매개 취약점 점검 언어로부터 점검 코드를 자동으로 생성하는 에이전트를 이용한 취약점 관리 시스템

김수용, 서정석, 김한성, 조상현, 임채호, 차성덕

한국과학기술원

Integrated Security Manager with Agent-based automatic vulnErability checking code generating scanner from intermediate vulnerability checking Language(ISMAEL)

Su Yong Kim, Jeong Seok Seo, Hansung Kim, Sanghyun Cho,

Chaeho Lim, Sung Deok Cha

Korean Advanced Institute of Science and Technology

요 약

악의의 침입자로부터 시스템을 보호하기 위한 첫 번째 단계는 시스템의 취약점을 분석하는 일이다. 기존의 시스템 취약점 분석 방법은 주로 네트워크 기반 취약점 점검 도구에 의존해 왔다. 하지만, 네트워크 기반 취약점 점검 도구는 대상 시스템의 제한된 정보만을 이용하여 취약점을 점검하기 때문에 시스템의 모든 취약점에 대한 검사가 불가능하다는 단점이 있다. 호스트 기반 취약점 점검 도구를 사용하면 시스템 내부의 모든 정보를 이용할 수 있지만, 시스템의 OS 종류나 버전에 따라 각기 다른 호스트 기반 취약점 점검 도구를 개발해야 한다는 단점이 있다. 또한, 호스트 기반 취약점 점검 도구들은 많은 호스트들을 동시에 점검하기 힘들다는 점이 문제로 지적되고 있다. 본 논문에서는 호스트 기반 취약점 점검 도구를 에이전트로 구현하여 대상 시스템에 설치하고, 하나의 관리 프로그램에서 여러 에이전트들을 관리함으로써 동시에 많은 호스트의 취약점들을 관리할 수 있는 모델인 ISMAEL을 제시한다. 또한 ISMAEL은 OS에 맞는 여러 호스트 기반 취약점 점검 도구들을 개발해야 하는 문제를 해결하기 위해 OS에 독립인 부분만을 뽑아내고, 그 외 OS에 종속된 부분은 Library 형태로 제공하여, OS에 독립인 부분에서 이 Library를 참조하여 특정 취약점 점검 코드를 자동 생성하고 이를 실행하여 취약성 여부를 판단할 수 있는 구조를 채택하고 있다.

I. 서론

시스템은 일단 해킹을 당하게되면, 사후 조사가 힘들뿐만 아니라 완전한 복구가 불가능한 경우가 많다. 이런 이유로 안전한 시스템 관리를 위해서는 시스템의 취약점들을 사전에 파악하고 취약점을 보완하는 것이 중요하다. 초기에는 대상 시스템의 OS에 독립적이어서 개발이 쉽고 여러 시스템의 취약점을 동시에 점검할 수 있기 때문에 네트워크 기반 취약점 점검 도구들이 주로 개발되었다. 대표적인 네트워크 기반 취약점 점검 도구들은 Satan, Nessus 등을 들 수 있다. 하지만, 네트워크 기반 취약점 점검 도구들은 네트워크를 통해서 일어나는 침입에 악용될 수 있는 취약점만을 점검할 수 있기 때문에 시스템 내부 사용자에 의해서 일어나는 침입에 악용될 수 있는 취약점을 점검할 수 없다. 다시 말해서, 네트워크 기반 취약점 도구들은 시스템의 모든 정보를 이용하여 취약점을 점검하지 않고, 네트워크를 통해 시스템으로부터 얻을 수 있는 정보만을 이용하여 시스템의 취약점을 점검하기 때문에 그 기능에는 근본적인 한계를 지닐 수밖에 없다.

이런 네트워크 기반 취약점 점검 도구의 한계로 인해 시스템의 모든 자원과 정보를 이용하여 시스템의 취약점을 점검하는 호스트 기반 취약점

점검 도구가 필요하다. 하지만, 시스템 내부의 취약점을 점검하는 방법이 OS에 따라 다르고, 같은 OS에서도 버전에 따라 다를 수 있으며, 심지어 동일한 버전의 OS가 설치된 시스템에서조차도 설치된 Application에 따라 점검 방법이 다르다는 점이 호스트 기반 취약점 점검도구를 구현하는 것을 무척 어렵게 만들었다. 하지만, 대부분의 침입이 내부 사용자에 의해 이루어진다는 점과 외부침입자의 경우에도 보안 개념이 없는 일반 사용자의 계정을 먼저 획득한 후에, 시스템 내부의 취약점을 이용하여 루트 권한을 획득하는 경우가 많기 때문에 시스템 내부의 취약점을 사전에 확인하여 이를 제거해 주는 일은 중요하다.

호스트 기반 취약점 점검 도구의 또 다른 문제점은 다수의 시스템을 동시에 효율적으로 점검할 수 없다는 점이다. 한 조직에서 관리해야 하는 시스템의 수가 급격히 증가하고 있기 때문에 다수의 호스트 기반 취약점 점검 도구들을 효율적으로 관리하기 위한 방법이 필요하다.

Satan 이후로 많은 취약점 점검 도구들이 개발되어 왔지만, 새로운 취약점들을 점검하기 위한 기법들이 계속해서 갱신되지 못함에 따라 대부분의 취약점 점검 도구들이 사장되었다. 취약점 점검 기법들이 갱신되지 못한 이유는 새로운 취약점 점검 기법들을 추가하기 위해서는 취약점 점검 도구의 소스 코드를 직접 수정해야 했기 때문에 개발자나 몇몇 옹호자들에 의해서만 취약점 점검 기법들이 갱신될 수 있었기 때문이다. Nessus는 이를 보완하기 위해 새로운 취약점들을 점검하기 위해 스크립트 코드만 작성하면 되도록 했지만, 일반 사용자가 새로운 취약점을 점검하기는 여전히 어렵다.[1]

많은 조직들은 내부적으로 서로 다른 보안 정책을 가지고 있는 경우가 많다. 따라서 각 조직은 그들이 정한 보안 정책대로 실제 시스템들이 작동하고 있는지 항상 점검할 필요가 있다. 예를 들면, A라는 조직에서는 FTP 를 허용하지 않는 보안 정책을 세웠다 하더라도 이를 기술하고 계속해서 검사해줄 방법이 없다면, 보안 정책이 제대로 수행되고 있는지 판단할 수 없다. 따라서 취약점 점검 기법을 쉽게 기술할 수 있는 수단을 제공함으로써 이를 통해 보안 정책의 일관성 점검에도 이용할 수 있다.

2 장에서는 기존의 취약점 점검 도구들을 중심으로 관련 연구를 기술하고, 3장에서는 본고에서 제시하는 ISMAEL의 구조와 작동원리를 기술한다. 4장에서는 ISMAEL의 적용 사례를 보여주고, 5장에서는 본 고에서 제시하는 ISMAEL의 장점과 앞으로의 가능성에 대해 언급한다.

II. 관련연구

시스템의 취약점을 점검하기 위한 연구는 크게 두 가지 방향 - 네트워크 기반 취약점 점검 방식과 호스트 기반 취약점 점검 방식- 으로 진행되고 있다. 네트워크 기반 취약점 점검 방식은 대상 호스트에 독립적이며, 여러 대상 호스트를 동시에 점검하기 쉽다는 큰 장점에도 불구하고, 많은 단점을 지니고 있다. 제한된 정보만을 이용하여 대상 호스트의 취약점을 점검하므로, 점검 결과가 부정확할 수 있고, 취약점 점검을 위해 대상 호스트에서 운영되고 있는 데몬의 정상적인 활동을 방해하는 검사를 수행해야 하는 경우도 있다. 또한, 내부 사용자에 의한 내부 공격에 이용될 수 있는 취약점들을 점검하는 것이 불가능하다는 점은 네트워크 기반 취약점 점검 도구의 가장 큰 단점이다. 이에 반해 호스트 기반 취약점 점검 방식은 호스트에 존재하는 모든 정보를 이용할 수 있기 때문에 정확하며, 빠르게 취약점을 점검할 수 있지만, 대상 호스트의 OS에 의존적이기 때문에 OS와 OS 버전에 따라 각기 다른 점검 도구를 구현해야 한다는 단점을 지니고 있다. 기존에는 구현의 편리성으로 인해 네트워크 기반 취약점 점검 도구들이 주로 개발되었지만, 내부 사용자에 의한 보안 등이 강조되면서 호스트 기반 취약점 점검 도구의 필요성이 논의되고 있다.

1) 취약성 점검기술 및 침입시도 탐지기술 개발[2]

2000년 12월에 한국정보보호센터가 주관하고 시큐아이닷컴㈜과 인터넷시큐리티㈜가 공동 연구하여 발표한 '취약점 점검기술 및 침입시도 탐지기술 개발' 에 관한 1차년도 연구개발 보고서는 취약점 점검과 침입 탐지를 모두 수행할 수 있는 모델을 제시한다. 가장 큰 특징은 네트워크 기반 취약점 점검 기법과 호스트 기반 취약점 점검 기법을 통합하는 형태를 취했다는 점이다. 보고서에서 제안하는 모델은 취약점 점검과 침입시도 탐지를 수행하기 위한 모델이지만, 여기에서는 취약점 점검 부분만을 논의한다.

보고서에서 제안하는 모델은 점검 대상이 되는 호스트에 에이전트 시스템을 설치하고, 관리자는 취약점 스캔 관리 시스템을 통해 여러 에이전트 시스템을 통합 관리하는 형태이다. 에이전트 시스템에는 네트워크 취약점 점검 모듈과 시스템 취약점 점검 모듈, 그리고 소스코드 보안 취약점 점검 모듈로 이루어진다. 네트워크 취약점 점검 모듈은

취약점 스캔 관리 시스템의 컨트롤에 따라 네트워크 취약점을 점검하는 기능을 가진다. 시스템 취약점 점검 모듈은 취약점 스캔 관리 시스템의 컨트롤에 따라 시스템 취약점을 점검한다. 소스코드 보안 취약점 점검 모듈은 취약점 스캔 관리 시스템의 컨트롤에 따라 소스코드 취약점을 점검한다. 세개의 에이전트 모듈들은 취약점 DB를 주기적으로 최신 취약점 DB로부터 update받는다.

2) COPS (Computerized Oracle and Password System)[3]

COPS는 Unix의 SUID program 이나 부적절한 파일 퍼미션문제와 잘못된 패스워드 등을 점검하는 호스트 기반 취약점 점검 도구이다.

3) Nessus [4]

Nessus 는 네트워크 기반 취약점 점검 도구이다. NASL이라는 스크립트 언어를 이용하여 새로운 취약점을 기술할 수 있는 것이 특징이다. 이런 새로운 취약점을 스크립트 언어를 사용하여 기술하기 때문에 새로운 취약점을 추가하여도 Nessus 프로그램을 새로 컴파일하거나 수정할 필요가 없다는 장점이 있다.

III. ISMAEL

ISMAEL은 Vulnerability Scanner Manager (VSM)와 Vulnerability Scanner Agent (VSA), Vulnerability Check Writer (VCW) 그리고 Vulnerability DataBase (VDB)로 구성되어 있다.

관리자가 시스템들을 ISMAEL을 이용하여 관리하고자 한다면, 시스템들의 OS에 맞는 VSA를 설치하고, VSM에서 VSA가 설치된 시스템의 IP 주소와 Port 번호만을 등록해주면, VSM은 해당 VSA로부터 시스템 정보를 받아온다. 시스템 정보를 바탕으로 VSM은 시스템의 OS 종류와 버전에 맞는 취약점들을 VDB에 요청한다. VDB는 시스템에서 검사를 수행할 필요가 있는 취약점 항목들을 VSM에게 넘겨주고, VSM은 이를 Tree 형태로 관리자에게 보여준다. 관리자는 취약점 항목들 중 점검하고자하는 취약점 항목들을 선택하여 검사수행버튼을 누르면, VSM은 Intermediate Language로 기술된 취약점 점검 방법을 VSA에게 넘겨주고, VSA는 이 정보를 이용하여 시스템에서 실행가능한 취약점 점검 코드를 생성한다. 이를 실행하여 취약성 여부를 VSM에 보내주고, VSM은 이 정보를 바탕으로 취약한 항목과 취약하지 않은 항목, 아직 점검하지 않은 항목으로 분류하여 관리자에게 보여준다.

호스트 기반 취약점 점검 도구들이 OS 종류와 버전에 따라 변하는 부분은 시스템 내의 정보를 얻어오는 부분이다. 그렇기 때문에 이 부분과 OS 종류와 버전에 관계없이 공통인 부분을 분리하고, 관리 프로그램이 호스트 기반 취약점 점검 도구에게 취약점 점검을 위해 Intermediate Language에 의해 기술된 점검 방법을 넘겨주고, 각 에이전트들은 Intermediate Language에 의해 기술된 점검 방법의 실제 코드를 담고 있는 Library를 호출하는 취약점 점검 코드를 자동 생성하고 이를 실행하여 취약성 여부를 판단함으로써 다른 OS 용 호스트 기반 취약점 점검 도구로의 포팅을 용이하게 하였다.

VSM과 VSA간에 교환되는 정보는 보안상 대단히 중요하기 때문에 암호화되어 통신하는 것이 필요하다. 이 부분은 차후 구현될 예정이다. 통신에 사용되는 Data encoding은 차후 다른 보안 product와의 상호운영을 대비하여 XML로 구현되어 있다. XML 문서는 사람이 읽기에도 쉽고 확장성이 좋은 장점이 있다.

1. VSM (Vulnerability Scanner Manager)

VSM은 관리 프로그램으로써, VSA가 설치된 여러 시스템의 취약점들을 관리할 수 있는 기능을 제공한다. 현재 Visual C++로 구현되어 있는 VSM은 크게 두 가지 Manager들로 구성된다. VSA를 새로 등록하거나 기존의 VSA를 삭제해 주는 Agent Manager와 등록된 VSA로부터 시스템의 정보를 받아 해당 시스템에 맞는 취약점 점검 항목을 VDB로부터 추려내서 표시해 주고, 관리자가 원하는 취약점들에 대해서 해당 시스템들의 취약점을 점검하는 Vulnerability Manager로 구성된다.

1) Agent Manager

Agent Manager는 Agent들의 추가, 삭제 및 정보 변경을 담당한다. 각 Agent들은 그룹별로 분류되어 관리되며, 여러 Agent들을 관리할 수 있도록 했다. 또한, 차후 그룹별 다른 보안 정책 수립이 가능한 기능을 추가할 예정이다.

2) Vulnerability Manager

리눅스 계열 VSA와 솔라리스 계열 VSA, 윈도우 계열 VSA는 각기 해당 시스템에서 수행해야 하는 취약점 점검 항목이 다를 수밖에 없다. 또한, 같은 계열의 운영체제라 하더라도 일부 취약점들은 특정 버전에만 취약한 것으로 알려져 있다. 따

라서 시스템에 따라 다른 취약점 점검 항목을 수행할 수 있도록 Vulnerability Manager는 해당 Agent들에 맞는 취약점 항목들만 제공해주는 역할을 한다.

2. VSA (Vulnerability Scanner Agent)

VSA는 관리하려는 대상 시스템에 하나씩 설치되어 VSM를 통해 관리자가 내리는 명령을 해당

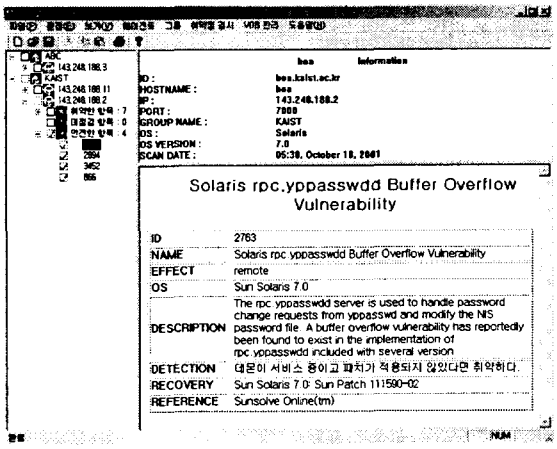

그림 1 : VSM 화면

시스템에서 수행하고 그 결과를 VSM에 보고해 주는 역할을 한다. 현재 VSA는 솔라리스 7.0에서 C로 구현되어 있으며, 제공하는 기능은 크게 세 가지이다. VSM이 요청할 때, 설치되어 있는 시스템에 대한 일반적인 정보(기계 종류, OS 종류와 버전)를 넘겨주는 기능과 VSM이 취약점 점검을 요청할 때 취약점 점검을 수행하기 위한 코드를 생성하고, 생성된 코드를 컴파일하고, 실행하고 그 결과를 VSM에 보고하는 기능을 제공한다.

1) Information Scanner

Information Scanner는 VSA가 설치된 시스템의 정보를 모아서 VSM에 제공해 주는 역할을 한다. 이는 관리자가 관리하고자 하는 시스템에 VSA만 설치해 주면 VSM이 자동으로 VSA의 정보를 얻어 오게 함으로써 관리 시스템의 수가 증가할 때 발생하는 관리자 부담을 최소화 할 수 있다. 또한, VSM에 취약점 검사를 수행할 때마다, VSA로부터 시스템 정보를 자동 갱신하게 함으로써 VSA에 설치된 시스템의 OS가 바뀌더라도 이를 자동 감지하고 이에 맞는 취약점 검사를 수행하도록 한다.

2) Vulnerability Scanner

VSM은 Information Scanner가 제공한 시스템 정보로부터 설치된 시스템에서 점검해야 하는 항목을 VDB로부터 추출하여 관리자에게 보여 준다. 의 정보를 모아서 VSM에 제공해 주는 역할을 한다. 이는 관리자가 관리하고자 하는 시스템에 VSA만 설치해 주면 VSM이 자동으로 VSA의 정보를 얻어 오게 함으로써 관리 시스템의 수가 증가할 때 발생하는 관리자 부담을 최소화 할 수 있다. 또한, VSM에 취약점 검사를 수행할 때마다, VSA로부터 시스템 정보를 자동 갱신하게 함으로써 VSA에 설치된 시스템의 OS가 바뀌더라도 이를 자동 감지하고 이에 맞는 취약점 검사를 수행하도록 한다.

3) Automatic Vulnerability Checking Code Generator & Executor

VDB에는 각 취약점에 대해서 여러 정보들과 Intermediate Language 로 기술된 취약성 점검 방법이 있다. 관리자가 VSM을 통해 특정 취약점을 점검하고자 할 때, VSM은 VDB로부터 취약점이 존재하는지를 점검할 수 있는 Intermediate Code를 받아서 이를 VSA에 넘겨준다. Automatic Vulnerability Checking Code Generator는 이로부터 해당 시스템의 OS 종류와 버전에 적합한 코드를 자동 생성한다. 생성된 코드를 컴파일하고 실행하여 취약성 여부를 VSM에 넘겨주는 기능을 담당한다.

3. VCW (Vulnerability Check Writer)

VCW는 기존에 많은 취약점 점검 도구들이 새로운 취약점의 갱신이 이루어지지 않음에 따라 현재에는 거의 사용되지 않고 있는 문제점을 해결하기 위해서 추가된 모듈이다. VCW는 여러 기관에서 발표하는 보안권고문을 참조하여 VCW의 GUI를 통해 새로운 취약점 점검 기법을 일반 사용자들도 손쉽게 추가할 수 있도록 한다.

또한 이 모듈은 FTP 나 Telnet과 같은 특정 서비스들이 허용하지 않는 보안 정책이 수립될 경우 이를 이런 보안 정책이 관리하는 사이트들에서 잘 지켜지고 있는지를 점검하기 위해 유용하게 쓰일 수 있다. 차후 Policy Writer로 확장할 계획이다.

4. VDB (Vulnerability DataBase)

VDB는 여러 OS들과 OS 버전에 대한 취약점들이 저장되어 있는 Database 이다. MySql Server 버전 3.23.41으로 구현되어 있으며, VSM은 ODBC를 이용하여 VDB에 접근한다. 이와 같은 구조는

VDB를 하나만 두고, 여러 VSM들이 VDB에 접근하여 취약점 정보를 얻을 수 있기 때문에 새로운 취약점이 새로 나왔을 때, 하나의 VDB만을 갱신해 줌으로써 모든 VSM이 갱신된 정보를 이용할 수 있다는 장점이 있다.

VDB에 저장되는 취약점 정보는 취약점 ID, 이름, 효과, 해당되는 OS 종류와 버전 정보, Description, 취약성 여부 탐지 방법, 취약점 제거 방법, 참고문헌, 취약점 점검 방법이 들어간다.

여러 OS 종류와 버전에서 취약하다고 알려진 취약점들을 하나의 취약점으로 기술하기 위해서 가장 힘든 부분이 취약점 점검 방법이다. 취약점 점검 방법은 OS 종류나 버전마다 다르기 때문에 이를 공통된 한 가지 표현으로 기술하기가 쉽지 않기 때문이다. ISMAEL에서는 Intermediate Code를 이용하여 이를 해결하였다. 즉, 특정 취약점을 점검하기 위해 각 시스템들이 취약한 파일이나 취약한 데몬이 존재하는지를 점검하는 방식은 OS 종류나 버전에 따라 다르지만, 점검해야 하는 항목은 동일하다. 따라서, 점검해야 하는 항목을 Intermediate Code로 표현함으로써 하나의 취약점을 점검 방법만을 달리해서 여러 번 기술하는 낭비를 없앨 수 있다. 또한, OS가 다른 VSA들이 Intermediate Code로 표현되는 함수들에 대한 기능을 library로 제공해 주기만 하면, 해당 Intermediate Code를 이용하여 VSA 자신에게 맞는 취약점 점검 코드를 생성할 수 있다.

예를 들면, kcms_configure의 취약점이 존재하는지를 점검하기 위해서 kcms_configure 파일이 존재하고 해당 패치가 되었는지를 점검하기 위한 Intermediate Code 는 DoesFileExist ("kcms_configure") and IsNotPatched ("111400-01") 이다. 이 Intermediate Code를 받은 VSA는 이 함수를 호출하여 데몬이 서비스 중인지를 점검하는 코드를 작성한다.

```
main()
{
        if(IsDaemonServiced("kcms_configure")
                && IsNotPatched("111400-01"))
                return 취약함;
        else
                return 취약하지않음;
}
```

IV. 적용사례

VSA가 구현되어 있는 Solaris 7.0 이 설치되어 있는 Sparc 머신에서 실험을 수행하였다. 취약점 점검 DB에는 Solaris 7.0에 해당하는 11가지 취약점이 기술되어 있었다. 11가지 취약점은 모두 일반 사용자가 루트 권한을 획득할 수 있는 취약점이었는데도 불구하고, 이 중 7가지 취약점에 대해서 취약한 것으로 나타났다. 검사 시간은 11가지 취약점을 검사하는데 30초 가량이 소요되었다.

V. 개선점

앞으로 ISMAEL의 개선해야할 사항은 C로 구현되어 있는 VSA를 Java로 포팅하는 일과 VSA와 VSM 간의 통신을 암호화하는 일이다. 현재 Solaris 7.0에서 C로 구현된 VSA는 여러 OS로의 포팅을 감안하여 OS마다 달라져야하는 부분(실제 취약점을 점검하는 부분)을 모두 Library 형태로 VSA로부터 분리시켰다. 이런 구조로 다른 OS로의 포팅을 쉽게 만들었지만, VSA를 Java로 작성할 경우 VSA를 전혀 수정할 필요 없이 다른 OS에서 사용할 수 있고, Library 만을 추가해 주면 된다.

VSA를 C로 작성한 이유는 속도 면에서 가장 빠른 언어이기 때문이지만, Java의 속도가 계속해서 개선되고 있으며, 취약점 점검 도구를 실제 구현하여 실행해 본 결과 점검 시간이 대체로 짧은 시간 내에 가능했으며, 긴 시간을 요하는 작업들도 대부분 속도가 느린 장치(하드 디스크 등)를 검색하는 데 걸리는 시간이므로 사용하는 프로그래밍 언어의 Performance와는 상관이 없었다.

한편, VSM과 VSA 간의 통신에는 VSA가 설치된 시스템의 취약점을 포함하고 있을 수 있기 때문에 통신 내용을 암호화하는 것이 반드시 필요하다.

VI. 결론

ISMAEL을 구현하고, 실제 시스템에 적용해 보면서 다양한 OS에 맞는 호스트 기반 취약점 점검 도구인 VSA의 바람직한 모델을 제시하였다. 또한, ISMAEL의 장점은 다음과 같다.

첫째, 효율적으로 많은 시스템들의 취약점들을 사전에 파악하여 이를 보완함으로써 시스템의 안전성을 높일 수 있다.

둘째, VSM을 통해 여러 호스트 취약점 점검

에이전트들을 동시에 관리할 수 있다.

셋째, VCW에 의해 취약점 점검 기법(넓게 보면, 각 사용자에게 Customized 된 보안 정책)을 기술할 수 있는GUI 환경을 제공함으로써 새로운 취약점이 발견되더라도 여러 보안 사이트에서 제공하는 Advisory 를 토대로 취약점 점검 기법을 일반 사용자들도 기술할 수 있게 하였다.

넷째, 기존의 취약점 점검 기법을 기술하는 방식으로 사용되었던 직접 코딩이나 스크립트 언어를 사용하지 않고 제한된 GUI Interface 를 사용함으로써 따로 Debugger를 제공할 필요가 없고, 관리자 역시 점검 기법의 기술이 적절한지에 대한 문법적 검증 절차를 생략할 수 있다.

다섯째, 다수의 VSM이 하나의 취약점 DB를 사용함으로써 취약점 업데이트가 용이하고 동일한 DB를 각 VSM 마다 설치해야 하는 비효율성을 제거할 수 있다.

http://salmosa.kaist.ac.kr/~sykim/ISMAEL/에서 VSM과 VSA, VWC에 대한 실행 파일을 다운 받을 수 있다.

참고문헌

[1] Renaud Deraison, "The Nessus Attack Scripting Language Reference Guide Version 1.0.0pre2", http://www.nessus.org/doc/nasl.html, Apr 2000

[2] 박정현외 35명, "취약성 점검기술 및 침입시도 탐지기술 개발 제 1차년도 연구개발 보고서", 정보통신부, 2000. 12.

[3] Daniel Farmer, Eugene H. Spafford, "The COPS Security Checker System", Purdue University Technical Report CSD-TR-993, Jan. 1994.

[4] Renaud Deraison, Jordan Hrycaj, Nessus, Available at http://www.nessus.org/