제1회 정보통신표준화 우수논문집

03 장려상 / 일반부문

IT839전략의 안전한 실현을 위한 소프트웨어 보안 표준

The Software Security Standards for the Secure Realization of IT839 Strategy

김홍근 / 한국정보보호진흥원

HongGeun Kim / Korea Information Security Agency

IT839전략의 안전한 실현을 위한 소프트웨어 보안 표준

The Software Security Standards for the Secure Realization of IT839 Strategy

김홍근 / 한국정보보호진흥원
HongGeun Kim / Korea Information Security Agency

요 약

네트워킹, 상호 운용, 통합, 스마트 디바이스, 지능형, 유무선 서비스, 멀티미디어 콘텐츠 등의 IT839 키워드에 존재하는 공통 분모는 소프트웨어이다. IT839의 진전은 소프트웨어의 크기와 양을 증가시키고, 이에 비례하여 보안 취약점의 양도 증가시키고 있다. 보안 취약점을 감소시키기 위한 개발 프로세스의 보안 프랙티스(practice) 표준화는 소프트웨어 보안의 출발점이다. 소프트웨어 자가 방어를 위한 실행시간 침입탐지 모듈 표준, 소프트웨어 보안 취약점 위험도 등급 표준, 보안 취약점 정보 관리 절차 표준, 부정 소프트웨어에 대한 사용자 대응을 지원하기 위한 부정 소프트웨어 데이터베이스와 소프트웨어 설치와 제거 표준 등은 소프트웨어 집약 시스템으로서의 IT839 서비스와 제품의 안전한 구현을 위하여 우선적으로 추진해야할 과제이다. 본 논문에서는 IT839 소프트웨어의 보안을 위한 표준화 주제들을 발굴하고 그 필요성을 제시한다.

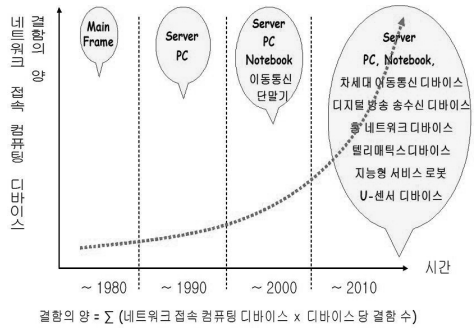
Ⅰ. 서론

유비쿼터스 네트워킹 환경의 구축과 "사람 · 사물 · 환경 속에 컴퓨터 기능이 내재된 디지털 컨버전스의 실현"에 따라 경제적 이익과 사회적 편리함이 증대될 것이다. 그러나 네트워크의 융합에 따라 인터넷을 이용한 사이버 공격으로 인한 피해가 방송망, 통신망으로 확대될 수 있으며, 다양한 디지털 이동기기 전반에 걸쳐 전파되는 악성 코드의 유포, 유비쿼터스 센서에 의한 개인정보의 수집과 오 · 남용으로 인한 프라이버시 침해 등이 사회적인 이슈로 등장할 것으로 예상된다. 또한, 통신 인프라와 생활 가전을 연동하는 홈 네트워크 서비스가 활성화됨에 따라 홈 네트워크 정보단말, 가전기기, 센서 등에 대한 보안공격이 예상되며, 이는 일상생활에 심각한 위협을 미칠 가능성이 있다. 따라서 IT839 전략에 의한 유무선 통합 환경과 디지털 컨버전스 서비스의 안전한 구현을 위하여, 보안 위협과 취약점에 대한 분석과 이에 대한 적절한 보안 대책이 강조되고 있다[1].

네트워킹, 상호 운용, 통합, 스마트 디바이스, 지능형, 유무선 서비스, 멀티미디어 콘텐츠 등의 IT839 키워드에 존재하는 공통 분모는 소프트

웨어이다. 오늘날 취약한 소프트웨어는 인체의 암과 같이 악성 코드에 감염되어 건강한 소프트웨어에 피해를 입힌다. 감염된 소프트웨어는 자기 복제를 하고, 네트워크를 타고 다른 시스템으로 전파된다. 2003년 1월에 발생한 MS SQL 슬래머 웜은 인터넷 서비스를 마비시켜 전자거래 등의 중단 사태를 초래하였다.

지금까지 컴퓨터와 네트워크에서 발생한 보안침해는 모두가 그들 시스템이 수행하는 소프트웨어의 문제라고 할 수 있다. 다시 말해 모든 시스템의 보안 문제는 그들 시스템들이 수행하는 프로그램인 소프트웨어의 보안을 통하여 근원적인 문제의 해결책을 찾을 수 있음을 의미한다. 그러나 불행하게도 운영중인 소프트웨어들은 수많은 설계 오류들과 구현상의 버그들을 내포하고 있어서 이들에 의한 보안상의 위험은 예측이 불가능하다. “Firewalls and Internet Security”을 저술한 Steve Bellovin은 “Any program, no matter how innocuous it seems, can harbor security holes[2]”이라는 표현을 통하여 진정으로 안전한 소프트웨어의 필요성을 역설하였다. 미국 CMU 소프트공학연구소 SEI의 조사[3]에 의하면, 소프트웨어 보안 사고의 90% 이상이 알려진 보안 결함을 악용하여 발생하며, 45개의 e-비즈니스 응용 프로그램의 분석 결과, 보안 결함의 70%가 설계 오류로부터 발생하였다. 또한 경험과 능력을 가진 소프트웨어 엔지니어도 코드의 9라인마다 평균 한 개의 결함을 발생시키고, 소프트웨어가 출시될 무렵, 백만 라인 길이의 코드에는 약 1,000개~5,000개 사이의 결함이 내포된 것으로 추정한다. 이 모든 결함이 보안 취약점으로 연결되는 것은 아니지만, 이중에 단 1%만이라도 보안 취약점을 허용한다면, 백만 라인 길이의 코드에 10개~50개의 보안 취약점이 존재한다는 것을 의미한다.



(그림 1) IT839 진전에 따른 보안취약점의 증가

(그림 1)에서 보는 바와 같이 IT839의 진전은 소프트웨어의 크기와 양을 증가시키고, 이에 비례하여 보안 결함의 양도 증가시킬 것이다. 상기에서 살펴본 바와 같이 IT839의 안전성을 높이기 위해서는 소프트웨어의 보안이 중요하다. 특히 소프트웨어의 보안 결함을 감소시키기 위하여 개발자와 사용자가 실천해야 하는 보안 프랙티스(practice)의 표준화는 소프트웨어 보안의 출발점이다. 본 논문에서는 IT839 전략의 성공적인 달성을 위한 소프트웨어 보안의 중요성과 보안 표준화 이슈를 살펴본다. 2장에서는 IT839의 실현에 따른 소프트웨어의 특성과 소프트웨어 보안에서의 보안 취약점의 중요성을 살펴보고, 3장에서는 소프트웨어 보안 취약점 감소 등을 위한 시급한 표준화 이슈 등을 제안한다.

Ⅱ. IT839와 소프트웨어 보안

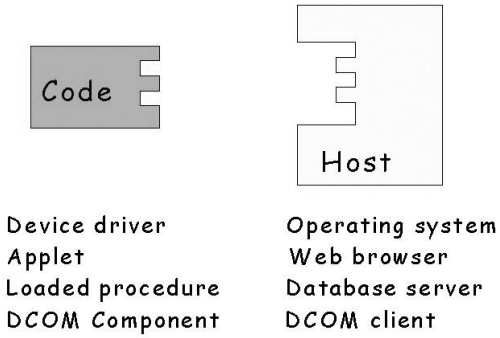
1. IT839, 소프트웨어, 보안

IT839의 진전과 함께 소프트웨어 형태도 변화할 것이다. 소프트웨어의 특징과 이에 따른 보안 취약점을 <표 1>에 정리하였다. 첫째, 스마트 디바이스, 지능형 디바이스를 위한 소프트웨어의 역할이 증가할 것이다. 보통의 하드웨어 디바이스에 컴퓨팅 유닛을 접목한 스마트 오브젝트에서부터 인공지능 소프트웨어를 탑재한 지능형 디바이스가 보편화할 것이다. 이들 지능형 디바이스에 네트워크 접속 기능을 내장하면 네트워크 형 DTV, 텔레매틱스 터미널, 홈 네트워크 터미널, USN 디바이스 등이 만들어진다. 둘째, 가상 머신(Virtual Machine) 사용의 확산에 따른 확장 가능한 소프트웨어(그림 2) 모듈이 증가할 것이다. 다양한 플랫폼에 동일한 코드를 실행할 수 있게 하는 가상 머신이 널리 보급되어, 전기와 같이 단지 플러그를 꽂는 것으로 컴퓨팅이 가능해 진다. 이는 컴퓨팅의 아웃소싱을 의미하며, 이에 따른 보안 우려도 발생할 수 있다. 셋째, 네트워킹의 보편화로 인해, 대부분의 코드가 모바일 코드가 될 것이다. 이는 네트워크가 곧 컴퓨터인 시대를 의미하며, 모든 코드가 네트워크를 기반으로 작동될 것이다. 소프트웨어를 구현하는 프로그래밍 언어 기반의 보안 모델이 보다 중요해지고, 여기에 대한 공격도 증가할 것이다. 넷째, 모든 소프트웨어 기능을 한꺼번에 설치하는 것이 아니라, 소프트웨어 기능이 필요에 따라 배달되고 설치되는 선택적 소프트웨어[4]에 대해, 사용하는 기능에 따라 요금이 부과될 것이다. 따라서 사용자에 대한 인증과 사용기록 인증에 대한 수요가 증가할 것이다.

많은 IT839 서비스와 제품은 소프트웨어 집약(Software Intensive) 시스템이다. 소프트웨어 집약 시스템의 보안은 소프트웨어의 보안 품질에 직접 영향을 받는다. 또한 IT839의 실현은 일상생활에서 발생하는 데이터의 집적화를 광범위하고 폭발적으로 증가시킬 것이다. 사이버 공간에서 이루어진 행동은 어떤 경우이든 기록되어 집적화 될 수 있다. 예컨대 구매한 식료품, 잠자는 시간, 요리한 음식, 처방 받은 의약품, 시청한 멀티미디어 목록 등 방대한 정보들이 기록될 것이다. 집적된 데이터를 보호하는 것은 소프트웨어이다. 파일 시스템, 데이터베이스 시스템 등의 인터페이스를 통해 데이터를 접근하게 된다. 따라서 집적 데이터의 보호는 소프트웨어의 보안이 관건이다.

<표 1> IT839 소프트웨어의 특성과 보안 취약점

소프트웨어 특성	보안 취약점
내장형 스마트 지능 소프트웨어	부정확한 디바이스 인증 취약점, 패치 관리의 어려움
확장 가능한 소프트웨어	불완전한 가상머신의 보안 메커니즘의 취약점
네트워크 기반 모바일 소프트웨어	이동 코드 인증 취약점
선택적 소프트웨어	사용자 위장, 과금 우회 취약점, 불법 복제 취약점

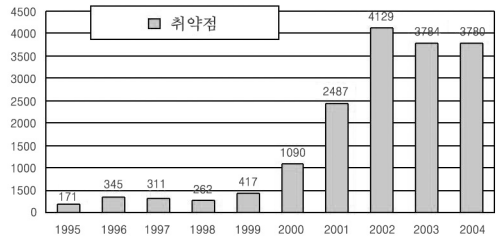


(그림 2) 확장 가능한 소프트웨어[5]

2. 소프트웨어 보안 취약점의 증가

소프트웨어 보안 취약점이란 소프트웨어의 기능 명세, 설계 또는 구현 단계의 오류나 시동, 설치 또는 운영상의 문제점으로 인하여 지니게 되는 보안 결함으로 정의된다[6]. 공격자가 비정상적인 권한의 획득으로 인해 정보 유출, 조작, 파괴 등이 발생하며, 정상적인 사용을 방해하는 등의 보안 침해를 허용하는 통로이다. 운영체제, 응용 프로그램 등의 소프트웨어에는 다양한 설계 및 구현상의 알려지지 않은 보안 취약점이 존재한다. 소프트웨어에 존재하는 보안 취약점은 보안 침해 사고의 주요한 원인을 제공하므로 소프트웨어의 개발과 이용 과정에서 소프트웨어의 보안 취약점을 식별하고 제거하여, 공격의 여지를 최소화한 소프트웨어를 개발하거나 이용하기 위한 절차, 기술, 도구 등이 필요하다. 즉, 공격자가 보안 취약점을 악의적으로 이용하기 전에 개발자 혹은 사용자가 먼저 보안 취약점을 발견하여 제거할 수 있어야 한다. 공격자는 가장 취약한 링크를 악용하므로 이러한 노력은 모든 소프트웨어에 표준적으로 적용되어야 한다.

(그림 3)은 미국의 CMU 대학교의 CERT/CC에서 집계한 소프트웨어 보안 취약점의 연도별 증가를 보여주고 있다[7]. 2005년 상반기 6개월 동안에 발견된 취약점 수는 2,875개로 집계되어, 2005년도는 가장 많은 취약점이 발견된 연도로 기록될 전망이다.



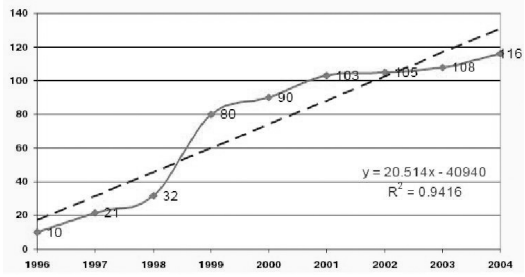
(그림 3) 소프트웨어 보안 취약점 수의 증가

소프트웨어 보안 취약점 증가 원인의 중심에는 소프트웨어 크기의 증가가 있다. 문서편집기나 웹 브라우저와 같은 대중적 소비시장을 갖는 소프트웨어 제품이 보여왔던 진화 과정을 돌이켜보면, 한마디로 그 크기가 커졌다는 것이다. 모든 소프트웨어가 그렇듯이 버전업 과정을 통해 기능이 향상되고 추가되어야 시장에서 경쟁할 수 있었다. 크기의 증가와 함께, 다양한 기능과 기술의 추가로 인해 코드의 복잡도가 증가하였으며, 이에 따라 그만큼 결함도 증가하게 되었다.

소프트웨어 산업이 시작된 이래로 소프트웨어 제품 개발 시의 보안에 대한 고려는 부족하다. 오늘날에도 최초 설계에서부터 보안을 고려한 제품을 만나기란 여전히 어렵다. 상업용 소프

트웨어 벤더들은 그들이 개발하여 배포하는 소프트웨어 속에 이미 보안 문제로 알려져 있는 버그들을 여전히 포함시킨 채로 공급하고 있다. 예를 들어 1988년에 발생한 "인터넷 웜" 사건에서 사용된 "버퍼 오버플로(buffer overflow) 취약점"은 그 보안상의 문제점이 널리 알려졌음에도 불구하고, 계속해서 새로운 소프트웨어에 포함되고 있다. 보안 경험과 사고를 가진 프로그래머는 전체 소프트웨어 인력에서 극소수를 차지하고 있다. 현재 소프트웨어 보안 연구에 참가하는 과학기술자 수가 적기 때문에, 연구를 통해 얻은 결과 및 경험 역시 적고 이를 바탕으로 학생들을 교육시키는데 어려움이 존재하게 된다. 즉, 학생들은 소프트웨어 보안기술에 대한 충분한 경험적 지식과 강의 자료를 가지고 학습하지 못하기 때문에 양질의 소프트웨어 보안 교육을 받을 수 없다. 이와 같이 양질의 보안기술 교육을 받지 못한 학생이 졸업 후 소프트웨어 시스템을 개발할 때 보안 취약점이 포함된 소프트웨어가 개발되는 것이다.

보안 취약점의 증가에 따라 침해의 수준과 다양성, 전체 침해의 수가 증가하고 있다. 미국의 ICSA Lab에서 보고한 1996년부터 2004년까지의 바이러스 감염 조사 자료[8]에 의하면, 컴퓨터 바이러스의 감염율이 매년 증가하고 있음을 볼 수 있다. 자료의 조사 대상이 미국 기업 보안 담당자에 국한한 것이기는 하나, 한국에서의 사정도 다르지 않다. 한국정보보호진흥원 KISA의 2004년 "정보시스템 해킹바이러스 현황 및 대응 보고서"에서, 2004년 일 년 동안 국내 백신업체와 KISA에 접수된 웜 · 바이러스 피해 신고는 총 107,994건으로, 2003년 85.023건에 비하여 27% 증가하였다.



(그림 4) 천대의 PC에 대한 월간 바이러스 감염 횟수 (ICSA Lab[8])

〈 표 2〉 국내 웜 · 바이러스 피해 신고 건수 (KISA

구분	2002년	2003년	2004년
피해신고 접수건수	38,677	85,023	107,994

Ⅲ. 소프트웨어 보안 표준 이슈

1. 소프트웨어 보안 취약점 감소를 위한 개발 과정에서의 프랙티스 표준

소프트웨어 보안 취약점이 발생하는 주요 원인은 개발 과정에서의 기능명세, 설계 또는 구현 단계에서 의도하지 않은 오류에 있다. 소프트웨어의 개발과정에서 소프트웨어의 보안 취약점을 식별하고 제거하여, 공격의 여지를 최소화한 소프트웨어를 개발하기 위해 표준화된 절차, 기술, 도구 등이 필요하다. 또한 소프트웨어 수명주기(life cycle)의 전반부에서 보안 결함을 찾

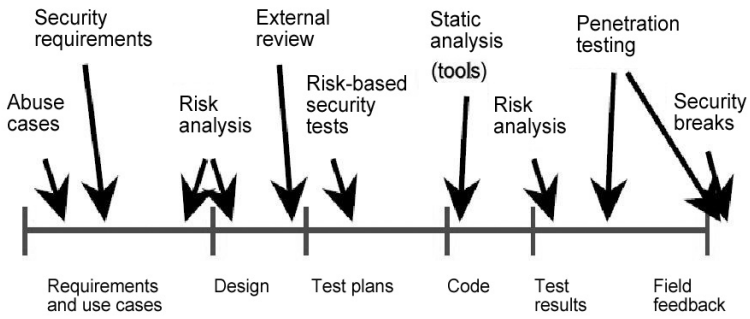
아 수정하였을 경우, 수명주기 후반부에서 보안 결함을 찾아내는 것 보다 많은 비용과 시간을 절약할 수 있다. 예를 들면 요구분석 단계에서 결함을 수정하는 비용과 시간이, 개발 후 결함을 수정하는 비용과 시간에 비해 100배 정도의 비용과 시간을 절감할 수 있다[10]. 이는 소프트웨어의 보안 취약점을 감소시키기 위한 노력은 개발 과정에 집중되어야 한다는 것을 의미한다. 따라서 소프트웨어 집약적인 IT839 서비스 및 제품의 보안의 성패는 소프트웨어 보안 취약점을 최소화 하는 노력에 달려있다. 기능으로 요구되는 보안 기능(인증, 보안감사, 접근통제 등)이 설계명세에 반영되는 것도 중요하지만, 일반 기능이든 보안 기능이든 소프트웨어로 구현된다면 어느 경우든지 보안 취약점을 포함할 수밖에 없다.

개발 단계에서 보안 취약점을 감소시키기 위한 프랙티스를 개발하기 위한 노력은 크게 2가지로 분류할 수 있다. 첫째는 소프트웨어 개발 공정에서의 프랙티스를 개선하여, 소프트웨어의 결함을 감소시키고, 궁극적으로 보안 취약점을 줄이는 것이다. CMU 소프트웨어공학연구소의 TSP(Team Software Process)[3]를 적용하여 개발된 소프트웨어의 결함은 프로그램 1000라인 당 평균 0.06개로 조사되었다. 수학적 모형과 형식 논리를 소프트웨어의 사양, 설계, 구현, 시험 등에 적용하는 형식적 방법론(formal method)[9]은 분명 결함을 감소시키기는 하지만 효과성에 대해서는 아직 결론이 나지 않은 상태에 있다. 하드웨어 정밀공정을 위한 청정실(cleanroom) 개념과 유사한 청정실 소프트웨어 공학을 적용하면, 1000라인 당 0.1개 결함을 가지는 것으로 조사되었다. SSE-CMM(System Security Engineering Capability Maturity Model), iCMM(integrated Capability Maturity Model) 등과 같은 공정 모델(Process Model)을 적용하여 소프트웨어 설계와 구현 상의 결함을 감소시킬 수 있다[10].

소프트웨어 보안 취약점은 코드 상의 입력검증 실패, 조건문 오류, 구성 오류 등으로부터 발생하기도 하지만, 부적절한 인증, 권한 인가의 실패, 잘못 구현된 암호 알고리즘, 데이터 보호의 누락 등으로부터 발생하기도 한다. 전자의 경우에는 소프트웨어 개발 공정에서의 프랙티스를 개선하는 것으로 상당 부분 해결된다. 후자와 같은 보안 취약점을 해결하기 위한 별도의 보안 프랙티스가 개발되어 표준적으로 실천되어야 한다. 일례로 위협 모델링(Threat Modeling)은 침해를 가할 위협을 식별하고, 정의된 위협의 등급에 따라 위협 완화 전략을 개발하여, 설계?구현?시험 단계에 반영하는 보안 분석 방법론이다. (그림 5)는 McGraw 등[11]이 제시한 소프트웨어 개발 공정에서 보안 취약점을 감소시키기 위한 프랙티스들을 나열한 것이다.

현재까지 소프트웨어 개발 전반에 대하여 최적화된 보안 프랙티스(best practice)는 존재하지 않는다. 일부 단계에서의 프랙티스가 존재하기는 하지만, 소프트웨어 개발 현장에 적용되어 그 효과성을 널리 입증받은 프랙티스는 없다. IT839가 국가 경제 질서와 국민의 안녕에 지대

한 영향을 미친다면, 국가는 개발자들이 준수해야 할, 소프트웨어 보안 취약점 감소를 위한 최적의 보안 프랙티스를 개발하고 표준화를 주도해야 한다. 단기적으로는 국내외의 제시된 방법론과 이에 따른 프랙티스의 효과성을 검증하여 표준화하고, 중장기적으로는 전 분야의 표준 프랙티스를 개발하는 프로젝트를 추진해야 할 것이다.



(그림 5) 소프트웨어 개발 단계에서의 보안 취약점 감소를 위한 프랙티스[11]

2. 소프트웨어 실행시간 침입탐지 모듈 표준

대부분 일정 규모 이상의 소프트웨어의 경우에 보안 취약점을 완전히 제거할 수 없다. 존속하는 보안 취약점을 공격자가 발견하여 악용할 수 있는 가능성은 언제나 있다. 따라서 소프트웨어가 보안 정책을 위반하거나 비정상적인 행위를 보이면, 실행시간 행위제어를 통해 시스템의 안전성을 보장하여야 한다. 이러한 행위 제어를 위해서 소프트웨어의 실행 궤적을 추적하여 비정상 또는 보안 침해 행위를 판단하기 위하여 침입탐지시스템[12]이 사용된다. 기존의 일반적인 호스트 기반 침입탐지시스템은 하나의 탐지 시스템이 호스트 내에서 실행하는 다수의 소프트웨어를 감시한다. 그러나 이러한 방식은 운영체제에 추가적으로 포함시키거나, 응용 프로그램으로 설치되어 감시 대상 개별 소프트웨어의 세세한 설계 특성을 반영하지 못한다.

소프트웨어의 복잡도가 높아질수록, 소프트웨어의 자가 치료(self-healing) 또는 자가 방어(self-protecting) 기능에 대한 요구가 높아지고 있다[13]. 이는 복잡도의 증가에 따른 필연적인 결함이나 공격의 증가에 대응하기 위하여, 실행 중의 문제점을 발견하고, 진단하여 수리하는 능력을 소프트웨어 스스로 가져야 한다는 것을 의미한다. 보안의 측면에서, 소프트웨어의 보안 취약점을 악용하여 보안 침해를 일으키는 상황을 방지하기 위하여 별도의 침입탐지 시스템을 설치하는 대신, 소프트웨어의 기본 기능으로 침입탐지 모듈을 장착하여 스스로의 실행 궤적을 감시하고, 비정상 행위가 발생하면 시정하는 능력을 갖는 것을 의미한다. 시스템 전체를

감시하는 침입탐지시스템이 아니라, 개별 프로그램 별로 침입탐지 모듈을 내장하는 방식이다. 이는 리눅스 운영체제에 적재가능한 커널 모듈(loadable kernel module) 형태로 침입탐지 기능을 구현하는 방식과 유사하다. 실행시간 행위 제어를 위한 침입탐지 기능을 정의하고, 어떤 응용 소프트웨어에도 내장할 수 있도록 모듈을 표준화하여, 소프트웨어 개발 과정에서 침입탐지 모듈을 용이하게 포함하도록 해야 할 것이다.

3. 소프트웨어 보안 취약점 위험도 표준

소프트웨어는 개발 과정과 운영 과정에서 공격자가 악용(exploit)할 수 있는 보안 취약점을 포함하고 있다. 소프트웨어의 보안 취약점은 발견되면, 해당 소프트웨어 벤더에게 비밀리에 신고되거나 인터넷에 공개된다. 소프트웨어 벤더는 인터넷으로부터 수집된 취약점 정보 또는 신고된 취약점 정보를 자체적으로 검증하여 취약점의 존재 사실을 확인하고, 취약점의 위험도를 산정하여 패치 파일의 개발 우선순위를 정하고, 이에 따라 패치를 개발한다. 보안 취약점 패치 파일은 기술적인 세부 사항을 설명한 보안 게시판(security bulletin) 또는 보안 권고문(security advisory)에 포함되어 정기적 혹은 비정기적으로 공개된다. MS, Oracle, Cisco 등의 소프트웨어 벤더는 정기적으로 보안 권고문을 발표한다.

BugTraq 등의 인터넷 메일링 리스트나 제품 벤더가 발표한 보안 게시판 정보를 모니터링 하여 공개된 보안 취약점 정보를 수집하고, 자체적인 테스트 · 검증 · 확인 작업을 거쳐, 보안 취약점 정보를 벤더 중립적이며 신속하게 보안 커뮤니티에게 제공하는 웹사이트에서도 소프트웨어 벤더와 유사한 보안 취약점의 권고문을 발표한다. 영리 단체인 Secunia, ISS X-force, SecurityFocus 등이 운영하는 웹사이트, 미국 정부의 지원을 받는 비영리 조직이 운영하는 Mitre(CVE), NIST(ICAT) 웹사이트 등이 대표적이다. 미국 정부가 운영하는 US-CERT 웹사이트에서도 국가적으로 중요한 취약점에 대한 보안 권고문을 발표한다.

보안 권고문은 사용자에게 패치의 설치 여부와 우선순위를 결정하는데 필수 정보이므로 매우 중요하다. 보통 보안 권고문에는 교정한 보안 취약점에 대한 위험도(Risk 또는 Severity) 등급을 표시하여, 취약점을 패치하지 않았을 경우의 위험 정도를 사용자에게 알린다. 그런데, <표 3>에서 보는 바와 같이 동일한 취약점에 대해서 보안 권고문에 표시된 위험도 등급이 상이하고, 보안 권고문 작성 주체에 따라 표시된 위험도 등급 체계가 달라서 사용자에게 혼란을 준다. 사세르(Sasser) 웜에 의해 악용된 CAN-2003-0533 취약점(MS 운영체제의 Local Security Authority Subsystem Service에서의 버퍼오브플로 취약점)에 대한 보안 권고문을 비교하여 보면, MS는 'Critical' 등급을 부여하였으나, US-CERT의 보안 권고문에는 비교적 낮은 점수인 35.44로 표기하고 있다. US-CERT의 보안 권고문은 CAN-2003-0567 취약점

(Cisco IOS에서의 서비스거부공격 취약점)에 대한 위험도 점수를 73.50으로 표기하여, CAN-2003-0533의 취약점 점수 33.44 보다 훨씬 더 위험한 것으로 판단하고 있다. CAN-2003-0567 취약점은 US-NIST ICAT MetaBase에서 'Medium', Secunia 보안 권고문에서는 'Moderately Critical'로 표기하여 위험도가 그리 심각하지 않은 것으로 표기하고 있다. 반면 ISS X-Force에서는 'High'로 위험도 등급을 표시하였다.

〈표 3〉 보안 권고문에 표시한 보안 취약점 위험도에 대한 상이한 표현 사례

보안권고문 주체 / 취약점 (CVE 이름)	MS 보안 게시판	시스코 보안 권고문	시만텍 보안 권고문	US-CERT 취약점 노트	US-NIST ICAT 메타베이스	ISS X-Force 데이터베이스	Secunia 보안 권고문
CAN-2003-0533	Critical	N/A	N/A	35.44	High	High	Highly critical
CAN-2003-0567	N/A	–	N/A	73.50	Medium	High	Moderately critical
CAN-2004-0487	N/A	N/A	Medium	3.94	High	High	Moderately critical

제품 벤더 Cisco는 보안 권고문에서 위험도 등급을 표시하고 있지 않다. Cisco를 포함하여 Sun, IBM 등의 메이저 벤더가 발표하는 보안 권고문에는 취약점에 대한 위험도를 제공하지 않는다. CAN-2004-0487 취약점(Norton AntiVirus 2004 ActiveX Control 취약점)은 제품 벤더인 Symantec의 보안 권고문에는 'Medium'으로 위험도 등급을 표시하였으나, NIST ICAT, ISS X-Force에서는 높은 등급, Secunia에서는 중간 등급, US-CERT에서는 낮은 등급 등으로 제각각 다른 위험도 등급을 부여하고 있다.

현재 소프트웨어 벤더가 부여하는 취약점의 위험도는 예외 상황을 제외하고, 보안 권고문의 수명주기 동안 일정하게 유지된다. 해당 취약점에 대한 악용 코드나 인터넷 웜이 공개되어 취약점에 대한 악용 가능성이 높아져 취약점에 대한 위험도가 증가하여도 고정된 값을 유지한다. 따라서 보안 취약점에 가해지는 동적인 위험 수준을 나타내는데 부족하다.

위에서 살펴본 바와 같이, 보안 취약점 위험도 등급 산정이 표준화되지 않아서 사용자 혼란을 가중시키는 중요한 문제로 부각되고 있다[17]. 현재까지 제품 벤더와 보안연구자 모두가 공통적으로 인정하는 취약점 위험도 산정 방식이 존재하지 않는다. 시스템에 여러 벤더의 소프트웨어를 설치 운영하는 사용자나 시스템 관리자 입장에서는 객관적이고 중립적인 취약점 위험도 정보가 필요하다. 왜냐하면 보안 취약점의 위험성에 대한 이해를 높이고, 패치 파일의 우선순위와 설치 일정을 수립하는데 취약점의 위험도 등급이 필수적이기 때문이다. 또한 언론의 입장에서는 제품 벤더가 발표한 보안 패치의 긴급성과 우선순위를 알리는데 취약점의 위험도 등급이 유용한 지표 역할을 한다. <표 4 >는 보안 취약점의 위험도를 결정하는 요인들의 예시이다.

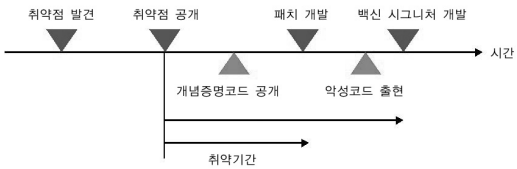
〈 표 4〉 보안 취약점의 위험 등급을 결정하는 세부 항목

항 목			설 명
악용 영향	원격코드 투입	시스템권한으로 실행	외부에서 투입한 코드를 시스템(루트) 권한으로 실행
		사용자 또는 제한된 권한으로 실행	외부에서 투입한 코드를 사용자 또는 제한된 권한으로 실행
		악성코드 저장	특정 영역에 트로이목마, 바이러스 등의 악성코드 저장 (재 부팅으로 자동 실행될 수 있음)
	권한 변경	시스템 권한	시스템 권한으로 상승
		사용자, 제한된 권한	사용자 또는 제한된 권한 획득
	정보 노출	보안 정보	패스워드, 암호 키 등과 같은 보안 관련 정보 노출
		시스템 정보	코어 파일, 로그 파일 등의 시스템 운영에 관련된 정보 노출
		사용자 정보	사용자 파일에 대한 접근이나 DB 액세스를 허용
		간접 정보	악용에 간접적인 시스템 또는 사용자 정보 노출
	서비스 거부	시스템 크래쉬	해당 취약점이 존재하는 소프트웨어가 실행되는 시스템 중단
		소프트웨어 크래쉬	해당 취약점이 존재하는 소프트웨어 중단
		자원 소모	해당 취약점이 존재하는 소프트웨어가 실행되는 시스템의 자원 소모
악용 복잡도	접근 위치	원격	인터넷에서 접근 가능
		지역	같은 조직 내, 보안 영역 내부의 시스템에서 접근
	인증	필요 없음	시스템 또는 소프트웨어에 대한 인증이 필요 없음
		필요	시스템 또는 소프트웨어에 대한 인증이 필요
	활성화	디폴트 실행	디폴트 설정
		사용자 실행	사용자 또는 관리자에 의한 수동 설정 또는 활성화, 특수한 설정
	사용자 개입	필요 없음	사용자의 개입 없이 공격 가능
		필요	공격자와 사용자의 상호작용이 필요
악용 가능성	악용코드 공개	자동 공격 도구	자동화된 코드(웜, 바이러스) 활동, 코드 필요 없음
		작동가능코드	기능적인 exploit 코드가 공개(대부분 작동가능)
		개념증명코드	개념증명 코드가 공개
		공개되지 않음	악용코드가 공개되지 않음
	악용 사례 보고	있음(다수)	악용 사례가 벤더, 보안회사, CERT 등에 단시간 내에 다수 보고
		있음(소수)	악용 사례가 벤더, 보안회사, CERT 등에 소수 보고
		없음	악용 사례가 보고되지 않음
악용 방어 수단	–	네트워크 차단	port 봉쇄 등의 라우터 또는 방화벽 기본 설정으로 악용 차단
		구성설정 변경	소프트웨어의 실행 옵션을 조정, 레지스트리 값 변경
		제한된 실행환경	제한된 실행 환경에서의 해당 소프트웨어의 실행
		보안 시스템	시스템/네트워크 보안 시스템의 탐지 패턴 존재

4. 취약점 정보 관리 절차 표준

공격에 이용될 수 있는 소프트웨어 보안 취약점이 알려지면, 다음과 같은 두 가지 대응 중에서 한가지가 이루어질 때까지 취약한 소프트웨어에 대한 공격은 지속된다. 취약한 소프트웨어를 만든 벤더가 취약점의 패치를 배포하고, 시스템 관리자나 사용자가 패치를 시험 · 검증하고 설치하거나, 앤티-바이러스 소프트웨어 회사들이 취약점을 이용하는 공격 코드나 악성 코드에

대한 탐지 패턴을 개발하고, 사용자 컴퓨터의 항-바이러스 소프트웨어의 탐지 패턴 DB에 반영한다. 이와 같은 적절한 대응방법 없이 보안 취약점 정보가 공개되면, 해킹에 악용될 가능성이 있으므로 사용자들에게 피해를 줄 위험성이 존재한다. (그림 6)에서 보는 바와 같이, 취약점의 공개 후에 통상 1~2주 내에 개념증명코드(Proof of Concept Code)가 발표되고, 이 이후에 실제적인 공격 코드인 웜 또는 바이러스가 출현한다. 따라서 공격이 증가되는 주요한 원인으로 취약점 공개의 증가를 주목해야 한다.



(그림 6) 취약점 공개와 취약기간

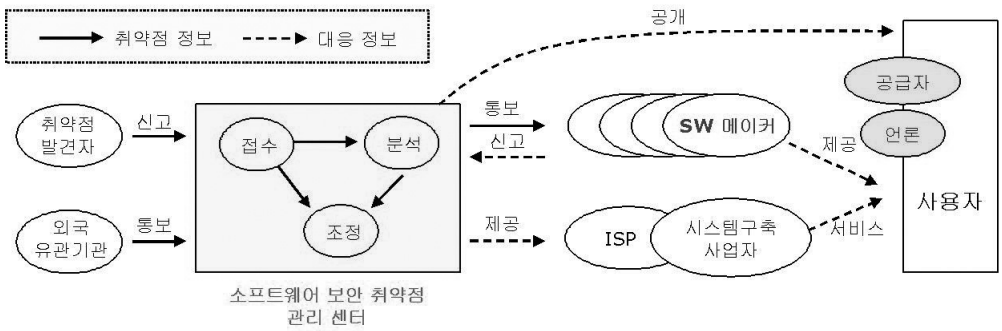
따라서 제품 개발자가 취약점 회피 또는 수정 방법을 결정할 때까지는 취약점이나 공격 방법이 인터넷상에 유통되는 것을 차단하도록 하는 것이 중요하다. 취약점 정보가 공개되지 않도록 강제하는 것은 현실적으로 불가능하지만, 공개되는 취약점 정보의 양을 제한하기 위한 제도적 접근이 필요하다. 따라서 다양한 기기의 소프트웨어 및 웹 어플리케이션 등의 취약점 정보를 체계적으로 수집 · 관리할 수 있는 시스템과 제도가 국가적, 국제적으로 마련되어야 한다. 구체적으로 소프트웨어의 보안 취약점 정보 취급기준, 취약점 수집 · 대처를 위한 신고관리 절차를 포함하는 소프트웨어 보안 취약점 정보관리 인프라가 (그림 7)과 같이 구축되어야 한다.

• 보안 취약점 정보 신고 접수 : 보안 취약점 정보 접수기관을 지정하여 취약점 정보가 방치되거나 폭로되는 것을 막는 효과를 기대할 수 있다. 접수기관은 소프트웨어 보안 취약점 발견자 본인이 바라지 않는 이상 발견자의 이름, 연락처 등의 정보를 제품 개발자에게는 제공하지 않고 발견자의 대리로서 기능하여, 발견자가 제품 개발자에 비해 법적 분쟁으로 불리한 입장에 놓이는 것을 막는 효과를 기대할 수 있다.

• 보안 취약점 공개 시기 조정 : 하나의 보안 취약점이 복수의 소프트웨어에 영향을 주는 경우, 취약점 공개 시기를 조정할 필요가 있다.

• 보안 취약점 정보 관리 통제 및 대응방법 적용의 신속화 : 취약점 정보는 기밀 보관 유지를 전제로 하고 필요한 관계자에게만 알릴 필요가 있다. 또한, 시스템 구축 사업자, 인터넷 서비스 제공자(ISP), 사용자 등을 위해서 취약점 대응방법을 공개하여 발견된 보안 취약점에 신속하게 대응하도록 유도하는 것이 중요하다. 신고된 취약점 정보에 대한 기밀유지협약을 체결하여, 관계자 그룹과 그 이외를 나누어, 그룹 내 · 외의 정보 입 · 출력을 통제하는 장치가 필요하다.

• 보안 취약점 대응방법, 신고 건수 등의 통계 데이터의 처리 및 공지 : 소프트웨어 제품의 취약점 관련 정보의 신고 건수나 처리 상황 등에 관한 데이터를 통계처리하고 공지하여 관련 자들이 취약점 위험성을 적절히 평가할 수 있도록 지원한다.



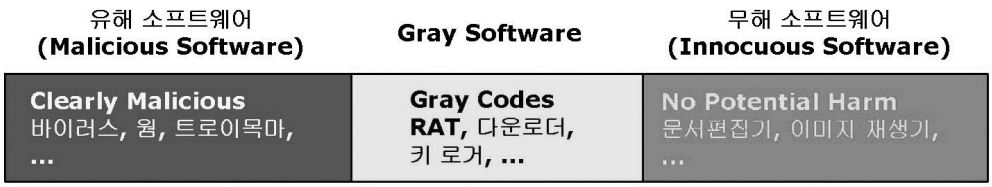
(그림 7) 소프트웨어 보안 취약점 정보 관리 인프라

5. 부정 소프트웨어 필터링 표준

소프트웨어에 구현되는 기능은 문서 편집, 정보 검색, 열린 네트워크 포트 탐지, 키보드 입력 기록, 원격 접속, 데이터 삭제, 메시지 전송 등 다양하게 존재한다. 소프트웨어에 구현되는 기능은 소프트웨어 개발자의 의지에 의해 다양하게 결정된다. 상용 소프트웨어의 보급으로 개발자와 사용자가 일반적으로 다르다. SI 프로젝트를 통해 개발된 소프트웨어의 경우에도 개발자와 사용자는 다른 것이 대부분이다. 개발자가 작성한 소프트웨어 관련 문서를 통해, 사용자는 소프트웨어에 구현된 기능을 이해하고 사용한다. 컴퓨터 바이러스, 인터넷 웜, 트로이목마 등과 같이 해를 끼칠 목적으로 만들어진 악성 소프트웨어를 제외하고, 소프트웨어는 (그림 8)과 같이, 소프트웨어에 포함된 기능의 사용 의도에 따라 해를 끼칠 가능성이 있는 소프트웨어와 그렇지 않은 소프트웨어로 구분할 수 있다. 사용 컨텍스트(context)에 따라 무해한(innocuous) 소프트웨어와 유해한(malicious) 소프트웨어의 중간에 위치하는 소프트웨어를 그레이(Gray) 소프트웨어[14,15]라고 부르며, 현재 인터넷에서 유통되는 그레이 코드를 사용 용도에 따라 <표 5>에 분류하였다. 예를 들어 다운로더(downloader)는 운영체제 또는 앤티-바이러스 소프트웨어 등의 자동 갱신을 위해 사용하는 기능으로 개발되었으나, 악성 소프트웨어를 원격 컴퓨터로부터 전송받아 사용자의 허락이나 인지없이 설치하는데 사용될 수 있다.

이러한 악의적인 목적으로 사용되는 그레이 소프트웨어를 부정 소프트웨어라고 정의하면, 부정 소프트웨어에 대한 사용자의 인식과 대응이 중요하다. 컴퓨터 바이러스, 웜과 같이 악의적인 행위를 특성화한 시그너처(signature)에 의해 진단될 수 있는 악성 소프트웨어와는 다르게, 그레이 코드 기능은 순기능적으로도 사용될 수 있기 때문에 일률적인 시그너처 적용은 가능하지 않다. 그레이 코드가 포함된 소프트웨어가 사용되는 사용자 환경의 컨텍스트에 따라 부정

소프트웨어가 되기도 하며, 합법적인 소프트웨어가 되기도 한다. 일반적으로 부정 소프트웨어는 사용자 허락을 받지 않고 컴퓨터에 설치되거나, 루트 킷(root-kit) 등의 기술을 이용하여 운영체제 내부에 잠복하여 악성코드 탐지시스템에 발각되지 않는 등의 기술로 무장되어 있다.



(그림 8) 위해 관점에서 분류한 소프트웨어 종류

〈 표 5〉 그레이 코드의 기능 분류

기능	설명	순기능	역기능
모니터링	사용자 행위 또는 사용자 정보 관찰(수집)	부모/기업의 합법적 관찰	ID, 개인정보 노출 컴퓨터 속도 저하, 보안 침해 기회제공
광고	광고성 컨텐츠 디스플레이	광고성 정보 제공	성가심과 생산성 지장, 컴퓨터 속도 저하 불쾌한 컨텐츠 디스플레이
원격 제어	원격 제어 또는 액세스	자신의 데이터에 대한 원격 액세스, 원격 기술지원 및 고장수리	컴퓨터 자원 도용, 대량 메일 발송 DDOS 공격 가담, 음란 자료 서비스
다이얼링	모뎀을 사용한 원격 접속	원격 서비스 액세스	전화료 부과
시스템 구성 설정 변경	시스템 또는 사용자 정보 설정/변경	시스템 커스터마이징	적절한 동의 없이 시스템 무결성 훼손
자동 내려받기	사용자 개입 없이 소프트웨어 다운로드/설치	자동 갱신 또는 시스템 자동관리	비인가 소프트웨어 설치

부정 소프트웨어는 비단 인터넷 파일 내려받기 뿐만 아니라 통상적인 소프트웨어의 배포 체계를 통해서도 전파될 수 있으므로, 부정 소프트웨어에 대응하기 위하여 사용자들은 자신의 컴퓨터에서 설치되어 실행되는 소프트웨어를 인식?확인할 수 있어야 한다[15]. 소프트웨어 개발자와 벤더에서 사용자에게로 소프트웨어가 배포되는 과정에서, 소프트웨어의 설치에 대한 사용자의 인지와 이해를 높이기 위해서 소프트웨어 배포 과정에서의 표준이 필요하다. 다음은 소프트웨어 작동에 있어서의 표준이 요구되는 항목들이다.

- 소프트웨어 목적 · 출처 · 동작 등에 대한 명시적 정보 제공 방법
- 소프트웨어의 실행 시작 및 중지 등의 작동에 대한 절차

– 소프트웨어의 설치와 제거(비활성화) 절차

또한 사용자에 대한 부정 소프트웨어 정보 제공을 위한 신뢰 사이트를 구축하여, 부정 소프트웨어 데이터베이스를 운영하여야 한다. 소프트웨어 사용자로부터의 신고 · 접수 · 검증 · 검색 등을 위한 사회적 헬프 데스크(help desk)를 운영하여 사용자의 부정 소프트웨어 대응을 지원하는 국가적 인프라를 구축하는 것이다. 이는 부정 소프트웨어에 대한 사용자들의 표준적 대응을 유도하여, 소프트웨어 사용에서의 신뢰를 높일 것이다.

6. 소프트웨어 보안강도의 기준과 측정 절차 표준

소프트웨어 시스템은 안전한가, 소프트웨어 시스템은 얼마나 안전해져야 하는가, 보안 기능을 장치하여 안전이 얼마나 향상될 수 있는가 등의 질문은 소프트웨어 보안 강도에 대한 측정을 요구한다. 소프트웨어 시스템의 보안 강도는 공격자가 주어진 위협 시나리오를 실현하기 어려운 정도로 정의할 수 있다. 어려움이란 공격자의 노력뿐만 아니라 시간, 장비 등과 같은 자원의 필요성도 포함한다. 공격자의 목표를 성취하기 위해 요구되는 시간의 양은 미리 알기 어려우며, 확률적이다. 예를 들어, 바이너리 코드에서 버퍼 오브플로 취약점을 찾아낼 확률은 매우 작지만 0은 아니다. 개발 과정 전반에 걸쳐 보안 강도를 측정하여, 제품으로 출시되기 전에 개발자에 의해 보안 강도가 향상되기 위한 "보안 강도 기준과 측정" 표준이 필요하다. 한편, 사용자의 입장에서도 구매하려는 소프트웨어의 보안 강도를 측정할 수 있거나, 제시된 보안 강도에 따라 소프트웨어를 구매한다면, 자신의 정보기술 환경과 비즈니스 니즈에 따라 취사선택할 수 있는 융통성이 주어진다.

Ⅳ. 결 론

사회전반이 네트워크기반에 의해 운영되는 상황이 폭발적으로 증가하는 상황에서 보안과 안전이 대단히 중요하며, 이를 다루는 정보보호 기술은 공공기술로서 중요하게 다뤄져야한다. 정보보호는 해킹과 같은 네트워크 공격으로부터 정보통신기반 보호, 정보화 진전에 따른 각 분야의 정보통신 시스템의 안전성 확보, 인터넷 등 개방형 네트워크의 확산에 따른 보안 취약점 대비, 전자상거래에서의 전자인증 등을 통한 신뢰성 확보수단 제공 등 국민의 일상생활, 국가 경제활동 및 국가안보에 무엇보다도 중요한 기술 영역이다.

지금까지의 정보보호 연구는 전통적인 암호 기술과 네트워크 경계선 방어 기술을 중심으로 이루어져 왔다. 전개된 시스템에 대한 보안 관리의 측면에서 방화벽 시스템, 항-악성코드 시스템, 침입탐지시스템 등과 같은 네트워크 기반의 보안 시스템을 설치하고, 백업이나 패스워드 관리와 같은 운영적 측면에 집중해왔다. 이는 보안에 취약한 시스템에 대한 공격자의 접근을 차단하려는 노력이다. 그러나 이러한 노력들은 기술

적으로 완벽한 접근 차단이 어렵고, 지속적인 갱신을 필요로 한다는 한계를 가지고 있다. 보다 근본적인 접근은 처음부터 보안에 취약하지 않은 또는 덜 취약한 시스템을 개발하는 것이다. 시스템 자체에 대한 보안성을 개선시키기 위한 연구는 아직 현장에 적용할 만큼 가시적인 성과를 내지 못하고 있다.

IT839 전략의 실현에 따라 다양한 정보통신 서비스 및 제품들이 전개될 것이다. 소프트웨어 집약 시스템으로서의 IT839 서비스와 제품은 소프트웨어의 보안 품질에 의해 그 보안성이 결정된다고 해도 과언이 아니다. 소프트웨어의 기능 명세, 설계 또는 구현 단계의 오류나 시동, 설치 또는 운영상의 문제점으로 인하여 발생하는 보안 취약점은 보안 침해 사고의 주요한 원인을 제공하므로, 소프트웨어의 개발과 이용 과정에서 소프트웨어의 보안 취약점을 감소시키기 위한 노력이 필수적이다.

개발 단계에서 코드 상의 입력검증 실패, 조건문 오류, 구성 오류, 부적절한 인증, 권한인가의 실패, 잘못 구현된 암호 알고리즘, 데이터 보호의 누락 등으로부터 발생하는 소프트웨어 보안 취약점을 감소시키기 위하여 소프트웨어 개발 공정에서의 효과적인 보안 프랙티스를 개발하여 적용하여야 한다. 이들 보안 프랙티스는 가장 취약한 링크를 찾아 공격하는 공격자의 속성에 비추어 모든 소프트웨어 개발자들이 준수하는 표준으로 이행되어야 한다. 또한 소프트웨어 보안 취약점은 완전히 제거될 수 없기 때문에, 보안 취약점을 악용하는 보안 침해에 실시간으로 대응하기 위하여, 개별 소프트웨어의 실행시간 행위 제어를 위한 침입탐지 기술을 표준화하여, 모든 응용 소프트웨어에 장착할 수 있도록 해야 한다.

사용 중인 소프트웨어에서 보안 취약점이 발견되는 경우, 사용자는 이에 대한 대처를 신속히 행하여야 하는데, 보안 취약점의 위험성에 대한 이해를 높이고, 패치 파일의 우선순위와 설치 일정을 수립하는데 필수적인 소프트웨어 보안 취약점 위험도 등급에 대한 표준 마련이 시급하다. 또한 소프트웨어 벤더가 취약점 회피방법이나 수정 방법을 배포할 때까지는 취약점이나 공격방법이 인터넷상에 유통되는 것을 최소화하기 위한 보안 취약점 정보 관리 절차에 대한 표준화가 정책적으로 추진되어야 한다. 인터넷 등의 공중 네트워크에 산재하는 소프트웨어에는 보안 침해를 야기하는 것들이 무수히 많다. 원하지 않는 유해한 그레이 코드인 부정 소프트웨어에 대한 사용자의 효과적인 대응을 위하여, 소프트웨어 설치와 제거를 포함하연 작동에 대한 표준이 요구된다. 소프트웨어를 구매하려는 사용자의 비용 효과적인 지출을 지원하기 위한 소프트웨어 보안강도 표준이 제도적으로 추진될 필요가 있다.

정보기술 환경의 오작동이나 보안 침해를 발생시키는 위협의 중심에 인간의 의도적 또는 비의도적인 행위가 존재한다. 인간의 실수 또는 오?남용을 줄일 수 있는 정보시스템의 보안 프랙티스 표준의 중요성은 더욱 증대하고 있다. 보안은 자동차에 있어서 제동장치와 같다. 제동장치의 목적은 자동차를 정지하기 위함이 아니라

자동차를 더 빨리 달릴 수 있도록 하기 위함이다. 도로에 떨어져 있는 위험물, 난폭한 운전자, 고장난 자동차 등에 의한 사고를 피하는데 제동장치를 이용한다. 속도가 높을수록 더 강력하고 효과적인 제동 장치가 필요하듯이 더 강화된 보안은 IT839가 그려낼 유비쿼터스 컴퓨팅 세상에서 더 많은 자유와 믿음을 줄 것이다. 보안은 소수의 특별한 소프트웨어에 적용되는 것이 아닌, 모든 소프트웨어의 생명주기 동안 표준으로 실천되어야 한다.

>> 참고문헌

[1] 정보통신부, 안전한 u-Korea 구현을 위한 중장기 정보보호 로드맵, 2005년 5월.

[2] William R. Cheswick & Steven M. Bellovin, Firewalls and Internet Security : Repelling the Wily Hacker, Addison-Wesley, 1994.

[3] CMU Software Engineering Institute, The Team Software Process and Security, http://www.sei.cmu.edu/tsp/tsp-security.html

[4] Greg Hoglund & Gray McGraw, Exploiting Software: How to Break Code, Addison-Wesley, 2004.

[5] David Walker, An Introduction to Proof-Carrying Code, Lecture Note, http://www.cs.princeton.edu/~dpw/talks/PCClecture.ppt

[6] W. A. Arbaugh, W. L. Fithen, & J. McHugh, "Windows of Vulnerability: A Case Study Analysis", IEEE Computer, vol. 33, pp. 52-59, December 2000.

[7] CERT/CC Statistics, Vulnerabilities reported, http://www.cert.org/stats/cert_stats.html

[8] ICSA Labs 10th Annual Computer Virus Prevalence Survey, http://www.trusecure.com/cgi-bin/ct_download.cgi ESCD=w0206&file=VPS2004.pdf

[9] S. L. Pfleeger & L. Hatton, "Investigating the Influence of Formal Methods," Computer, February 1997, pp. 33-43.

[10] National Cyber Security Partnership Task Force, Issues Report on Security Across the Software Development Life Cycle, April 2004, http://www.cyberpartnership.org/

[11] Kenneth R. van Wyk, & Gary McGraw. "Bridging the Gap between Software Development and Information Security," IEEE Security and Privacy, vol. 03, no. 5, pp. 75-79, September/October, 2005.

[12] ISO/IEC WD 18043: Guidelines for the implementation, operation and management of intrusion detection systems(IDS), April 2002.

[13] Computer Research Association, Grand Research Challenges in Information Systems Final Report, September 2003, http://www.cra.org/reports/gc.systems.pdf

[14] Anti-Spyware Coalition, Anti-Spyware Coalition Definitions and Supporting Documents, July 2005, http://www.antispywarecoalition.org/documents/index.htm

[15] MS, Windows AntiSpyware(Beta): Analysis approach and categories, March 2005, http://www.microsoft.com/athome/security/spyware/software/isv/analysis.mspx

[16] Microsoft Security Response Center, Security

Bulletin Severity Rating System, November 2002. http://www.microsoft.com/technet/security/bulletin/rating.mspx

[17] National Infrastructure Advisory Council, Common Vulnerability Scoring System, October 12, 2004. http://www.dhs.gov/interweb/assetlibrary/NIAC_CVSS_FinalRpt_12-2-04.pdf

>> 저자소개

김 홍 근(HongGeun Kim)
Email : hgkim@kisa.or.kr
Tel : +82-2-405-4760
Fax : +82-2-405-5119

1985. 2 : 서울대학교 전자계산기공학과 학사
1987. 2 : 서울대학교 전자계산기공학과 석사
1994. 2 : 서울대학교 컴퓨터공학과 박사
1994. 5~1996.5 : 한국전산원 전산망안전보안센터장
1996. 5~현재 : 한국정보보호진흥원 팀장/단장/연구위원
주관심분야 : 컴퓨터/소프트웨어 보안, 침입탐지