알림, 설계변화요구에 관한 보안관련 사항 문서화, 위협모델을 통해 찾아진 취약성을 위한 작업 목록 작성 등이 포함된다.

2.4 구현 단계

이 단계에서는 사용자가 소프트웨어를 안전하게 사용할 수 있도록 도와주는 사용자가 사용할 문서들과 도구들을 개발하게 된다. 구현 단계에서 보안 및 프라이버시 문제점을 발견하고 제거하기 위하여 개발 best practice를 수립하고 따르도록 하며, 필수 보안활동으로는 최신버전의 빌드 도구 사용, 금지된 API 사용 회피, 'Execute' 허가를 통한 SQL 안전하게 사용, 저장된 프로시저에서 SQL 사용 등이 포함된다. 또한 권고 보안활동으로는 안전하게 소프트웨어를 사용하기 위해 필요한 사용자 정보 식별, 사용자 중심의 보안 문서 계획, 보안 형상관리에 관한 정보 생성, 자동화된 금지 API 변환 실행, 프로젝트 팀 전체와 모든 모범사례와 정책에 대해 정의, 문서화, 토론 등이 포함된다.

2.5 검증 단계

검증 단계에서는 코드가 이전 단계에서 설정한 보안과 프라이버시를 지키는지 확인해야 한다. 이것은 보안 및 프라이버시 테스팅과 보안 푸쉬(security push), 문서 리뷰를 통하여 이루어진다. 보안 푸쉬는 팀 전체에 걸쳐 위협 모델 갱신, 코드 리뷰, 테스팅에 초점을 맞춘 작업이다. 공개 배포용 프라이버시 검사 역시 이 단계에서 완수된다. 검증 단계에서 필수 보안활동으로는 커널-모드 드라이버를 위한 테스팅 완료, COM 객체 테스팅 수행, 인증된 사이트 크로스 도메인 스크립팅을 위한 테스팅, 애플리케이션 검증 테스트 수행, 파일 fuzzing 수행, 위협모델 검토 및 수정 등이 포함되며, 권고 보안활동으로는 보안 테스팅 계획 완료, 침투 테스팅 수행, 시큐어 코드 검토, 보안 푸쉬를 시작하기 전 모든 코드에 대한 우선순위 결정, 보안 문서 계획서 검토 등이 포함된다.

3. CLASP 방법론

CLASP(Comprehensive, Lightweight Application Security Process)[5] 방법론은 구조적이고, 반복적이고, 예측가능한 방법으로, 기존에 우리가 가지고 있는 혹은 새로 시작하는 소프트웨어 개발 라이프사이클에서 보안을 구축하기 위한 정형화된 최적의 실행지침(best practices)을 포함하며, 핵심 활동기반(activity-driven)과 프로세스 컴포넌트의 역할기반으로 설정된다. Secure Software Inc 사에서 개발된 CLASP 방법론은 보안 제품 개발과 관련된 계획, 설계, 코딩 등을 포함하는 테크닉으로 단계별 도구와 방법이 결합된 형태로 제공되며, 특히 그림 2와 같은 5가지의 활동 관점을 제시함으로써 이들 관점 내에서의 활동을 정의하고, 각 관점별 활동간에 유기적인 결합 과정을 제공한다는 점에서 다른 개발 방법과 차별된다. 첫째, CLASP의 개념 관점에서는 각 개발 프로세스를 계획하고 평가하

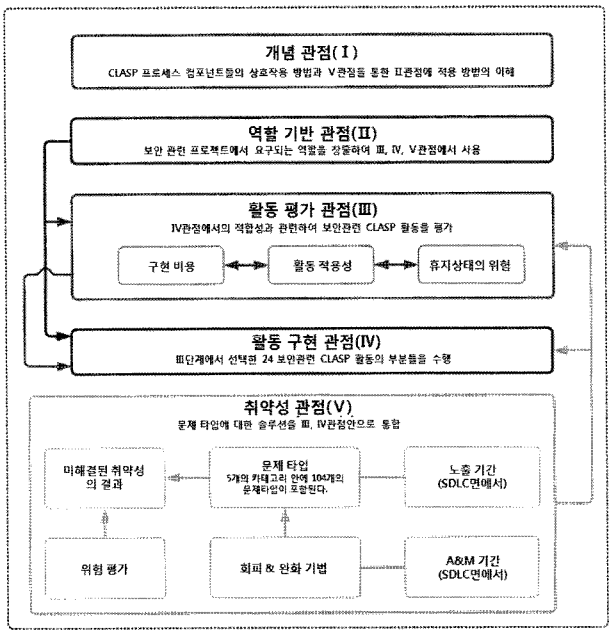


그림 2 CLASP의 5가지 관점

는 단계에서, CLASP 프로세스 컴포넌트들의 상호작용과 취약성 관점을 통해서, 어떻게 역할기반 관점에 적용하는지를 기술한다. 둘째, 역할기반 관점에서는 프로젝트팀의 각 구성원이 24개의 보안과 관련된 CLASP 활동들에 대해 각 역할을 정의한다. 셋째, 활동평가 관점에서는 CLASP의 보안관련 활동들에 대해서 타당성을 평가할 수 있도록 도와줌으로써, 프로젝트 매니저와 프로세스 엔지니어링 팀의 부담을 덜어주는 것이다. 넷째 활동구현 관점에서는 활동 평가단계에서 선택한 24가지 보안관련 CLASP의 활동들을 수행한다. 마지막으로 취약성 관점에서는 애플리케이션 소스코드에서 보안과 관련된 취약성을 5개의 카테고리(범위 및 타입 에러, 환경 문제, 동기화 및 타이밍 에러, 프로토콜 에러, 일반 논리 에러)로 분류하고 각 카테고리별로 해당하는 취약성에 대해서 취약성의 발견시기, 취약성의 결과, 위험평가, 공격을 위한 자원, 회피와 완화방법, 침투 가능성, 영향 받는 프로그래밍 언어와 운영체제 등을 핵심적으로 설명한다.

3.1 CLASP의 보안 취약성

CLASP는 어플리케이션에서 보안 취약성을 정의한다. 이러한 보안 취약성은 공격자가 사용자의 시스템 내에서 주어지지 않은 권한을 가정하고, 그것을 활용해서 사용자의 시스템 운영을 규제, 중요한 데이터를 위태롭게 만든다. 실제적으로 보안 취약성은 그것을 관리하는 보안 정책이 위반되었을 때, 소프트웨어 어플리케이션에서 발생한다. CLASP는 어플리케이션 소스코드에서 보안 취약점의 기본 형태로서, 기초적인 104개의 문제 타입을 식별한다. 개별적인 문제 타입은 그 자체가 취약성이 아니고, 소스코드에서 취약성을 발생시키는 보안 조건을 만드는 문제들의 조합이다. 또한, CLASP에서는 하나 또는 그 이상의 기본적인 보안 서비스 즉, 권한부여, 기밀성, 인증, 서비스 거부, 책임, 부인방지 등이 실패했을 때, 보안이 깨질 수 있는 취약성의 결과를 정의한다.

다음 그림 3은 SDLC에서 보안과 관련된 취약성이 발생할 수 있는 측면과 공격의 대상이 될 수 있는 운영 시스템의 포인트를 보여준다.

3.2 CLASP의 최적의 실행지침

보안 취약성들이 만약에 제품에 존재하는 어플리케이션 소스코드에 만들어진다면, 그것은 엄격한 비즈니스의 조직의 책임이 된다. 보안 취약성이 깨진 결과의 관점(view)에서, 어플리케이션 보안의 최적의 실행지침을 사용하는 것은 합리적인 대안이 아니다.

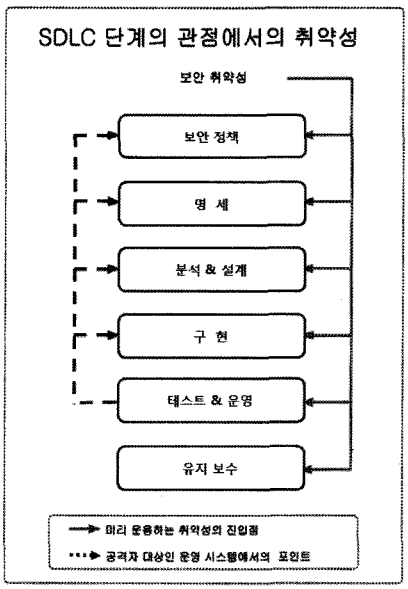


그림 3 SDLC 각 단계에서의 취약성

소프트웨어 어플리케이션 보안의 최적의 실행지침은 보안 취약성에 강한 소프트웨어 어플리케이션을 만들고, 배치하는 개발팀을 가이드하기 위해서, 반드시 신뢰할 수 있는 프로세스를 가져야만 한다. 다음은 7개의 CLASP의 최적의 실행지침이다.

- o 인식 프로그램 제정하기
- o 어플리케이션 평가 수행하기
- o 보안 요구사항을 획득하기
- o 안전한 개발 프랙티스 실행하기
- o 취약성 개선 절차를 구축하기
- o 측정기준들을 정의 및 모니터하기
- o 작업의 보안 가이드라인 출판하기

4. Seven-Touchpoint 방법론

Gary McGraw가 제안한 Seven-Touchpoint[6] 방법론은 실무적으로 검증된 방법론 중 하나로서 보안 강화 기법을 지원하는 중요한 기법 중 하나로 인정받고 있다. 이 방법론의 핵심은 개발 생명주기를 구성하는 단계의 활동 중 보안 기능과 직접 혹은 간접으로 관련된 활동이 존재한다는 발견으로부터 시작된다. 만일 보안 관련 활동들을 개별적으로 구분할 수 있고, 이들 활동들이 별도로 관리된다면 소프트웨어의 보안성은 그렇지 않은 경우보다 더욱 강화될 수 있다는 것이 McGraw의 발견이다. 이러한 개념에 그는 7개의 중점 관리 대상, 즉 터치 포인트를 제안하여 개발자에게 집중적인 관리를 하도록 요구한다. SDLC 내의 개발 단계와 이와 관련된 7개의 보안 강화 활동은 그림 4에 표현되어 있다.

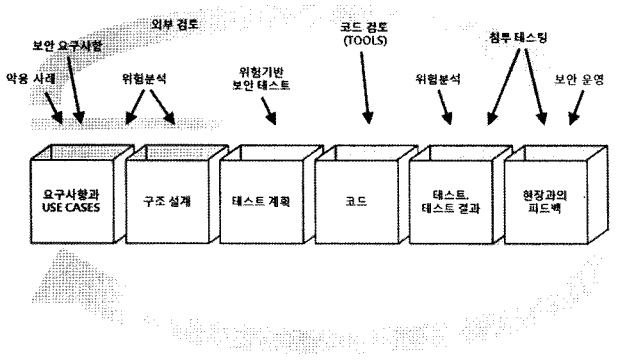


그림 4 Seven-Touchpoint 방법론

McGraw가 제안하는 7개의 터치 포인트는 다음과 같다.

- o 코드 검토(code review)
- o 아키텍처 위험 분석(architectural risk analysis)
- o 침투 테스트(penetration testing)
- o 위험기반 보안 테스트(risk-based security tests)
- o 악용 사례(abuse cases)
- o 보안 요구(security requirement)
- o 보안 운영(security operation)

SDLC 각 단계별로 집중 관리되어야하는 각각의 터치포인트 보안 활동은 다음과 같다.

4.1 요구사항과 유즈케이스 단계

이 단계에서의 보안강화 활동으로는 보안 요구, 악용 사례, 위험분석이 있다. 즉, 악용사례와 위험분석을 통해서 보안요구사항에 대한 정의와 명세를 하고, 악용 사례에 대한 정의 및 케이스 예를 작성한다.

4.2 구조설계 단계

공격저항 분석(attack resistance analysis), 모호성 분석, 허점 분석 등을 통해 위험요소를 분석한다. 첫째, 공격저항 분석에서는 체크리스트를 분석 방법으로서, 공격 패턴, 분석과정에서의 취약성, 이미 알려진 공격에 대한 정보를 사용하여 아키텍처의 위험 분석요소를 설계한다. 둘째, 모호성 분석은 새로운 위험을 발견하기위해 요구되는 새로운 활동을 캡처하는 하위 프로세스이며, 최소한 적어도 두 번의 분석과 어느 정도의 경험이 요구된다. 또한 각 팀 구성원을 위해서 병렬로 개별적인 분석활동을 수행한다. 이러한 개별적인 분석활동 후에 이해하는 것을 통합하는 단계에서 동시에 각 팀의 활동이 완성된다. 셋째, 약점 분석은 외부 소프트웨어 종속성에 미치는 영향을 이해하기위한 프로세스다. 현재 소프트웨어는 일반적으로 복잡한 미들웨어 프레임워크나 .NET, J2EE 등에 생성되며, 거의 모든 코드는 DLL같은 외부 라이브러리 또는 glibc 같은 공통의 언어 라이브러리에 포함된다. 또한 분산된 코드는 아키텍처 예외 기준이 된다.

4.3 테스트 계획 단계

이 단계에서는 공격 패턴, 위험 분석 결과, 악용 사례를 기반으로 위험기반 보안 테스트를 수행한다. 위험기반 보안 시험은 공격자 입장에서 생각하는 것이 필수적이다. 보안 테스트를 할 경우 소프트웨어 아키텍처, 공통 공격, 공격자의 심리 상황을 고려하는 것은 필수적이다.

4.4 코드 단계

이 단계에서는 구현 오류(implementation bug)에 중점을 두며 특히 소스코드에 존재하는 취약성을 발견할 목적으로 수행되는 코드 정적 분석에 중심을 둔다. 정적 분석에 사용되는 소스코드 취약성의 분류 구조는 CVE(Common Vulnerability Enumeration), CLASP 뿐만 아니라 여러 도구 벤더들에 의해서도 제공되고 있다.

코드 검토는 취약성 확인을 위해 필요 사항이기는 하지만 충분 사항은 아니다. 다시 말해서 코드 검토를 통해 많은 오류들(예를 들면 C, C++에서 발견되는 버그와 같은 오류들)을 발견할 수는 있지만 아키텍처에 관한 오류를 발견하기 위해서는 다른 방식의 접근이 필요로 한다. 따라서 집중적인 접근을 위해서는 코드 분석과 아키텍처 분석이 총체적으로 수행되어야 한다.

4.5 테스트, 테스트 결과 단계

이 단계에서는 위험 분석 및 침투 테스팅을 수행한다. 침투 시험은 미리 설정된 환경에서 계획된 블랙박스 테스트로 수행 된다. 따라서 침투 테스팅에 실패한 소프트웨어이거나 부실하게 구성된 침투 테스트를 통과한 소프트웨어는 심각한 위험에 처할 수 있다. 침투 테스트의 장점은 실제 작동 환경에서의 필드 소프트웨어에 대한 좋은 이해를 제공해준다는 점이다.

4.6 현장과의 피드백 단계

소프트웨어 보안은 네트워크 보안 효과에 큰 혜택을 받는다. 잘 통합된 보안 운영은 네트워크 보안 관리자로 하여금 소프트웨어 보안 운영에 적극적으로 참여하여 세븐 터치포인트를 잘 활용하도록 하며 개발자들이 간과한 보안 운영의 경험들을 개발자들과 공유할 수 있게 한다. 아무리 정교하게 소프트웨어를 구축하였다 하더라도 공격은 항상 발생한다. 따라서

시스템의 행위를 이해함으로써 방어에 대한 좀 더 많은 지식을 쌓을 수 있다. 보안 운영을 통해 얻은 공격자와 공격 도구에 대한 경험과 지식은 개발자에게 다시 피드백 되어야 한다.

5. TSP-Secure 방법론

1996년, Watts Humphrey는 TSP(Team Software Process)[7-9] 프로세스의 최초 버전을 개발하였다. 그의 목적은 엔지니어들이 일관되게 효율적인 작업을 할 수 있도록 돕는 영업 프로세스를 제공하는 것이었다. 이후 Humphrey는 9개의 TSP 버전을 더 개발하였다. 이 중 TSP-Secure 방법론은 예측 가능하게 개발된 소프트웨어의 보안을 개선하기 위한 방법으로, 소프트웨어 애플리케이션의 보안에 좀 더 직접적인 초점을 맞추도록 TSP를 확장하였으며, 기존의 개발절차로 만든 소프트웨어보다 더 안전한 소프트웨어 개발을 돕기 위해 설계되었다.

5.1 TSP 팀워크 프로세스

대부분의 엔지니어링 프로젝트에 팀이 필요하다. 비록 작은 하드웨어나 소프트웨어는 개인에 의해 개발될 수 있지만, 현대의 시스템의 복잡성과 크기, 짧은 스케줄의 요구로 인하여 한 사람이 대부분의 엔지니어링 작업을 하는 것은 더 이상 실용적이지 않다. 시스템 개발은 팀 활동이며 팀의 효율성이 엔지니어링의 질을 결정한다. 팀워크는 연습이 필요하고 특별한 기술을 필요로 한다. 팀은 공통적인 과정이 필요하다; 그들은 동의에 의한 목표가 필요하다. 그리고 그들은 효과적인 지도와 리더십이 필요하다.

5.2 TSP 프로세스

TSP 프로세스의 주요한 요소들은 그림 5에서 볼 수 있다.

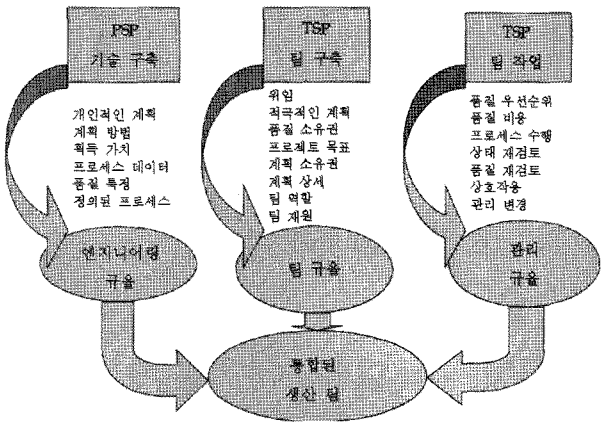


그림 5 TSP-Secure 주요요소 및 팀 구축

구성원들이 TSP 팀에 참여하기 전, 그들은 어떻게 훈련된 작업을 할 수 있는지 알아야만 한다. 개인 소프트웨어 프로세스(PSP : Personal Software Process)의 훈련은 TSP를 사용하기 위한 지식과 기술을 제공하기 위해 요구된다. PSP 훈련은 세부적인 계획을 어떻게 만드는지, 프로세스 데이터를 모으고 사용하는 것, 가치를 얻는 계획을 만들어내는 것, 얻어진 가치를 프로젝트 추적에 사용하는 것, 제품의 품질을 측정하고 관리하는 것, 그리고 운영 프로세스를 선별하고 사용하는 것의 학습을 포함한다. 엔지니어들은 TSP 팀 구축에 참여하거나 정의된 TSP 프로세스를 따라가기 전 이러한 기술들을 훈련받아야 한다. 출시에서, 모든 팀 구성원들은 프로젝트를 수행하기 위해 계략, 프로세스, 그리고 계획을 개발한다. 출시를 완료한 후에, 팀은 그들의 작업을 위해 정의된 프로세스를 따른다.

5.3 시큐어 시스템 개발을 위한 TSP-Secure

시큐어 시스템 개발을 위한 TSP-Secure는 시큐어 시스템 영역에 대하여 TSP를 강화한 응용연구이다. TSP 프로세스에 시큐어 어플리케이션을 위한 설계 원칙인 보안 설계 프로세스, 보안 구현 프로세스, 보안 검토 및 검사 프로세스, 보안 테스트 프로세스, 보안 관련 대응방안 등을 통합하여 TSP-Secure 프로세스를 개발하고 있다. TSP-Secure는 아래와 같은 세가지 목적을 갖는다.

- o 안전한 시스템 개발 관습을 지원한다.
- o 잠재적인 보안결함의 가능성을 예측한다.
- o 새로운 위협에 능동적으로 대응한다.

시큐어 시스템 개발을 위한 TSP-Secure는 파일럿 프로젝트로 진행중이며, 현재산업계와 정부간 긴밀한 협력아래 연구되고 있다.

6. 결론 및 요약

최근 인터넷에 연결된 소프트웨어에 대한 의존이 심화되면서 해킹, 바이러스, 프라이버시 침해, 사이버 테러등과 같은 보안문제가 날로 심각해지고 있다. 따라서 소프트웨어의 안전성뿐만 아니라 소프트웨어 내에 존재하는 약점들로부터 안전하게 자산을 보호하고 결함이 없는 시큐어 소프트웨어를 개발하는 일이 시급하다. 시큐어 소프트웨어 개발을 위해서 기존의 소프트웨어 개발생명주기의 각 개발 단계별로 보안 요소를 강화시키는 활동들을 고려함으로써, 소프트웨어 전체의 보안성을 강화하는 체계화된 개발방법론이

요구된다. 본 연구에서는 이러한 시큐어 소프트웨어 개발을 위한 소프트웨어 개발방법론의 최근 동향을 소개하였다. 시큐어 소프트웨어 개발을 위해서는 이러한 개발방법론의 체계화된 정립뿐만 아니라, 소프트웨어를 만드는 개발자와 관리자들의 의지 및 인식이 우선 요구되어져야한다.

참고문헌

[1] Karen Mercedes Goertze, Theodore Winograd, et al (DACS). for Department of Homeland Security and Department of Defense Data and Analysis Center for Software. Enhancing the Development Life Cycle to Produce Secure Software: A Reference Guidebook on Software Assurance, October 2008.

[2] Karen Mercedes Goertzel, Theodore Winograd, et al(IATAC, DACS). Software Security Assurance Software Security Assurance State-of-the-Art Report (SOAR), July 2007.

[3] Michael Howard and Steve Lipner, The Security Development Lifecycle, P.352, Microsoft, 2006.

[4] Microsoft Security Development Lifecycle(SDL), http://msdn.microsoft.com/en-us/security/cc448177.aspx.

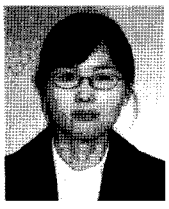
[5] CLASP(Comprehensive Lightweight Application Security Process), http://searchsoftwarequality.techtarget.com/searchAppSecurity/downloads/clasp_v20.pdf.

[6] Gary McGraw, Software Security: Building Security In, P.448, Addison-Wesley Professional, 2006.

[7] Humphrey, W. S. Managing the Software Process. Reading, MA: Addison-Wesley, 1989.

[8] Humphrey, W. S. A Discipline for Software Engineering. Reading, MA: Addison-Wesley, 1995.

[9] Humphrey, W. S. Introduction to the Personal Software Process. Reading, MA: Addison-Wesley, 1997.



이 송 희

1998 순천향대학교 컴퓨터공학과 학사
2002 순천향대학교 전자상거래학과 석사
2009 고려대학교 컴퓨터학과 박사
현재 고려대학교 컴퓨터 · 정보통신연구소 연구교수
관심분야 : 정형기법, 네트워크보안, 정보보호기술, 소프트웨어 공학
E-mail : shlee@formal.korea.ac.kr

최 진 영

서울대학교 컴퓨터 공학과 학사. Dept. of Mathematics and Computer Science, Drexel Univ. 석사. Dept. of Computer and Information Science, University of Pennsylvania 박사
1996~현재 고려대학교 컴퓨터·전파통신공학부 교수
관심분야 : 정형기법, 임베디드 실시간 시스템, 프로그래밍 언어, 프로세스 대수, 소프트웨어 공학
E-mail : choi@formal.korea.ac.kr

강 인 혜

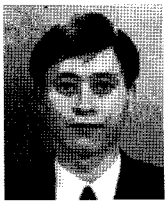
1987 서울대학교 전자계산기공학과 학사
1989 서울대학교 전자계산기공학과 석사
1997 University of Pennsylvania 컴퓨터학과 박사
1998~2000 숭실대학교 정보과학대학 객원교수
2000~2002 (주)시큐아이닷컴 부장
2003~현재 서울시립대학교 기계정보공학과 부교수
관심분야 : 정형기법, 소프트웨어공학, 정보보호
E-mail : inhye@uos.ac.kr

서 동 수

1989 중앙대학교 컴퓨터공학과 학사, 석사
1994 Univ. of Manchester, Dept. of Computation 석사, 박사
1994~1998 전자통신연구원, 선임연구원
1998~현재 성신여자대학교 IT학부 교수
관심분야 : 소프트웨어공학, 정보보호기술, 정형기법
E-mail : dseo@sungshin.ac.kr

안 재 영

1998 중앙대학교 전자공학과 학사
1998~1999 삼성전자 연구원
2002 중앙대학교 전자공학과 인터넷보안 전공 석사
2001~현재 한국인터넷진흥원 선임연구원
관심분야 : 소프트웨어 보안개발방법론
E-mail : jyahn@kisa.or.kr



한 근 희

1986 서울산업대학교 컴퓨터학과 학사
1988 한양대학교 컴퓨터학과 석사
2006 고려대학교 컴퓨터학과 박사
2006~현재 행정안전부 정보보호정책과 근무
2002~현재 건국대학교 정보통신대학원 겸임교수
관심분야 : 인터넷 보안, 통합보안관리, 모바일 보안, 차세대 인터넷 등
E-mail : hankeunhee@nate.com, khhan@korea.kr