네트워크 생존성 평가 시뮬레이터를 위한 취약성 데이터베이스 구축

신동훈°, 고경희, 김형종, 김동현
한국정보보호진흥원
{dhshin°, khko, hjkim, dhkim}@kisa.or.kr

Korea Information Security Agency

DongHoon Shin°, KyoungHee Ko, HyungJong Kim, DongHyun Kim

요 약

오늘날, 가용성과 중단없는 서비스 제공에 대한 관심 고조, 개방화되는 시스템, 중앙집중적 단일화된 관리와 통제 적용의 어려움으로 시스템이 주어진 임무를 수행해 나갈 수 있는 능력인 생존성이 요구되고 있다. 정보통신기반의 생존성을 평가하기 위해서는 평가 방법론을 정의해야하고 이를 뒷받침해주는 기술적인 연구가 수행되어야한다. 특히, 복잡 다양한 시스템들로 구성된 현재의 정보통신기반 네트워크의 특성과 실제 공격을 통해 테스트하기 어려운 현실을 감안하여, 시뮬레이션 기법을 사용한 생존성 평가기술이 연구되고 있다. 보안 시뮬레이션은 공격자, 네트워크 모델, 성능평가 모델, 원인-결과 모델들로 이루어져 있으며 이 중 원인-결과 모델이란 모델들 자체의 동적 정보이며 모델들간을 이어주는 관계 데이터이다. 본 논문에서는 원인-결과 모델링을 위해 단위 취약점(Atomic Vulnerability)을 이용한 취약점 분석 방법을 제안하고 이를 토대로 한 시뮬레이션용 취약성 데이터베이스인 VDBFS의 구축과정과 결과를 소개한다.

1. 서 론

행정, 국방, 금융, 통신, 운송, 전력 등 공공 인프라의 운영 및 관리에 정보통신 시스템에 대한 의존도가 심화되고 있으며 전 세계가 인터넷으로 연결됨에 따라 해킹, 컴퓨터 바이러스 유포 등 전자적 침해행위가 국가 안보에 새로운 위협 요소로 대두되었다. 주요 정보통신 시설의 교란·마비는 사회 기반시설의 기능 마비를 초래하여 막대한 경제적 손실과 사회적 혼란을 초래할 수 있다. 전통적으로 보안이 정보의 기밀성에 중점을 두었던 것에 반해 현재는 가용성과 중단없는 서비스의 제공에 관심이 고조되고 있으나 시스템이 개방됨에 따라 중앙에서 단일한 관리와 통제를 적용하기 어렵게 되었다. 이제 취약성 분석·평가를 통한 사전 예방에서 더 나아가 위협을 탐지하고 조직이 허용하는 범위에서 감내할 수 있는 생존성(survivability)이라는 척도로 평가하는 것이 필요하게 되었다.
각 정보통신기반의 외부 공격에 대한 생존성 평가는 안전한 정보통신기반 구축에 필수 불가결한 요소이다. 생존성이란 공격, 오류, 사고가 발생하였을 때 시스템이 시기 적절한 방법으로 주어진 임무를 수행해 나갈 수 있는 능력을 말한다.[1] 정보통신기반의 생존성을 평가하기 위해서는 평가 방법론을 정의해야하고 이를 뒷받침해주는 기술적인 연구가 수행되어야한다. 특히, 복잡 다양한 시스템들로 구성된 현재의 정보통신기반 네트워크의 특성과 침투 시험(penetration test)과 같은 실제 공격을 통해 테스트하기 어려운 현실을 감안했을 때 시뮬레이션 기법을 사용한 생존성 평가기술이 적합하다.

이러한 시뮬레이터의 모델의 동적 특성을 나타내주는 데이터와 모델들간에 일어날 수 있는 사건의 인과론적 연결 고리를 만들어 주는 데이터를 이용하여 시뮬레이션이 구동된다. 이것을 원인-결과 모델이라 한다. 따라서 원인-결과 모델을 구현한다는 것은 공격자 모델을 위한 공격 정보, 호스트와 보안시스템 모델을 위한, 주어진 운영체제, 서비스 환경, 보안 정책 등의 속성에서 생길 수 있는 취약점 정보, 그 취약점을 완화시키거나 제거할 수 있는 방어 솔루션 정보들을 모으는 것이다. 이 작업은 교육·연구용, 취약점 스캐닝, 침입탐지, 소프트웨어 공학, 컴퓨터 포렌식스에서 사용되고 있는 취약성 데이터베이스(이하 DB)를 구축([2])하는 것과 유사하지만 더 나아가 시뮬레이션의 요구 사항도 만족시켜야 한다.
본 논문에서는 보안 시뮬레이션용 취약성 DB로서의 요구 사항을 만족시키기 위해 단위 취약점(AV : Atomic Vulnerability)에 기반한 취약점 분석 방법을 제안한다. 그리고 이를 적용하여 취약성 DB인 VDBFS(Vulnerability Database for Simulation)를 구현한 과정과 그 결과를 소개하고자 한다.

2. 단위 취약점을 이용한 취약점 분석 방법

2.1 AV와 CV의 정의

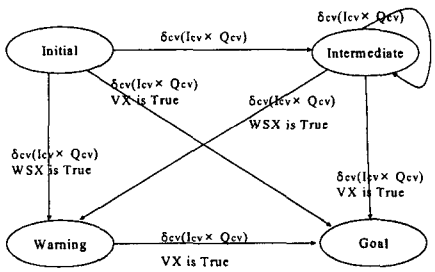
원인-결과 모델이란 시뮬레이션이 동작하는 원리를 모델링한 것이다. 코헨의 논문[3]에서처럼 본 과제의 시뮬레이션 세계도 우연이 아닌 인과율에 지배받는다. 본 논문에서 취약점은 일반적인 취약성 DB에서 식별번호를 부여하고 있는 하나의 토픽이며, CVE 코드[4] 하나에 대응되는 것이다. 취약점이란 보안 정책에 위배를 가져오는 오류를 말하며, 이것을 CV(Compound Vulnerability)라고 칭한다. 그리고 이 CV가 악용 가능한 세부 원인을 단위 취약점 AV(Atomic Vulnerability)라 한다.

2.2 CV 표현 형식

시스템 S가 가지고 있는 취약점들의 집합을 CV라 할 때, CV의 원소 cv_i는 5개의 구성요소를 갖는다. Icv_i는 다음 AV의 표현 형식에서 설명한다. CV의 상태와 상태전이는 (그림1)과 같다. Warning 상태는 Intermediate 상태의 하나인데 특히 Goal 상태로 전이되기 바로 직전의 상태이다. wsx_i는 이 Warning 상태를 정의하는 표현식이고 vx_i는 Goal 상태를 정의하는 표현식이다. 이 표현식은 4개의 이항 연산자를 이용한다. AV를 호출함으로서 그 표현식 값(True, False)을 계산한다.

CV = {cv_1, cv_2, ..., cv_n}
cv_i = {Icv_i, Qcv_i, δcv_i, wsx_i, vx_i}
1. 공격에 사용되는 입력 집합 Icv_i
2. CV의 상태 집합 Qcv_i = {Normal, Intermediate, Warning, Goal}
3. CV의 상태 전이 함수 δcv_i : $Icv_i \times Qcv_i \rightarrow Qcv_i$
4. 경고 상태 취약점 표현식 wsx_i ::= <AV> | <AV><Operator><AV>
5. 취약점 표현식 vx_i ::= <AV> | <AV><Operator><AV>
   이때, Operator의 집합 Operator = {and, or, sand, por}
   단위 취약점의 집합 AV = {av_1, av_2, ..., av_n}

- AND Relation (AND) : 두 AV 모두가 참인 경우에만 악용되는 취약점을 표현할 때 사용
- OR Relation (OR) : 두 AV 중 하나라도 참인 경우 악용될 수 있는 취약점을 표현할 때 사용
- Probabilitic OR Relation (POR) : 두 AV 중 하나라도 참인 경우 악용될 수 있는 취약점을 표현할 때 사용. 특히 두 AV에 가중치가 존재하는데 여기서 가중치란 단위 취약점에 대해 시스템이 어느 정도 취약한지를 정량화한 것이다.
- Sequential AND Relation (SAND) : 두 AV가 순서적으로 악용되어야 하는 취약점을 표현할 때 사용



(그림1) CV의 상태와 상태 전이도

2.3 AV 표현 형식

단위 취약점 av_i는 6개의 구성요소를 갖는다.

AV = {av_1, av_2, ..., av_n}
av_i = {Name, Type, Category, δav_i, Qav_i, Iav_i}
1. Type = {Fact, NonProb, Prob}
2. Category = {Generic, Application-Specific, System-Specific}
3. 상태전이함수 δav_i : $Q_{Initial\_State} \times Iav_i \rightarrow Q_{Final\_State}$
4. 상태집합 $Q = Q_{Initial\_State} \cup Q_{Final\_State}$
   where
   $Q_{Initial\_State}(\subset Q)$ = {None, Normal, *S_IdentifiedStateName(AppName)*}
   $Q_{Final\_State}(\subset Q)$ = {None, *S_IdentifiedStateName(AppName)*}
5. 공격 입력을 이루는 액션의 집합 Iav_i = {i_1, i_2, ..., i_i}
   where i_i ::= <System_Directive AppName [Parameter]>

본 연구에서의 추상화 수준은 시뮬레이션 방법론과 관계가 있다. AV를 이용해서 분석한 취약점 데이터가 사용되는 시뮬레이션에서 보안 관점에서 중요한 상태와 상태 전이 함수를 정의한 개념 레벨(conceptual level)로 모델링[5]을 수행한다. 따라서 상태 및 입력은 보안 관점에서 중요한 상태, 보안 정책에 위배되는 상태와 입력으로 정의 한다

2.4 AV와 CV를 이용한 취약점 분석 예제

대표적인 취약점 부류인 버퍼 오버플로우 취약점의 AV를 정하고 그 AV를 조합하여 CV를 만드는 과정에 대해 설명한다.

[예제] amd 버퍼 오버플로우 취약점 (CVE-1999-0704)

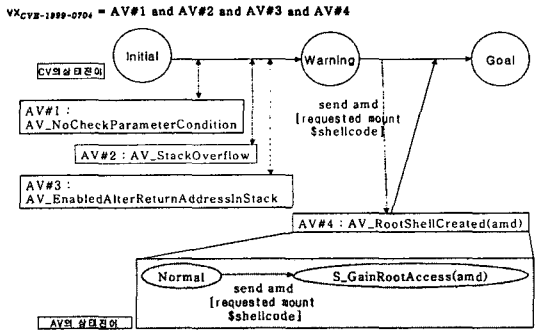
amd는 자동적으로 파일시스템을 마운트/언마운트 해주는 데몬이다. 이 데모의 로깅 함수에 리모트에서 악용가능한 버퍼 오버플로우 취약점이 존재한다. 버퍼 오버플로우는 가장 대표적인 취약점 혹은 공격방법을 지칭하는 이름이다. 'egg'라 알려진 특별한 어셈블리 코드를 삽입하여, 스택을 오버플로우시키고 리턴 주소를 변환하여 원하는 코드를 실행할 수 있다. 버퍼 오버플로우의 근본 원인은 다음과 같이 분석할 수 있다.[6]

- 버퍼로 데이터를 복사할 때 경계(bound)를 확인하지 않는다.
- 사용자가 리턴 주소를 변경하는 것이 가능하다.
- 입력 데이터가 정확한 형태인지 확인하지 않는다.
- 실행 가능한 형태의 워드(word)를 확인하지 않는다.

이 원인들은 세 가지 Fact 타입 AV로 표현한다. 첫째, 셋째, 넷째 원인은 조건 체크(condition check)에 대한 것으로 묶을 수 있으며, 두 번째 원인은 독립적으로 표현하였다.

- NoCheckParameterCondition
- StackOverflow
- EnabledAlterReturnAddressInStack

레드햇 리눅스 6.0을 설치하고 amd 데몬이 수행되고 있다면 이 시스템은 세 가지 Fact 타입 AV가 참이 되면서 바로 Warning 상태가 된다(그림2 참조). 이 때 공격자가 공격 입력을 보내면서 관리자 권한을 획득한다(AV#4). 따라서 취약점의 표현식 $VX_{CVE\text{-}1999\text{-}0704}$은 네 개의 단위 취약점이 AND된 식이다. CV와 AV 두 수준에서의 상태 전이도는 아래 (그림2)와 같다.



(그림2) amd 버퍼 오버플로우 취약점 분석

3. VBFS의 구축 결과

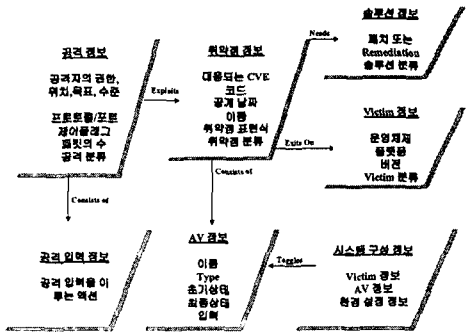
3.1 VDBFS 개략의 스키마

본 연구에서 제안한 VDBFS의 스키마는 크게 공격에 대한 정보를 가지고 있는 공격 DB와 취약점과 공격의 대상이 되는 시스템에 대한 정보, 상태 변화를 가지고 있는 취약성 DB로 나뉜다.

우선, [취약성 DB]는 운영체제, 운영체제의 버전, 하드웨어 플랫폼, 설치된 소프트웨어 종류 및 버전 정보 등의 <Victim 정보>와 취약점과 밀접한 소프트웨어 환경설정 파일에 대한 정보를 가지고 있는 <시스템 구성 정보>, 해당 CVE 코드, 취약점의 악용이 성공한 후의 결과, 발표 날짜 등의 <일반적인 취약점 정보>, 앞에서 설명한 것처럼 AV와 CV를 이용하여 취약점을 분석한 결과를 담고 있는 <AV 정보>, 취약점에 대한 해결방안인 <솔루션 정보>로 구성된다.

[공격 DB]는 공격자의 권한, 목표와 수준 정보, 공격 가능한 위치 정보, 프로토콜, 포트, 제어 플래그 등 공격자가 공격 패킷을 생성할 때 사용되는 <공격 정보>와 <공격 입력 정보>로 구성된다.

아래 (그림3)은 VDBFS 개략의 스키마이다. 상위층에 있는 공격 정보, 취약점 정보, Victim 정보, 솔루션 정보는 일반적인 취약성 DB의 스키마와 같다. 그림 하위층에 있는 공격 입력 정보, AV 정보, 시스템 구성 정보는 시뮬레이션을 위해 특화된 스키마이다.



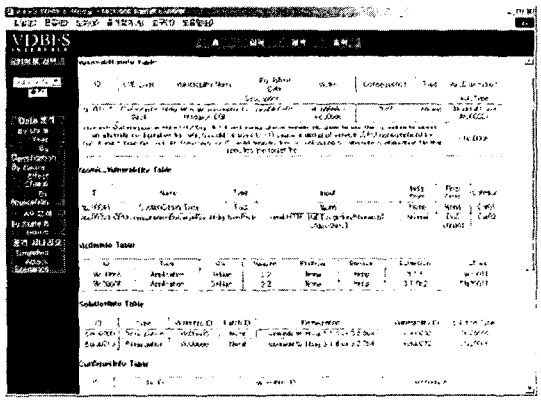
(그림3) VDBFS 개략의 스키마

3.2 VDBFS의 구축 결과

이상의 절차로 2002년 5월의 ICAT 메타베이스를 기준으로 약 30%의 취약점을 선정하여 CVE 코드 기준, 282개의 취약점을 분석, 입력 완료하였다. [표1]의 VDBFS(개수) 컬럼에 운영체제별 입력 개수를 정리하였다. 괄호안의 숫자는 ICAT 대비 VDBFS의 취약점 보유비율이다. 시뮬레이션에서 사용되는 라우터 모델을 위해 특별히 5개의 Cisco 라우터와 스위치 관련 취약점과 Apache, BIND, Oracle, zlib와 같이 모든 운영체제에서 동작하는 응용 프로그램은 따로 선별하여 37개를 분석하였다. 하나의 취약점이 여러 운영체제에서 나타날 수 있기 때문에 아래 표의 VDBFS 데이터 개수는 중복을 포함하고 있다. (그림4)는 구축된 VDBFS의 취약점 출력 화면이다.

[표1] 운영체제별 분석갯수, ICAT과 비교

	운영체제	버전	벤더	ICAT (개수)	VDBFS (개수)
유닉스	FreeBSD	4.x	FreeBSD	126	50(40%)
	AIX	4.3.x	IBM	106	13(12%)
	HP-UX	11.x	HP	125	20(16%)
	Solaris	7 (sparc,x86) 8 (sparc,x86)	Sun	178	54(30%)
리눅스	Debian	2.x	Debian	86	37(37%)
	RedHat	7.x	RedHat	175	54(31%)
윈도우즈	Windows NT	4.0	Microsoft	292	87(30%)
	Windows 2000		Microsoft		
기타	router	CiscoCatalyst	3500 XL	-	5
	application	-	-	-	37



(그림4) VDBFS 취약점 출력화면

4. 결론 및 향후 연구계획

본 논문에서는 시뮬레이션용 취약성 DB의 구축을 위해 단위 취약점을 이용한 취약점 분석 방법을 제시하였다. 이 분석 방법을 통해 시뮬레이션 모델 자체, 또는 모델들 사이에서 필요한 데이터를 생성하여 취약성 DB인 VDBFS를 구축하였고 이를 시뮬레이션에 적용하였다. 분석되는 취약점의 개수가 늘어나면 시뮬레이션의 적용 대상, 시나리오가 좀 더 다양화될 것이고, 이 시뮬레이션의 결과는 현재 연구중인 생존성 평가 방법론의 정립에서 평가에 필요한 정보를 제공하는데 활용될 것이다.

현재는 이미 발표된 보안 취약점들을 분석하여 단위 취약점을 정의하지만, 향후 이미 분석되어 존재하는 단위 취약점들을 조합하여 이것이 알려지지 않은 보안 취약점의 발견에 일조할 수 있는지에 대한 연구가 필요하다.

또한 솔루션 정보의 확대가 필요하다. 정보통신기반에 대한 다양한 사이버 테러에 대응하기 위해서는 시스템 및 네트워크 전반에 걸쳐 적용할 수 있는 일관성있는 방어 메커니즘 수립이 요구되며, 수립된 방어 메커니즘이 정보통신기반의 필수 서비스를 외부 공격이 있을 때 일정 수준의 기능 및 성능을 유지시키는지를 평가하는 연구가 필요하다. 이를 위해 VDBFS를 확대 적용하여 방어 메커니즘 지식베이스를 구축하고 있다.

참고문헌

[1] Howard F. Lipson, David A. Fisher, "Survivability - A New Technical and Business Perspective on Security", Proceedings of the 1999 Workshop on New Security Paradigms, 1999.

[2] Pascal C. Meunier, Eugene H. Spafford, "Final Report of the 2nd Workshop on Research with Security Vulnerability Databases", Technical report, CERIAS Purdue University, 1999.

[3] Fred Cohen, "A Preliminary Classification Scheme for Information System Threats, Attacks, and Defenses; A Cause and Effect Model; and Some Analysis Based on That Model", September, 1998

[4] http://cve.mitre.org/

[5] Nong Ye, J. Giordano, and J. Feldman, "A process control approach to cyber attack detection", Communications of the ACM, Vol. 44, No. 8, 2001, pp. 76-82.