안전한 소프트웨어 개발·도입을 위한 보안 가이드 V1.0

2008. 12

한국정보보호진흥원
Korea Information Security Agency

주 의 사 항

이 가이드 사용에는 어떠한 제한도 없지만 다음과 같은 사항에 주의하여야 합니다.

○ 문서 내에 언급된 상표, 제품명 등에 대한 권리는 각 상표 또는 제품을 소유한 해당 기업에 있으며, 설명을 위해 특정 회사 제품명이나 화면이 표시된 경우 가이드의 고유 목적 외에 어떠한 다른 목적도 없으며 그렇게 이용되어서도 안 됩니다.

○ 문서 내에 기술된 템플릿, 예시 등은 각 사용자, 기업 등에 있을 수 있는 고유한 환경을 모두 고려하지 않았으므로 실제 환경에서는 그대로 적용되지 않을 수 있습니다. 그러므로 이용자는 본 가이드에서 주어진 기술 세부사항을 활용하여 자신의 환경에 맞게 조정하는 작업이 필요하며, 세부사항 내용의 오류로 인해 발생하는 피해에 대하여 본 가이드의 발행기관은 책임을 지지 않습니다.

목 차

제 1 장 서 론 ····· 1

제 1 절 배 경 ····· 1

제 2 절 권고(안) 목적 및 적용범위 ····· 3

제 3 절 문서의 구성 ····· 4

제 4 절 용어 정의 ····· 5

제 2 장 국내·외 소프트웨어 보안 기술 동향 ····· 7

제 1 절 소프트웨어 개발 방법론 ····· 7

제 2 절 보안이 강화된 소프트웨어 개발 방법론 ····· 19

제 3 절 소프트웨어 취약점 시험 기술 ····· 29

제 3 장 소프트웨어 개발과정의 정보보호 ····· 43

제 1 절 일반적 보안 원칙 권고(안) ····· 43

제 2 절 소프트웨어 개발단계별 보안절차 ····· 46

제 4 장 소프트웨어 도입과정의 정보보호 ····· 87

제 1 절 소프트웨어 도입 단계에서 보안 ····· 87

부 록 ····· 99

1. 소프트웨어의 취약점 점검 및 오류 검증 도구 ····· 99

2. 안전한 소프트웨어 개발을 위한 자료 목록 ····· 110

<표 차례>

[표 2-1] SDLC 모형의 분류 ····· 10
[표 3-1] 보안요소별 보안등급 구분 (예시) ····· 49
[표 3-2] 보안정책 검토 형식 (예시) ····· 50
[표 3-3] 보안계획 수립시 주요 검토 사항 (예시) ····· 50
[표 3-4] 보안계획서 양식 (예시) ····· 51
[표 3-5] 보안 계획 검토 결과서 (예시) ····· 53
[표 3-6] 시작단계 전체 보안활동에 대한 점검 목록 (예시) ····· 53
[표 3-7] 기술적인 측면의 보안 요구사항 (예시) ····· 56
[표 3-8] 관리적인 측면의 보안 요구사항 (예시) ····· 56
[표 3-9] 분석단계 전체 보안활동에 대한 점검 목록 (예시) ····· 58
[표 3-10] DMZ 방화벽의 네트워크 접근통제 구성 (예시) ····· 63
[표 3-11] 개발 관련 주요 서버보안 항목 (예시) ····· 64
[표 3-12] 개발 관련 서버의 네트워크 서비스 관리표 (예시) ····· 65
[표 3-13] 개발 관련 서버의 접근 계정 관리표 (예시) ····· 65
[표 3-14] 개발 관련 주요 물리적 보안 항목 (예시) ····· 66
[표 3-15] 개발 관련 주요 응용 프로그램통제 보안 항목 (예시) ·· 67
[표 3-16] 애플리케이션 보안 설계시 반영 항목 (예시) ····· 69
[표 3-17] 설계단계 전체 보안활동에 대한 점검목록 (예시) ····· 72
[표 3-18] 개발환경 보안(네트워크) 구현사항 점검목록 (예시) ····· 81
[표 3-19] 개발환경 보안(서버) 구현사항 점검목록 (예시) ····· 82
[표 3-20] 애플리케이션 보안 설계 구현사항 점검목록 (예시) ····· 82
[표 4-1] 소프트웨어 도입 결정시 포함되어야 할 항목 ····· 90
[표 4-2] 소프트웨어 도입 평가 기준 ····· 92

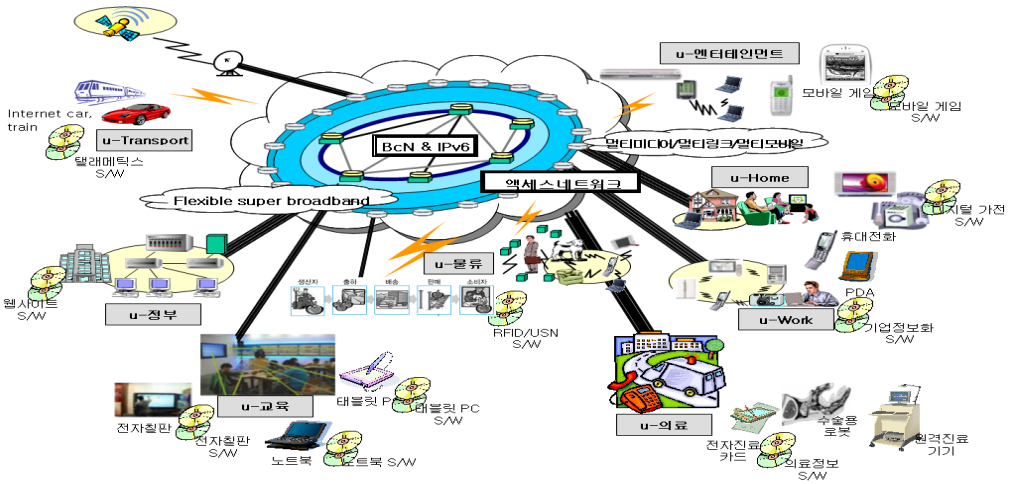
<그림 차례>

(그림 1-1) 미래 정보이용 환경 변화에 따른 소프트웨어 이용 범위의 확대 ···· 1
(그림 1-2) 소프트웨어 단계별 보안 탐지 비용 비율 비교 ····· 2
(그림 2-1) 일반적인 SDLC ····· 8
(그림 2-2) 10개의 PSP 프로그램 과제 ····· 12
(그림 2-3) TSP의 착수식 프로세스 ····· 13
(그림 2-4) Cleanroom 프로세스 모형 ····· 15
(그림 2-5) Correction by Construction 프로세스 ····· 17
(그림 2-6) CLASP의 뷰와 상호작용 ····· 21
(그림 2-7) SDLC 단계의 취약점 ····· 22
(그림 2-8) CLASP 요소의 적용 ····· 23
(그림 2-9) SDL을 통한 소프트웨어 개발 프로세스의 개선 ····· 25
(그림 2-10) 소프트웨어 생명주기에서의 취약점 시험 적용시기 ·· 31
(그림 3-1) 주요 보안 활동이 포함된 애플리케이션 SDLC ····· 46
(그림 3-2) 시작단계의 보안 프로세스 ····· 48
(그림 3-3) 분석단계의 보안 프로세스 ····· 55
(그림 3-4) 설계 단계의 보안 프로세스 ····· 61
(그림 3-5) 네트워크 구성도 (예시) ····· 62
(그림 3-6) 구현단계의 보안 프로세스 ····· 73
(그림 3-7) 시험단계의 보안 프로세스 ····· 85
(그림 4-1) 소프트웨어 도입 단계별 보안활동 ····· 88

제 1 장 서 론

제 1 절 배 경

IT제품의 지능화, 그린화, 무선화에 힘입어 소프트웨어의 중요성이 한층 증가함에 따라 소프트웨어의 보안 취약점을 이용한 공격위협 또한 커지고 있다. 네트워크 인프라가 전국적으로 구축되면서 발달한 온라인 환경은 웹 소프트웨어의 활성화를 가져왔지만, 본질적으로 개방된 웹상에서는 소프트웨어의 취약점을 이용한 보안 사고의 위험성이 더욱 높다. 또한 휴대폰 등의 모바일 단말기에 제공되는 임베디드 소프트웨어의 경우는 제품 출시 후 발견된 취약점에 대한 보완이 어려워, 제거되지 않은 보안 취약점이 지속적으로 발생할 수 있다.

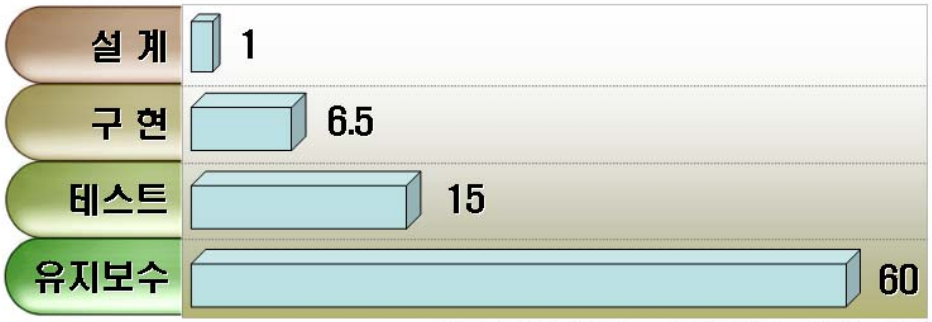


(그림 1-1) 미래 정보이용 환경 변화에 따른 소프트웨어 이용 범위의 확대

CNET 보고에 따르면 2004년 휴대폰에서 바이러스가 발견된 이후 전 세계적으로 200여종의 휴대폰 바이러스가 발견되었으며, 휴대용 게임기를 통한 PC의 원격 조종이 가능해지는 등 융·복합 단말기에 내재된 소프트웨어

취약점 문제가 급증하고 있는 추세이다. 더 나아가 유비쿼터스 사회에서 통신, 자동차, 의료 등 일생상활에서 소프트웨어의 이용범위가 확대됨에 따라 소프트웨어 보안문제는 개인정보 유출 및 산업자원 손해 등 경제적, 사회적 손실을 초래할 수 있는 가능성이 더욱 높아지고 있다.

기존의 소프트웨어 개발 기업들은 촉박한 개발 기간과 부족한 예산, 보안 인식 부족으로 인해 개발 단계에서의 소프트웨어 보안 문제를 등한시 해왔다. 미국 CMU 소프트웨어 공학연구소는 보안 취약점의 70%가 설계과정의 오류로부터 발생한다고 보고하였으나, 이러한 설계과정의 오류는 대부분 제품 완성 후 베타테스트나 보안 업데이트 및 패치로 보완하고 있는 추세이다. KrCERT의 보고서에서 '05년 대비 '06년 응용 프로그램 보안 업데이트는 약 41%, 보안 취약점은 약 79% 증가하였음을 볼 때, 소프트웨어 개발 이후 취약점을 보완하는 보안 업데이트는 계속적으로 증가하는 보안 취약점을 완벽하게 해결하기에는 시간과 비용문제에서 많은 어려움이 있다.



(그림 1-2) 소프트웨어 단계별 보안 탐지 비용 비율 비교

보다 근본적인 해결 방안으로 소프트웨어 개발단계에서 보안을 적용하여 취약점을 제거하는 방식이 제기되고 있다. 미국 Microsoft의 사례를 살펴보면 개발단계에서 SDL(Security Development Lifecycle) 적용 시 보안 취약성이 50 ~ 60% 정도 감소한다고 보고되고 있다. 또한 소프트웨어 개발단계에서

취약점 제거 시 유지보수 단계의 비용보다 60배 절감된다는 연구결과를 볼 때 소프트웨어 개발 초기부터 보안을 강화시키는 방안은 비용과 취약점 감소 측면에서 많은 이점을 가지고 있다. 또한 소프트웨어 개발과정에서 Secure 소프트웨어 엔지니어링을 적용할 경우 ROI가 21% 이상 향상된다는 것이 전문가들의 지적이다.

소프트웨어 배포 단계에서의 보안성 확보 미흡도 문제이다. 소프트웨어 품질인증, 표준적합성 시험, 보안 IT제품 보안성 평가에 대한 규제만 마련되어 있을 뿐, 일반 소프트웨어 제품에 대한 보안성 검증 규제는 존재하지 않는다. 특히 누구라도 제작하여 배포가 가능한 공개 소프트웨어에 대한 안전성 검증 미흡은 국내외 공개 소프트웨어 이용 및 도입 활성화 지연 요인으로 작용하고 있다. 소프트웨어 출시 후에도 소프트웨어 성능 및 기능에 편향되어 품질 향상이 진행되기 때문에 실질적으로 소프트웨어에 내재된 취약점은 현실적으로 제거되기 어려운 환경에 있다.

따라서 개발자 및 관리자에게 개발과정에서 취약점을 제거하고 소프트웨어 도입 시에 보안취약점들을 점검하여 조치 할 수 있는 방법론을 제공하고 필요성을 인식할 수 있도록 이러한 지침의 개발이 필요한 시점이라 하겠다.

제 2 절 권고(안) 목적 및 적용범위

1. 권고(안) 목적

본 권고(안)은 소프트웨어 개발 및 이용에 관계된 사람들에게 안전한 소프트웨어의 개발 및 운영을 위한 상위수준의 방법론 및 지침을 제공한다. 소프트웨어 개발단계부터 도입단계까지 소프트웨어 보안 취약점 감소를 위한 Secure 소프트웨어 엔지니어링 기법을 중심으로 개발공정별 보안 방법론 및 보안요건, 점검도구 등을 제공하며, 소프트웨어 도입단계에서의 보안 점검 항목 등을 규정하였다.

본 권고(안)은 소프트웨어 개발과정 및 도입단계에서 취약점 제거에 필요한 상위수준의 절차를 서술하여 관계자들이 주지하여 소프트웨어를 개발할 수 있도록 최소한의 필요성을 인식하고 본 권고(안)을 활용하여 각자의 개발환경에 맞는 독립적인 개발방법론을 작성할 수 있도록 도움을 주기위한 목적으로 개발되었다. 소프트웨어 개발 성격에 부합하는 좀더 구체적이며 세부적인 점검항목 및 절차는 이후 추가적으로 개발이 진행될 예정이다.

2. 적용 범위

소프트웨어 개발 및 도입에 대한 별도의 지침이나 보안 엔지니어링 방법론이 마련되어 있지 않은 개발업체 및 도입에서 이 권고(안)을 적절하게 활용할 수 있으며 주 대상 독자는 다음과 같다.

- 소프트웨어를 실제 구현하는 부서의 개발자 및 관리자
- 소프트웨어 개발 계획 및 설계를 담당하는 소프트웨어 엔지니어 및 설계자
- 소프트웨어 환경의 보안 감리와 평가를 수행하는 정보보호 컨설턴트
- 소프트웨어 도입과 운영, 유지보수를 담당하는 시스템/네트워크 관리자
- 소프트웨어 도입하는 이용기관

제 3 절 문서의 구성

본 권고(안)은 총 4장과 부록으로 구성되어 있다. 각 장의 구성과 내용을 살펴보면 다음과 같다.

제1장에서는 소프트웨어 개발·도입 보안권고(안)의 배경, 목적 및 적용 범위, 문서의 구성, 용어정의에 대해 설명한다. 제2장에서는 국내·외 소프트웨어 정보보호 동향을 중심으로 일반적인 소프트웨어 개발 공정 및 방법론과

Secure SDLC 개발 동향에 대해 분석한다. 제3장에서는 소프트웨어 개발 과정에서 개발자가 지켜야할 개발 보안 방법론, 체크리스트 등을 제시하여 소프트웨어 개발 생명주기(SDLC)에 걸쳐 취약점을 감소시키기 위한 방안을 제시한다. 제4장에서는 소프트웨어를 사용하고자 하는 기관에서 소프트웨어 도입시 보안 측면의 점검사항을 제시하여 개발 이후 발생할 수 있는 보안 취약점을 최소화할 수 있도록 안내한다.

부록에서는 소프트웨어 개발 과정에서 실무자들이 실제 업무에 활용하기 쉽도록 다음 내용을 첨부하였다.

[부록 1]에서는 소스코드 보안 취약성 점검도구. 소스코드 오류 검사 도구, 실행코드 보안 취약점 점검 및 방지 도구, 소프트웨어 모델 체킹 도구들을 목록화하여 제공한다. [부록 2]에서는 안전한 소프트웨어 개발을 위한 국외가이드 및 보고서, 관련도서와 관련 홈페이지를 제공한다.

본 권고(안)에서는 개발/운영되는 소프트웨어가 네트워크 및 인터넷에 항상 연결되어 있다고 가정한다.

제 4 절 용어 정의

본 권고(안)에서 사용되는 용어의 정의는 다음과 같다.

1. “소프트웨어 개발 생명주기(Software Development Life Cycle : SDLC)” 이라 함은 주어진 예산과 자원으로 개발 방법, 개발 환경 그리고 개발 관리에 대한 포괄적인 접근 방법을 설정하여 보다 높은 품질의 소프트웨어를 만들어 내기 위한 개발 프로세스를 말한다.

2. “안전한 소프트웨어 개발 생명주기(Secure SDLC)“이라 함은 안전한

소프트웨어를 개발하기 위해 SDLC 상에서 보안을 강화한 개발 프로세스를 말한다.

3. “소프트웨어 취약점 점검 및 오류 검증 도구” 이라 함은 소프트웨어 컴포넌트가 지니고 있는 보안상의 문제점 및 소스 코드 오류를 미리 점검하여 발생할 수 있는 보안 사고를 사전에 예방할 수 있는 도구를 말한다.

4. “소프트웨어 취약점 시험 기술 “이라 함은 소프트웨어 시스템 및 소프트웨어 시스템의 컴포넌트들이 상호 작동하는 방식을 시험하는 기술을 말한다.

5. “소프트웨어 개발 단계별 Template“은 안전한 소프트웨어 개발 공정을 위해 단계별로 작성해야할 문서의 양식을 말한다.

6. “보안계획서” 라 함은 소프트웨어 개발의 시작단계에서 보안 담당자가 안전한 소프트웨어 개발을 위해 보안목적, 보안계획 등을 정의한 문서를 말한다.

7. “보안 요구사항 분석서” 라 함은 보안 담당자가 소프트웨어 개발의 분석 단계에서 해당 소프트웨어에서 요구되는 기술적, 관리적 보안 요구사항을 세부항목별로 분석한 문서를 말한다.

8. “보안 구현 설계서” 라 함은 보안 요구사항이 반영된 소프트웨어 설계문서로 해당 소프트웨어의 기능명세서, 기본설계서, 상세설계서를 통칭한다.

제 2 장 국내·외 소프트웨어 보안 기술 동향

제 1 절 소프트웨어 개발 방법론

소프트웨어를 개발함에 있어 개발 공정 단계에서 불안전하게 개발된 소프트웨어는 향후 프로그램 업데이트나 유지 보수를 수반해야 한다. 이러한 비합리적인 소프트웨어 개발로 인한 비용 부담이나 업무 손실은 점점 증가하고 있으며 소비자의 신뢰를 떨어뜨리는 중요한 요인이 된다. 특히, 보안 문제를 간과한 소프트웨어 개발은 추후 보안 사고로 이어지게 되고 그로 인한 피해 규모는 예측하기도 힘들다. 그러나 아직 상당수의 프로그램 개발자들이 적절한 보안 점검이나 개발 절차를 지키지 않는 경우가 많다.

본 절에서는 소프트웨어 개발자가 오류를 최소화 하여 안전한 프로그램을 개발하면서 프로젝트 업무를 효과적으로 수행하기 위한 소프트웨어 개발 방법을 알아보고자 한다.

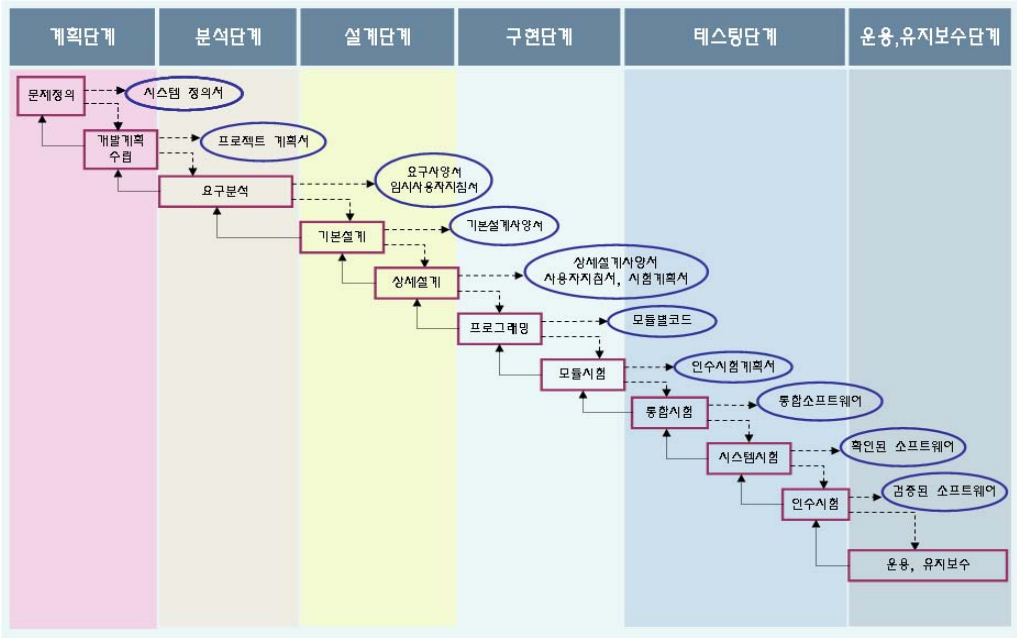
1. 일반적인 SDLC

SDLC(Software Development Life Cycle)란 소프트웨어 개발 생명 주기라고 하며, 소프트웨어를 개발하기 위한 계획부터 구현 및 사용이 끝나서 완전히 폐기될 때까지의 전과정을 단계적으로 분류하여 정의한 것을 말한다. SDLC는 주어진 예산과 자원으로 개발 방법, 개발 환경 그리고 개발 관리에 대한 포괄적인 접근 방법을 설정하여 보다 높은 품질의 소프트웨어를 만들어 내기 위한 개발 프로세스이다. SDLC에서는 개발의 시작과 끝을 중요시 하는데 그 중심은 소프트웨어 개발 과정이 아니라 사용자의 요구를 전략적으로 계획하기 위한 준비 단계와 사용자가 시스템을 사용하고 이익을 얻을 수 있는 운용 단계를 중시한다.

SDLC는 문제의 유형이나 개발 방침 등에 따라 다양한 패러다임이 존재하는데 폭포수 모델, 원형 모델, 나선형 모델, 반복적 개발 모델 등이 있다. 이러한 패러다임은 나름대로의 장·단점을 가지고 있다. 현재 가장 널리 사용되고 있는 폭포수 모델은 각 단계별로 산출물을 통하여 그 과정을 검증하고 이를 기초로 다음 단계의 작업을 수행하는 방법으로 표준화된 양식과 문서를 통하여 프로젝트의 관리적 측면을 강조한 모델이다.

폭포수 모델은 소프트웨어 개발 시 프로세스에 체계적인 원리를 도입한 방법론으로서 기존에 존재하는 시스템을 보완하는데 효과적이다. 또한, 이 모델이 적용된 사례가 많고 가장 오래되고 널리 사용되는 방법으로서 단계별 정의와 산출물이 명확하다는 특징이 있다.

폭포수 모델은 소프트웨어의 개발 단계를 계획 단계, 분석 단계, 설계 단계, 구현 단계, 테스팅 단계, 운용 및 유지보수 단계로 나누고 있다. 폭포수 모델의 절차는 (그림 2-1)과 같다.



(그림 2-1) 일반적인 SDLC

가. 계획 단계

사용자는 소프트웨어의 필요성을 파악하고 이를 개발하기 위한 타당성을 먼저 검토한다. 소프트웨어 개발이 타당하다면 사용자는 이를 제안 요청서에 의해 개발자에게 요청하게 된다. 개발자는 주어진 문제의 실현 가능성을 타진하고, 시스템의 성격을 파악하여 소요되는 비용과 기간을 예측한다. 사용자가 제시한 문제에 대한 정의와 해결 방안 및 이익 분석, 그리고 방법별로 필요한 비용, 자원, 기간 등을 체계적으로 분석한다.

나. 요구 분석 단계

사용자 요구나 주어진 문제를 정확히 분석하여 이해하는 과정으로 구현될 시스템의 기능이나 목표, 제약사항 등이 정확히 파악되어야 한다. 요구 분석의 목적은 최종 시스템의 기능, 성능, 사용의 용이성, 이식성 등을 파악하는 것이다. 또한, 요구 분석 결과인 요구 분석서는 사용자가 요구한 요소가 잘 반영되었는지 확인하기 위한 목적도 있다. 따라서 요구 분석서는 사용자와 개발자의 의사소통 수단으로 사용될 수 있으므로 정확하고 간결한 것이 좋으며 상호 이해하기 쉽고 개발 목표의 일관성을 유지하도록 작성되어야 한다.

다. 설계 단계

설계에는 시스템 구조 설계, 프로그램 설계 그리고 인터페이스 설계 등이 있다. 시스템 구조 설계는 시스템을 이루는 각 모듈과의 관계와 전체적인 구조를 설계하는 것이고 프로그램 설계는 각 모듈 안에서의 처리 절차나 알고리즘을 설계하는 것을 말한다. 또한, 사용자 인터페이스 설계는 시스템 사용자에게 보이거나 통신하기 위한 부분을 설계하는 것이다. 이러한 설계 단계의 결과물은 설계 명세서가 되며 여기에는 소프트웨어 전반적인 구조, 각 모듈의 기능 그리고 모듈 사이의 관계를 나타낸다.

라. 구현 단계

프로그래밍을 하는 단계로 각 모듈의 코딩과 디버깅이 이루어지고 그 결과를 검증하는 단위(모듈) 시험을 실시한다.

마. 시험 단계

개발된 각 모듈들을 통합시키며 시험하는 통합 시험(integration test), 사용자의 요구사항을 만족하는지 알아보는 시스템 시험(system test). 그리고 사용자가 직접 자신의 사용 현장에서 시스템을 검증해 보는 인수 시험(acceptance test) 등이 있다.

바. 운용(operation) 및 유지보수(maintenance) 단계

사용자가 개발된 소프트웨어를 인수하여 이용하면서 문제점을 발견하였을 경우 이를 수정하거나 새로운 기능을 추가해 보다 유용한 소프트웨어로 발전시키는 단계이다. 유지보수는 잘못된 것을 수정하는 수정 유지보수, 시스템을 새 환경에 적응시키는 적응 유지보수, 새로운 기능을 추가하는 추가 유지보수, 미래의 시스템 관리를 위한 관리 유지보수 등이 있다.

SDLC 모델은 다양한 방법으로 개선되었으며 다음과 같이 분류할 수 있다.

[표 2-1] SDLC 모형의 분류

구분	주요 내용 및 특징	장·단점
폭포수 모델	- 각 단계별로 개발 완료 후 다음 단계를 수행 - 각 단계의 정확한 수행과 검증	- 고전적인 소프트웨어 개발 모델로 많은 검증 과정 - 요구분석 시간이 많이 걸리고 개발의 유연성이 떨어짐 - 개발기간동안 결과물을 제시 못함

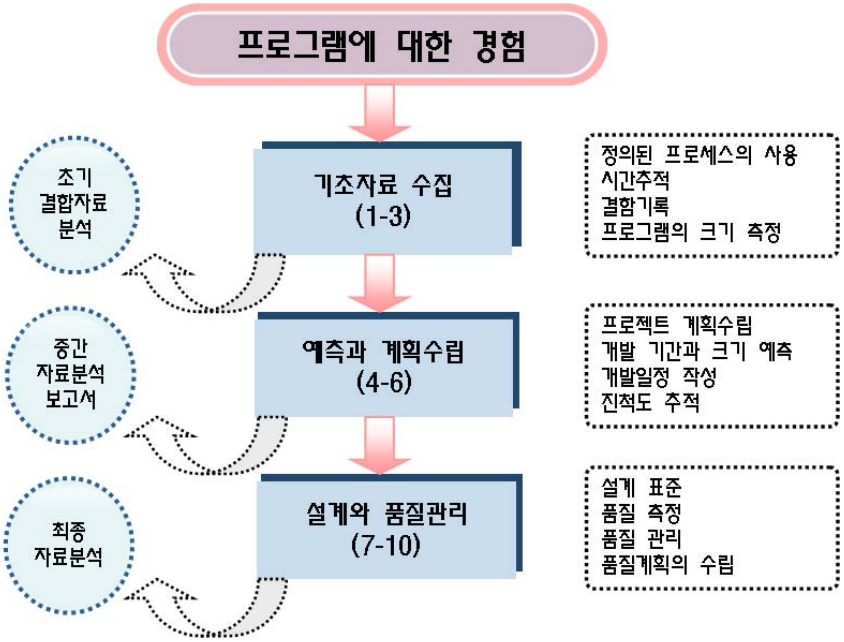
원형 모델	- 사용자 요구분석의 어려움을 해결하기 위해 실제 개발될 소프트웨어시제품 제작 - 중간 과정을 점검할 수 있는 계획표(일정표)나 결과물이 없음	- 제품의 신뢰성이 낮고 성능은 비교적 좋지 못함 - 쉽고 빠르게 시제품을 만들 수 있는 개발 모형
나선형 모델	- 폭포수 모델과 원형 모델의 장점에 새로운 요소인 위험 분석을 추가하여 만든 모델 - 시스템 개발 시 생기는 위험을 최소화하여 관리하고자 하는 것이 주목적	- 대규모 시스템 개발에 가장 현실적인 소프트웨어 공학 기법 - 개발과 유지보수의 구분이 없음 - 개발이 복잡하여 많은 사용자를 대상으로 하는 상업용 제품 개발에는 적용하기 힘듦
점증적 모델	- 시스템을 여러 번 나누어 배포하는 방법 - 개발되어 운용되고 있는 시스템과 개발이 진행 중인 시스템이 함께 공존	- 배포하는 동안 예상치 못한 문제를 신속히 해결 - 새로운 배포판에 기능 추가 가능

2. 기타 소프트웨어 개발 공정 방법론

가. PSP와 TSP

PSP(Personal Software Process)와 TSP(Team Software Process)는 소프트웨어의 품질을 관리할 수 있는 체계적이고도 효과적인 훈련을 개발자에게 제공함으로써 결함이 적은 소프트웨어를 개발할 수 있도록 지원하고 있다. PSP에서는 개발자들이 원래 계획한 예산과 기간내에 시스템을 개발할 수 있도록 지원하기 위해서는 지금까지의 잘못된 관행을 찾아서 이를 해결하기 위한 품질 개선 프로세스를 제공한다.

PSP 훈련은 소프트웨어 개발자가 우수한 작업을 할 수 있도록 10개의 프로그램 과제로 구성되어 있다.

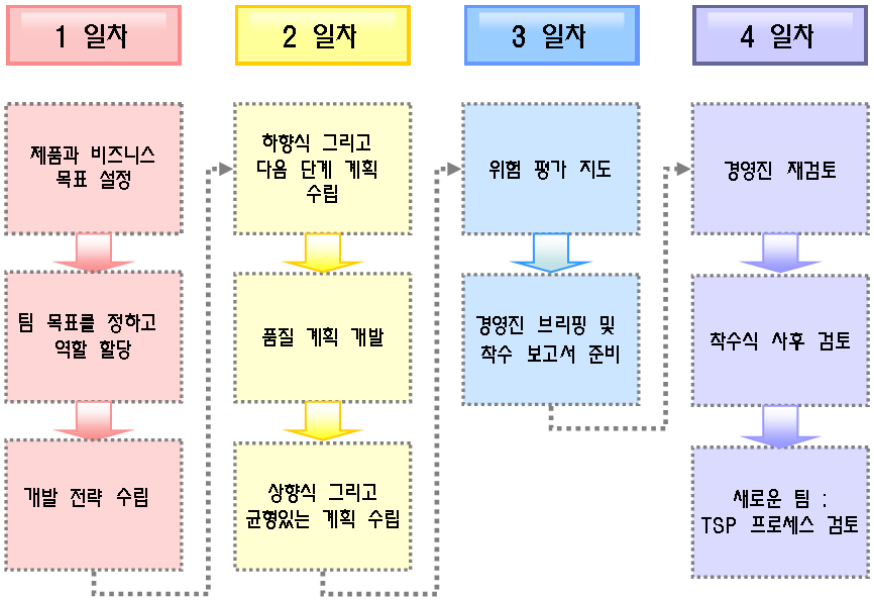


(그림 2-2) 10개의 PSP 프로그램 과제

PSP 훈련은 전반부에서는 3개의 프로그램 과제를 통해 실시되는데, 개선 활동에 필요한 기초자료를 수집하고 결함자료를 분석하기 위해 정의된 프로세스를 사용하는 방법과 시간을 추적하고, 결함을 기록하는 방법, 프로그램의 크기를 측정하는 방법을 훈련한다. 중반부에서는 3개의 프로그램 과제를 통해서 예측과 계획수립을 위한 중간 자료 분석 보고서를 작성한다. 이를 위해 프로젝트의 계획을 수립하고, 개발 기간과 프로그램의 크기를 예측하며, 예측된 개발 기간과 프로그램 크기를 이용하여 개발 일정을 작성하는 방법과, 개발계획과 실제 개발 진행상황을 비교하여 달성도를 추적하는 방법을 훈련하게 된다. 끝으로 종반부에서는 4개의 프로그램 과제를 통해서 설계와 품질관리와 관련된 최종 자료 분석 보고서를 작성하는데 필요한 설계 표준의 작성법, 소프트웨어 품질의 측정과 관리 방법, 소프트웨어 품질관리를 위한 계획수립 방법을 훈련한다.

TSP는 소프트웨어 개발 팀에게 동기를 부여하여 효과적인 소프트웨어 개발 팀을 구성하고 유지하는 훈련 방법을 제공한다. 일단 팀 구성원들의

준비가 끝나면 착수식(launch)을 하게 되는데 일반적으로 4일간 진행되며, 이 기간 동안 팀은 팀 목표를 설정하고, 팀 구성원간의 역할을 할당하며, 업무수행을 위한 계획 등을 설정하게 된다. 착수식이 종료되면, 팀은 실제 개발 작업에 들어가고, 팀은 주별 회의를 통해서 주 단위로 개별 개발자들의 개발 진행 상황과 팀의 개발진행 상황을 점검하고 필요한 조치를 취하게 된다. 일차적으로 팀 작업이 지나면 소프트웨어에 대한 요구사항의 변경, 팀 구성원의 변경, 팀 구성원의 학습효과에 의한 생산성 향상 등을 반영할 수 있도록 재착수식을 수행한다. TSP 프로세스는 반복적이고 점진적인 개발 전략을 따르고 있고, 각각의 단계 또는 사이클은 전 단계에서 얻어진 지식에 의존하여 계획되기 때문에 주기적인 재착수 회의는 필수적이다. 최고 경영자는 팀 착수식에 반드시 참석하여 계획하고 있는 프로젝트의 중요성을 개발 팀에서 인식시키고, 개발 팀은 최선을 다하여 실행 가능한 개발 계획을 수립하고 최고 경영진을 포함한 이해 당사자에게 수립된 계획의 타당성을 설명하고 승인받을 수 있어야 한다.



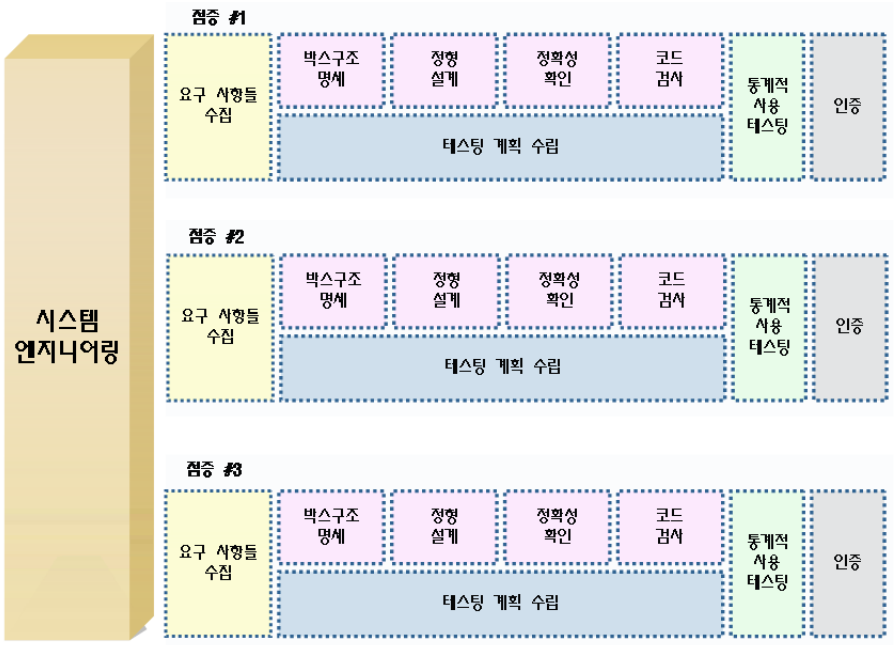
(그림 2-3) TSP의 착수식 프로세스

일단 TSP 팀이 작업에 착수하게 되면 팀 구성원들이 계획을 따라 다음과 같은 업무를 추진하게 된다.

- 팀 지휘
- 프로세스 훈련
- 문제점 추적
- 의사소통
- 경영진 보고
- 계획 유지보수
- 프로젝트 완료 추정
- 팀 작업량의 재분배
- 프로젝트 재착수식
- TSP 품질관리

나. Cleanroom

Cleanroom은 소프트웨어 개발 과정에서 엄격하고 정밀하게 오류를 검사하고 개발하여 오류가 적은 프로그램을 만드는 것을 최종 목표로 한다. Cleanroom 프로세스의 기본 원칙은 처음에 코드 점증(increment, 실행 가능한 원형)을 정확하게 작성하고 시험 전에 그들의 정확성을 확인해서 비교적 비용이 많이 드는 결함 제거 프로세스에 대한 의존성을 제거하는 데 있다. 정형 방법(formal method)에서 택한 접근법을 확장시킨 Cleanroom 접근법은 통계적인 품질 관리에 대한 기법을 강조한다.



(그림 2-4) Cleanroom 프로세스 모형

- 점증 계획 수립 : 점증적인 소프트웨어 개발 전략을 채택한 프로젝트의 계획을 개발한다. 이 단계에서 각 점증에 대한 기능, 계획된 규모, 그리고 Cleanroom 개발 일정이 결정된다.

- 요구 사항 수집 : 사용자 수준의 요구 사항에 대해 보다 구체적인 내용을 서술한다.

- 박스 구조 명세 : 박스 구조를 사용하는 명세 방법에 대해 박스 기능을 기술한다. 박스는 세부 사항의 어떤 수준에서 시스템(혹은 시스템의 어떤 관점에서)을 캡슐화한 것을 말하는 것으로 블랙 박스(black box), 상태 박스(state box), 클리어 박스(clear box)가 사용된다.

- 정형 설계 : 박스 구조 접근법을 사용한 Cleanroom 설계에서는 명세의 자연적이고 단절이 없는 확장이다. 비록 두 개의 활동들 간에 명백한 구분이 가능해도 명세는 아키텍처와 절차 설계가 유사하게 되도록 반복적으로 정제된다.

- 정확성 확인 : Cleanroom 팀은 설계와 코드상의 엄격한 정확성 확인 작업을 수행한다. 확인은 최상위 수준의 박스 구조로 시작해서 설계의 세부 사항과 코드로 옮겨간다.

- 코드 생성, 검사, 확인 : 특수 언어로 표현된 박스 구조 명세를 적합한 프로그래밍 언어로 변환된다. 또한 표준적인 검토 또는 검사 기법은 코드와 박스 구조의 어의 일치와 코드의 구문적 정확성을 확인하는데 사용되며 이 경우 정확성 확인은 소스 코드로 처리된다.

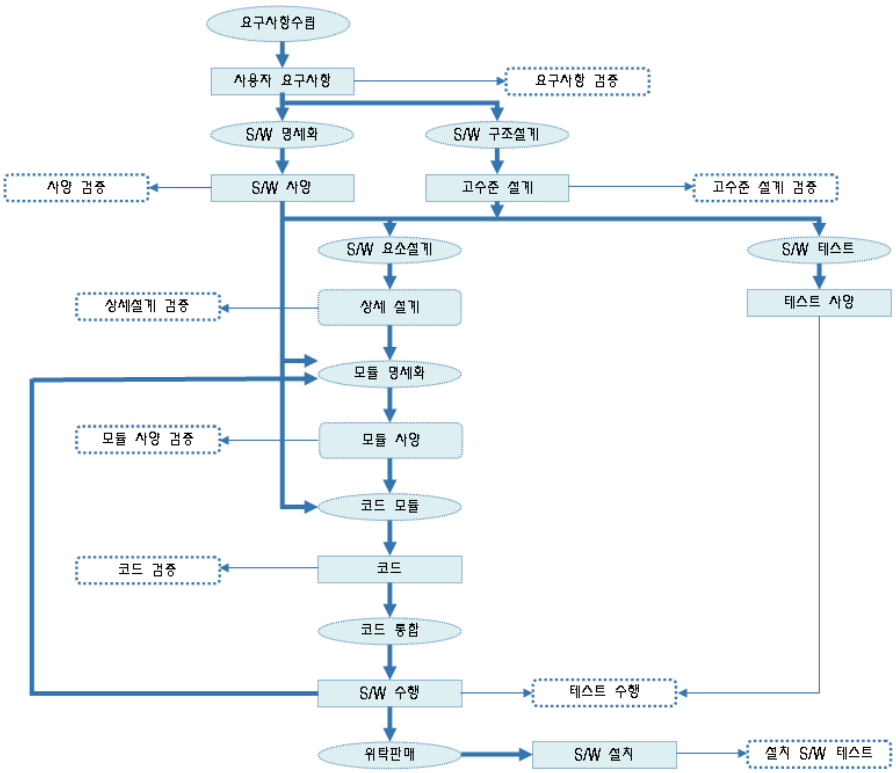
- 통계적 테스트 계획 수립 : 소프트웨어의 계획된 사용법이 분석되면, 사용법 확률 분포를 조사하는 일련의 테스트 사례들이 계획되고 설계된다. 소프트웨어는 사용 시나리오의 집합을 정의해서 각 시나리오에 대한 사용 확률을 결정해고 그 확률들을 따르는 랜덤 테스트를 정의해서 테스트 한다. 이 활동은 명세, 정형 설계 및 정확성 확인 및 코드 생성 절차와 병행하여 추진한다.

- 통계적 사용법 시험 : 컴퓨터 소프트웨어의 철저한 시험이 불가능한 것을 명심해서, 반드시 유한한 개수의 테스트 사례를 설계할 필요가 있다. 이 단계에서는 위에서 조사한 모든 가능한 프로그램의 사용법 확률 분포로부터 유도된 일련의 테스트를 수행한다.

- 인증 : 확인, 검사, 사용법 시험이 완료되어 모든 오류는 교정이 되었으면 점증은 통합을 위한 준비로서 인증된다.

다. Correction by Construction

Correction by Construction는 높은 무결성 소프트웨어 개발을 위해 정형 기법에 조기 검증 과정과 결점 제거 과정을 접목한 프로세스이다.



(그림 2-5) Correction by Construction 프로세스

- 사용자 요구사항 : 소프트웨어의 목적, 제공하는 기능 보안, 안전 그리고 성능과 같은 비함수적 요구사항을 서술한 것이다.

- 소프트웨어 명세서 : 블랙박스로 표현되는 소프트웨어의 동작을 완벽하고 자세하게 기술한 것이다. 여기에는 소프트웨어의 내부 구조에 관한 정보는 없다.

- 고 수준의 디자인 : 소프트웨어의 구조를 기술한 것이다.

- 상세 디자인 : 프로세스 구조 혹은 데이터베이스 스키마 등과 같은 소프트웨어의 다른 측면에서의 운영을 기술한 것이다.

- 모듈 명세서 : 각 소프트웨어 모듈에 의해 캡슐화된 상태와 동작을 정의한 것이다.

- 코드 : 각 모듈의 수행 코드를 만든다.

- 통합 소프트웨어 : 부분적인 동작을 제공하는 소프트웨어를 통합한 것이다. 일반적으로 초기 구조화는 단지 기반 구조 소프트웨어와 일부 응용 기능을 담고 있다. 각 통합화 작업은 연속적인 코드에 대한 테스트 작업과 같이 작용한다.

- 소프트웨어 설치 : 설치 프로그램이란 동작 환경하에서 최종 구조화되고 환경 설정이 되고 설치되는 소프트웨어를 말한다.

제 2 절 보안이 강화된 소프트웨어 개발 방법론

보안이 강화된 SDLC(Secure SDLC) 기법에서는 반복적인 위험 평가, 영향 분석 및 보안 모델링, 구성요소의 보안 평가, 통합/어셈블리 옵션 시제품화(prototyping) 및 소스 코드의 보안 코드화 와 보안 심사가 병행되는 통합 보안 실험을 지원해야 한다. SDLC 기법을 선정하고 응용함에 있어서, 개발자 및 통합자는 프로그래밍의 처음부터 보안에 초점이 맞추어져 있는지 혹은 취득하고 재사용하는 구성요소에 대한 적절한 보안 평가가 되고 있는지 등을 고려해야 한다. Microsoft나 Oracle사는 각각 보안 개발 생명주기 기법(Security Development Life Cycle)과 소프트웨어 보안 인증 프로세스(Software Security Assurance Process)를 통해 자사 소프트웨어 개발 프로세스의 보안을 강화하고 개발자들을 대상으로 소프트웨어 제품에서 보안이 갖는 중요성과 의미에 대한 교육을 실시하고 있다.

1. CLASP

CLASP(Comprehensive, Lightweight Application Security Process)은 Secure Software사에서 개발한 것으로 소프트웨어 개발 생명주기의 초기 단계의 보안을 강화한다는 구체적인 목표를 가지고 정의된 최초의 생명주기 프로세스로서 정확성과 품질 인증에 주안점을 둔 정형 프로세스이다. CLASP은 시스템의 코드를 작성하기 전에 적절한 애플리케이션 보안 문제를 명시하고 접근하는 것을 시작하기 위한 일련의 기법 및 실천 방법을 제시한다. CLASP은 활동 중심, 역할 기반의 프로세스로 구성된 집합체로서 다음과 같은 특징을 가지고 있다.

가. 기존 개발 프로세스에 대한 CLASP의 적응력

CLASP은 기존 응용 개발 프로세스에 보안 관련 활동(activity)을 집적하고자 설계되었다. 각각의 CLASP 활동은 프로세스 요소로 구분하지만

하나 이상의 특별한 프로젝트의 역할과 연동된다. 이와 같은 방법으로, CLASP은 프로젝트 관리자, 보안 감사 책임자, 개발자, 설계자, 시험 책임자 등 프로젝트 참여자에게 보안에 관한 지침을 제공한다.

나. CLASP 취약점 목록

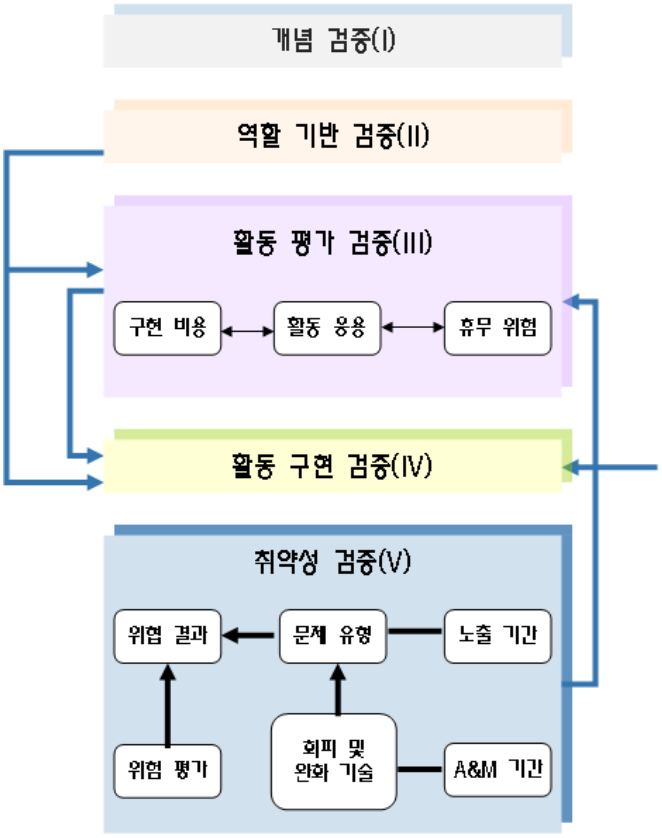
CLASP은 프로그램의 설계나 코딩 오류를 찾아내거나 개선하기 위해 개발 팀에게 취약점 목록을 제공한다. 이러한 목록의 기본은 아주 유연한 분류에 있는데 이를 통해 개발팀은 문제의 형태, 문제 형태의 범주, 노출 시기, 회피나 완화 시기 등 많은 가능성으로부터 취약점 목록 정보를 쉽게 찾게 된다.

다. 자동화된 분석 툴

CLASP의 취약점 목록에 있는 정보들은 소스 코드의 정적인 분석 기술을 사용한 자동화 툴을 이용하여 시행된다.

CLASP의 구조 및 프로세스 요소간의 종속성을 살펴보면 다음과 같다. 먼저 CLASP 프로세스는 아래와 같은 5개의 상위 수준의 검증(View)를 제공하며 이러한 뷰는 활동(Activity)들로 세분화되며 활동은 프로세스 요소(Process Component)를 포함한다. 즉, CLASP 프로세스는 View-Activity - Process Component 구조로 된 계층 구조이다. 다음 그림은 CLASP 검증(view)와 그 상호 작용을 보인 것이다.

- 개념 검증(Concepts View)
- 역할 기반 검증(Role-based View)
- 활동 평가 검증(Activity-Assessment View)
- 활동 구현 검증(Activity-implementation View)
- 취약성 검증(Vulnerability View)

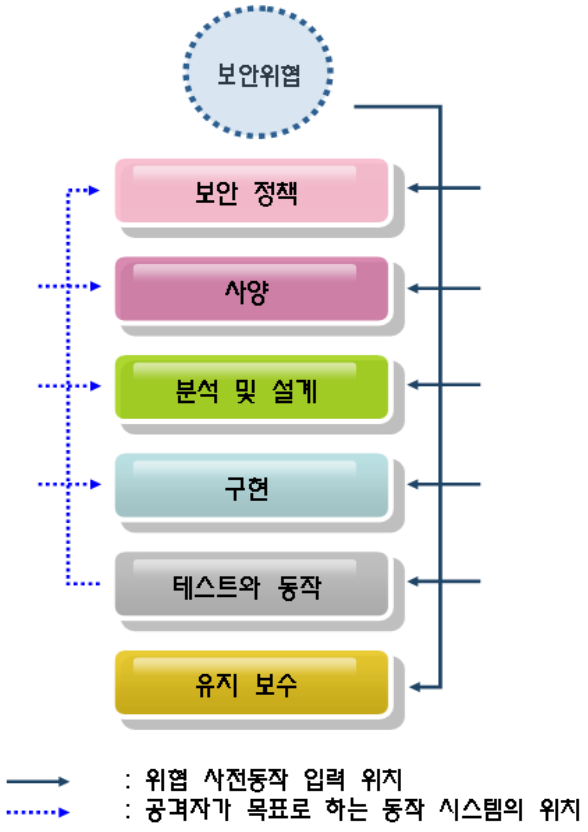


(그림 2-6) CLASP의 뷰와 상호작용

CLASP에서 최고의 실천 방법(best practice)은 모든 보안 관련 소프트웨어 개발 활동의 기본이 되며 다음의 7가지 실천 방법이 있다.

- 기관의 인식 프로그램
- 응용 평가 수행
- 보안 요구사항 인지
- 안전한 개발 업무의 구현
- 취약성 개선 과정의 수립
- 평가 척도(metric)의 정의 및 점검
- 운영상 보안 지침서 발행

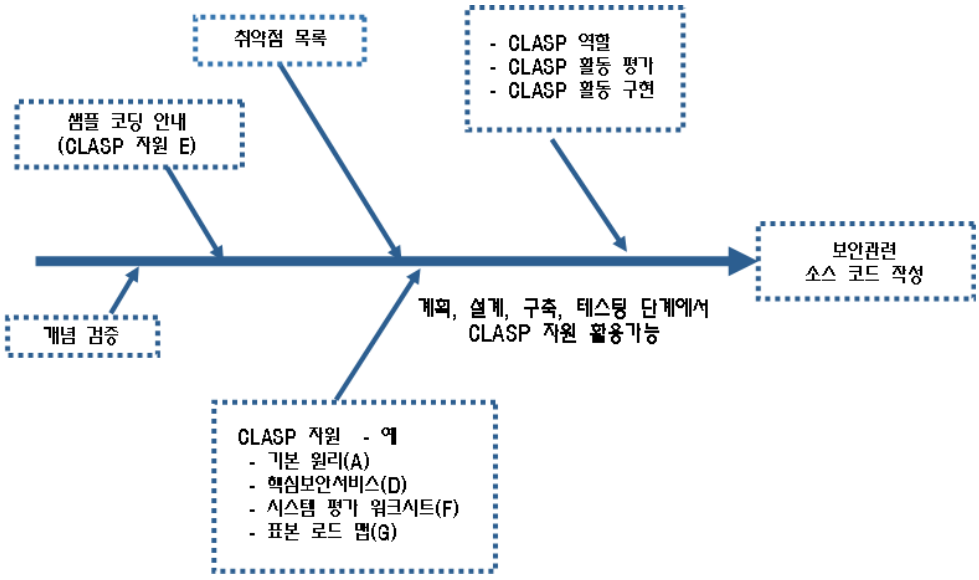
CLASP에서는 보안 취약성을 소프트웨어 환경에서의 결함이라고 정의하는데 응용 소스 코드에 보안 위협성을 근간으로 104개의 중요한 문제 유형을 명시하고 있다. 하나의 문제 유형은 그 자체로는 보안 위협이 되지 않는 경우도 있지만 종종 다른 문제와 조합되어 문제가 되기도 한다. 또한, 104개의 문제 유형을 5개의 고수준 레벨의 범주로 분류하였는데, 따라서 하나의 문제는 한 개 이상의 부모(parent) 범주를 갖게 된다. CLASP에서는 다음과 같은 6가지의 보안 서비스에 대한 위협의 결과를 정의한다. 그림은 소프트웨어 개발 단계에서 어느 단계가 보안과 관련된 위협이 발생하는지 그리고 운영 시스템의 어느 시점에서 공격이 이루어지는지를 명시하고 있다.



(그림 2-7) SDLC 단계의 취약점

- 공증(자원 접근제어)
- 기밀성(데이터나 다른 자원)
- 인증(접속 인식 및 무결성)
- 가용성(서비스 거부)
- 책임성
- 부인 봉쇄

다음 그림은 CLASP 자원 E인 표본 코딩 안내(Sample Coding Guideline, CLASP 자원 E)을 사용하여 CLASP의 여러 요소를 적용하는데 가능한 순서를 도시한 것이다.



(그림 2-8) CLASP 요소의 적용

1) CLASP 프로세스의 개요를 알기 위해 CLASP 개념 점검을 읽는다.
2) CLASP 개념 검증에는 프로세스의 페이지 설명에 주목해야 하는데 여기에는 도식 설명, 104개 문제 유형의 위치, 5개의 고수준, 소스 코드 관련 카다로그 그리고 보안 취약성에 대한 결과 등을 포함하고

있다.

3) CLASP 자원 E인 Sample Coding Guideline을 읽고 소프트웨어 개발 프로젝트와 관련된 부분을 찾는다.
4) 계획, 설계, 구축, 시험 등을 통해 나머지 CLASP 자원을 활용한다.
5) 가장 중요한 단계로서 CLASP 취약성 검증에 있는 104개의 문제 유형 중 일부를 선택하기 위해 Sample Coding Guideline을 사용한다.
6) CLASP 역할 기반 검증에 적용한다. 이것은 Sample Coding Guideline 중에서 선택된 부분을 적용하는데 연관된 프로젝트의 역할을 제공하며 설계자나, 보안 전문가, 감시 책임자, 구현 전문가에게 지침을 할당하게 된다.
7) 활동 평가 검증을 이용한 평가를 할 때는 Sample Coding Guideline 중에서 선택된 취약점의 부분을 고려한다. 또한, CLASP 활동 구현 검증에 포함된 24개의 활동 중 원하는 부분을 선택하게 된다.

2. Microsoft의 SDL

빌 게이츠가 2002년 1월에 적성한 Trustworthy Computing 메모로 시작하여, Microsoft는 보안 수준이 더욱 높은 소프트웨어를 만들기 위한 프로세스 개선 작업에 착수했다. 이러한 프로세스 보안 및 개인정보보호 요소를 SDL(보안 개발 생명주기: Security Development Lifecycle)이라 한다. Microsoft에 따르면, 지금까지 SDL하에서 개발된 소프트웨어는 SDL를 채택하기 전에 개발한 동일한 소프트웨어의 버전에 비해 취약성이 50 퍼센트가 감소했다고 한다. 단, SDL은 아직 Windows XP와 SQL Server의 개발에만 적용되었으며 Microsoft의 다른 제품들은 제작 및 유지관리 프로세스의 보안 향상의 혜택을 아직 누리지 못하고 있다.

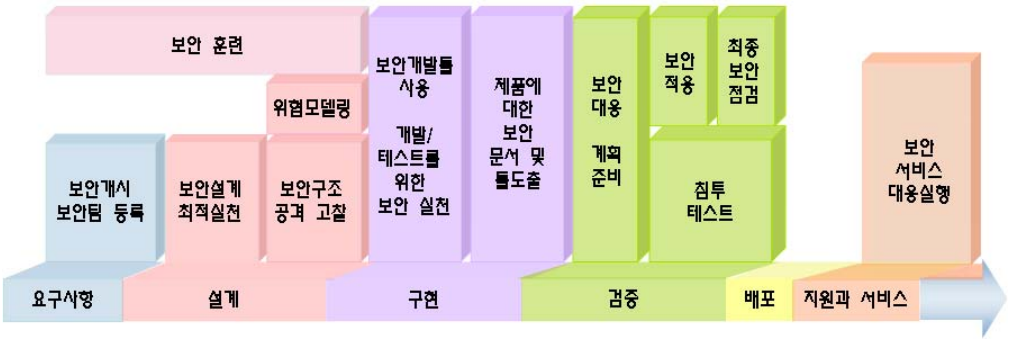
Microsoft사는 필수 인프라를 보호해야 하는 필요성을 느끼거나 컴퓨팅에 대한 폭넓은 신뢰를 구축하고 보존해야 하는 필요성을 느끼는 모든 소프트웨어 업체들에게 보안은 핵심적인 요구사항이라고 보고 있다. 모든 소프트웨어 업체들이 직면하는 커다란 과제 중 하나는 패치를 통한 업데이트와 부담스러운

보안 관리를 경감할 수 있는 보안 수준이 높은 소프트웨어를 만드는 것이다. Microsoft에 따르면, 소프트웨어 업계가 현재의 보안 개선 요구를 충족할 수 있는 능력을 갖는 열쇠는 실질적으로 개선된 보안을 제공하는 반복 가능한 프로세스를 이행하는 데 있다고 한다고 본다. 따라서 프로세스의 목적은 설계, 구현 및 문서화로 인한 보안 취약성의 수를 최소화하고 이러한 취약성을 최대한 개발 생명주기의 초기에 감지하고 제거하는 것이다.

Microsoft에서는 보안 수준이 높은 소프트웨어의 제작에는 다음 3가지 요소가 필요하다고 보고 있으며 이중에서 SDL의 성공에 중요한 점은 자사 개발자들을 대상으로 한 보안 개발을 위해 필요한 기법 및 기술에 관한 지속적인 교육 및 훈련이었다.

- 반복 가능한 프로세스
- 엔지니어 교육
- 측정 기준 및 책임성

SDL은 보안 관련 작업 및 산출물을 기존의 소프트웨어 개발 생명주기에 접목한다. 그림은 SDL의 단계를 매우 간단하게 묘사하고 있으며 SDL의 주요 단계와 요구되는 작업 및 산출물을 설명하면 아래와 같다.



(그림 2-9) SDL을 통한 소프트웨어 개발 프로세스의 개선

가. 요구사항 단계

보안을 “처음부터“ 고려해야 하는 필요성은 보안 시스템 개발의 기초적인 개념이다. 요구사항 단계에서, 제작 팀은 중앙 보안 팀과 연락하여 계획에 진행됨에 따라 연락 창구, 자원 및 가이드의 역할을 하는 보안 자문가를 배정해 줄 것을 요청한다. 보안 자문가는 계획을 검토하고, 권장사항을 제안하고, 보안 팀이 제품 팀의 일정을 지원하기 위한 적절한 자원을 계획하도록 보장함으로써 제품 팀을 보조한다. 보안 자문가는 프로젝트 구상 단계부터 최종 보안 점검 및 소프트웨어 출시될 때까지 보안 팀과 제품 팀의 연락 창구 역할을 계속 유지한다. 요구사항 단계에서는 제품 팀이 보안을 어떻게 개발 프로세스에 통합할 것인지를 생각하고, 주요 보안 목표를 결정하고, 그 밖의 방법으로 계획 및 일정 중단을 최소화하면서 소프트웨어 보안을 극대화한다. 또한 개발 팀은 소프트웨어의 보안 기능 및 인증 조치가 소프트웨어와 함께 사용될 가능성이 높은 다른 소프트웨어와 어떻게 통합될 것인지를 고려해야 한다.

나. 설계 단계

설계 단계에는 소프트웨어의 전체적인 요건과 구조를 제시한다. 보안의 관점에서, 설계 단계에서 고려하는 주요 요소는 다음과 같다.

(1) 보안 구조 및 설계 지침 정의 : 소프트웨어의 전체적인 구조를 보안의 관점에서 정의하고, 올바른 기능 수행이 시스템 보안에 필수적인 소프트웨어 부분을 확인한다.
(2) 소프트웨어 공격 외형 요소 기록 : 소프트웨어가 완벽한 보안을 실현하는 것은 불가능하므로, 대부분의 사용자가 사용할 기능만 모든 사용자에게 기본적으로 노출시키고, 그러한 기능을 가능한 한 최소한의 권한 수준만으로 설치할 수 있도록 하는 것이 중요하다.
(3) 위협 모델링 작업 수행 : 위협 모델링 프로세스는 구조화된 기법을 사용하여 각 자산에 피해를 줄 수 있는 위협과 피해가 가해질 가능성을 확인하고 위험을 완화하기 위한 대응 조치를 파악하는데

도움을 준다. 가장 위험이 높은 구성요소가 가장 심도 있는 코드 점검과 가장 철저한 시험을 필요로 하므로, 이 프로세스는 코드 점검 및 보안 시험을 진행하는 데에도 도움이 된다.

다. 구현 단계

구현 단계에 제품 팀은 소프트웨어를 코드화하고, 테스트하고, 통합한다. 보안 결함 및 약점을 제거하거나 예방 조치를 하게 되면 소프트웨어의 최종 버전에 남아있게 될 가능성을 크게 줄인다. 적용되는 SDL 요소는 다음과 같다.

(1) 위협 모델링 작업을 통해 어떤 구성요소가 가장 위험이 높은 지를 파악한다.
(2) 동료가 점검하거나 코드 검사 전에 코드화 및 시험 표준과 올바른 실천방법을 적용한다.
(3) 퍼징(fuzzing) 도구 등을 이용한 보안 시험 도구를 적용한다.
(4) 정적 분석 코드 스캔 도구와 바이너리 검사 도구를 적용한다.
(5) 보안 코드 심사를 실시한다.

제품의 사용자를 위해 올바른 보안 실천방법을 문서화하고, 필요할 경우 사용자가 보안 수준을 확인하는데 도움이 되는 도구를 제작한다.

라, 검증 단계

검증 단계는 소프트웨어가 기능적으로 완전하여 베타 시험에 돌입하는 시점이다. 제품 팀은 이 단계에서 소프트웨어가 배포된 후에 직면할 위협에 대응할 수 있음을 보증하기 위해 침입 시험을 실시할 수 있다. 타사의 전문 침입 시험 실시자도 이 시점에서 동원될 수 있다.

마. 배포 단계

배포 단계에는 소프트웨어에 대한 최종 보안 점검을 실시하는데 이는

중앙 보안 팀이 조직을 위해 실시하는 독립적인 소프트웨어 점검이다. 이를 위한 작업에는 처음에 보안 버그인 것으로 확인되었으나 심층 분석을 통해 보안에 영향을 미치지 않는 것으로 밝혀진 버그를 점검하여 분석이 올바로 이루어졌는지를 확인하는 작업이 포함된다. 최종 보안 점검에는 유사 소프트웨어에 영향을 미치는 것으로 새롭게 보고된 취약성에 저항할 수 있는 소프트웨어의 능력을 점검하는 작업도 포함된다. 주요 소프트웨어 버전에 대한 최종 보안 점검에는 침입 테스팅과 추가적인 코드 점검, 그리고 잠재적으로 중앙 보안 팀을 보완하는 독립적인 외부 계약업체에 의한 보안 점검 및 침입 시험이 필요하다.

바. 지원 및 서비스 단계

개발 과정에서 SDL을 사용함에도 불구하고 아직도 취약성으로부터 완전히 자유로운 배송 소프트웨어를 지원하지 못하기도 한다. 따라서 제품 팀은 배송 소프트웨어에서 새로 발견되는 취약성에 대처할 준비를 해야 한다. 대응 프로세스의 일부는 취약성 보고서 평가를 준비하고 적절할 때 보안 권고 및 업데이트를 발표하는 것을 포함해야 한다. 또한, 대응 프로세스는 각각의 보고된 취약성에 대한 사후 검토를 실시하고 필요에 따라 SDL 프로세스, 교육 및 도구 사용을 업데이트하는 것과 같은 조치를 취한다.

제 3 절 소프트웨어 취약점 시험 기술

소프트웨어 취약점 시험은 소프트웨어 시스템이 작동 및 상호 동작하는 방식, 그리고 소프트웨어 시스템의 컴포넌트들이 작동 및 상호 동작하는 방식을 시험한다. 본 장에서는 취약점 시험 목적과 시기 선정에 대해 간략하게 살펴본 후, 화이트박스 시험 기법, 블랙박스 시험 기법, 사후 보안분석, 취약점 시험 도구에 대한 정리 분석한다.

1. 취약점 시험 목적과 시기

가. 소프트웨어 취약점 시험의 목적

소프트웨어 취약점 시험의 주요 목적은 소프트웨어에 다음과 같은 특징이 있는가의 여부를 검증하는 데 있다.

(1) 의도적인 오류 발생시에도 안전한 상태를 유지할 수 있는 능력: 소프트웨어의 무결성이나 가용성, 그리고 민감한 데이터 및 프로세스의 기밀성이 의도적인 오류 (예를 들어 일반적으로 소프트웨어 실행 파일의 변경을 시도함으로 인해 발생하는 오류나 소프트웨어에 대한 서비스 거부 공격 등)에 의해 기능이 저하되지 않아야 한다.

(2) 악용될 수 있는 약점의 부재: 소프트웨어에 공격자 (사람 또는 악성 코드)가 소프트웨어 자체나 소프트웨어가 사용하는 데이터, 소프트웨어 실행 환경의 보안 특성을 저해하기 위해 활용할 수 있는 오류, 백도어, 숨겨진 기능 등의 약점이 없어야 한다. 웹 콘텐츠 자체는 주민등록번호와 같은 민감한 정보를 표시하는 것처럼 안전하지 않은 방식으로 표시할 수도 있고, 콘텐츠가 외부 데이터베이스나 기타 출처로부터 유입될 수도 있기 때문에 모든 웹 콘텐츠를 개발자가 통제할 수 있는 것은 아니다. 하지만 개발자가 통제할 수

있는 콘텐츠는 공격자들이 민감한 정보에 접근할 수 없도록 해야 한다.

(3) 예측 가능한 보안 행위: 소프트웨어는 소프트웨어 자체, 환경, 데이터의 보안 특성을 저하시키지 않으면서 지정된 기능을 수행하여야 한다. 이에는 소프트웨어가 정상 동작 상태에서 부주의나 의도적인 공격 패턴에 의해 “휴면“ 상태로 갑자기 전환되지 않도록 하는 것 등도 포함된다.

(4) 보안 인지 오류 및 예외 처리: 소프트웨어는 정상상태의 오류 및 예외 처리 절차를 안전하게 수행할 수 있어야 한다. 그리고 안전하지 않은 상태에서 예외 처리 기능이 의도하지 않은 비정상적인 현상이나 의도적인 오류에 반응하지 않아야 한다.

(5) 보안 요구사항의 적절한 이행: 소프트웨어가 지정된 보안 규격을 충족시켜야 한다. 소프트웨어가 상기 4가지 목적을 달성하더라도, 보안 규격을 만족하지 못한다면 설치된 후에도 여전히 취약할 수 있다. 예를 들어, 소프트웨어가 상기 요건들을 만족한 상태로 다운로드 한 모바일 코드를 실행 전에 이 다운로드한 코드의 서명을 제대로 검증하지 못하는 경우가 생길 수 있다. 만약, 이 휴대폰 코드가 악성 코드인 경우 더 큰 위험성에 노출되게 되는 것이다.

소프트웨어 취약점 시험에서 최종적으로 확인된 사항은 담당자들이 충분한 근거를 통해 소프트웨어의 배포 및 사용 방식을 결정하고, 개발 중에 발생한 보안상의 문제점을 리엔지니어링을 통해 해결할 때까지 상쇄시킬 수 있도록 설치 시 적용해야 하는 요건들을 면밀히 검토 확인하여야 한다.

나. 취약점 시험의 시기 선정

소프트웨어 취약점 시험은 주로 소프트웨어가 개발 완료된 후(심지어는 배포된 후)에야 시행되어 왔다. 통상적으로 이러한 시험은 잘 알려져 있는

취약점을 악용하여 설치된 시스템에 대한 침투를 시도하는 “타이거팀(Tiger Team)“이 참여하며, 침투에 성공하면 취약점의 가능한 원인을 문서화하여 소프트웨어를 재구성하고, 부적절한 파일 허용을 재설정하고, 패치를 적용할 수 있게 하는 것이다. 하지만, 이 시험 방식은 특히 시기의 관점에서 볼 때 소프트웨어 생명주기에서 지나치게 늦은 시기에 이루어지며, 따라서 공격자들을 항상 뒤따라가는 형태일 수 밖에 없다는 것이다. 따라서 취약점 시험은 소프트웨어 시스템의 생명주기 전체에 걸쳐 시행하는 것이 바람직하다.

(그림 2-10)는 생명주기 각 단계에 걸친 다양한 취약점 시험 기술의 적용시기가 제안되어 있다.

(그림 2-10) 소프트웨어 생명주기에서의 취약점 시험 적용시기

생명주기에 걸친 취약점 시험 내용에는 다음과 같은 것들이 포함되어야 한다.

- 요구사항, 아키텍처, 설계 규격, 문서화에 대한 보안 검토
- 초기 모듈, 오픈 소스 소프트웨어, 아웃소싱한 상용 소프트웨어 및 재사용한 컴포넌트에 대한 소스 코드 보안 검토
- 바이너리 인수하거나 재사용한 소프트웨어의 조립/통합에 대한 (소프트웨어 평가 및 선정 절차의 일부로서 실시하는) “블랙박스“ 보안 분석 및 시험
- 다양한 설계 옵션에 따른 컴포넌트의 보안 동작을 파악하고, 컴포넌트들 간의 인터페이스 보안과 실행 환경에서의 통합/조립 및 외부 개체 간의 인터페이스 보안을 확인하기 위한 조립/통합 옵션 프로토타입 시험
- 최종 선정 또는 도입된 통합/조립 옵션의 통합 취약점 시험

2. 화이트 박스 보안 시험 기법

화이트박스(White Box) 시험 및 검토는 소스 코드를 대상으로 하는 시험 기법이다. 이러한 기법들은 최대한 이른 시기에 시행해야 하며, 전체 생명주기에 걸쳐 반복적으로 시행하여 소규모 코드를 대규모 코드베이스에 통합되기 전에 수정할 수 있도록 해야 한다.

화이트 박스 시험은 또한 작은 단위의 컴포넌트(예를 들어, 개별 모듈 또는 함수 등)에서 먼저 시행하고 나중에 전체 소프트웨어 시스템 차원에서 시행할 때 효율성이 더 높아진다. 이러한 방식으로 시험을 반복하면 최초 단계의 시험 중에 작은 컴포넌트에 있는 오류가 발견될 가능성이 높아지기 때문에, 마지막에 전체 시스템 코드 검토를 시행할 때 컴포넌트들 간의 인터페이스 및 관계를 나타내는 코드 묶음 간의 통합에 주력할 수 있게 된다.

화이트 박스 시험은 초기 코드만으로 한정되지 않아야 하며, 소프트웨어

시스템에 사용되는 모든 오픈 소스 컴포넌트까지도 대상으로 해야 한다.

가. 코드 보안 검토

코드 검토 (또는 감사)에서는 소프트웨어에 사용된 코딩 관행을 조사한다. 이러한 검토의 주요 목적은 보안상의 오류와 그 해결책을 찾기 위한 것이다. 소스 코드 보안 검토는 담당자의 전문지식에 대한 의존도가 매우 높기 때문에 개발팀의 보안 전문가, 기업의 위험성 관리 및 인증 팀, 또는 독립적인 전문 기관이 시행하도록 하여야 한다. 필요로 하는 검토 시간을 줄이기 위해, 위험성 분석에서 가장 취약한 것으로 확인된 코드만 검토하도록 할 수도 있다.

코드 보안 검토를 시행하기 위한 기법은 수동에서 완전 자동까지 다양하다. 수동 검토에서는 담당자가 오류 및 기타 취약점을 찾아낼 때까지 자동화된 코드 검토 도구의 도움 없이 모든 코드를 검사한다. 이것은 담당 인력 자원이 많이 소요되기도 하지만, 초기 단계에서 가장 완벽하고 정확한 결과가 도출되는 경우가 많다. 완전 자동화된 검토에서는 자동화된 코드 검토 도구가 모든 코드 검사를 수행하며, 담당자의 역할은 결과를 해석하는 것이다.

완전 자동화된 검토의 문제점은 관련 도구가 이미 프로그램화된 패턴 목록에 한해서만 정확한 결과가 나온다는 것이다. 또한 자동화된 코드 검토는 여러 코드 간의 관계에서의 모든 취약점을 파악할 수 없지만, 이러한 관계의 파악을 지원하는 도구들이 현재 개발되고 있다.

수동 및 완전 자동식 코드 검토의 중간에 해당되는 방법으로 반자동 검토가 있다. 이 방식은 담당자가 자동화된 도구를 수동 검사의 보조 수단으로 사용한다. 이러한 도구들은 특정한 패턴이 포함되어 있는 코드를 찾아내서 담당자가 이러한 코드로 수동 검사의 대상을 축소시킬 수 있게 한다. 또한 자동화된 도구들은 대규모의 코드 베이스를 수동 검토에 비해 매우 짧은 시간 내에 탐색할 수 있으며, 동시에 일관된 결과와 척도를 제공한다.

코드 검토에서는 또한 의심되는 악성 코드에 중점을 두고 악성 로직의 징후 및 위치를 파악할 수 있어야 한다. 쉘 스크립트로 작성된 의심되는 악성 코드의 경우 담당자는 다음과 같은 것들을 찾아보아야 한다.

- 코드가 악용될 수 있는 코드임을 나타내는 설명의 존재
- 호스트 로그인을 가능하게 하는 코드
- 파일 소유권이나 허용 수준을 변경하는 코드

※ 코드 보안 검토에 활용 가능한 도구 목록은 부록 1. 소프트웨어의 취약점 점검 및 오류 검증 도구의 ‘소스코드 오류 검사 도구’ 참조

나. 소스 코드 오류 주입

소스 코드 오류 주입 방식은 프로그램에 비침투적인 방식으로 변경을 가하여 소프트웨어 소스 코드를 조작하고, 조작된 프로그램을 컴파일 및 실행하여 조작된 코드가 실행될 때 상태가 어떻게 변하는가를 관찰하는 것이다. 이러한 방식을 통해, 소프트웨어가 (의도적인 오류에 의한 경우를 포함하여) 강제적으로 비정상적인 상황에 놓이게 되었을 경우의 행동 양상을 파악하는 데 사용된다. 특히 포인터 및 배열의 올바르지 못한 사용, 위험한 호출의 사용 등을 파악하는 데 유용하다.
소스 코드 오류 주입은 코드 도입 절차에 걸쳐 반복적으로 사용할 때 가장 효과적이다. 새로운 위협 (공격 유형 및 침입 기법)이 발견되었을 때에는 소스 코드를 다시 수정하여야 한다.

다. 컴파일 시간 및 런타임 결함 탐지

컴파일 시간 탐지 및 런타임 탐지는 코드 검토시 발견되지 않은 코드 오류를 탐지하고 표시 (또는 제거)하기 위해 방법이다. 모든 기본 컴파일

러는 간단한 컴파일 시간 탐지가 가능하도록 되어 있다. 이러한 검사는 간단한 오류를 확인하기에는 적합하지만 더 복잡한 취약점을 확인할 수 있을 정도로 정교하지는 않다.

반면에, 일부 컴파일러에는 복잡한 보안 특성을 확인하기 위한 전체 프로그램 검증 기능이 포함되어 있다. 이러한 검증은 컴파일 전에 수립해야 하는 공식적인 규격을 기반으로 한다. 프로그램 검증은 C 및 C++ 프로그램과 라이브러리의 오류 또는 포맷 스트링 공격이나 버퍼 오버플로우 공격 등을 확인하는 데 가장 많이 사용된다.

※ 컴파일 시간 및 런타임 결함 탐지에 활용 가능한 도구 목록은 부록 1. 소프트웨어의 취약점 점검 및 오류 검증 도구의 ‘실행코드 보안 취약성 점검 및 방지 도구’ 참조

3. 블랙 박스 보안 시험 기법

블랙 박스(Black Box) 보안 시험 기법은 실제 코드를 검사하는 것이 아닌 소프트웨어의 실행파일을 대상으로 하며, 기본적으로 소프트웨어의 출력 및 행동양상을 관찰하는 것이다. 인수 및 재사용한 블랙 박스 컴포넌트의 경우에는 블랙 박스 시험이 유일한 시험 기법이다. 하지만 소스 코드에서 컴파일한 실행 파일 시험에도 유용하게 쓰일 수 있다. 담당자가 실행중인 소프트웨어를 관찰하여, 특정한 컴포넌트가 다른 컴포넌트, 환경, 사용자로부터 다양한 입력을 받을 때에 화이트 박스 시험 중에 수립된 가정을 확인할 수 있기 때문이다. 블랙 박스 시험은 또한 담당자가 다양한 사용자 입력 및 환경 구성, 상태 변경에 대한 소프트웨어의 대응방식을 관찰할 수 있게 해 준다. 이러한 소프트웨어 및 외부 환경 및 사용자 간의 상호작용은 화이트 박스 시험에서는 확인할 수 없다.

가. 소프트웨어 침입 시험

소프트웨어 침입 시험에서는 담당자가 각 바이너리 컴포넌트나 소프트웨어 시스템 전체를 대상으로 하여 컴포넌트들 간의 취약점 악용을 통해 소프트웨어 시스템, 데이터, 환경 자원을 침해할 수 있는가의 여부를 파악한다. 침입 시험에서는 기능 시험에서 간과된 중요한 보안상의 취약점을 찾아낼 수도 있다.

침입 시험 시나리오는 소프트웨어 설계의 잠재적인 보안 문제점을 대상으로 해야 하며, 소프트웨어의 위험성 분석 및 최악의 사례 시나리오(예를 들어, 적대적인 인증된 사용자)에 의해 가장 발생률이 높거나 가장 치명적인 것으로 확인된 위협(공격, 침입)를 재생성하는 시험이 수반되어야 한다. 시험 기획자가 요건 및 설계 규격, 도입 문서, 사용자 및 관리자 설명서, 하드웨어 도면에 대한 정보를 접할 수 있도록 하는 것이 가장 이상적이다.

나. 실행파일의 보안 오류 주입

실행파일의 오류 주입은 기존의 시험 기법에서 찾아낼 수 없었던 안전을 위협하는 오류를 찾아내기 위한 기법이다. 안전 오류 주입은 소프트웨어에 스트레스를 가하고, 컴포넌트들 간의 상호작용 문제점을 일으키고, 실행 환경에서의 오류를 시뮬레이션하는 데 사용된다. 보안 오류 주입은 유사한 방식을 활용하여 소프트웨어에 대한 의도적인 공격, 그리고 공격자가 악용할 수 있는 의도하지 않은 오류에 의한 상황을 시뮬레이션 한다.

보안 오류 주입은 담당자가 소프트웨어가 공격에 대처하는 방식을 더 완벽하게 이해하도록 보안 침입 시험의 보조수단으로 활용할 때 가장 유용하다. 보안 오류 주입에는 데이터 조작(즉, 실행 환경 컴포넌트가 소프트웨어 시스템에게 전달되거나 소프트웨어 시스템의 컴포넌트가 다른 컴포넌트에게 전달되는 데이터 유형 변경 등)이 수반된다. 오류 주입은 컴포넌트 자체의 행동양상, 그리고 전체 소프트웨어 시스템에 대한 보안 오류의 영향을 파악할 수 있다.

특히 환경상의 오류는 실제 공격 시나리오를 가장 적절하게 반영하기

때문에 시뮬레이션에 매우 유용하다. 하지만 주입되는 오류는 실제 공격을 시뮬레이션하는 것으로 한정되지 않아야 한다. 침입 시험과 마찬가지로, 오류 주입 시나리오는 담당자가 모든 가능한 운영 환경에서의 소프트웨어 시스템의 보안 양상, 상태, 보안 특성에 대해서 최대한 완벽하게 이해할 수 있도록 설계되어야 한다.

다. 퍼지 시험

퍼지 시험(Fuzzy Testing)은 잘못된 데이터가 환경을 통해 소프트웨어 시스템에 입력되거나 한 프로세스에 의해 다른 프로세스에 입력된다는 점에서 오류 주입과 유사한 기법이다. 퍼지 시험은 몇몇 입력 조합을 시험 대상에게 제시하여 그 반응을 관찰하기 위한 프로그램이나 스크립트인 조작기(Fuzzer)에 의해 시행된다. 조작기는 소프트웨어에게 주로 유효한 입력을 수정해서 만드는 반무작위식 입력을 제공한다. 다른 시험 기법과 마찬가지로, 퍼지 시험도 효과적으로 이루어지려면 담당자가 대상 소프트웨어 시스템, 그리고 이것이 환경과 상호작용하는 방식을 완벽하게 이해하고 있어야 한다. 대부분의 조작기는 특정 프로그램에서 사용하도록 작성되기 때문에 재사용하기가 쉽지 않다. 하지만 취약점 탐색기 및 오류 주입기 등의 더 일반적인 도구가 찾아낼 수 없는 보안상의 오류를 찾아낼 수 있는 이점이 있다.

라. 역엔지니어링 시험

실행파일의 역엔지니어링(Reverse Engineering)은 역어셈블리나 역컴파일을 통해 이루어진다. 역어셈블리에서는 어셈블러 코드를 바이너리 실행파일로부터 재구성하여 담당자가 어셈블러 단계의 코드에서 확인할 수 있는 보안 관련 코드 오류 및 취약점을 찾아낼 수 있도록 한다. 역어셈블리는 담당 엔지니어가 역어셈블러에 의해 생성된 관련 어셈블러 언어를 완벽하게 이해하고 있어야 하며, 검토 담당자가 어셈블리 언어의 보안 오류를

찾아내기가 고수준의 언어에 비해 훨씬 어렵기 때문에 매우 어려운 작업이 될 수 있다.

역어셈블리와는 달리, 역컴파일은 바이너리 실행파일로부터 고수준 언어의 소스코드를 생성한다. 역컴파일된 소스 코드는 보안 코드 검토 및 기타 화이트 박스 시험에 사용할 수 있다. 하지만 역컴파일을 통해 생성한 소스 코드는 원래 소스 코드만큼 탐색 및 이해하기가 쉽지 않다. 따라서 역컴파일된 코드의 보안 코드 검토는 원래 소스 코드의 검토에 비해 훨씬 어렵고 더 많은 시간이 걸린다.

2가지 기법 모두 위험성이 매우 높은 것으로 간주되는 매우 높은 가치의 소프트웨어로 엄격하게 제한되는 경우에만 실용적일 가능성이 높다. 여러 상용 소프트웨어 제품들은 역엔지니어링을 방지하기 위해 다양한 퍼지 시험을 사용하며, 이것은 역어셈블리/역컴파일 시험에 필요한 노력을 배가시켜 실용성을 저하시킨다. 또한, 상용 소프트웨어의 배포 라이선스에서 명시적으로 역엔지니어링을 금지하는 경우도 있다.

마. 애플리케이션 취약성 자동 스캔

애플리케이션 취약성 자동스캔에는 상용 또는 오픈 소스 스캐너가 사용된다. 이 스캐너는 실행중인 애플리케이션을 검색하여 스캔 도구의 취약점 징후 데이터베이스에 저장된 알려져 있는 취약점과 관련된 패턴과 일치하는 입력/출력 패턴 및 행동양상을 찾는다. 취약점 스캐너는 기본적으로 자동화된 패턴 대조 도구이며, 자동화된 코드 검토 도구와 유사하다. 또한 자동화된 코드 검토 도구와 마찬가지로, 취약점 스캐너도 각 취약점이 모여서 커진 위험성을 고려하거나 특정한 행동양상 또는 입력/출력의 예측할 수 없는 조합에 의한 취약점을 파악하기가 어렵다. 하지만 여러 취약점 스캐너는 취약점들의 결합에 대한 몇몇 메커니즘을 제공하고 있다.

여러 웹 애플리케이션 취약점 스캐너들은 징후 기반의 탐색 기능과 함께 공격자들이 사용하는 기법을 모방하여 웹 애플리케이션의 취약점을 스캔하는 기능(퍼지 시험과 유사)을 제공한다. 이 기법은 자동화된 정기적

애플리케이션 검사라고도 한다. 이 스캐너들은 사용자의 로그인 정보를 사용하여 액세스 정책이 적절하게 준수되고 있는가의 여부를 확인하는 기능부터 애플리케이션 파괴를 위해 유효하지 않은 입력이나 악성 입력을 애플리케이션에 제공하는 기능까지 다양한 기능을 제공할 수 있다. 서명 기반의 스캐너와 마찬가지로, 이러한 스캐너들은 알려져 있는 유형의 공격만을 탐지할 수 있으며 각 취약점들의 결합에 따라 커진 위험성을 고려하지 못할 수도 있다.

취약점 스캐너는 네트워크 또는 호스트 기반일 수 있다. 네트워크 기반의 스캐너는 네트워크를 통해 대상 애플리케이션에서 원격으로 실행되며, 호스트 기반의 스캐너는 대상 애플리케이션과 동일한 장비에 설치해야 한다. 호스트 기반의 스캐너는 일반적으로 보안 구성의 검증과 같은 더 정교한 분석을 시행할 수 있다. 탐색 과정은 자동이지만 도구의 거짓 양성 비율이 높은 경우가 많기 때문에, 담당자는 결과를 적절하게 해석하고 실제 취약점을 파악할 수 있을 정도로 충분한 애플리케이션 개발 및 보안 지식을 갖고 있어야 한다.

이 도구들은 바이러스 스캐너와 마찬가지로 담당자가 도구를 구매/입수했을 당시 공급업체에게 알려져 있었던 취약점 징후를 기반으로 하고 있다. 따라서 담당자는 꾸준히 취약점 징후 데이터베이스의 업데이트를 다운로드 하도록 주의를 기울여야 한다.

취약점 스캐너는 인수하거나 재사용한 바이너리 소프트웨어의 초기 보안 검사에서 상용 소프트웨어에 널리 퍼져 있는 일반적인 취약점을 파악하는데 가장 효과적으로 사용할 수 있다. 또한, 애플리케이션 침입 시험을 시행하기 전에 취약점 탐색을 시행하면 애플리케이션에 일반적인 취약점이 포함되어 있지 않음을 확인하여 이러한 취약점의 파악을 위한 침입 시험 시나리오를 생략할 수 있다.

※ 애플리케이션 취약성 자동 스캔에 활용 가능한 도구 목록은 부록 1. 소프트웨어의 취약점 점검 및 오류 검증 도구의 ‘소스코드 보안 취약성 점검 도구’ 참조

4. 사후 보안 분석

사후 보안 분석은 배포된 소프트웨어가 공격당한 후에 공격자가 악용한 소프트웨어의 기능이나 인터페이스의 취약점을 파악하는 데 사용된다. 배포 전의 취약점 시험과는 달리, 사후 보안 분석은 존재하거나 존재하지 않을 수 있는 오류를 찾는 것이 아니라 입증된 취약점을 파악하고 분석하는데 중점을 둔다. 사후 분석은 컴포넌트 내부, 컴포넌트들 사이, 컴포넌트 외부의 3가지 분석으로 구성된다.

컴포넌트 내부의 사후 분석은 악용된 취약점이 컴포넌트 자체 내에 있다고 의심되는 경우에 사용된다. 이러한 분석을 지원하는 도구는 취약점이 있는 지점을 지적할 수 있도록 컴포넌트의 행동양상 및 상태에 대한 정적 및 동적인 정보를 분석가에게 제공한다.

컴포넌트들 사이의 사후 분석은 취약점이 컴포넌트들 사이의 인터페이스에 있다고 의심되는 경우에 사용된다. 이러한 분석을 지원하는 도구는 컴포넌트들 간에 사용되는 통신 또는 프로그램 인터페이스 메커니즘 및 프로토콜에 대한 정보를 제공하며, 각 컴포넌트가 이러한 메커니즘/프로토콜을 도입하는 방식들 간의 차이점도 파악한다.

컴포넌트 외부의 사후 분석은 취약점이 전체 시스템의 행동양상이나 특정 환경에 있다고 의심되는 경우에 사용된다. 이러한 분석을 지원하는 도구는 소프트웨어 시스템 및 주변 환경의 감사, 이벤트 로그에 대한 정보를 제공한다. 시스템 단계의 보안 관련 행동양상에 대한 기록도 분석하여 공격자가 대상으로 한 소프트웨어의 구성 및 환경과의 상호작용에 대한 취약점을 파악한다.

5. 소프트웨어 취약점 시험 도구

소프트웨어 취약점 시험 도구는 담당자가 소프트웨어 프로그램의 다음과 같은 점들을 확인할 수 있도록 설계되어 있다.

(1) 철저하게 규격에 따라 설치되었는가의 여부 (지정된 모든 보안 기능의 실행 여부 포함)
(2) 올바르게 그리고 안전하게 작동하는가의 여부, 즉 예상되는 취약점을 대상으로 한 (공격자의) 악용이나 침해, 또는 알려지지 않은 취약점의 파악을 위한 공격자의 시도에 대한 내성

대부분의 취약점 시험 도구는 화이트 박스 코드 검토, 블랙 박스 취약점 스캔, 침입 시험, 보안 지향적인 오류 주입 등의 특정한 시험 기법이나 방식을 도입한다. 불행히도 소프트웨어 애플리케이션, 서비스, 프로그램의 보안을 자동으로, 그리고 포괄적으로 조사할 수 있는 단 하나의 도구는 존재하지 않는다. 현재의 상용 및 오픈 소스 취약점 조사, 보안 오류 주입, 코드 보안 검토 도구는 복잡하지 않은 소프트웨어 시스템, 소스 코드 파일, 실행 환경의 일반적인 기본 오류 중 그 일부분만을 찾아내서 더 나은 코딩 및 설치 절차에 대한 일반적인 조언을 제공할 뿐이다. 이러한 도구들은 인수 또는 재사용한 소프트웨어 및 다양한 컴포넌트 조립품 옵션의 초기 보안 조사, 그리고 전체 애플리케이션의 보안 조사에 가장 유용하게 활용할 수 있을 것이다.

일반적으로 취약점 시험 도구는 특정한 단계, 기법, 시험 방식만을 지원하도록 설계된다. 애플리케이션 단계의 소프트웨어 시험에 맞춰진 도구는 소프트웨어의 실행 환경, 미들웨어, 운영체제, 네트워킹 컴포넌트 등의 보안 취약점을 검증하는 도구로 보완되어야 한다. 이것은 특히 실행중인 소프트웨어의 동적인 보안 분석을 통해 확인된 소프트웨어 시스템 자체나 그 환경에서 발생한 보안상의 취약점을 파악하는 데 매우 중요하다.

상용 또는 오픈 소스로 널리 보급되어 있는 소프트웨어 및 애플리케이션 취약점 시험 도구의 주요 유형은 다음과 같다.

■ 컴파일러 오류 점검 도구

- 모델 확인, 이론 검증, 제약 기반 및 정적 분석 도구, 특성 기반의 시험 도구와 같은 공식적인 검증 도구
- 수동 또는 반자동식 코드 검토 지원 도구, 소스 코드 스캐너, 동적 코드 분석 도구
- 소스 코드 오류 주입기
- 바이트 코드 스캐너
- 역엔지니어링 도구 (역컴파일러, 역어셈블러, 바이너리 스캐너 등)
- 바이너리 오류 주입기
- 조작기(Fuzzer)
- 침입 시험 도구
- 실행 환경 스캐너 (네트워크, 운영체제, 데이터베이스, 웹 서버 스캐너 포함)
- 애플리케이션 스캐너
- 소스 코드 오류 주입 도구
- 완력 시험이기, 버퍼 오버런 탐지기, 입력 검증기 등의 기타 관련 도구

※ 소프트웨어 취약점 시험에 관계된 도구는 본 가이드의 ‘[부록 1] 제 1 절 소프트웨어의 취약점 점검 및 오류 검증 도구’를 참조 하십시오.

제 3 장 소프트웨어 개발과정의 정보보호

제 1 절 일반적 보안 원칙 권고(안)

본 절에서는 소프트웨어 개발자가 안전한 개발환경을 마련하기 위하여 준수가 필요한 일반적인 원칙을 제시한다.

1. 개발시스템의 소프트웨어 설치목록 관리

개발과 관련되지 않은 소프트웨어나 충분히 검증되지 않은 소프트웨어의 사용에 대해서도 적절한 보안 통제가 이루어져야 한다. 특히, Active-X나 플러그인과 같이 다른 응용 소프트웨어의 일부분으로 설치되는 경우에도 설치 소프트웨어 현황을 목록으로 만들어 지속적으로 모니터링하고 관리해야 한다.

2. 유해 소프트웨어로부터의 보호

개발시스템이 인터넷과 연결되어 있을 시, 개발과 관련되지 않은 인터넷 접속이나 서비스 활용을 자제하여 악성코드 감염이나 외부의 불법적인 침입 등으로부터 개발 시스템을 보호할 필요가 있다.
인터넷과 연결된 개발시스템은 컴퓨터 바이러스, 트로이 목마, 논리 폭탄, 네트워크 웜 등과 같은 유해 소프트웨어의 침입에 취약하다. 따라서, 개발자는 이러한 유해 소프트웨어에 대한 위험을 인식해야 하며, 개발과 관련되지 않은 인터넷접속의 제한이나 유해 소프트웨어를 탐지하고 제거하기 위한 적절한 통제 방안들을 도입하여야 한다.

3. 직무 분리 및 권한 없는 접근 차단

시스템에 대한 고의적 또는 우연한 사고의 발생 가능성을 줄일 수 있는 한 가지 관리 기법으로 직무 분리(segregation of duties)가 있다. 직무 분리는 특정 개발업무 수행에 관한 책임 및 권한을 한 사람에게 전적으로 맡기는 것이 아니고 개발업무를 여러 단계로 구분하여 접근을 통제하는 것이다. 개발 시스템에는 소스코드나 설계문서 등 민감한 자료가 존재하기 때문에 직무 분리나 접근통제, 암호화, 사용자 인증 등을 통해 사고를 예방해야 한다.

4. 개발시스템과 운영시스템의 분리

소프트웨어는 개발과 테스트 단계를 거쳐서 개발이 완성되고 그 후에 운영된다. 그런데 개발이나 테스트 활동들은 소프트웨어에 대해 수정을 가할 수 있기 때문에 고의적 또는 비고의적으로 시스템에 대한 심각한 문제를 일으킬 수 있다. 만일 권한없는 개발 인력이나 테스트 인력이 운영 시스템에 접근할 수 있다면 불법적 코드나 미검증 코드를 삽입하거나 운영 데이터를 변조할 수 있다. 또한, 운영시스템은 외부의 접근에 쉽게 노출될 수 있으므로 운영시스템에서 개발시 소스코드나 설계문서 등의 자료 유출 위험이 있다. 필요시 소스코드 및 문서 유출 방지 솔루션을 활용할 수 있다.

5. 시스템 및 네트워크 보안 소프트웨어 설치 운영

운영시스템과 마찬가지로 개발시스템에도 백신, 침입차단시스템, 침입탐지시스템 등의 네트워크 보안 소프트웨어를 설치하여 운영해야 한다. 특히 무선통신(무선랜, HSDPA, Wibro 등)에 의한 접근은 면밀히 모니터링하고 통제해야 한다.

6. 개발시스템 및 개발 도구 업데이트

개발시스템에 대한 보안패치는 각종 소프트웨어 및 운영체제 등에서 발견되는 보안상의 취약성을 보완해주는 것으로써 새로운 취약성에 대한 보안패치가 발표되는 즉시 시스템에 적용하여 보안조치를 취해야 하며, 개발 도구에 대한 패치(업데이트)도 고려함으로써 보안사고를 사전에 예방할 수 있도록 해야 한다.

7. 정기적인 자료 백업

사이버 공격이나 재난으로부터 개발관련 자료를 안전하게 보호하기 위하여, 별도의 시스템에 정기적으로 백업을 해야한다. 이동형 저장장치에 백업하는 경우, 이동형 저장장치에 대한 보호조치를 마련해야 한다.

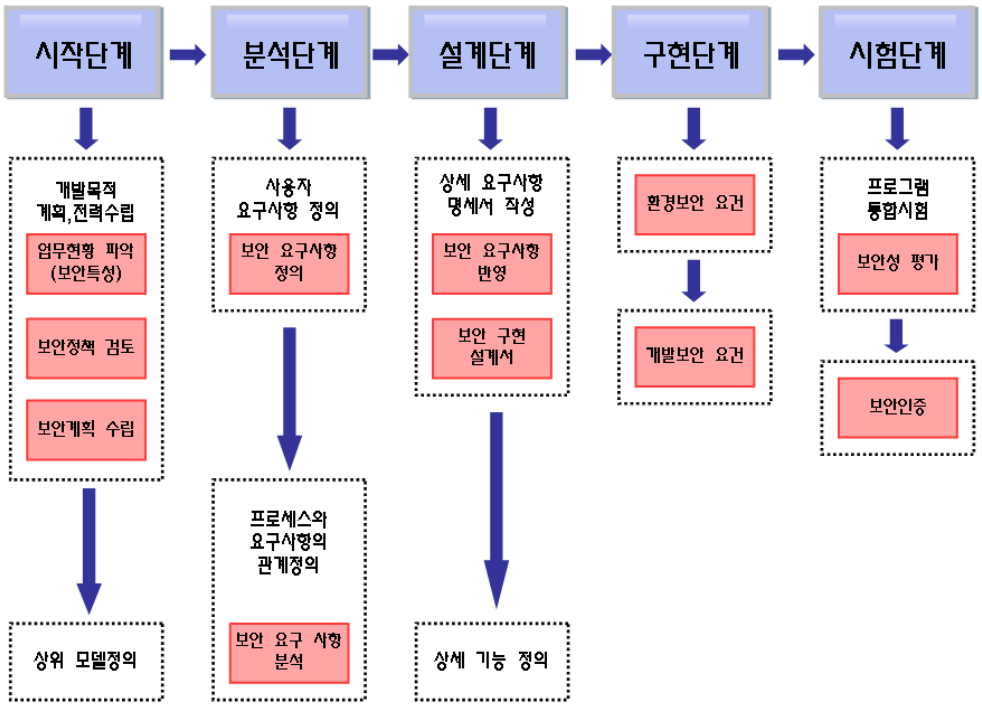
8. 개발시스템 저장정보의 안전한 파기

개발시스템의 매각이나 폐기를 위해 외부로 반출하는 경우, 저장장치를 분리하여 별도로 파기하거나 복구할 수 없도록 완전삭제를 해야 한다.

위와 같은 사항에 대해 개발자는 개발 시스템 및 네트워크 환경 등 개발환경에 대해 정기적으로 모니터링하고 예방조치를 취해야한다. 또한, 개발 시스템이 고도의 정보보호 기능을 가지고 있다고 하더라도 실제 개발업무에서는 허점이 발생할 수 있으므로 개발 시스템에 대한 관리절차의 문서화 등을 포함하는 적절한 관리체계가 병행되어야 한다.

제 2 절 소프트웨어 개발단계별 보안절차

본 절에서는 일반적인 소프트웨어 개발주기 과정의 시작, 분석, 설계, 구현, 시험 5단계에서 각 단계별 소프트웨어 개발활동에서 수행되어야 하는 보안 활동을 제시한다.



(그림 3-1) 주요 보안 활동이 포함된 애플리케이션 SDLC

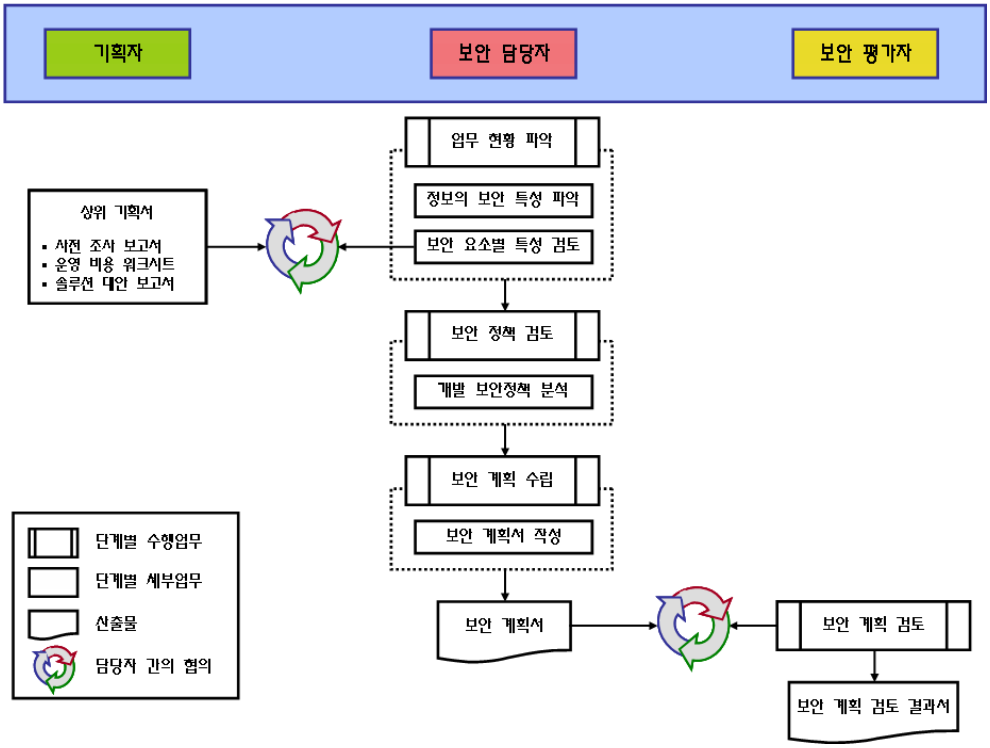
위 그림에서 붉은색 사각형은 SDLC 각 단계에 따른 일반 애플리케이션 소프트웨어 개발과정에서 요구되는 보안활동과 보안활동 수행 시 요구되는 자료를 나타낸다. (개발되는 애플리케이션 특성에 따라 요구되는 자료는 일부 달라질 수 있다). 안전한 소프트웨어 개발을 위해서는 보안 담당자는 일반적인 소프트웨어 개발과정에서 요구되는 자료 이외에 보안 프로세스 수행시 요구 자료들을 개발자 또는 프로젝트 기획자를 통해 산출하고 보안 평가자는 이에 대해 확인하여야 한다.

상기 각 단계에서 보안 활동들은 개발된 애플리케이션이 서비스(운영)되는 환경을 기준으로 정의되어야 한다. 즉 애플리케이션은 개발기관에서 바로 운영되거나 애플리케이션 개발을 발주한 기관에서 운영될 수 있으므로 실제 운영되는 환경을 기준으로 각 보안 활동들이 수행되어야 한다.

소프트웨어 개발의 각 단계별 보안활동이 수행되고 검토결과가 개발자에게 피드백되어 보완되면 해당 애플리케이션은 안전성을 확보할 수 있다.

1. 시작 단계

시작 단계에서는 개발되는 애플리케이션의 개발 목적을 정의하고 업무 현황 파악과 애플리케이션 개발과정에서 적용되어야하는 보안정책을 검토하여 위험분석 및 조직 전체에 적용되는 보안계획 등을 작성한다.



(그림 3-2) 시작단계의 보안 프로세스

가. 업무현황 파악

애플리케이션 개발이 시작되면, 보안 담당자는 개발되는 애플리케이션에서 운영될 업무에 대한 기본적인 현황을 파악하여 전산화되는 정보가 어느 정도의 보안 등급을 가지고 있는지 파악해야 한다. 애플리케이션 구축 단계에서 전산화될 정보의 보안 특성을 파악하지 않으면 애플리케이션 구축 시에

계획되지 않은 별도의 보안예산 소요, 사업 일정 지연, 관련기관의 통제 등 여러 가지 문제에 직면할 수 있으며, 전산화될 정보의 보안 특성에 따라 향후 단계에서 검토 되어야 할 보안 고려요소나 보안 대책에 차이가 발생할 수 있다.

보안 요소별 특성 검토는 보안 담당자가 애플리케이션 기획자와 함께 기획서를 기반으로 새로 개발되는 정보자산의 특성을 분석하여 각 정보자산의 등급을 구분한다. 각 정보 자산은 다음과 같이 기밀성, 무결성, 가용성 측면을 구분하여 현황을 파악한다.

[표 3-1] 보안요소별 보안등급 구분 (예시)

보안요소	보안 등급 구분
기밀성	*고객비 / 극비 / 비밀 / 대외비 / 일반정보*
무결성	*높음 / 중간 / 낮음*
가용성	*1분 / 1시간 / 1일 / 1주 / 1달*

나. 보안 정책 검토

보안 정책 검토는 조직의 보안 규정 및 보안 지침 등의 정보보호 관련 보안 정책을 검토하여 애플리케이션 개발에 적용할 수 있는 보안계획을 수립하기 위함이다.

보안 담당자는 보안조직 내 존재하는 People / Process / Technology 측면의 보안정책 문서 중에서 개발 시스템에 적용할 수 있는 보안정책을 발취하여 기술하며 정책에 대한 출처 및 세부 내용, 요구 수준들을 모두 분석하여 작성한다. 이 보안정책에 대한 분석 자료는 요구사항 반영 및 보안설계에도 사용된다.

[표 3-2] 보안정책 검토 형식 (예시)

보안정책 항목	존재 여부	요구 수준	세부 내용	보안정책 출처
개인정보 처리	*존재*	*필수*	*개인 정보에 대한 수집과 저장 폐기에 관한 지침*	*개인정보보호 지침*
…	…	…	…	…

다. 보안 계획 수립

보안 계획 수립은 개발 중인 애플리케이션에서 발생할 수 있는 위협 식별을 통해 조직의 환경에 적합한 위험관리 계획을 수립하여 시간 및 비용 효과적인 방법으로 보안대책을 적용하기 위한 것으로 보안 담당자가 새로이 개발되는 애플리케이션의 업무현황 파악문서 및 보안 정책 검토 자료를 기반으로 보안정책에 부합하도록 작성한다.

다음은 보안계획 수립시 검토되어야 할 주요 사항은 다음과 같다.

[표 3-3] 보안계획 수립시 주요 검토 사항 (예시)

분야	검토할 사항	검토방법
기능성	사용 및 관리 업무상의 보안 요구 사항들이 실제 구현 시 반영이 적용되지 않은 경우나 기대 이하로 적용된 경우	정보의 특성 및 보안정책 검토 결과에 따른 주요사항 확인
비용	프로젝트 보안 비용이 미 반영되어 보안 기능의 구현이나 적용이 어려운 경우	일반 개발비용 산정 사항 확인/ 검토
관리	프로젝트 보안 관리에서 부적합한 직무 분리에 의한 문제 및 통제 관리 부족으로 보안 문제가 발생하는 경우	업무정의, 직무분리, 통제 프로세스 고려사항 여부 확인
호환성	기존 시스템과의 보안 기능 호환성 고려 부족으로 통합 관리 및 확장 시 보	기존 업무환경의 보안관련 사항 및 호환성 검토

	안 관리체계 문제가 발생하는 경우	
신뢰성	애플리케이션의 자체의 보안 신뢰성이 부족하여 보안상 문제가 발생되는 경우	기밀성, 무결성, 가용성 기능 사항 반영에 대한 제한 사항 검토

o 보안계획서 작성

보안계획서는 보안 담당자에 의해 작성되어야 하고 보안 평가자에 의해 적정성이 평가되어야 한다. 보안계획서 작성에는 보안 목적과 정보의 기밀성, 무결성, 가용성, 기록성, 신뢰성을 기초로 한 보안 요구사항, 조직의 하부구조와 책임, 애플리케이션 개발 및 구매 시 보안 기능의 설치, 비상사태에 대한 대응책, 법적인 책임 등이 언급되어야 한다.

[표 3-4] 보안계획서 양식 (예시)

대상	*개인정보 DB*	**책임자**	*김보안*
관련 보안정책	*개인정보 처리*		
위협		**보안대책**	**소요예산**
주민번호 등 개인 정보 노출로 인한 개인정보 도용		*주민번호는 SEED로 암호화 하여 저장 하여야 한다.*	*100만원*
침해 시 영향범위		**보안사고 발생 시 대응책**	
이름, 주민번호, 이메일 노출		■ *개인 정보 노출 시 공지사항에 게시* ■ *개인 정보 노출 시 각 개인에게 메일을 이용하여 공지*	

라. 보안 계획 검토

보안 계획 검토는 보안 평가자에 의해서 수행되어야 하며 보안계획서 작성 시 고려해야 하는 사항에 대해 검토한다. 애플리케이션 보안계획은 애플리케이션과 제공되는 서비스, 정보 등의 보호를 위한 세부적인 규칙이며,

애플리케이션 개발 보안계획과 조직 전체의 보안계획이 조화를 이루어야 한다. 보안 평가자는 보안계획서가 [표 3-4] 보안계획 수립시 주요 검토 사항을 적절하게 반영하였는지 평가하여야 한다.

아래 표는 '보안 계획 검토결과서'로서 보안 평가자에 의해 보안 계획을 평가하는데 사용할 수 있는 항목 및 평가양식을 나타낸다.

[표 3-5] 보안 계획 검토 결과서 (예시)

분야	검토항목	검토된 내용	검토 결과
기능	정보의 특성	*극비 기밀성 요구*	*적절*
	보안 정책	*암호화된 DB 사용*	*부적절*
비용	비용 산정지침	*100만원 이내*	*초과*
관리	업무 정의	*개인정보 취급*	*적절*
	직무 분리	*개인정보 취급*	*적절*
	통제 프로세스	*개인정보 처리 지침*	*적절*
호환성	기본 시스템 환경 정보	-	*N/A*
	제안된 시스템 정보	-	*N/A*
	시스템 통합성	-	*N/A*
신뢰도	기능, 관리, 호환 등 고려	-	*N/A*

※ N/A : Not Applicable (적용 대상이 아닌 경우)

보안 담당자는 보안 평가자가 검토한 결과 '부적절'로 판정되는 항목에 대해서는 보안계획서를 보완하여 검토결과가 '적절'로 판정될 수 있도록 필요한 조치를 취해야 한다.

마. 시작 단계 보안활동에 대한 점검(Review)

시작 단계에서 수행되어야 하는 전체 보안활동에 대한 점검은 보안담당자, 보안 평가자에 의해 수행되고, 그 결과는 애플리케이션 기획자와 공유되어 기획서에 반영되어야 한다. 시작단계 전체 참여자에 의해 전체 보안활동에 대한 점검이 이루어지고 반영계획 등이 수립된 후에 분석단계로 이행되어야 시작단계의 안전성을 확보할 수 있다.

다음 표는 시작 단계 전체 보안활동에 대한 점검목록 및 점검 양식을 나타낸다.

[표 3-6] 시작단계 전체 보안활동에 대한 점검 목록 (예시)

Review 항목	중요도[1]	적용 여부	특이사항	보완계획 (일정)	비고
1. 업무 현황 파악					
- 보안 요소별 특성 검토	매우 중요	*X*			
2. 보안 정책 검토					
- 보안 정책	매우 중요	*N/A*	*해당되는 관련 보안정책 없음*		
3. 보안 계획 수립					
- 보안 계획서 작성	매우 중요	▲			
4. 보안 계획 검토					
- 보안 계획 검토	매우 중요	●			

(● : 보안 적용됨, ▲ : 부분 적용됨, X : 적용 안됨, N/A : 해당 사항 없음)

1) Review 항목에 대한 중요도는 각 기업 및 기관의 프로젝트에 따라 달라질 수 있다.

☞ 고려 사항(Tip)

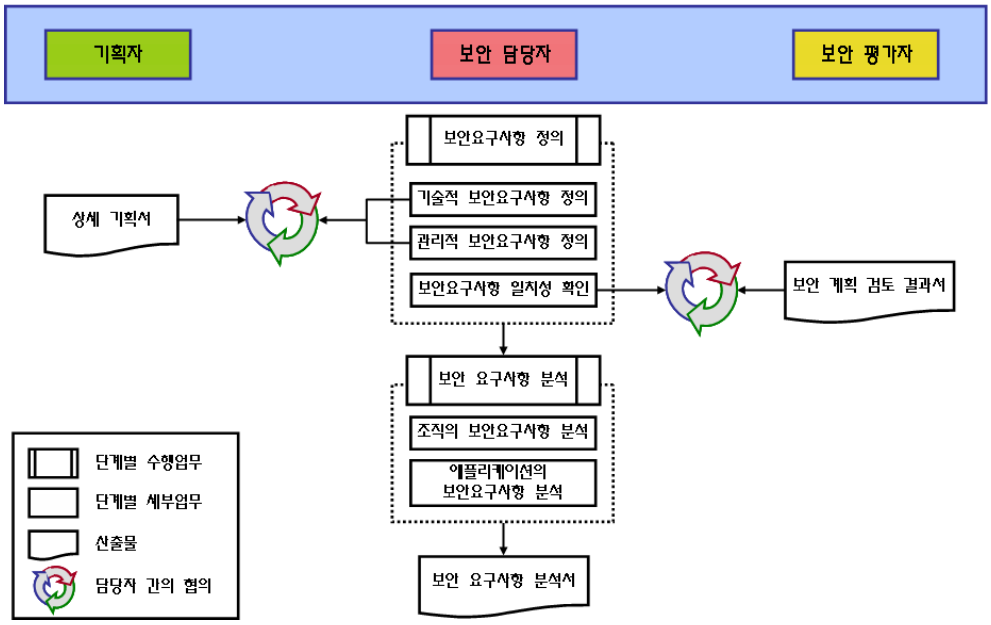
- 시작 단계에서는 실제 개발자들이 고려해야 할 사항은 적지만 보안 담당자가 업무 현황을 파악하기 위해 구축 되어야 할 전산 시스템의 보안 등급을 구분할 때 기획자의 의견과 개발자의 의견이 서로 조율되어야 차후 원활한 애플리케이션 개발진행이 가능하다.

- 정보보호 정책은 기본적으로 '정보통신망 이용촉진 및 정보보호 등에 관한 법률' 및 '개인정보보호 관련법' 에서 지정하는 법적 요건을 갖추도록 한다.

  ※ 정보보호 관련 법령, 지침 및 해설서 등의 자료 참조 사이트
  ⇒ http://www.moleg.go.kr/main/main.do (법제처, 국가법령정보센터)
  ⇒ http://www.kisa.or.kr (한국정보보호진흥원, 주요사업별 자료실)

2. 분석단계

분석 단계에서는 시작 단계의 진행사항을 기초로 구체적인 보안 요구사항을 분석하는 단계이다. 이 단계에서는 애플리케이션 사용자의 보안 요구 사항을 정의하고 개발 프로세스와 보안 요구사항과의 관계 분석을 수행하여야 한다.



(그림 3-3) 분석단계의 보안 프로세스

가. 보안요구사항 정의

보안요구사항 정의는 보안 담당자에 의해 수행되며 애플리케이션 기획자와 상세 기획서를 기반으로 개발되는 애플리케이션에 대해 기술적 측면, 관리적 측면으로 분류하여 보안 요구사항을 정의한다. 정의된 보안 요구사항에 대해 보안 평가자는 시작단계에서 수행된 '보안 계획 검토결과서' 내용을 기반으로 보안요구사항과 보안 계획간의 일치성을 확인하여야 한다.

o 기술적인 측면의 보안요구사항 정의

기술적인 측면의 보안 요구사항 정의는 애플리케이션이 서비스되는 환경에서 안전한 서비스 제공에 관계된 통신과 운영관리, 애플리케이션 보안(시스템 개발), 서버보안, 네트워크보안에 대한 보안 요구사항을 정의한다.

[표 3-7] 기술적인 측면의 보안 요구사항 (예시)

분야	요구사항 정의
통신과 운영관리	■ *http protocol을 이용 정보 전달* ■ *전송 데이터의 128bit SEED 암호화*
어플리케이션 보안	■ *사용자 입력 값 검증* ■ *파일 처리 검증* ■ *사용자 인증과 권한 관리* ■ *개인정보 중 주민번호 암호화 저장*
서버 보안	■ *최신 OS 설치* ■ *최신 보안 패치 유지*
네트워크 보안	■ *DMZ 구성* ■ *개발 네트워크 분리*

o 관리적인 측면 보안요구사항 정의

관리적인 측면의 보안 요구사항 정의는 애플리케이션이 서비스되는 환경에서 안전한 서비스 제공을 위한 보안정책, 보안조직, 인력보안, 자산분류와 통제, 준거성, 사업지속관리, 물리적 보안을 정의한다.

[표 3-8] 관리적인 측면의 보안 요구사항 (예시)

분야	요구사항 정의
보안 정책	■ *운영을 위해 추가적인 어플리케이션 보안 운영 지침 필요*

	■ *정책서를 모든 직원에 전달 필요* ■ *개인정보보호 지침에 따른 개인정보에 대한 보호*
보안 조직	■ *외주 계약의 보안 요건 적용 필요*
인력 보안	■ *보안을 포함한 직무 책임* ■ *정보보호 교육 및 훈련* ■ *보안 사고의 보고*
자산 분류와 통제	■ *모든 중요한 자산의 목록 및 등급 작성, 유지*
법적요건	■ *정보통신망 법에 의거하는 요구사항 준수*
사업지속관리	-
물리적 보안	■ *출입 카드 설치* ■ *보안 카메라 설치*

나. 보안 요구사항 분석

보안 요구사항 분석은 관리적, 기술적, 물리적 측면에서 정의된 보안 요구사항에 대해서 조직의 보안 요구사항 분석과 애플리케이션 보안 요구사항 분석으로 구분할 수 있다. 보안 요구사항 분석 절차는 조직의 보안 요구사항을 먼저 파악한 후, 이를 애플리케이션 보안 요구사항에 반영하는 것이 일반적이다.

o 조직의 보안 요구사항 분석

조직의 보안 요구사항 분석은 애플리케이션을 운영할 조직의 보안 요구사항으로서 시작단계에서 수행한 보안요소별 보안등급 분류, 보안정책 검토결과, 보안계획서의 내용이 기술적, 관리적 측면의 보안요구사항에 정확히 반영되었는지 분석하는 단계이다.

o 애플리케이션의 보안 요구사항 분석

애플리케이션 보안 요구사항 분석은 애플리케이션 구축을 통해 해당

조직의 목적 달성이나 소관 업무의 전산화시에 요구되는 보안사항을 분석하는 것이다. 조직의 보안 요구사항 범위 내에서 애플리케이션의 이용자, 운영자, 처리 데이터의 종류, 운영환경 등을 고려하여 애플리케이션 구현 시에 적용되어야 하는 보안사항을 분석한다.

조직의 보안요구 사항을 분석한 후 이를 토대로 설계단계에서 보안 요구사항을 반영하여 설계하고 만약 요구사항에 대한 설계단계에서의 미반영 요건이 있을 경우 미반영 사유를 보안 담당자에게 통보 및 필요시 미반영 요구사항의 대체 안을 제시한다.

다. 분석 단계 보안활동에 대한 점검(Review)

개발 되는 애플리케이션에 대해 기술적, 관리적, 물리적 측면에 대한 조직의 보안 요구 사항을 분석하여 보안 요구사항 분석서를 작성한다.

분석 단계에서 수행되어야 하는 전체 보안활동에 대한 점검은 보안담당자, 보안 평가자에 의해 수행되고, 그 결과는 애플리케이션 기획자 및 개발자와 공유되어 설계단계에서 반영되어야 한다. 분석단계 전체 참여자에 의해 전체 보안활동에 대한 점검이 이루어지고 반영계획 등이 수립된 후에 설계단계로 이행되어야 설계단계의 안전성을 확보할 수 있다.

다음 표는 보안 담당자 및 보안 평가자에 의해 수행되어야하는 분석단계 전체 보안활동에 대한 점검목록 및 점검 양식을 나타낸다.

[표 3-9] 분석단계 전체 보안활동에 대한 점검 목록 (예시)

Review 항목	중요도	적용 여부	특이사항	보완계획 (일정)	비고
1. 보안 요구사항 정의					
- 통신과 운영 관리	매우 중요	…	…	…	…
- 어플리케이션 보안	매우 중요				
- 서버 보안	매우 중요	X			
- 네트워크 보안	매우 중요				
- 보안 정책	매우 중요	N/A			
- 보안 조직	중요	▲			
- 인력 보안	매우 중요				
- 자산 분류와 통제	매우 중요	●			
- 법적요건	매우 중요				
- 사업 지속 관리	중요				
- 물리적 보안	중요				
2. 보안 요구사항 분석					
- 조직의 보안 요구사항 분석	매우 중요				
- 애플리케이션의 보안 요구사항 분석	매우 중요				

(● : 보안 적용됨, ▲ : 부분 적용됨, X : 적용 안됨, N/A : 해당 사항 없음)

☞ 고려 사항(Tip)

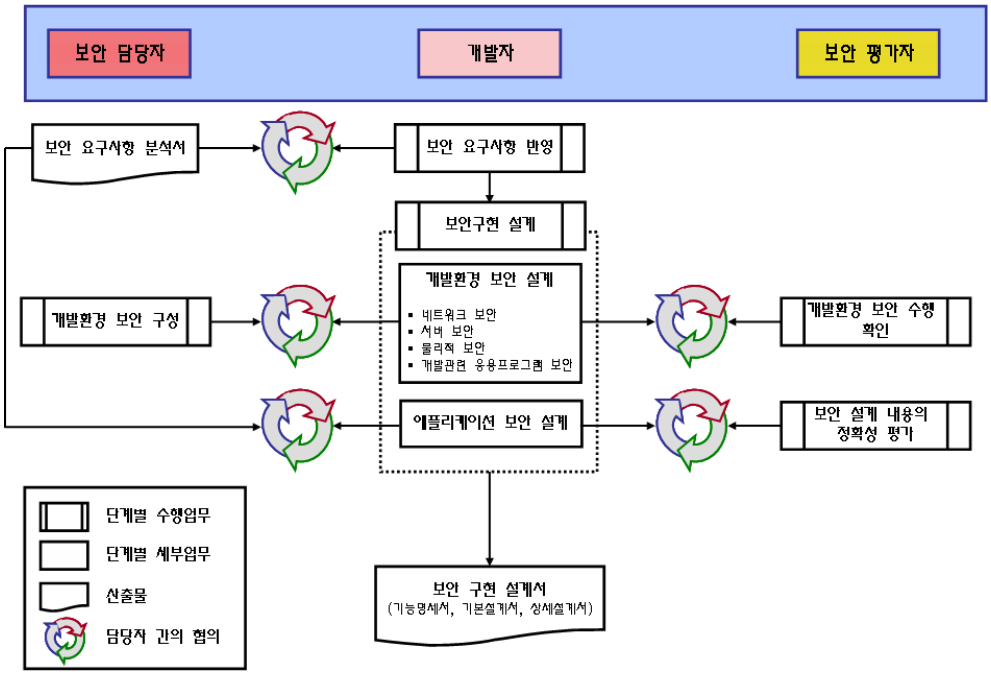
- 개발자 개발되는 애플리케이션의 모든 동작 범위에 대해 보안 담당자들에게 전달해야 보안 담당자들이 기술적인 측면의 보안 정의 사항에 대해 정확한 작성이 가능하다.

- 분석 단계의 보안 요구사항 분석서가 향후 개발 단계에서 작성 되는 산출물의 기초 자료가 되므로 빠짐없이 작성 되어야 한다.

■ 물리적인 보안 또한 보안의 중요 범주에 들어가므로 이에 대한 보안 요건도 보안 요구사항 분석시 반드시 고려하여야 한다.

3. 설계 단계

애플리케이션 설계 단계에서는 분석 단계에서 파악한 보안 요구사항 분석서의 요구사항이 반영되며, 개발위험과 통제를 고려하여 보안구현 설계서(기능명세서, 기본설계서, 상세설계서)를 작성하여 애플리케이션 설계에 반영하여야 한다.



(그림 3-4) 설계 단계의 보안 프로세스

가. 보안 요구사항 반영

애플리케이션 개발자는 애플리케이션 분석단계에서 정의된 보안 요구사항 분석을 통해 나타난 항목을 검토 후, 다음의 사항을 고려하여 설계에 적절히 반영하여야 한다.

- 보안 요구사항은 보안 계획이나 표준 등에 적합하여야 한다.
- 애플리케이션 개발자는 먼저 보안 요구사항들을 검토하여 설계 가능성

여부를 판단하여야 한다.

- 애플리케이션 개발자는 보안 요구사항을 분석하여 설계에 적절히 반영하여야 한다.

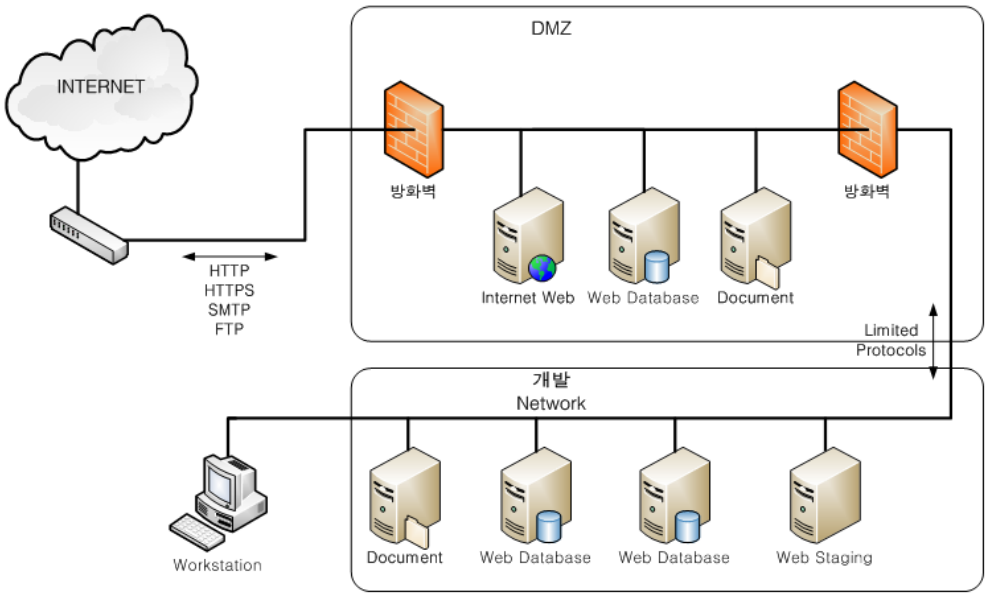
나. 보안구현 설계

보안구현 설계서는 보안 요구사항 분석서를 파악하여 개발환경에 대한 환경 보안과 애플리케이션 보안 설계에 대하여 정의한다.

o 개발환경 보안

개발환경 보안요건은 개발 환경에 대한 네트워크 및 서버의 보안사항에 대하여 정의한다. 또한 서버 및 DB의 데이터 백업 및 복원에 대한 보안, 접근통제 관련 물리적 보안 내용에 대하여 정의한다.

▷ 네트워크 보안



(그림 3-5) 네트워크 구성도 (예시)

개발환경보안 요건 중 네트워크 보안은 외부 사이버 공격으로부터 개발환경 보호를 위해 그림과 같이 개발 네트워크와 DMZ 네트워크를 구분하고 방화벽을 설치하는 등의 네트워크 보안환경을 구성하고 개발과 운영에 필요한 네트워크 접근경로를 구성하여야 한다. 개발환경의 네트워크 보안환경은 보안 담당자가 네트워크 관리자, 애플리케이션 기획자 및 개발자와 함께 구성하여야 하며 보안 평가자는 네트워크 보안에 대해 적정성을 평가하여야 한다.

다음 표는 개발환경 네트워크 보안을 위해 DMZ 구간의 방화벽에서 개발자원에 접근하는 주체별 접근통제를 위한 접근통제 구성 방식을 나타낸다.

[표 3-10] DMZ 방화벽의 네트워크 접근통제 구성 (예시)

Source	Destination	Service	OPEN 기간	목적
127.0.0.X (개발실)	*신규 App. IP*	*http*	*~ 1개월*	*웹 개발*
		Terminal service	*~ 1개월*	*개발 작업*
		Netbios	*~ 1개월*	*파일 공유*
10.10.10.X (외부개발업체)	*신규 App. IP*	*Ftp*	*~ 유지보수 기간 동안*	*콘텐츠 업로드*
신규 App. IP	*DB IP*	*SQL NET*	*서비스 기간*	*DB Query*
Any	*신규 App. IP*	*http*	*~ 1개월*	*애플리케이션 시험 서비스*

▷ 서버 보안

개발/운영시 외부 사이버 공격으로부터 개발 관련 서버 보호를 위해 필요한 서버의 네트워크 서비스와 접근계정에 대한 통제를 수행하여야 한다. 서버 보안환경은 보안 담당자가 서버 관리자, 애플리케이션 기획자 및

개발자와 함께 구성하고 보안 평가자는 안전하게 서버보안이 수행되는지 확인하여야 한다.

다음 표는 개발환경 서버 보안을 위해 Windows 환경의 개발 관련 서버에서 수행되어야 하는 주요 서버보안 항목을 나타낸다.

[표 3-11] 개발 관련 주요 서버보안 항목 (예시)

■ 애플리케이션 서버로 기본적인 보안이 강화된 Windows 2000 Advanced Server 사용여부 ■ 애플리케이션 서버가 SP 2와 보안 Hotfix에 대해 최신 버전까지 적용되었는지 여부 ■ 애플리케이션 서버에 대해 향후 통합보안관제 시스템과 연결한 시스템 및 보안 이벤트에 대한 통합 보안 관리 적용 계획 여부 ■ 개발시에 외부망에 개방해야 하는 서비스를 서버에서 접근통제를 실시하는지 여부 등

서버 보안 관련 네트워크 서비스와 접근계정에 대한 통제를 통해 책임추적 및 문제분석을 통한 개발자료 보호를 위해 보안 담당자는 다음과 같은 서버 보안 관리표를 작성하여 관리하고 주기적인 감사를 통해 계획된 바와 같이 적합한 서비스, 계정이 운영되는지 확인하여야 한다.

[표 3-12] 개발 관련 서버의 네트워크 서비스 관리표 (예시)

서버명	신규 Web 서버	
Service	사용 기간	목적
http	*2008.8 - 10*	*웹 개발*
Terminal service	*2008.8 - 10*	*개발 작업*
Netbios	*2008.8 - 10*	*파일 공유*
Ftp	*2008.8 - 10*	*콘텐츠 업로드*
http	*2008. 11 ~*	*애플리케이션 시험 서비스*

[표 3-13] 개발 관련 서버의 접근 계정 관리표 (예시)

서버 대상명	신규 Web 서버		
계정/권한	사용자 명	OPEN 기간	목적
Security / administrator	*홍길동 대리*	*2008.8-10*	*보안 관리*
Develop / administrator	*김한국 과장*	*2008.8-10*	*개발 작업*

▷ 물리적 보안

물리적 보안은 개발/운영시 개발 관련 네트워크 장비 및 서버가 위치 물리적 공간에 출입하는 인력에 대한 통제를 수행하는 것으로 애플리케이션 개발 조직의 보안 정책과 연계하여 관리적·기술적 통제를 수행하여야 한다. 물리적 보안환경은 보안 담당자가 애플리케이션 기획자 및 개발환경 관리자와 함께 구성하고 보안 평가자는 안전하게 물리적 보안이 수행되는지 확인하여야 한다.

다음 표는 개발환경 물리적 보안을 위해 필요한 주요 보안 항목을 나타낸다.

[표 3-14] 개발 관련 주요 물리적 보안 항목 (예시)

■ 개발 관련 인력(파견 인력 포함)들이 어플리케이션 개발을 지정된 프로젝트 공간에서 진행하는지 여부 ■ 개발 프로젝트 공간에 대한 카드키 출입통제 여부 ■ 출입 구역대해 폐쇄회로 등을 통한 모니터링 및 오비 방문자 기록 여부 등

▷ 개발관련 응용 프로그램 보안

개발 관련 응용 프로그램 보안은 애플리케이션 개발시 허가되지 않은 프로그램 구현 통제, 허가되지 않은 시스템 접근 통제, 개발 단계와 운영단계의 구분 등에 대한 통제 사항을 정의하여 기술한다. 개발관련 응용 프로그램 보안 통제항목은 보안 담당자가 개발자에게 각 항목에 대하여 설명하고 이에 대한 통제를 실시하여야 한다.

다음 표는 개발관련 응용 프로그램 보안을 위해 필요한 주요 보안 항목을 나타낸다.

[표 3-15] 개발 관련 주요 응용 프로그램통제 보안 항목 (예시)

○ 허가되지 않은 프로그램의 통제

- 개발 단계에서 개발자나 관리자에 의해 허가되지 않은 프로그램(트랩도어, 트로이 목마, 바이러스, 백도어 등)의 삽입에 대한 통제
- 외부에서 입력되는 모든 프로그램은 바이러스 검사
- 애플리케이션 내에 무결성 체크 프로그램을 설치하여 허가되지 않은 데이터나 부정확한 데이터의 입력 체크
- 개발된 애플리케이션은 애플리케이션 개발에 참여하지 않은 관리자와 기술요원에 의한 시험

○ 허가되지 않은 시스템 접근 통제

- 애플리케이션 개발자는 사용이 허가된 유틸리티를 이용하여 서버에 접근 하도록 통제
- 허가된 개발자만이 시스템에 접근할 수 있도록 접근통제 기능을 시스템 내 구현
- 시스템에 대한 모든 접근시도나 접근 후에 행해진 모든 활동에 대한 기록(Logging)

○ 개발 단계와 운영 단계의 명확한 구분

- 개발 단계에서 사용되는 시스템과 데이터는 운영 단계에서 사용되지 않도록 명확히 구분
- 애플리케이션 개발요원과 운영 요원 사이에는 명확한 임무 분리
- 개발된 애플리케이션을 시험하거나 운영 단계로 이동할 때 개발요원 배제.(특수한 경우 참여 가능)

o 애플리케이션 보안 설계

애플리케이션 보안 설계는 분석 단계에서 수행된 보안 요구사항 분석서의 내용을 애플리케이션의 상세설계 명세서에 반영하는 것이다. 보안 담당자는 애플리케이션 개발자와 협의를 통해 보안 요구사항 분석서의 내용을 최적화하여 상세설계 명세서에 반영하여야 하며 보안 평가자는 상세 설계 명세서에 반영된 보안 설계 내용의 정확성을 평가하여야 한다.

애플리케이션 보안은 기밀성, 무결성, 가용성, 책임추적성 측면에서 설계되어야 하며 설계시 비용 대 위험간의 상쇄효과를 고려해야 한다. 즉, 위험을 감소시키기 위해서는 정보보호 구현비용이 초래되고, 반면 위험을 방치하면 또한 조직에 손실을 미칠 수 있는 가능성이 있기 때문에 애플리케이션의 이용자, 목적, 다루는 정보 등에 기반하여 적절한 수준에서 위험을 감소시킬 수 있도록 설계에 반영하여야 한다.

다음은 애플리케이션 보안 설계시 기밀성, 무결성, 가용성, 책임 추적성 각 측면에서 설계에 반영되어야 하는 일반적인 보안 항목을 나타낸다.

[표 3-16] 애플리케이션 보안 설계시 반영 항목 (예시)

○ 기밀성을 위한 설계
■ 사용자 컴퓨터 시스템 내에서의 침해에 대응한 설계
- 암호화, 쿠키 정보의 암호화, 메모리 저장
■ 개방 네트워크로의 전송 중 침해에 대응한 설계
- 암호화, SSL over Web 접속, link 암호화 장비
■ 저장 장치 내에서의 침해에 대응한 설계
- DB내 접근제어 정책 구현
- 특별한 구성관리 요구사항 요구

○ 무결성을 위한 설계
■ 사용자 컴퓨터 시스템 내에서의 변경에 대응한 설계
- Integrity check into the back-end processing
- 전자서명
■ 개방 네트워크로의 전송 중 침해에 대응한 설계
- 암호화(전자서명), 체크섬(CRC check, MIC) 등
■ 저장 장치 내에서의 변경에 대응한 설계
- 강력한 평가 메커니즘, 접근제어 정책
- 감사로그 관리
- 암호 체크섬(변경가능성이 낮은 데이터는 MIC 사용)

○ 가용성을 위한 설계
■ 내부 네트워크(LAN) 가용성을 위한 설계
- 내부 DNS의 취약성 존재 고려
- 하드웨어 구성상(예: 내부라우터의 구성상의 오류)의 문제 고려
■ 저장 장치 가용성을 위한 설계
- Clustered System
- 데이터 백업

○ 책임 추적성을 위한 설계
■ 식별 및 인증(I&A): 사전적 대책
- 사용자 수, 사용자에 대한 통제, 정보의 민감도 등 고려
- 지식 기반, 소유물 기반, 신체 기반 접근통제 방식
■ 감사(Audit): 사후적 대책
- 어떤 로그 정보를 어떻게 저장할 것인가 결정
- 로그 정보에 대한 보호

보안 담당자는 애플리케이션 기획자 및 개발자가 작성하는 기능명세서, 기본 설계서, 상세설계서에 대해 상기 언급된 일반적인 보안 항목들을 분석하고 관련자와 협의하여 보안항목들을 설계에 반영하여야 한다.

보안 담당자 및 개발자는 보안을 고려한 애플리케이션 기능명세서를 다음과 같은 방식으로 작성한다.

- 요구사항 분석 단계에서 식별된 모든 시스템 정보보호 요구사항을 만족하는 정보보호 기능을 포함한 기능 명세서를 작성한다.
- 모든 정보보호 기능 및 외부 인터페이스의 목적, 방법, 효과, 예외사항, 오류메시지 등을 문서화한다.

보안 담당자 및 개발자는 보안을 고려한 애플리케이션 기본설계서를 다음과 같은 방식으로 작성한다.

- 기능명세서에 포함된 모든 정보보호 기능을 포함한 기본 설계서를 작성한다.
- 정보보호 기능을 구성하는 모든 하위시스템에 대하여 그들이 제공하는 정보보호 기능을 문서화한다. 또한 정보보호 하위시스템과 비정보보호 하위시스템의 구분 기준을 문서화한다.
- 정보보호 하위시스템의 모든 구성 요소를 하드웨어, 펌웨어, 소프트웨어의 구성 계층과 요소별로 문서화한다.
- 정보보호 하위시스템의 모든 인터페이스의 목적, 방법, 효과, 예외사항, 오류메시지 등을 문서화한다.

보안 담당자 및 개발자는 보안을 고려한 애플리케이션 상세설계서를 다음과 같은 방식으로 작성한다.

- 기본 설계서에 포함된 모든 정보보호 하위시스템의 구성 모듈을 포함하는 상세 설계서를 문서화한다.
- 모든 정보보호 모듈의 목적, 방법, 내부 및 외부 인터페이스, 효과, 예외사항 및 오류 메시지를 문서화한다.

- 정보보호 기능이 제공되는 방법을 정보보호 기능과 모듈간의 상호관계 및 종속관계를 정의함으로써 문서화한다.
- 핵심 정보보호 기능을 수행하는 모듈과 기타 모듈을 구분한다.
- 접근 통제 기능이 모든 자원을 통제하고 독립적으로 운영되는지 확인하고 문서화한다. 또한 접근 통제 기능은 분석이 용이하도록 단순하게 설계한다.

보안 담당자는 애플리케이션 기획자 및 개발자와 협의를 통해 기능명세서, 기본설계서, 상세설계서에 반영된 보안사항들이 적절하게 반영되었고 동작하는지 확인할 수 있도록 애플리케이션 시험계획서 등에 보안기능 시험계획을 반영하여야 하며, 보안 평가자는 기능명세서, 기본설계서, 상세설계서에 상기 사항들이 적절히 반영되었는지 타당성, 적절성, 명료성 등을 평가하여 적절치 않을 경우 피드백을 제공하여 보완되도록 해야한다.

다. 설계단계 보안활동에 대한 점검(Review)

설계단계의 보안 활동은 분석단계에서 산출된 보안 요구사항 분석서를 기반으로 애플리케이션을 안전하게 개발하기 위한 네트워크, 서버 등의 개발환경 보안과 보안 요구사항을 반영한 애플리케이션보안설계를 수행하는 것이다. 설계단계 전체에 대한 보안활동에 대한 점검이 이루어지고 반영계획 등이 수립된 후에 구현단계로 이행되어야 구현단계의 안전성을 확보할 수 있다.

다음 표는 보안 담당자 및 보안 평가자에 의해 수행되어야하는 설계단계 전체 보안활동에 대한 점검목록 및 점검 양식을 나타낸다.

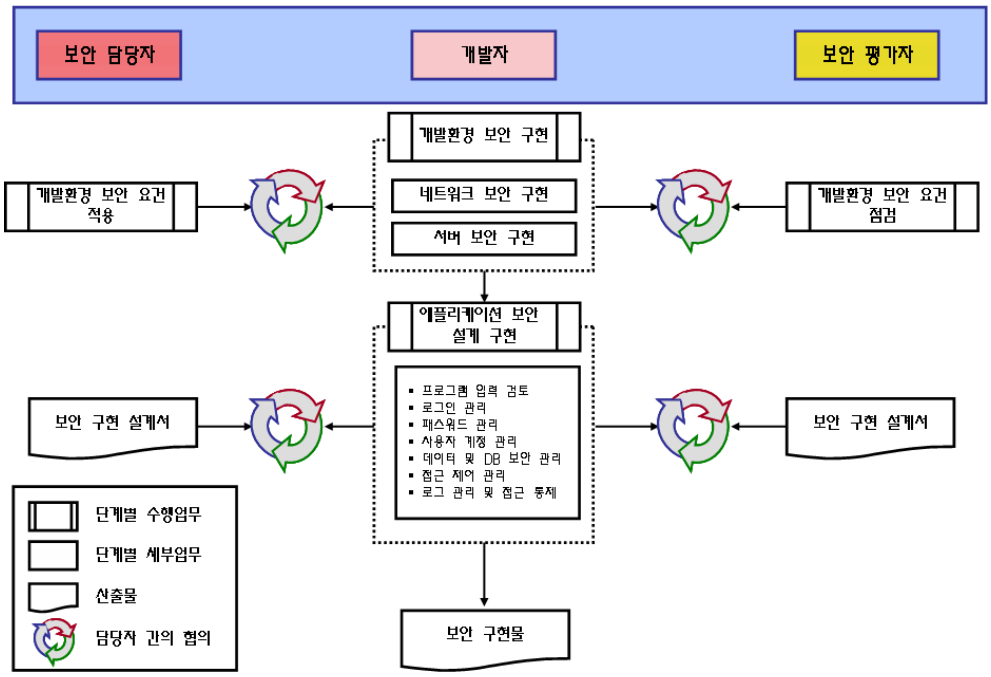
[표 3-17] 설계단계 전체 보안활동에 대한 점검목록 (예시)

Review 항목	중요도	적용 여부	특이사항	보완계획 (일정)	비고
1. 개발환경 보안 (네트워크 보안)					
- 내부망, 외부망, DMZ 분리구성	매우 중요				
- DMZ 방화벽 접근통제 구성	매우 중요				
2. 개발환경 보안 (서버 보안)					
- 네트워크 서비스 관리표 작성·운영	중요				
- 서버 접근계정 관리표 작성·운영	보통				
3. 개발환경 보안 (물리적 보안)					
- 물리적 보안항목 준수여부	보통				
4. 개발환경 보안 (개발관련 응용프로그램 보안)					
- 허가되지 않은 프로그램의 통제	중요				
- 허가되지 않은 시스템 접근통제	중요				
- 개발 단계와 운영 단계의 명확한 구분	중요				
5. 애플리케이션 보안 설계					
- 기능명세서에 보안 적용 설계	매우 중요				
- 기본설계서에 보안 적용 설계	매우 중요				
- 상세설계서에 보안 적용 설계	매우 중요				
- 시험계획에 보안기능 시험 반영	매우 중요				

(● : 보안 적용됨, ▲ : 부분 적용됨, X : 적용 안됨, N/A : 해당 사항 없음)

4. 구현 단계

어플리케이션 개발단계 중 보안 구현단계는 설계단계의 개발환경 보안 계획에 따라 환경을 구성하고 보안구현 설계서의 내용에 따라 실제 구현시 보안 요구 사항들을 정의하고 세부적인 활동 내역을 제시한다.



(그림 3-6) 구현단계의 보안 프로세스

가. 개발환경 보안 구현

개발환경 보안 구현은 설계단계의 개발환경 보안 요건들을 개발시에 실제 구성하여 운영하는 것으로 보안 담당자 및 보안 평가자는 계획된 사항들이 정확히 반영되었는지 점검하고 확인하여야 한다.

보안 담당자 및 보안 평가자는 설계단계에서 계획된 개발환경 보안 관련 네트워크, 서버 보안 구현에 대해 다음 항목들을 중심으로 점검하여야 한다.

o 네트워크 보안

▷ 네트워크 정책 운영

- 유·무선/DMZ/개발망/운영망 등의 서비스 경계영역 구분
- 네트워크 구성상의 Inbound 및 Outbound 관리 정책
- 사용하지 않는 네트워크 포트에 대한 Deny 정책
- 정책의 등록/변경/삭제에 대한 History 관리 정책

▷ 네트워크 분리 및 DMZ(De Military Zone) 구성

- 서비스 망과 개발 망에 대한 명확한 분리 구성
- 서비스 망에 대한 DMZ 구성

▷ 침입탐지/침입차단 시스템 적용 및 보안관리

- 침입탐지 및 침입차단 시스템을 통한 접근통제 및 모니터링
- 침입탐지 및 침입차단 시스템 접근통제 및 모니터링 Rule 관리

▷ DB Zone 분리

- 운영 및 개발 서버 Zone 과 분리된 DB Zone 구성
- DB Zone에 대한 접근통제

▷ 전송 데이터 보호

- 애플리케이션의 중요 정보에 대한 암호화(SSL, VPN 등 활용)

o 서버 보안

▷ 네트워크 서비스 보안 관리

- 개발서버에서 개발을 위해 허용된 서비스에 대해서만 Open
- 서비스 중인 서버는 서비스 제공에 필요한 Port만 Open

▷ 기본 시스템 보안 설정 및 보안패치

- Banner등 외부로 시스템 설정상황이 노출되는 것을 방지

- 시스템의 불필요한 권한 설정을 제거
- 최신 시스템 취약점에 대한 대응조치 등의 시스템 Hardening
- 시스템 OS 및 서비스 소프트웨어에 대한 최신 보안패치 적용

▷ 사용자 계정 관리

- 개발에 관련된 계정 이외의 사용자에 대한 통제
- 소프트웨어 설치 시에 사용되는 Default 계정 및 Guest 계정 삭제

▷ 악성코드 차단

- 바이러스, 웜 등의 악성코드 제거를 위한 백신 소프트웨어 설치
- 백신 소프트웨어를 통한 실시간 감시

나. 애플리케이션 보안 설계 구현

애플리케이션 보안 설계 구현은 설계단계의 기능명세서, 기본설계서, 상세설계서 등에 반영된 보안 요구사항 들을 애플리케이션 개발자가 직접 구현하는 것으로 보안 담당자 및 보안 평가자는 설계된 내용들이 정확히 구현되었는지와 소스코드 등 구현물 내에 취약점 등에 대해 개발자의 구현물을 점검하고 확인하여야 한다.

※ 구현물(프로그램 소스코드 등)의 취약점 점검에 활용 가능한 도구들은 '부록 1. 소프트웨어의 취약점 점검 및 오류 검증 도구' 를 참조하십시오.

보안 담당자 및 보안 평가자는 구현물 점검시에 애플리케이션 보안 설계 내용 이외에 아래 항목들을 추가로 점검하여야 하며 개발자에게 구체적인 구현 방법 등을 알려줄 수 있어야 한다.

o 프로그램 입력 검토

▷ 프로그램 버그에 의한 경쟁상태 발생 가능성 검토

- 공격자가 구동한 악성 프로그램이 프로그램 버그를 악용한 경쟁 상태를 유발하여 시스템 권한(Set-User ID가 붙은 경우 Root, Bin등…)을 획득하는 상황 방지

▷ 소스코드 내 민감한 정보 삽입

- 개발자는 소스코드에 ID 또는 패스워드, 시스템 정보, DB정보 등이 포함 되거나 주석으로 포함되지 않도록 처리

▷ 소스코드 내 악성코드 삽입

- 프로그램 개발 시 보안 정책을 우회할 수 있거나 프로그램 자체 기능에 위해를 가할 수 있는 코드, 개인 정보의 유출코드, Covert Channel등을 생성 할 수 있는 코드, 백도어/트랩 도어 등의 코드 삽입방지

▷ 실행 결과의 예외 처리

- 예기치 않는 입력값에 대한 정확한 예외상황 처리

o 로그인 관리

▷ 로그인 실패 시 원인 표기 제한

- 사용자의 로그인 실패 시 로그인 실패 이유를 설명없이 세션을 종료하거나 재입력 대기

▷ 잘못된 패스워드 입력 회수 제한

- 일정 횟수 이상 패스워드를 잘못 입력할 경우 일시적인 사용 중지 또는 세션 종료

▷ 최근 로그인 정보 표시

- 사용자가 로그인시 가장 최근에 성공적으로 로그인한 날짜, 시각 등의 정보를 화면에 표시

▷ 일정시간 경과시 자동 로그오프

- 사용자로부터 일정시간 동안 입력이 없을 경우 자동 로그오프 또는 세션 종료 처리

▷ 로그인 상태 유지시 세션 방식 적용

- 웹 애플리케이션의 경우 브라우저에 저장된 Cookie 방식보다는 서버 측에 일부 정보를 저장하여 상호 대조할 수 있는 세션(Session) 방식 사용

o 패스워드 관리

▷ 패스워드 최소 길이 제한

- 패스워드의 최소 길이를 통제할 수 있도록 최소 패스워드 길이값 정의

▷ 추측 가능한 패스워드 사용 통제

다음과 같은 추측 가능한 패스워드가 사용되지 못하도록 한다.

- 사용자 개인 정보(주민등록번호, 휴대폰 번호 등)와 관련된 패스워드
- 사용자 ID와 동일한 패스워드
- 동일한 문자 또는 숫자가 반복되는 패스워드
- 문자 또는 숫자만으로 구성된 패스워드

▷ 패스워드 변경 및 확인시 보안 적용

- 패스워드 변경시 현재 패스워드를 확인한 후 변경 사항 적용

▷ 패스워드 파일 접근통제 및 암호화 저장 여부 진단

- 애플리케이션에 저장된 패스워드 파일은 관리자도 알아볼 수 없도록 암호화된 형태로 저장

▷ 이전 부여 패스워드 일정기간 재사용 통제

- 애플리케이션의 인증시스템은 관리자가 이전에 사용했던 패스워드를 일정기간 동안 재사용하지 못하도록 하는 기능 구현

▷ 패스워드 사용기간 설정

- 패스워드를 사용할 수 있는 최소기간을 설정하여 일정 기간이 지나면 사용자로 하여금 강제로 패스워드를 변경토록 구현

▷ 인증(O.T.P, 생체인증, PKI)의 강화 작용

- 전자상거래, 금융관련 서비스 등 특별하게 인증의 강화가 필요할 경우 O.T.P, 생체인증, PKI등의 강화된 인증 기능 적용

o 사용자 계정 관리

▷ 추측 가능 및 Default 계정 사용 통제

- 애플리케이션 관리자 계정의 경우 “admin”, “root”, “administrator” 등 추측 가능하거나 Default 계정을 사용하지 않도록 구현

▷ 불필요한 계정에 대한 주기적 검토 및 삭제

- 일정기간 동안 사용하지 않는 계정에 대한 일시 사용중지 및 사용중지 된 후 휴면 계정에 대한 삭제 조치

o 데이터 및 DB 보안 관리

▷ 주요 데이터 및 자료 파일 등의 DB 저장 유무

■ 주요 데이터를 접근통제가 수행되는 DB에 저장

▷ 데이터의 보안등급 분류 설정

■ 애플리케이션에서 사용되는 데이터에 대한 보안등급 설정 및 보안등급에 따른 보안대책 수준 구현

▷ 클라이언트와 서버간의 데이터 암호화 전송 여부

■ 애플리케이션의 서버와 사용자간에 사용되는 연결 및 전송되는 자료의 보안성 강화를 위해 필요시 암호화 전송 구현

▷ 암호화 Key 관리 정책

■ 데이터베이스에 자료 저장 시 암호화를 사용하였을 경우 키 관리 정책 및 복구에 대한 안전성을 확인

o 애플리케이션 접근제어 관리

▷ 사용자 그룹별 접근권한 통제

■ 애플리케이션의 사용자를 그룹별로 구분하여 접근권한에 대한 등급을 설정하여 접근 권한 관리

▷ 관리자 접근 통제

■ 강력한 애플리케이션 서비스 관리자의 접근 통제 (예: 애플리케이션 서버에서 관리자에 대한 접근 날짜, 접근 시간, IP 주소별로 통제 적용)

▷ 다중 접속권한 제한

■ 단일 사용자 ID의 다중 온라인 세션(Session) 생성 제한

▷ 인증절차를 우회한 접근통제

■ 모든 애플리케이션 접근 방법에 대한 인증 실시

▷ 인가된 등급이상의 정보에 대한 접근 통제

- 특정 수준의 정보에 대한 접근권한을 부여 받은 사용자는 해당수준 또는 그 이하의 정보에만 접근되도록 구현

▷ 개발 시 사용된 Sample & Backup 코드 접근 제한

- 애플리케이션 개발 시 사용된 Sample 소스 코드 및 Backup 코드 등에 대한 접근 제한

o 로그 관리 및 접근 통제

▷ 보안등급별 로깅 설정(비밀/일반)

- 로그의 중요도에 따른 분류가 가능하도록 구성
- 정보의 비밀등급 설정에 따라 생성, 수정, 삭제, 조회 기능 관리
- 분류정보에 따른 로그인 정보, 오류정보, 기밀정보 접근 기록은 별도 관리되도록 구현
- 비밀로그정보는 기밀정보 접근기록, 관리자 모듈 접근 정보, 콘텐츠 유료 정보 생성 및 삭제 정보, 상거래 수행 정보 등을 포함도록 구현

▷ 보안등급별 로그 보관기간 설정

- 모든 사용자 ID의 응용시스템 로그인 및 이용기록 저장
  (사용자 ID, 단말 ID, 사용시간, 사용 정보 명, 로그인 실패, 출력 내역 등이 포함)

▷ 로그 접근 통제

- 보안 담당자의 사전 승인 없는 경우 애플리케이션의 로그파일에 대한 접근을 제한
- 로그 파일에 대한 무결성 확보 및 백업된 로그파일은 수정이 불가능하도록 구현

다. 구현단계 보안활동에 대한 점검(Review)

구현 단계의 보안활동에 대한 점검은 보안 담당자에 의해 수행되고, 보안 평가자에 의해 평가되어 그 결과가 애플리케이션 개발자에게 피드백 되어야 한다. 구현단계 보안활동에 대한 점검이 이루어지고 보완된 이후에 시험단계로 이행되어야 구현단계의 안전성을 확보할 수 있다.

다음 표는 보안 담당자 및 보안 평가자에 의해 체크되어야 하는 구현단계 전체 보안활동에 대한 점검목록 및 점검 양식을 나타낸다.

[표 3-18] 개발환경 보안(네트워크) 구현사항 점검목록 (예시)

Review 항목	중요도	적용 여부	특이사항	보완계획 (일정)	비고
1. 네트워크 정책 운영					
경계영역 구분 정책	매우 중요				
Inbound 및 Outbound 정책	매우 중요				
최소권한 원칙 정책	매우 중요				
History 관리 정책	매우 중요				
2. 네트워크 분리 및 DMZ 구성					
개발/운영 망 분리	매우 중요				
DMZ 구성	매우 중요				
3. 침입탐지 / 침입차단 시스템 적용 및 보안 관리					
침입탐지/침입차단시스템 을 통한 접근통제 및 모니터링	매우 중요				
침입탐지/침입차단 시스템 Rule 보안 관리	매우 중요				
4. DB Zone 분리 진단					
DB Zone 구성	매우 중요				
DB Zone 접근통제	매우 중요				
5. 데이터 전송보안 적용					
중요정보 암호화(SSL, VPN 등) 적용	중요				

(● : 보안 적용됨, ▲ : 부분 적용됨, X : 적용 안됨, N/A : 해당 사항 없음)

[표 3-19] 개발환경 보안(서버) 구현사항 점검목록 (예시)

진단항목	중요도	적용 여부	적용 시기
1. 네트워크 서비스 보안 관리			
불필요한 네트워크 서비스 중지 및 관리	매우 중요		
2. 기본 시스템 보안 설정 및 보안패치			
Banner 수정	중요		
시스템의 불필요한 권한 설정을 제거	매우 중요		
최신 시스템 취약점에 대한 대응조치 등의 시스템 Hardening	중요		
시스템 OS 및 서비스 소프트웨어에 대한 최신 보안패치 적용	중요		
3. 사용자 계정 관리			
불필요하거나 Default 사용자 계정 통제 및 삭제	매우 중요		
4. 악성코드 차단			
최신 패턴의 백신 소프트웨어 설치	매우 중요		
백신 실시간 감시 적용	중요		

(● : 보안 적용됨, ▲ : 부분 적용됨, X : 적용 안됨, N/A : 해당 사항 없음)

[표 3-20] 애플리케이션 보안 설계 구현사항 점검목록 (예시)

Review 항목	중요도	적용 여부	특이사항	보완계획 (일정)	비고
1. 보안 요구 사항					
프로그램 입력 검토	매우 중요				
Race Condition 발생 방지	매우 중요				
Buffer Overflow 발생 방지	매우 중요				
소스 코드 내 민감한 정보 삽입 금지	매우 중요				

소스 코드 내 악성 코드 삽입 금지	매우 중요				
실행 결과의 예외 처리	중요				
2. 로그인 관리					
로그인 실패 시 원인 표기 제한	매우 중요				
잘못된 패스워드 입력 회수 제한	중요				
최근 로그인 정보 표시	중요				
일정시간 경과 시 자동 로그오프 설정	매우 중요				
로그인 상태 유지 시 세션 방식 사용	매우 중요				
3. 패스워드 관리					
패스워드 최소 길이 제한	매우 중요				
추측 가능한 패스워드 사용 통제	매우 중요				
패스워드 변경 및 확인 시 보안 적용	매우 중요				
패스워드 파일 접근통제 및 암호화 저장 여부	매우 중요				
이전 부여 패스워드 일정 기간 재사용 통제	중요				
패스워드 사용기간 설정	중요				
인증(O.T.P, 생체인증, PKI)의 강화 작용	중요				
4. 사용자 계정 관리					
추측 가능 및 Default 계정 사용 통제	매우 중요				
불필요한 계정에 대한 주기적 검토 및 삭제	매우 중요				
5. 데이터 및 DB 보안 관리					
주요 데이터 및 자료 파일	매우 중요				

등의 DB 저장 유무					
데이터의 보안등급 분류 설정	매우 중요				
클라이언트와 서버간의 데이터 암호화 전송 여부	중요				
암호화 Key 관리 정책	중요				
6. 애플리케이션 접근 제어 관리					
사용자 그룹별 접근권한 통제	중요				
관리자 접근 통제	매우 중요				
다중 접속권한 제한	매우 중요				
인증절차를 우회한 접근통제	매우 중요				
인가된 등급이상의 정보에 대한 접근 통제	매우 중요				
개발 시 사용된 Sample & Backup 코드 접근 제한	매우 중요				
7. 로그 관리 및 접근 통제					
보안등급별 로깅 설정(비밀/일반)	매우 중요				
보안등급별 로그 보관기간 설정	매우 중요				
로그 접근 통제	매우 중요				

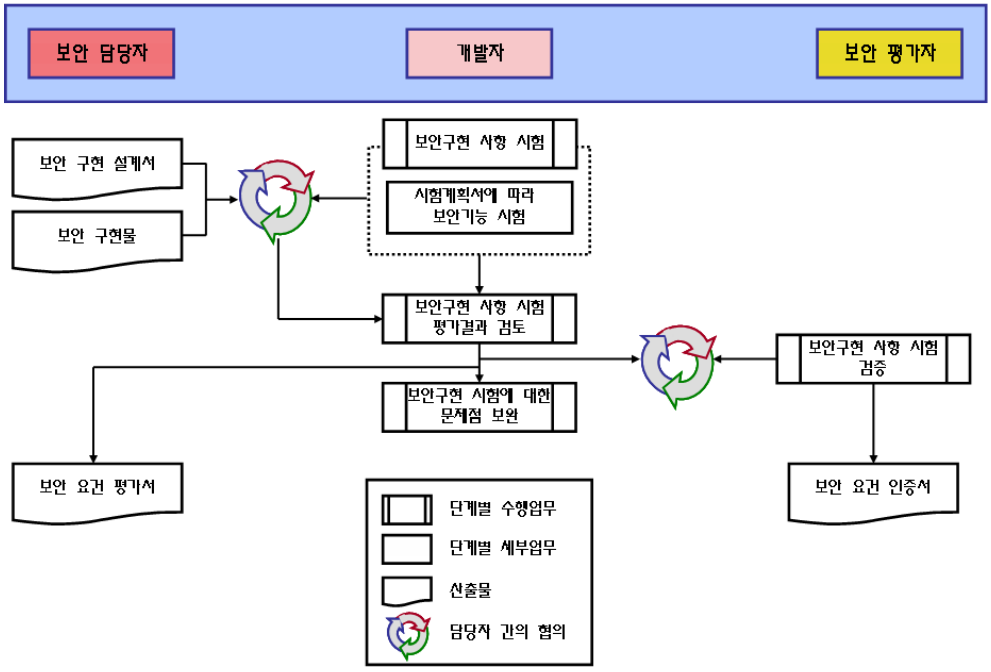
(● : 보안 적용됨, ▲ : 부분 적용됨, X : 적용 안됨, N/A : 해당 사항 없음)

☞ 고려 사항(Tip)

- 상기 언급된 애플리케이션 보안 설계 구현의 점검항목은 가장 기본적으로 고려가 필요한 항목들을 제시하고 있으므로, 개발자는 상기 항목들을 필요에 따라 가감하여 활용해야 한다.

5. 시험단계

시험 단계는 애플리케이션 개발주기에서 매우 중요한 단계이다. 개발된 애플리케이션은 운영 단계로 이동하기 전에 반드시 시험 단계를 거쳐야 한다. 시험 단계에서는 기능명세서, 기본설계서, 상세설계서에 반영된 보안사항을 시험하는 것으로, 시험계획 수립시에 설계단계의 보안사항을 점검할 수 있도록 수행되어야 한다.



(그림 3-7) 시험단계의 보안 프로세스

개발자와 보안 담당자는 시험계획을 기반으로 보안 구현 사항들을 시험하여 설계된 내용들이 정확히 반영되고 동작되는지 확인하여야 한다. 또한 시험계획 이외에 구현단계의 애플리케이션 보안 설계 구현 과정에서 제시된 추가항목 및 유사 애플리케이션에서 발생하는 최신 취약점들을 점검할 수 있도록 시험계획을 수립하여 시험하여야 하며, 시험단계에서 수행한 모든 결과는 문서화되고 개발자에게 피드백 되어야 한다.

시험단계에서 보안기능 시험을 위한 수행 절차는 다음과 같다.

- 애플리케이션 개발자 및 보안 담당자는 시험계획서에 따라 보안기능 시험 (시험의 객관성 및 보편성 강화를 위해 개발에 참여하지 않은 보안 담당자가 수행할 수 있음)
- 애플리케이션 개발자 및 보안 담당자는 애플리케이션 보안 설계 구현 과정에서 제시된 추가항목 및 최신 취약점 점검을 위한 시험계획서에 따라 보안기능 시험
  (시험의 객관성 및 보편성 강화를 위해 개발에 참여하지 않은 보안 담당자가 수행할 수 있음)
- 보안기능 시험이 완료되면 보안 담당자는 평가결과를 검토하여 보안 기능 동작의 적절성 여부를 판단
- 보안 평가자는 평가 절차에 따라 적절히 계획되고 실행되었는지, 평가 결과가 신뢰할 수 있는지 검증
- 보안 담당자 및 보안 평가자의 시험 결과를 개발자에게 피드백을 제공하여 문제가 있는 기능에 대해 보완토록 조치

보안 기능 시험이 완료되면 보안 전담 팀은 시험결과를 검토하여 해당 애플리케이션의 보안기능의 적절성 여부를 판단하고, 애플리케이션에 대한 내·외부 인증절차가 요구되는 경우 인증을 실시한다.

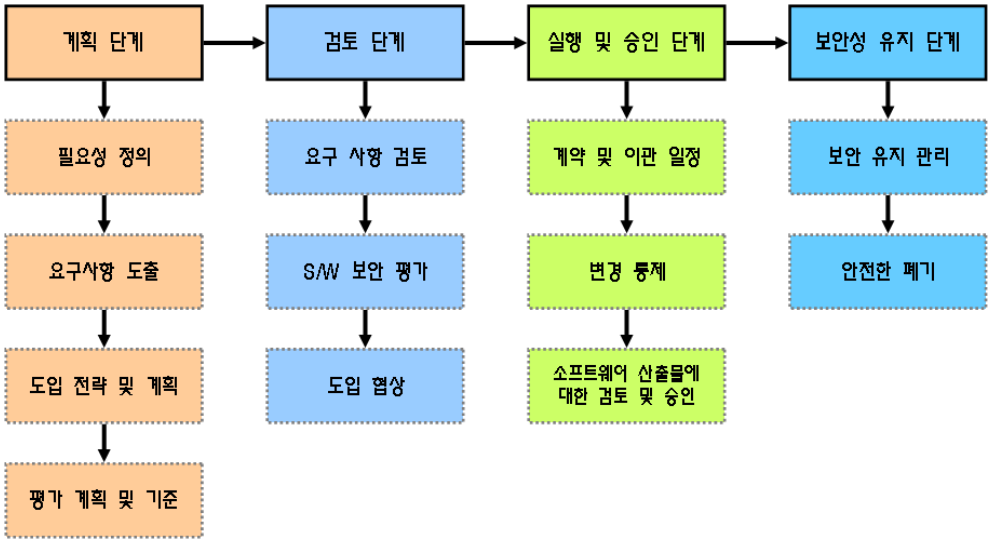
※ 소프트웨어 개발과정의 보안성 강화에 관계된 좀더 상세한 자료는 ‘부록 2. 안전한 소프트웨어 개발을 위한 자료 목록’을 참조하십시오.

제 4 장 소프트웨어 도입과정의 정보보호

제 1 절 소프트웨어 도입 단계에서 보안

소프트웨어 개발자들이 효율성 및 편의성에 치중함에 따라 개발과정에 정보보호를 위한 요구사항들이 반영되지 않아 발생할 수 있는 보안 취약점은 매우 다양하지만 실제 대응은 보안 솔루션을 도입하는 수준의 비교적 단순한 접근방식을 취하고 있다. 아울러 보안 소프트웨어(정보보호제품)에 대해서는 국가기관에서 보안 적합성 검토 및 보안성 평가를 수행하고 있지만 이외 소프트웨어에 대해서는 별다른 보안성 검토가 되고 있지 않은 실정이며 이미 도입 완료된 소프트웨어의 보안 취약점을 수정하는 것은 기존 운영되고 있는 서비스와 시스템에 악영향을 미칠 수 있을 뿐만 아니라 많은 추가비용이 수반될 수 있으므로 소프트웨어 도입을 담당하는 부서에서는 소프트웨어 도입 시 위험분석, 보안 취약성 점검 및 보안 평가 등을 실시하여 안전성을 검증해야 한다.

본 절에서는 안전한 소프트웨어 도입을 위해 사용자 관점의 고려사항들에 대해 도입을 계획하는 단계부터 보안을 고려하여 요구사항 검토 및 평가를 통해 실행을 거쳐 폐기 단계에 이르기까지 단계별로 고려하여야 하는 보안 권고사항을 제시한다.



(그림 4-1) 소프트웨어 도입 단계별 보안활동

1. 계획 단계

계획단계는 소프트웨어 제품을 도입하기 위한 필요성을 정의하는 것을 시작으로 요구사항 도출, 다양한 도입 전략과 연계한 위험을 식별하기 위한 전략 및 계획 수립, 마지막으로 평가 기준 및 계획 개발 등 4가지 중요 활동 분야를 포함하고 있다.

가. 필요성 정의

소프트웨어 도입의 필요성 정의는 소프트웨어 도입을 통해 해결하고자 하는 업무의 문제점에 대해 위험 분석 및 평가를 실시하여 소프트웨어 도입에 대한 요구사항을 결정하는 것이다.

위험관리는 위험을 식별하고 평가하여 감내할 수 있도록 경감시키기 위한 것으로, 초기의 위험평가는 도입할 소프트웨어에 요구되는 보안 범주와 보안통제의 기준을 결정하는데 도움을 주며 도입 담당 부서에서는 위험관리 측면에서 다음 항목에 대해 확인이 필요하다.

- 보호를 필요로 하는 대상
- 보호 대상을 안전하게 유지하기 위한 소프트웨어와 정보자산 분류
- 보호 대상에 보안 미적용시 문제점
- 소프트웨어가 비정상적으로 동작할 경우 업무 영향
- 잔존하는 위험 요소 및 효과적인 관리 방안
- 위험 통제 및 경감을 위한 유지관리 요소

또한 위험평가를 통해 식별된 위험을 감소시키기 위해서는 다음과 같은 절차가 포함된다.

- 위험을 처리(수용, 경감, 회피, 전가, 분담 등) 하기 위한 대체 수단 평가
- 수용 가능한 위험 수준으로 감소하기 위한 보호 전략 수립
- 위험 감소, 비용증가, 운영 효율성을 고려한 중재 방안 수립
- 보호 전략이 적용된 이후 잔존 위험 관리 방안 수립

나. 요구사항 도출

일반적으로 보안 요구사항은 다양하게 발생하므로 효과적으로 요구사항을 도출하기 위해서는 기술적, 관리적 측면과 보증 등으로 분류하고, 위험 평가를 통해 필요성이 정의되면 조직 전체에 공통적으로 적용 될 수 있는 다음과 같은 기본적인 보안 요구사항(기밀성, 무결성, 가용성)들이 정의되어야 한다.

- 악의적인 목적으로 취약점을 유도하는 설계 또는 구현 단계의 소프트웨어 보안 취약성에 대한 정의
  (예, 절대경로 우회, CSS, 오버플로우, SQL 인젝션 등)
- 기밀성, 무결성, 가용성에 대한 보안 수준을 선택하기 위한 세부 사항을 포함하는 보안 범주에 대한 설명

- 요구사항 달성을 위해 필요한 처리사항 및 근거
- 소프트웨어 도입 승인 기준을 정의하는 테스트 계획
- 감리 또는 감사 기관에 의한 코드 검사를 위한 고려사항
- 소프트웨어 컴포넌트에 대한 서술을 포함하는 기반 구조
- 모든 보안 설정 환경 선택사항들을 대한 가이드라인
- 소프트웨어 담당 직원 또는 부서의 적격성
- 소유권, 통제사항, 영향력과 관련하여 요구된 정보
- 조직 및 기관의 특별한 요구사항 또는 명령

다. 도입 전략 및 계획

소프트웨어 도입을 위해 도입 전략과 계획은 실제 구매 또는 계약전에 수립되어야 하며 조직의 보안 정책을 반영해야 한다.

아울러 전략 및 계획 수립 과정은 소프트웨어 도입에 관한 일정과 실행을 위한 역할 및 책임이 명시되어야 하며 도입 결정을 위해 포함되어야 할 내용으로는 다음과 같은 것들이 있다.

[표 4-1] 소프트웨어 도입 결정시 포함되어야 할 항목

항목	내용
보안 전문가 선정	도입 부서에서는 도입을 위한 모든 과정(계획, 요구사항 도출, 소스 코드 선택, 계약, 프로젝트 관리 등)에 적격의 보안전문가를 통해 수행될 수 있도록 요청해야 한다.
보안 범주 식별	보안 범주는 정보와 정보시스템을 위협하는 사고의 잠재적 파급효과를 기반으로 보안 요구사항을 도출하고 선택하는데 적용된다.

보안 요구사항	중요 보안 고려사항 뿐만 아니라 공급자 선택, 부서간 역할 및 책임, 프로젝트 관리 등에 대한 사항들이 기술되어야 한다. 예) 소프트웨어 개발 또는 시스템 통합 부문: 소프트웨어 보안 요구사항에는 수용 가능한 안전한 방식으로 운영될 수 있음을 보장하는 결정적인 근거(프로세스, 절차, 테스트 결과 등)와 의무이행사항이 제공되어야 하며 보안부서에서는 알려진 보안 취약점으로부터 안전하게 동작함을 결정하고 평가한다.
독립적인 테스트 계획	소프트웨어 구축, 안전성, 기능과 특별한 요구사항들에 대해 직접 확인을 위한 기관 자체의 테스트 계획이 기술되어야 한다.

라. 평가 계획 및 기준

평가 계획은 소프트웨어제품이 도출된 요구사항을 만족하는지 검증하기 위한 절차가 포함되며 도입할 때 보안 기준에 부합하도록 평가 기준을 기술한다.

예) 도입 부서 또는 기관에서는 유망 공급자에게 계약 협상에 앞서 소프트웨어의 안전한 품질 제공을 위해 평가될 수 있도록 일정 및 평가방법을 수립하여 요청하고 평가 결과에 의해 최상의 도입 후보자를 결정하는데 이용한다.

평가 기준은 다음과 같은 3가지 범주에서 수립될 수 있다.

[표 4-2] 소프트웨어 도입 평가 기준

항목	내용
직원과 조직	보안에 정통하지 않은 소프트웨어 개발자는 분석, 설계, 구현 관점에서 이용 가능한 취약점을 유발하는 결함을 인식하지 못하고 조직에서는 가용성에 치중하여 잘못 정의된 보안책임을 부여할 수 있으므로 개발자가 보안 지식을 소유하고 있는지와 조직에서 적절한 보안 지원 사항(교육훈련, 절차, 기준 등)을 제공하는지 평가 되어야 한다.
프로세스 성숙도	성숙한 보안 프로세스는 결함을 허용하지 않거나 조기 발견 조치의 가능성이 크기 때문에 소프트웨어 보안 신뢰성을 유지하고 개발 능력을 결정하기 위해 프로세스 성숙도가 평가 되어야 한다.
기술	소프트웨어 결함을 찾아내어 조치하기 위한 기술이 평가되어야 한다

2. 검토 단계

도입 검토 단계는 요구사항 검토, 소프트웨어 보안 평가, 도입 협상의 3단계가 있다.

가. 요구 사항 검토

도입 부서에서는 보통 과업 기술서를 준비하게 되는데, 과업 기술서는 소프트웨어 보안 요구사항 검토를 위해 다음과 같은 내용들이 고려되어야 한다.

- 소프트웨어 보안 신뢰성과 관련한 정의
- 보안 요구사항과 일반적인 프레임워크를 제공하는 보안 범주에 대한 정의
- 필요한 보안 요구사항(기능, 특성 등)과 이를 충족시킬 수 있는 근거

- 소프트웨어 구현과 연계된 보안 위험과 안전성을 관리하기 위한 정규 프로그램을 포함하는 위험 관리
- 프로그램 코드의 보안 상태를 결정하기 위한 독립적인 보안감리 또는 감사 기관을 통한 코드 감사
- 소프트웨어 보안 관점에서의 클래스, 오브젝트, 코드, 테이블 등을 포함하는 아키텍쳐 기술
- 보안 요구사항 만족 여부에 대한 접근 방법을 정의한 보안 테스트 계획
- 보안 환경 설정을 위한 선택사항에 대한 가이드
- 보안 요구사항이 지속적으로 유지하기 위한 패치 및 업그레이드

이외 권장되는 부가적인 항목으로는 다음과 같은 것들이 있다.

- 소프트웨어 보안과 관련한 공급자와 도입자의 법률적 책임
- 소프트웨어 개발 프로세스 주기의 보안 품질
- 소프트웨어 보안 승인 기준
- 핵심 보안 인력에 대한 신원과 소프트웨어 개발자의 자격 및 훈련
- 소프트웨어 보안 교육 및 훈련
- 소유권, 통제사항, 효과에 대한 요구된 정보
- 요구된 사전 설정 보안 특징
- 보안 미이행에 대한 보상 약관 등

나. 소프트웨어 보안 평가

도입 부서 또는 기관에서는 소프트웨어 보안 요구사항 충족여부 판단을 위해 담당자를 선정하여 기술성, 관리 및 편의성, 가격 등으로 분리하여 평가하고, 상황에 따라 배점 기준을 달리하여 평가점수에 따라 적합한 소프트웨어를 선정하여야 한다. 따라서 소프트웨어 선정시 반드시 보안 요구사항에 대한 적합성 여부를 평가를 통해 검증해야 한다.

다. 도입 협상

평가 결과에 따라 최적의 소프트웨어를 선택하여 도입을 위한 협상을 진행하며 도입 부서에서는 요구사항과 계약 조항들에 대한 협의를 실시한다.

3. 실행 및 승인 단계

실행 및 승인단계는 공급 부서 또는 업체의 개발된 소프트웨어 운영 상태를 모니터링하고 최종 결과물(서비스, 시스템, 제품 등)들이 보안 요구사항들 준수하고 있음을 승인하는 것을 포함한다.

가. 계약 및 이관 일정

계약 및 이관 일정은 소프트웨어 보안 요구사항의 이행을 위해 매우 상세한 업무를 포함해야 하며, 만약 작업 명세 일정(WBS)이 사용 되었다면 도입 부서 및 기관에서는 소프트웨어 보안 요구사항에 대한 이행이 작업 명세 일정에서 식별되도록 확실히 해야 한다.

나. 변경 통제

소프트웨어에 대한 변경 통제는 소프트웨어에 변경이 발생해도 보안 요구사항 등 준수되도록 통제되어야 한다.

다. 개발된 소프트웨어에 대한 검토 및 승인

개발된 소프트웨어의 보안 요구사항에 대한 이행 산출물로는 소프트웨어에 대한 위험관리 계획, 보안 인증서, 테스트 결과서가 포함 될 수 있으며, 승인 기준은 계약 또는 협정 사항에 포함되었거나 측정가능 하도록 명시되어야 한다.

도입 부서에서는 소프트웨어 보안 전문가를 통해 보안 요구사항 충족 여부가 확인될 때까지 도입 승인을 해서는 않되며, 아울러 자체적으로 소프트웨어의 보안성 테스트를 고려해야 한다.

4. 보안성 유지 단계

보안성 유지 단계는 지속적인 유지관리를 통해 보안취약점을 제거하고 새로운 보안동향에 맞게 추가적인 보완을 실시하는 것과 소프트웨어의 사용 중단 또는 신규 도입으로 기존 소프트웨어 폐기시 안전하게 관리해야 하는 폐기관리가 포함된다.

가. 보안 유지 관리

초기 보안 요구사항 또는 협의사항에 포함된 항목들을 지속적으로 강화할 수 있도록 세심한 주의가 필요며 보안 설정을 안전하게 유지하기 위해 다음과 같은 사항이 포함된다.

- 실행 가능한 소프트웨어 보안 지침에 대한 실행 환경 통제
- 소프트웨어 수정 재검증과 통합 이행
- 문제 분석, 재구축, 검토, 승인에 대한 자문
- 소프트웨어 보안 성능에 대한 예측(보안 취약점 발생 경향)
- 보안이 신뢰된 작업 환경과 관련 보고서
- 보안 및 보안 취약점 조치를 위한 보안 정책 및 절차에 대한 유지
- 소프트웨어 및 데이터 이관에 대한 실행 및 유지 그리고 폐기를 위한 정책 및 절차 등

나. 안전한 폐기

폐기 정책과 절차는 흔히 매우 쉽게 간과되는 부분으로 많은 조직과

부서에서는 안전하게 소프트웨어를 폐기하기 위한 정책과 절차를 수립하지 않고 있다. 도입 또는 유지 관리 부서에서는 소프트웨어에 대한 보안과 안전한 폐기를 데이터가 안전하게 이관되고 삭제 될 수 있도록 보안 정책과 절차를 확실히 수립해야 한다.

단일 소프트웨어가 아닌 다수의 소프트웨어들로 구성되는 집약적인 시스템은 폐기되거나 갱신될 때 새로운 시스템에 반드시 타당한 보안수단에 의해 데이터가 이관될 수 있도록 각별히 유의해야 한다.

참고문헌

[1] Richard Kissel, Kevin Stine, Matthew Scholl, Hart Rossman, Jim Fahlsing and Jessica Gulick, “NIST: Security Considerations in the Information System Development Life Cycle,” NIST SP 800-64 Rev2, 2008.

[2] Lyndon Oh, Michael Colon and Elaine Fedchak etc. "Software Security Assurance: A State-of-the-Art Report (SOAR)," IATAC, DACS, 2007.

[3] Elaine Fedchak, Thomas McGibbon and Robert Vienneau, "Software Project Management for Software Assurance," DACS, DACS Report Number 347617, 2007.

[4] Noopur Davis, "SECURITY IN THE SOFTWARE LIFECYCLE, Making Software Development Processes and Software Produced by Them - More Secure," DHS, CS&C, NCSD, 2006.

[5] Noopur Davis, "Secure Software Development Life Cycle Processes: A Technology Scouting Report," CMU/SEI, 2005.

[6] Jayaram K R and Aditya P. Mathur, "Software Engineering for Secure Software - State of the Art : A Survey," Purdue University, 2005.

[7] Ron Moritz and Scott Char etc. "Improving Security Across the Software Development LifeCycle," NCSP, 2004.

[8] John Neumann, Brian Mullins, Delores Cohen, Shelby S. Oakley, Christopher Miller, Gary Middleton, and Marie Ahearn, "Knowledge of Software Suppliers needed to manage risks," GAO, 2004.

[9] John Viega, "Secure Software Development Lifecycle," NCSS, 2004.

[10] Robert Seacord, "The CERT C Secure Coding Standard," Addison Wesley, 2008.

[11] Michael Cross, Mark Burnett and James Foster, "2008 Ultimate Secure Coding CD," Syngress Publishing, 2007.

[12] Michael Sutton, Pedram Amini, and Adam Greene, "Fuzzing: Brute Force Vulnerability Discovery," Addison Wesley, 2007.

[13] Brian Chess and Jacob West, "Secure Programming With Static Analysis," ddison Wesley, 2007.

[14] Gary McGraw, "Software Security: Building Security In," Addison Wesley, 2006.

[15] Mark Dowd, John McDonald and Justin Schuh, "The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities," Addison Wesley, 2006.

[16] Michael Howard, Steve Lipner, "The Security Development Life-cycle - A Process for Developing Demonstrably More Secure Software," Microsoft Press, 2006.

[17] Michael Howard, David LeBlanc and John Viega, "19 Deadly Sins of Software Security," McGraw-Hill, 2005.

[18] Robert C. Seacord, "Secure Coding in C and C++," Addison Wesley, 2005.

[19] Greg Hoglund and Gary McGraw, "Exploiting Software: How to Break Code," Addison Wesley, 2004.

[20] Mark G. Graff and Kenneth R. van Wyk, "Secure Coding: Principles and Practices," O'Reilly, 2003.

[21] Michael Howard and David LeBlanc, "Writing Secure Code, Second Edition," Microsoft Press, 2003.

부 록

1. 소프트웨어의 취약점 점검 및 오류 검증 도구

□ 소스코드 보안 취약성 점검 도구

소스코드 내에 버퍼오버플로우가 발생 가능한지 또는 포맷스트링, 포인터, 배열, 함수 반환 주소 등 메모리 관련 보안취약점이 존재하는지 등을 점검해 주는 도구이다. 상기 취약점들은 보안공격에 직접적으로 이용될 수 있기 때문에 사전 점검이 매우 중요하다.

1	도구명	ITS4(It's the Software Stupid Source Scanner)	주요 기능
	제작사	Cigital	- 취약한 함수에 대하여 점검(버퍼 오버플로우, 레이스컨디션 등) - 대체함수 제공, Risk 별로 함수 분류
	지원언어	C/C++	
	홈페이지	www.cigital.com/its4/download.php	
2	도구명	LCLint	주요 기능
	제작사	virginia univ.	- 정적 C 프로그램 점검 툴 - 프로그램 코드의 버그를 점검
	지원언어	C/C++	
	홈페이지	lclint.cs.virginia.edu/	
3	도구명	ckit	주요 기능
	제작사	bell-labs	- 변수, 구조체, 공용체 등의 범위, 타입 점검 - 개체의 크기와 메모리의 설계, 초기화
	지원언어	C	
	홈페이지	http://www.smlnj.org/doc/ckit/index.html	
4	도구명	Safer C	주요 기능
	제작사	Oakwood Computing Associates Limited	- check/use 되는 함수를 조사하여 점검. - C 언어의 레이스컨디션 점검
	지원언어	C	
	홈페이지	www.oakcomp.co.uk/	

5	도구명	RATS (Rough Auditing Tool for Security)	주요 기능
	제작사	Fortify Software Inc.	- 버퍼오버플로우, 레이스컨디션 점검. - 검사결과는 HTML 형식으로 출력 가능
	지원언어	C/C++/Perl/Python/PHP	
	홈페이지	www.fortify.com/security-resources/rats.jsp	
6	도구명	PScan	주요 기능
	제작사	ucdavis univ.	- printf 스타일 함수의 잘못된 사용을 점검 - 버퍼 오버플로우 점검
	지원언어	C	
	홈페이지	http://seclab.cs.ucdavis.edu/projects/testing/tools/pscan.html	
7	도구명	qaudit.pl	주요 기능
	제작사	SourceForge	- C/C++ 코드의 간단한 점검 - 버퍼오버플로우, 레이스컨디션, 실행 콜 등 점검
	지원언어	C/C++	
	홈페이지	http://freshmeat.net/projects/qaudit/	
8	도구명	formatstring	주요 기능
	제작사	codeproject	- 포맷스트링 함수 사용부분 점검
	지원언어	C/C++	
	홈페이지	www.codeproject.com/KB/string/FormatString.aspx	
9	도구명	Splint	주요 기능
	제작사	Virginia univ.	- 보안 취약성과 코딩 실수를 정적으로 확인 - 사용되고 있지 않은 변수 선언, 선언문의 충돌, 반환값 형태의 불일치, 무한루프 등을 검사
	지원언어	C	
	홈페이지	www.splint.org/	
10	도구명	Cyclone	주요 기능
	제작사	AT&T Labs Research and Greg Morrisett's group at Cornell University	- 버퍼 오버플로우, Dangling 포인터, 포맷 스트링 취약점 점검 - 패턴매칭, 데이터타입, 예외처리, 메모리 관리
	지원언어	C	
	홈페이지	www.rapid-finder.com/en/odpgate/845567/cyclone.html	
11	도구명	Vault	주요 기능
	제작사	Microsoft Research	- Dangling 포인터, 메모리 누수 등 점검
	지원언어	C/C++	
	홈페이지	research.microsoft.com/vault/	

12	도구명	Cqual	주요 기능
	제작사	Cqual	- dataflow와 type inference를 이용하여 분석 - 형태 수식자를 소스코드에 추가하며 Cqual은 추가된 형태 수식자의 천이가 미리 정의된 규칙에 따르는가를 검사
	지원언어	C	
	홈페이지	tbp.berkeley.edu/~jdonald/cqual/	
13	도구명	Flawfinder	주요 기능
	제작사	www.dwheeler.com/	- 취약성 데이터베이스를 참조하여 소스코드 점검 - 취약성 데이터베이스는 Python 스크립트 내부에 기술되어 있음 - "Hit" 개념사용(입력된 파일에서 발견되는 보안결점을 말함.)
	지원언어	C/C++	
	홈페이지	www.dwheeler.com/flawfinder/	
14	도구명	CodeWizard(상용)	주요 기능
	제작사	PARASOFT	- 코딩 룰과 소스코드의 상태를 조합 - 320 개에 이르는 코딩 룰을 사용해, 원시 코드 검사
	지원언어	C/C++	
	홈페이지	www.techmatrix.co.jp/products/quality/codewizard/	
15	도구명	Illuma(상용)	주요 기능
	제작사	Reasoning Inc.	- NULL pointer에의 부정 참조나 변수 배열외에 접근하는 취약성 요인을 검사
	지원언어	C/C++	
	홈페이지	http://www.reasoning.com/	
16	도구명	PC-lint/FlexLint(상용)	주요 기능
	제작사	Gimpel Software	- 구문 분석 기법에 의해 버그, 불일치, 중복코드 등의 소스코드 오류 검사
	지원언어	C/C++	
	홈페이지	http://www.gimpel.com/html/lintinfo.htm	
17	도구명	비주얼 스튜디오 PreFast(상용)	주요 기능
	제작사	Microsoft	- 정적 소스 코드 분석기가 포함 - 컴파일하는 동안 실행이 되며, 잠재적인 C/C++ 취약성을 알려주는 컴파일러 경고문을 생성
	지원언어	C/C++	
	홈페이지	www.microsoft.com/whdc/DevTools/tools/PREfast.mspx	

18	도구명	비주얼 스튜디오 FxCop(상용)	주요 기능
	제작사	Microsoft	- 어셈블리 수준에서 분석을 수행 - 마이크로소프트 닷넷 프레임워크 가이드라인(Microsoft .Net Frame-work Guidelines)에 따르는지 검사 - 안전한 닷넷 어셈블리를 위한 26개의 보안 테스트 포함
	지원언어	.NET	
	홈페이지	http://msdn.microsoft.com/en-us/library/bb429476(vs.80).aspx	
19	도구명	AppScan DE 플러그인(상용)	주요 기능
	제작사	Sanctum	- MS Visual Studio .NET 지원 - 개발업무를 수행하는 중에 작성 코드의 취약성 검사를 수행
	지원언어	C++/C#/J#	
	홈페이지	http://www.sanctuminc.com/solutions/appscan/de/index.html	
20	도구명	SecurityChecker(상용)	주요 기능
	제작사	Compuware DevPartner	- Visual Studio 2005 지원 - 소스코드 및 실행코드에 대해 보안 취약성 점검 - ASP.NET 응용 프로그램에 대한 알려진 보안 취약점 분석, Com-pile time/Run time/Integrity 분석 기능 제공
	지원언어	C#/Visual Basic.NET/ASP.NET	
	홈페이지	http://compuwarezone.co.kr/html/sub_pro_02_02.htm	

□ 소스코드 오류 검사 도구

소스코드 오류 검사 도구는 기본적으로 소스코드 내에 존재하는 메모리 관련 버그나 논리적 오류를 점검해 주는 도구이다. 공격자는 소스코드 내에 존재하는 버그나 오류 중 보안공격이 가능한 부분이 있는지를 찾아낼 수 있기 때문에 아래의 도구를 활용하여 사전에 점검할 수 있다.

1	도구명	Sparrow(상용)	주요 기능
	제작사	파수닷컴	- 메모리 누수(Leak), 버퍼 오버런, 버퍼 오버플로우 등 검사
	지원언어	C/C++	
	홈페이지	www.fasoo.com/	
2	도구명	Coverity Prevent(상용)	주요 기능
	제작사	Coverity, Inc.	- 정적 소스코드 오류 분석 - 중요 결함이나 보안 취약성 검사
	지원언어	C/C++/Java	
	홈페이지	www.coverity.com/html/coverity-prevent.html	
3	도구명	PolySpace(상용)	주요 기능
	제작사	Ecole Polytech	- 소스 코드에 런타임 에러가 존재하지 않는지 검사
	지원언어	C/C++	
	홈페이지	www.mathworks.com/polyspace/	
4	도구명	Klocwork(상용)	주요 기능
	제작사	Klocwork Inc.	- 중요 결함이나 보안 취약성 검사
	지원언어	C/C++/Java	
	홈페이지	www.klocwork.com/	
5	도구명	PurifyPlus(상용)	주요 기능
	제작사	IBM Rational 소프트웨어	- 포인터나 메모리 관리 관련 잠재적인 오류 검사 - 메모리 손상/유실 점검 - 테스트되지 않은 코드 범위 식별
	지원언어	C/C++/Java	
	홈페이지	www.ibm.com/software/awdtools/purify/	
6	도구명	Insure++(상용)	주요 기능
	제작사	Parasoft	- 힙/스택 메모리 검사, 배열 및 스트링 경계값 오류 검사, 메모리 누수, 포인터 오류 등 검사
	지원언어	C/C++/.NET/Java	
	홈페이지	www.parasoft.com/jsp/products/home.jsp?product=Insure	

7	도구명	CODECHECK(상용)	주요 기능
	제작사	abraxas software	- 소스 코드의 이식성, 유지보수성, 복잡도, 재사용성, 라이브러리/클래스 관리
	지원언어	C/C++	
	홈페이지	www.abxsoft.com/	
8	도구명	OSPC (Open Systems Portability Checker)(상용)	주요 기능
	제작사	Knowledge Software	- ANSI, POSIX 표준 적합성 검사 - 이식성 검사
	지원언어	ANSI-C/POSIX-C	
	홈페이지	www.knosof.co.uk/ospc.html	
9	도구명	C/Ada Verifier(상용)	주요 기능
	제작사	PolySpace	- 컴파일 시 잠재적인 런타임 에러 검사
	지원언어	C/Ada	
	홈페이지	www.embeddedstar.com/software/content/p/embedded309.html	
10	도구명	Fortify SCA(Fortify Source Code Analysis)(상용)	주요 기능
	제작사	fortify	- 소프트웨어 보안 취약점 검사 및 해결지원 - 정적분석과 동적분석 지원
	지원언어	-	
	홈페이지	www.fortify.com/products/sca/	
11	도구명	Ounce(상용)	주요 기능
	제작사	Once Labs	- 보안 취약점 검사 및 결함 추적 기능 제공 - 보안취약점에 대한 위험도 등에 대한 매트릭스 정보 제공
	지원언어	-	
	홈페이지	http://www.ouncelabs.com/solutions/ounce-source-code-analysis.asp	

□ 실행코드 보안 취약성 점검 및 방지 도구

실행코드 보안 취약성 점검 및 방지 도구는 실행코드를 실행시킨 후 보안 취약성을 점검하고, 예방해 주는 도구이다. 주로 메모리 관련 보안 취약성을 점검하며, 소스코드 보안 취약성 점검 도구에서 찾지 못한 취약성을 찾을 수 있다. 또한 방지 도구는 실행코드에 보안 취약성이

발견되더라도 공격자가 이를 이용하여 공격하지 못하도록 막아주는 역할을 한다.

가. 실행코드 보안 취약성 점검 도구

1	도구명	BFBTester	주요 기능
	제작사	sourceforge	- 명령어 라인 인자(argument) 오버플로우를 체크 - 환경 변수(environment variable)의 오버플로우 체크 - 안전하지 않은 tempfile 생성 감시
	지원언어	-	
	홈페이지	http://bfbtester.sourceforge.net/	
2	도구명	Sharefuzz	주요 기능
	제작사	@stake, sourceforge	- Unix 시스템에서의 환경변수 오버플로우 검출
	지원언어	-	
	홈페이지	http://sourceforge.net/projects/sharefuzz/	
3	도구명	FuzzerServer(상용)	주요 기능
	제작사	@stake, symantec	- format string이난 buffer/heap 오버플로우와 같은 통상적인 취약점을 발견하기 위해 마치 정상적인 웹 서버와 같이 동작하면서도 fuzzing 응답을 생성
	지원언어	-	
	홈페이지	http://www.atstake.com/research/tools/	
4	도구명	COMbust(상용)	주요 기능
	제작사	@stake, symantec	- ActiveX/COM/DCOM 컴포넌트를 윈도우즈 플랫폼에서 테스트 - COMBust는 주어진 컴포넌트에 의해 제공되는 인터페이스들을 찾아내고 fuzzing을 통해 테스팅을 위한 컴포넌트 기능을 수행 - 부적절한 입력 validation에 의한 보안 취약성 탐지
	지원언어	-	
	홈페이지	http://www.atstake.com/	
5	도구명	GDBProfiler	주요 기능
	제작사	Drexel univ.	- Linux/Unix/Windows 플랫폼에서 수행되는 동적 실행시간 응용 분석 툴 - GNU Debugger를 사용하여 C나 C++로 쓰여진 응용의 실행시에 행동을 모니터함
	지원언어	-	
	홈페이지	http://serg.mcs.drexel.edu/gdbprofiler	

6	도구명	MemWatch	주요 기능
	제작사	-	- 메모리 leak 검출 - 메모리 버퍼들의 overflow와 under-flow를 탐지 - 태크를 붙여 추적. 블록이 할당될 때, 그 주소 및 메모리 할당 정보가 데이터베이스에 기록되며, 블록이 끝날 때 메모리가 free되면서 데이터베이스에 기록된 정보와 일관되게 free되는지 검사
	지원언어	C	
	홈페이지	directory.fsf.org/project/memwatch/	
7	도구명	Bugscan(상용)	주요 기능
	제작사	hbgary	- 실행 파일의 취약점 종류와 위치, 취약점의 심각도, 그리고 그것을 수정하는데 도움이 되는 권장사항 제시
	지원언어	-	
	홈페이지	http://www.hbgary.com	

나. 실행코드 보안 취약성 점검 및 방지 도구

1	도구명	StackGuard	주요 기능
	제작사	Wirex	- 스택의 리턴주소가 수정되는 것을 막아 스택파괴 공격을 탐지하고 방어 - Canary 개념을 사용 - Canary 값이 변경된 경우 공격을 받은 것이라고 판단
	지원언어	C/C++	
	홈페이지	http://immunix.org/	
2	도구명	Bounds Checking	주요 기능
	제작사	Richard Jones and Paul Kelly, Imperial College	- 실행중 발생하는 버퍼 오버플로우 공격을 검출하여 조치하기 위해 컴파일러를 확장(ex: malloc 등의 표준 라이브러리 수정) - 메모리상의 배열, 포인터 등에 대해 실행을 추적하여 생성시 메모리 크기를 넘어가는지 검출 및 방어
	지원언어	C	
	홈페이지	http://www.doc.ic.ac.uk/~phjk/BoundsChecking.html	

3	도구명	Stack Smashing Protector	주요 기능
	제작사	IBM	- 스택에 대한 버퍼 오버플로우 공격 검출하기 위해 컴파일러를 확장 - 기본적인 스택 오버플로우 공격 방어 구조는 StackGuard와 동일 - 중간언어에 대해 검출기능을 삽입하여 대부분의 아키텍처 컴파일러를 지원 가능 - 스택에서 리턴주소의 변경, 인수나 프레임포인터의 변경 공격 등을 탐지
	지원언어	-	
	홈페이지	http://www.trl.ibm.com/projects/security/ssp/	
4	도구명	FreeBSD stack integrity patch(libparanoia)	주요 기능
	제작사	Alexandre Snarskii	- C 라이브러리에서 발생하는 버퍼 오버플로우 공격 검출 - libparanoia 설치시 표준 함수 라이브러리 교체 필요
	지원언어	C	
	홈페이지	http://www.lexa.ru/snar/libparanoia/	
5	도구명	Libsafe	주요 기능
	제작사	Avaya Labs	- C 라이브러리의 버퍼오버플로우 문제를 회피하기 위한 라이브러리 - 자동으로 버퍼의 안전한 상한 크기를 결정하며 버퍼 크기의 결정은 버퍼 관련 함수를 실행한 후 결정 - 함수가 실행될 때 버퍼의 최대크기를 결정.
	지원언어	C/C++	
	홈페이지	http://www.research.avayalabs.com/	
6	도구명	libformat	주요 기능
	제작사	Tim J. Robbins	- printf 계열 함수를 가로채 포맷스트링 공격을 막기 위한 라이브러리
	지원언어	C/C++	
	홈페이지	http://www.freebsdsoftware.org/devel/libformat.html	
7	도구명	Trustfile	주요 기능
	제작사	Ucdavis Univ.	- 동적으로 유닉스 파일 접근 점검을 위한 라이브러리 - Trust/Untrust User를 설정하여 특정 파일에 대한 접근 가능한 사용자만 접근하도록 허가.
	지원언어	C	
	홈페이지	ftp://nob.cs.ucdavis.edu/pub/sec-tools/trustfile.tar.Z	

□ 소프트웨어 모델 체킹 도구

소프트웨어 모델 체킹 도구는 소프트웨어의 취약성을 정형적으로 모델링하고 소프트웨어 코드의 제어 흐름이나 데이터 흐름을 분석하여 추상화된 설계 프로파일을 도출한 후 이 설계 프로파일이 취약성 모델과 일치하는지 검사한다.

1	도구명	Alloy Analyzer	주요 기능
	제작사	MIT's Software Design Group	- FOL에 기반한 간단한 구조적 모델링 언어로 작성한 모델을 분석
	지원언어	-	
	홈페이지	http://alloy.mit.edu/alloy4/	
2	도구명	Assertion Definition Language(상용)	주요 기능
	제작사	Sun	- C,C++ 프로그램을 위한 형식화 언어로 ADLT 도구를 이용하여 ADL 세부사항을 자동으로 시험
	지원언어	C/C++/Java	
	홈페이지	adl.opengroup.org/	
3	도구명	AST Toolkit	주요 기능
	제작사	Daniel Weise's group at Microsoft Research	- 프로그램 파싱 트리 분석
	지원언어	C/C++	
	홈페이지	http://research.microsoft.com/research/sbt/asttoolkit/ast.asp http://www.cs.washington.edu/home소프트웨어ei se/	
4	도구명	MOPS(MOdel checking Programs for Security)	주요 기능
	제작사	버클리 대학의 Open Source Quality 그룹	- C 소스 코드와 취약성을 각각 push down automata와 finite state automata 로 모델링하여 검사
	지원언어	C	
	홈페이지	http://www.eecs.berkeley.edu/~daw/mops/	

5	도구명	Blast(Berkeley Lazy Abstraction Software Verification Tool)	주요 기능
	제작사	The BLAST Team	- C 소스 코드와 취약성을 각각 control flow automata, monitor automata 로 모델링하여 검사 - lazy abstraction을 사용하여 C 소스 코드로부터 control flow automata를 생성하여 검사
	지원언어	C	
	홈페이지	http://mtc.epfl.ch/software-tools/blast/	
6	도구명	Bandera	주요 기능
	제작사	SAnToS Group at Kansas State University	- 자바 소스 코드를 Slicing 과 Abstraction Bindings 기법을 사용하여 추상화된 자바 코드를 생성한 후 Model Constructor를 사용하여 finite-state 모델을 생성한 후 PVS, Promela, SMV 와 같은 모델 체킹 도구를 사용하여 검증
	지원언어	Java	
	홈페이지	http://bandera.projects.cis.ksu.edu/	
7	도구명	SLAM Project	주요 기능
	제작사	Microsoft	- C 소스 코드를 분석하기 위하여 boolean program 으로 추상화하여 반복적인 정제(refinement)를 수행한 후 모델 체킹에 의하여 오류 추적
	지원언어	C	
	홈페이지	research.microsoft.com/slam/	
8	도구명	UPPaal	주요 기능
	제작사	Uppsala 대학과 Aalborg 대학	- timed automata 에 기초한 모델체킹
	지원언어	-	
	홈페이지	www.uppaal.com/	
9	도구명	SPIN	주요 기능
	제작사	Bell Lab	- CSP 에 기반을 둔 모델체킹
	지원언어	-	
	홈페이지	http://spinroot.com/spin/whatispin.html	
10	도구명	SMV 심볼릭 모델 체킹	주요 기능
	제작사	카네기 멜론 대학	- finite state machine 과 CTL 기반 모델체킹
	지원언어	-	
	홈페이지	http://www.cs.cmu.edu/~modelcheck/smv.html	

2. 안전한 소프트웨어 개발을 위한 자료 목록

□ 국외 가이드, 보고서

번호	제 목	발행기관	형태	발행년도
1	NIST: SP 800-64 Rev2, Security Considerations in the Information System Development Life Cycle - NIST의 위험관리 프레임워크 소개 - 소프트웨어 개발 라이프사이클별 보안 활동에 대해 소개하고 소프트웨어 보증, 서비스 지향 아키텍쳐, 재사용을 위한 보안 모듈 재사용 이슈 등에 대해 소개	NIST	가이드	2008
2	Software Security Assurance: A State-of-the-Art Report (SOAR) - 소프트웨어 보안성 보증관련 기관, 조직, 활동들에 대해 소개 - 소프트웨어에 대한 다양한 보안 위협을 분석하고, 여러 기관에서 채택하고 있는 소프트웨어 개발 라이프사이클에 대한 보호모델, 보호방법을 소개	IATAC, DACS	보고서	2007
3	Software Project Management for Software Assurance - 프로젝트 관리자가 프로젝트 관리 중에 소프트웨어 보증을 제공하는데 알아야 할 정보 제공 - 프로젝트 관리에 있어 소프트웨어 보증의 중요성을 소개하고, 소프트웨어 개발 계획수립, 추적 및 모니터링 단계에서의 위험 관리 방법과 개발 라이프사이클 과정에서의 보안활동에 대해 소개	DACS	보고서	2007
4	SECURITY IN THE SOFTWARE	DHS,	보고서	2006

	LIFECYCLE, Making Software Development Processes and Software Produced by Them - More Secure - 안전한 소프트웨어 개발 라이프사이클을 위한 프로세스 및 프로세스 개선 모델을 소개 - 소프트웨어 보안 체크리스트 및 안전한 개발 원칙을 제시 - 소프트웨어 보안성 향상을 위한 테스팅 기법 및 안전한 배포 방안 소개	CS&C, NCSD		
5	Secure Software Development Life Cycle Processes: A Technology Scouting Report - T-CMM/TSM(Trusted CMM/Trusted Software Methodology)와 SSECMM (Systems Security Engineering Capability Maturity Model)에 대해 소개하고, CMMI(Capability Maturity Model Integration)와 FAA-iCMM(Federal Aviation Administration integrated Capability Maturity Model)에 안전성 및 보안성을 제공하기 위한 방법을 소개 - 안전한 소프트웨어 개발을 위해 MS사에서 채택한 프로세스와 팀 소프트웨어 프로세스 등에 대해 소개	CMU/SEI	보고서	2005
6	Software Engineering for Secure Software - State of the Art: A Survey - 안전한 소프트웨어 개발을 위한 소프트웨어 공학의 최근 기술 동향 소개 - 소프트웨어 개발 단계에서 보안 요구사항을 만족시키기 위한 다양한 기술들에 대해 소개	퍼듀대학	보고서	2005
7	Improving Security Across the Software	NCSP	보고서	2004

	Development LifeCycle - 교육, 소프트웨어 프로세스, 패치관리, 동기부여 의 4개 서브그룹별로 소프트웨어 개발 라이프사이클 동안 소프트웨어의 보안성을 향상시키기 위한 방안 권고			
8	Knowledge of Software Suppliers needed to manage risks - 국외에서 개발된 무기체계 관련 소프트웨어를 미 국방성에 도입할 경우의 보안 취약점 이슈들을 제시하면서 위험관리의 필요성 권고	GAO	보고서	2004
9	Secure Software Development Lifecycle - 소프트웨어 제작자가 보안 소프트웨어를 제작하기 위해 따라야 하는 절차를 정의하고, 이러한 권고사항을 따르도록 동기를 부여하고 지원하는 방안 권고	NCSS	보고서	2004

□ 관련 도서

번호	도서명	출판사	출판년도
1	The CERT C Secure Coding Standard - 소프트웨어 개발자들이 자주 하는 실수와 안전하지 않은 코딩 유형을 정리하여 제공	Addison Wesley	2008
2	2008 Ultimate Secure Coding CD - 안전한 코딩을 위한 웹어플리케이션 보안, 버퍼오버플로우 공격의 탐지 및 예방, 역공학기술 등에 대해 소개	Syngress Publishing	2007
3	Fuzzing: Brute Force Vulnerability Discovery - 소프트웨어의 보안성을 시험하기 위한 퍼징(fuzzing) 시험방법에 대해 소개	Addison Wesley	2007
4	Secure Programming With Static Analysis - 소프트웨어 정적분석기술 및 안전한 소프트웨어 개발을 위한 정적분석 방법 소개	Addison Wesley	2007

5	Software Security: Building Security In - 위험관리 프레임워크 및 프로세스, 정적분석 도구를 이용한 코드검사, 침투 테스트, 보안취약성 테스트 등에 대해 소개	Addison Wesley	2006
6	The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities - 유닉스, 윈도우즈 시스템에서의 소프트웨어 보안 취약점 및 네트워크 어플리케이션 프로토콜에 대한 보안 취약점에 대한 식별, 감사, 보안성 평가 등에 대해 소개	Addison Wesley	2006
7	The Security Development Lifecycle - A Process for Developing Demonstrably More Secure Software - 소프트웨어 개발 라이프사이클 과정에서 소프트웨어의 보안성과 신뢰성을 향상시키기 위한 방법과 MS사에서 채택하고 있는 표준 개발 프로세스에 대해 소개	Microsoft Press	2006
8	19 Deadly Sins of Software Security - 다양한 개발언어에 대해 소스코드의 보안 취약점을 탐지하고 개선하는 방법에 대해 자세한 예를 들어 설명	McGraw-Hill	2005
9	Secure Coding in C and C++ - 버퍼오버플로우, 스트링 취약성, I/O 취약성 등의 소프트웨어의 보안 결함을 예방할 수 있는 방법에 대해 예를 들어 설명	Addison Wesley	2005
10	Exploiting Software: How to Break Code - 소프트웨어 역공학 방법 및 서버/클라이언트 소프트웨어에 대한 다양한 보안공격 사례 소개	Addison Wesley	2004
11	Secure Coding: Principles and Practices - 보안 아키텍쳐 및 설계, 구현과정에서의 모범사례 및 잘못된 사례를 소개하고 테스팅 및 위험 평가 방법론등을 소개	O'Reilly	2003
12	Writing Secure Code, Second Edition	Microsoft	2003

	- 안전한 소프트웨어 개발 라이프사이클을 위한 방법, 위협모델, 메모리 관련 취약점 예방 및 접근통제 권한설정, ActiveX/DCOM 등에 대한 보안, 안전한 소프트웨어 설치 등에 대해 소개하고, 보안 체크리스트를 제공	Press	

이 가이드 작성을 위하여 다음과 같은 분들께서 수고 하셨습니다.

2008년 12월

총괄 책임자	방송통신위원회 네트워크안전과	과장	윤혜주
	한국정보보호진흥원 IT기반보호단	단장	원유재
사업 참여자	방송통신위원회 네트워크안전과	사무관	황큰별
	한국정보보호진흥원 u-IT서비스보호팀	팀장	이완석
	한국정보보호진흥원 u-IT서비스보호팀	수석연구원	김호성
	한국정보보호진흥원 u-IT서비스보호팀	수석연구원	강필용
	한국정보보호진흥원 u-IT서비스보호팀	선임연구원	신동훈
	한국정보보호진흥원 u-IT서비스보호팀	연구원	이익섭
	한국정보보호진흥원 u-IT서비스보호팀	연구원	한재홍
집 필 자	호서대학교	교수	하재철
	숭실대학교	교수	이정현
	컴퓨터프로그램보호위원회	박사	신동명
	삼성 SDS	책임	노윤재
	LG데이콤	기술사	구자현
	NHN	과장	임만기
감 수	단국대학교	교수	조성제
	상명대학교	교수	한혁수
	한국소프트웨어진흥원	팀장	이혁재
	어울림정보기술	이사	김형률
	마이어스테크놀러지	이사	홍필한
	한국정보보호진흥원	선임	안성수
	한국정보보호진흥원	선임	손경호

안전한 소프트웨어 개발·도입을 위한 보안 가이드 V1.0

2008년 12월 인쇄
2008년 12월 발행

- 발행인: 황 중 연
- 발행처: 한국정보보호진흥원
  서울시 송파구 중대로 135(가락동 78)
  IT벤처타워(서관)
  Tel: (02) 405-5114
- 인쇄처: 한올 인쇄사
  Tel: (02) 2279-8494

<비매품>