단위 취약점 식별자 부여 자동화에 대한 연구☆

A study on automation of AV(Atomic Vulnerability) ID assignment

김 형 종*
Hyung-Jong Kim

요 약

단위 취약점(atomic vulnerability)은 기존의 취약점의 표현방법이 갖는 모호성을 개선하여, 취약점을 시스템적으로 표현하기 위한 취약점의 새로운 정의이다. 단위 취약점은 취약점의 유형, 위치, 결과 등에 따라 보다 세분화하여, 취약점을 의미를 중심으로 분석하고 자 할 때 필요한 정보로서, 기존의 취약점은 몇 개의 단위 취약점 식별자의 조합으로 표현된다. 현재 가장 대표적으로 사용되는 취약점 정보인 CVE(Common Vulnerability Exposure)의 경우, 취약점의 핵심적인 내용을 자연어 형태의 설명(description)을 통해 제시한다. 이러한 CVE의 설명 정보는, 정형화되어 있지 않아서 단위 취약점 분석을 위해서는 기존의 CVE 설명 정보에서 특정 단어들을 검색하여 데이터를 분류하는 자연어 검색 및 판단 기법이 필요하다. 본 논문에서는 자연어 검색 기법을 이용하여 단위 취약점 분석에 활용할 수 있는 소프트웨어를 설계하고 이를 실제 구현한 결과를 소개하고자 한다. 본 연구의 기여점은 설명위주의 취약점 표현을 정형화된 형태로 변환해 주는 소프트웨어 시스템의 개발에 있다.

Abstract

AV (Atomic Vulnerability) is a conceptual definition representing a vulnerability in a systematic way. AVs are defined with respect to its type, location, and result. It is important information for meaning based vulnerability analysis method. Therefore the existing vulnerability can be expressed using multiple AVs. CVE (common vulnerability exposures) which is the most well-known vulnerability information describes the vulnerability exploiting mechanism using natural language. Therefore, for the AV-based analysis, it is necessary to search specific keyword from CVE's description and classify it using keyword and determination method. This paper introduces software design and implementation result, which can be used for atomic vulnerability analysis. The contribution of this work is in design and implementation of software which converts informal vulnerability description into formal AV based vulnerability definition.

☞ Keyword: Meaning-based Vulnerability Identification, Vulnerability Assessment, Atomic vulnerability, Vulnerability Description Analysis, 의미기반 취약점 식별, 취약점 진단, 단위 취약점, 취약점 설명 분석

1. 서 론

소프트웨어의 기능이 다양해지고, 복잡해짐에 따라서 소프트웨어의 취약점 수도 급격히 증가하고 있다. 취약점은 소프트웨어 자체의 안정성을 해칠 뿐만 아니라, 이를 이용한 바이러스, 웜, 해킹 등에 의해서 2차적인 피해를 발생시킬 수 있다. 따라서 취약점을 체계적으로 분석하고 관리하기 위해서 표준화된 명명법을 개발하여 이용할 필요성이 높아졌다. MITRE[1]에서는 실제적인 국제표준으로 활용되고 있는 취약점 목록인 CVE(Common Vulnerability Exposures) 체계를 개발하여 새롭게 발견되는 취약점을 관리하고 있다[20]. 그러나 CVE는 단순히 취약점의 발생연도와 일련번호로 이루어진 식별자(CVE-ID)와 특정한 형식 없이 자연어로 기술되는 설명(description) 부분으로 이루어져 있다. 따라서

* 종신회원 : 서울여자대학교 컴퓨터학부 전임강사
hkim@swu.ac.kr
[2008/05/16 투고 - 2008/05/19 심사 - 2008/07/21 심사완료]
☆ 이 논문은 2008학년도 서울여자대학교 교내특별과제연구비의 지원을 받았음.

1) 미 연방정부의 필수 임무에 대한 기술 및 연구개발 지원 기관 (http://www.mitre.org)

취약점들 간의 연관 관계를 파악하거나 심도 있는 분석을 위해서는 현재의 CVE 체계만으로는 부족한 상황이다. 또한 취약점의 정도(severity)를 정량적으로 나타내기 어려워서, CVSS (Common Vulnerabilities Scoring System)라는 체계를 도입하여 CVE를 보완하여 사용하고 있다[21].

한편 취약점을 보다 체계적으로 분석하기 위하여, 단위 취약점(atomic vulnerability)이라는 개념이 제안되었다[4, 5]. 단위 취약점은 국제적인 표준으로 통용 되는 CVE 기반 취약점의 구성요소가 되는 단위 취약점을 정의하고 이들의 관계를 정의하여 취약점의 특성을 표현하다. 이렇게 취약점이 정의될 경우, CVE-ID와 함께 취약점 표현식을 사용하여, 취약점이 갖는 세부적인 특성과 각 특성간의 관계를 직관적으로 알 수 있게 된다.

이러한 단위 취약점 개념을 적용하기 위해서는 기존의 취약점을 단위 취약점을 이용하여 분류하여야 한다. 본 논문에서는 기존의 취약점 체계인 CVE 정보를 해당하는 단위 취약점으로 자동으로 맵핑하여 주는 방법 및 이를 적용한 소프트웨어와 그 동작 결과를 소개하겠다.

2. 기존연구 소개

단위 취약점의 표현에 근간이 되는 모델링 방법론인 DEVS(Discrete EVent System Specification) 형식론[1, 2, 3]은 B. P. Zeigler에 의해서 정의된 이산 사건 시뮬레이션 이론이다. DEVS 형식론은 시스템을 분석하여 모델을 만들기 위한 이론적 근간을 제공해준다. DEVS 형식론은 더 이상 나눌 수 없는 모델에 해당하는 atomic 모델과 atomic 모델과 또 다른 coupled 모델의 조합으로 구성될 수 있는 coupled 모델로 구성된다. 다음은 atomic 모델의 정의이다.

$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, t_a >$

단,

X : 입력 사건의 집합.

S : 순차적 상태의 집합.

Y : 출력 사건의 집합.

$\delta_{int} : S \rightarrow S$: 내부 전이 함수

$\delta_{ext} : Q \times X \rightarrow S$: 외부 전이 함수

$\lambda: S \rightarrow Y$: 출력 함수

$t_a : S \rightarrow R+0 \rightarrow \infty$: 시간 진행 함수

단, $Q = \{(s,e) \mid s \in S, 0 \leq e \leq t_a(s)\}$

e : 최근의 상태 전이 이후로 흐른 시간.

다음은 coupled 모델의 정의이다.

$DN = < D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, select>$

단,

D : coupled 모델의 구성요소 모델 i에 대한 이름의 집합.

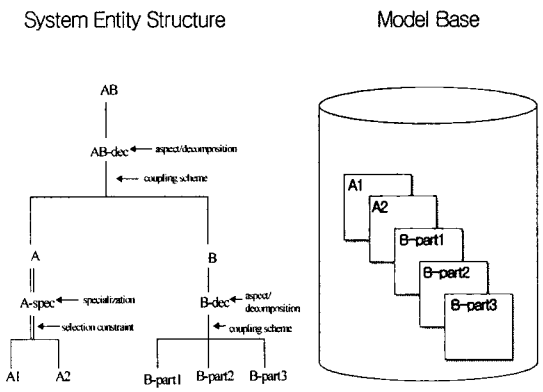
M_i : 구성요소가 되는 basic 모델.

I_i : 모델 i의 영향을 받는 모델들의 집합.

$Z_{i,j}$: I_i의 원소 각 j에 대해서 i에서 j로의 출력 번역 함수.

Select : 타이 브레이킹 함수.

Atomic 모델과 coupled 모델을 통해서 대부분의 시스템의 특성의 표현이 가능하다. 하지만, 시스템의 구조적인 특성을 표현하는데 있어서 coupled 모델이 갖는 한계가 존재한다. 이것은 시뮬레이션 환경이기 때문에 존재하는 문제로서, 시스템을 구성하는 컴포넌트가 될 수 있는 후보 모델에 대한 정의와 이들 중 시뮬레이션을 수행하기 위한 선택이 가능한 구성이 필요하게 된다. 이를 위해서 정의된 모델링 이론이 SES/MB(System Entity Structure/Model Base) 이다. 그림 1은 SES/MB를 표현한 그림이다. SES는 시스템의 구조를 나타내는 지식을 특정 형식으로 표현한 것으로써 모델들의 분할(decomposition), 분류(taxonomy)와 모델간의 연결 관계에 대한 정보를 가지고 있다. SES는 트리 형태로 계층적인 모델들을 정의하고, 트리의 말단은 모델 베이스의 모델로 나타낸다. SES안에는 시스템의 구조를 표현하기 위한 지식으로 entity,



(그림 1) SES/MB의 구성 예

aspect(decomposition), specialization의 3가지 형태의 노드가 있다.

Entity 노드는 실세계의 한 객체와 대응되며 aspect 또는, specialization을 자식으로 가질 수 있다. Aspect 노드는 그림 1의 한 수선 사이에 있는 노드로 나타나고, 이는 aspect노드의 자식 노드들이 aspect노드의 부모 노드의 분할된 형태라는 것을 의미한다. 즉, 그림 1의 A, B는 AB를 구성하는 원소라는 것을 의미한다. 본 DEVS 및 SES/MB 이론을 활용할 경우 정보보호 시스템의 다양한 조합의 효과를 모델링하고 시뮬레이션 하는 데에 매우 유익하다 [16].

취약점의 이름을 정하는 가장 대표적인 기법은 CVE (Common Vulnerability Exposures)에서 정의한 방법으로 단순히 취약점이 발견된 년도와 시점으로 고려해서 순차적으로 번호를 붙이는 형태로 정의된다. CVE-ID는 취약점을 식별하는데 있어서 혼란이 생길 때 나타나는 문제를 해결하기 위한 매우 간단한 명명법 이다. 2008년 8월 현재 약 31,000 여 개의 취약점이 존재하며, 매년 약 4~6,000개의 취약점이 NIST를 통해서 발표 되고 있다. 표 1은 CVE 취약점의 예 이다. 또한, NIST는 NVD (National Vulnerability Database)라고 하는 취약점 데이터베이스를 관리하며, 실시간으로 갱신되는 정보를 사용할 수 있도록 XML형태의 서비스를 제공해 주고 있다[19]. 또한, CVE-ID는 취약점 점검 도구 및 침입탐지 시스템의 탐지 규칙들과의 상호 연관성을 표현하기 위한 방법으로서 매우 유용하다[17, 18].

(표 1) CVE 취약점 예

CVE-ID	CVE-2002-0364
Description	Buffer overflow in the chunked encoding transfer mechanism in IIS 4.0 and 5.0 allows attackers to execute arbitrary code via the processing of HTR request sessions, aka "Heap Overrun in HTR Chunked Encoding Could Enable Web Server Compromise."
References	- BUGTRAQ:20020612 ADVISORY: Windows 2000 and NT4 IIS .HTR Remote Buffer Overflow [AD20020612] - URL:http://marc.theaimsgroup.com/?l=bugtraq&m=102392069305962&w=2 - CERT-VN:VU#313819 - URL:http://www.kb.cert.org/vuls/id/313819 - MS:MS02-028 - URL:http://www.microsoft.com/technet/security/bulletin/ms02-028.asp - OVAL:oval:org.mitre.oval:def:182 - URL:http://oval.mitre.org/repository/data/getDef?id=oval:org.mitre.oval:def:182 - OVAL:oval:org.mitre.oval:def:29 - URL:http://oval.mitre.org/repository/data/getDef?id=oval:org.mitre.oval:def:29 일부 생략
Status	Entry

CVE가 그동안 취약점의 통일성 있는 표현에 있어서 많은 기여를 하였지만, 취약점이 갖는 특성을 표현하는데 있어서 모호하다는 한계를 가지고 있다.

이러한 한계를 극복하기 위해 최근 CWE (Common Weakness Enumeration)이라는 개념을 통해 취약점의 세부적 특성을 명시하고자 하는 노력

이 시작 되었다[22]. 하지만 CWE 역시, 해당 취약점들의 상호 연관성을 표현해주지는 못하고 있다.

단위 취약점은 특정 취약점의 더 이상 나눠 질 수 없는 특성을 정의하기 위한 방법이다. 취약점은 단위 취약점과 이들 사이의 관계로 표현 된다. 이러한 취약점의 최소단위 개념은 [14]에서 먼저 언급되었다. 본 연구의 단위취약점은 [14]에서 제시한 취약점의 최소단위 개념을 기반으로 DEVS/SES 방법론을 사용한 시스템적인 표현을 제공한다.

본 연구에서 제안된 단위 취약점을 통한 취약점의 표현은 시스템 모델링 이론을 근간으로 하여 취약점의 악용 과정을 상태와 상태전이로 표현할 수 있다는 장점을 갖는다. 다음은 단위 취약점 AV의 정의이다[4, 5].

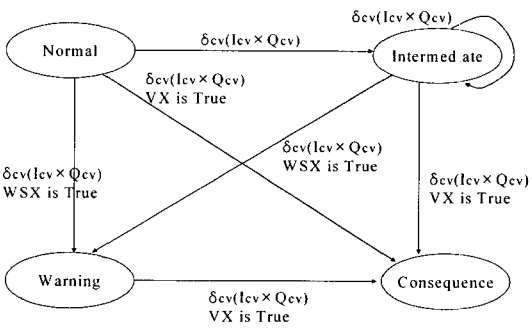
Atomic Vulnerability : AV = {I_{av}, Q_{av}, δ_{av}, Type}
where,
I_{av} = {I_{av1}, I_{av2}, ..., I_{avn}}
Q_{av} = Q(initial state) ∪Q(final state)
δ_{av} : I_{av} × Q(initial state) →Q(final state)
Type : {Fact, Deterministic, Probabilistic}

여기서, 입력 I_{av}에 의해서 시스템이 갖는 취약 특성이 활성화 되는 것이다. 이는 시스템의 취약점이 공격자의 특정 입력에 의해서 악용되는 상황을 모델링한 것이다. 일반적으로 취약점은 시스템의 잠재적 특성으로 공격자의 특정 입력을 통해서 활성화된다. 상태 집합인 Q_{av}는 특정 입력이 시스템에 도착한 경우 전이하게 될 시스템의 상태를 정의하고 있다. 상태전이는 외부 입력 I_{av}와 상태집합 Q_{av}로 정의된다. 단위 취약점은 3가지 유형으로 분류된다. 첫째, 외부의 어떤 입력도 받지 않는 Fact 형태, 둘째, 외부의 입력을 받아 상태 변화가 생기는 경우는 확률적으로 상태변화가 발생하는 Probabilistic 형태와 셋째, 외부의 입력에 의해 반드시 상태변화가 발생하는 Deterministic 형태이다. 이러한, 단위 취약점의 유형은 취약점을 분석하면서 도출된 것으로 대부분의 취약점의 특성을 포함 할 수 있다.

위에서 정의된 단위 취약점들과 이들 사이의 관계를 사용하여 취약점을 표현한다. 특히, 취약점 사이의 관계를 정의하기 위해서 취약점 표현식(VX: Vulnerability Expression)을 정의한다. 공격자의 행위는 결국 취약점을 악용하는 것으로 취약점의 악용이 성공할 경우 이를 참(True)으로 출력을 내주는 논리식을 취약점 표현식으로 정의하였다. 또한, 취약점 표현식은 공격자의 외부 입력에 대한 시스템의 상태 전이를 결정하는 연산으로 활용되어, 공격에 대한 시스템의 동적인 특성을 표현해 주게 된다. 취약점 표현식과 함께 위험상태 표현식(WSX: Warning State Expression)을 정의하여, 취약점이 악용되기 직전의 상태를 표현해 주도록 하였다. 아래는 단위 취약점을 사용한 취약점 표현식 및 위험상태 표현식에 대한 정의이다.

Vulnerability: V = {I_{cv}, Q_{cv}, δ_{cv}, WSX, VX}
where,
I_{cv} = {I_{cv1}, I_{cv2}, ..., I_{cvn}}
Q_{cv}= {Normal, Intermediate, Warning, Consequence}
δ_{cv} : I_{cv} × Q_{cv} → Q_{cv}
WSX : Warning State Expression
VX : Vulnerability Expression

이와 같이 단위취약점과 이의 조합으로 취약점을 표현할 경우, 기관이 운영 관리하는 네트워크 및 시스템이 갖는 취약점을 효율적으로 관리할 수 있을 뿐 아니라, 취약점을 통하여서 보안 관리 정



(그림 2) 취약점의 상태 및 상태 전이

책을 추출할 수 있을 것으로 기대된다. 특히, 기존의 정보보호모델[6, 7, 8] 및 정책표현 언어[9, 10]들에 이러한 개념을 추가하여 취약점을 고려한 정보보호 정책의 표현을 향후 연구로 주제로 고려할 수 있다.

그 밖의 시뮬레이션 기술을 취약점 및 침입 대응 부분에 사용한 예로서 [13]은 매우 추상화 수준이 높은 위협 및 공격의 관계에 대한 시뮬레이션 수행을 가능 하게 하였다. [15]는 공격과 이에 대한 탐지에 대해서 다양한 추상화 수준에서의 모델링 방법을 제시하고 있다. 이러한 연구들은 정보보호 분야의 모델링 및 시뮬레이션의 가능성을 제시하였지만, 구체적인 활용 방향을 제시하지는 못했다.

3. 자연어 검색 및 분석 알고리즘

3.1 취약점 자동 분석을 위한 5단계

CVE 취약점 설명을 단위 취약점으로 자동 매핑하기 위하여, 논문에서는 다음과 같은 방법을 사용

(표 2) CVE 취약점 정보의의 XML 스키마 (1/2)

구성 요소	설명	속성(필수여부)	자식 구성요소 (개수)
nvd	Root 구성요소	nvd_xml_version (필수): 제공되는 xml의 스키마 및 DTD의 버전 pub_date (필수): xml feed가 취합된 날짜	entry (0 or more)
entry	CVE 각 Entry의 Root 구성요소	type (필수): 취약점이 CVE 또는 CAN 이름을 갖는지 여부 표시 name (필수): CVE 전체 이름을 가짐 seq (필수): 취약점의 순차 번호 (name에서 CAN 또는 CVE를 제거한 것) nvd_name: NVD에서 지정한 이름 (추후 지정 예정) discovered: 취약점의 발견시점 published (필수): 취약점 발표 시점 modified: 가장 최근에 변경된 시점 severity: NVD에서 분석한 취약점의 심각도("High", "Medium", or "Low") CVSS_version: CVSS version 정보 CVSS_base_score: CVSS base score (0.0-10.0) CVSS_impact_score: CVSS impact score (0.0-10.0) CVSS_exploit_score: CVSS exploit (0.0-10.0) CVSS_vector: CVSS base vector로서 base score를 계산하는 데 사용된 속성 reject: 취약점의 이름이 CVE 사전에서 거부된 적이 있음을 표시 (always has value "1")	desc (exactly 1) impacts (0 or 1) sols (0 or 1) loss_types (0 or 1) vuln_types (0 or 1) range (0 or 1) refs (exactly 1) vuln_soft (0 or 1)
desc	모든 설명들에 대한 Wrapper Tag	None	descript (0 to 2)
descript	취약점에 대한 설명으로, 설명을 제공한 기관을 명시	source (필수): 취약점의 설명을 제공한 기관	취약점의 설명을 담고 있는 문자열
impacts	취약점의 파급효과의 설명 대한 Wrapper Tag	None	impact (1 or more)
impact	취약점의 파급효과에 대한 설명으로, 이를 제공한 기관을 명시	source (필수): 파급효과 설명 제공기관	취약점 파급효과 설명을 담고 있는 문자열
sols	취약점의 해결방법에 대한 Wrapper tag	None	sol (1 or more)
sol	취약점의 해결책에 대한 설명으로, 이를 제공한 기관을 명시	source (필수): 해결책을 제공한 기관	취약점 해결책 설명을 담고 있는 문자열

(표 2) CVE 취약점 정보의의 XML 스키마 (2/2)

구성요소	설명	속성(필수여부)	자식 구성요소 (개수)
loss_types	취약점이 악용됐을 때 피해의 특성을 명시. 4가지 형태를 가짐.	• None	avail (0 or 1) conf (0 or 1) int (0 or 1) sec_prot (0 or 1)
vuln_types	취약점의 유형에 대한 wrapper tag	• None	access (0 or 1) input (0 or 1) design (0 or 1) exception (0 or 1) env (0 or 1) config (0 or 1) race (0 or 1) other (0 or 1)
range	취약점을 악용하는 공격의 범위를 명시하기 위한 wrapper tag이며, range는 4개의 값을 가짐	• None	local (0 or 1) local_network(0 or 1) network (0 or 1) user_init (0 or 1)
refs	참고자료들에 대한 wrapper tag	• None	ref (0 or more)
ref	취약점에 대한 하나의 참고 자료 표시	• source (필수): 참고자료 제공기관 • url (필수): 참고자료의 하이퍼링크 • sig: Tool의 서명을 갖고 있음을 표시 • adv: 참고자료가 보안 권고임을 명시 • patch: 참고자료가 취약점에 대한 패치 정보를 갖고 있음을 명시	제공기관의 이름 문자열로 보유
vuln_soft	취약점을 갖는 SW 제품들의 명시를 위한 wrapper tag	• None	prod (1 or more)
prod	취약점 갖는 SW의 이름을 명시하며, SW의 버전 정보에 대해서 wrapper tag 역할을 함	• name (필수): 제품의 이름 • vendor (필수): 제조회사의 이름	vers (1 or more)
vers	취약점을 갖는 SW의 버전 정보	• num (필수): 버전 숫자 • prev: 표시된 버전 이전에 발표된 모든 SW가 해당 취약점을 갖고 있음을 명시 • edition: 취약점을 갖는 SW의 edition을 표시	None

하였다. 먼저 분석 대상 취약점 설명을 읽어 들인다. 다음으로 유형 결정을 위하여 키워드를 파싱하여 검색한다. 검색 결과에 따라 취약점 유형을 결정한다. 취약점 유형에 따라서 해당 키워드를 검색하여 필요한 정보를 수집하여, 단위 취약점으로 매핑 한다. 최종적으로 수집한 정보를 기반으로 매핑 결과를 제시하고, 이를 확인한다. 그림 3은 이러한 5개의 과정을 도식화 한 것이다.

단계 1. XML 데이터 파싱

NVD에서는 CVE 취약점 정보를 XML로 데이터베이스를 제공해 준다. NVD의 XML 스키마 정보를 활용하여 각 취약점의 설명(Description)정보를 추출한다. 이를 위해서 XML 파서를 사용한 정보추출이 요구된다. 표 2는 NVD의 XML 스키마 정보 중 2 레벨까지의 내용이다.

단계 2. Description 키워드 추출

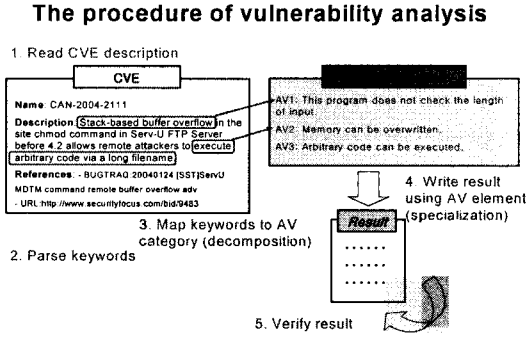
Description 정보를 파싱하여 키워드 추출을 수행한다. 키워드 추출은 취약점 정보를 면밀히 분석하여 추출된 값으로, 본 연구에서는 사람에 의해서 핵심적 키워드를 정의했다. 추출된 키워드는 제 3

단계 분할과 4단계 특화를 위한 기초 데이터로 활용된다. 분할/특화는 SES에서 제시하는 시스템의 개체표현에 사용되는 방법으로 본 연구에서는 취약점을 구성하는 세부적 구성요소의 역할을 하는 단위 취약점을 제시하기 위한 절차로 사용하였다.

본 연구 결과에서 제시하는 취약점 정보의 분석을 위해서는 표 2의 XML 스키마를 갖는 연도별 취약점 파일을 파싱하여 단위 취약점 추출을 위한 기본정보로 사용되는 Description 데이터를 읽어오는 기능을 구현해야한다.

단계 3. 분할(Decomposition)

취약점을 취약점 표현식으로 표현하는 것은 해당 취약점의 표시를 세분화된 수준으로 수행하고 이들 사이의 관계를 명시하는 것이다. 이렇게 취약점을 여러 개의 조각으로 나누는 개념은 SES의 분할과 같은 개념이며, 각 취약점의 조각은 단위 취약점으로 정의된다. 그림 4에서 제시된 버퍼 넘침 취약점의 분석 결과를 보면 취약점이 여러 개로 분할되고, 각 분할된 취약점들이 단위 취약점으로 나타나는 것을 볼 수 있다. 기존 취약점의 단위취약점으로의 표현을 위해서는, 이와 같이 유형 별로 분할이 먼저 정의된다. 본 연구에서는 분석된 약 1,000개의 취약점에서 7종류의 취약점 (버퍼 넘침, 예외처리 오류, 과도한 서비스 요청, 약한 인증 기술사용, Cross Site Script, 경쟁상태, 스푸핑, 설정



(그림 3) 단위 취약점 추출을 위한 취약점 분석 절차

에러)에 대한 분할을 정의하였다. 여러 개의 단위로 나누어진 취약점은 이들 사이의 관계를 명시하는 이진 연산자로 연결되게 된다. 여기서 사용되는 이진 연산자는 AND, OR, POR, PAND, SAND 이다. AND 연산자는 두개의 단위 취약점이 참(True)인 경우에 참이 되는 경우이며, OR 연산자는 둘 중 하나만 참이면 결과가 참이 된다. POR의 경우 연산대상인 단위 취약점 중 Probabilistic 형태가 있을 경우, 확률적 특성을 고려한 OR 연산을 위한 것이다. 즉, 두개의 단위 취약점 중 하나라도 악용될 확률이 일정 수준인 경우 결과가 참이 된다. PAND는 두개의 단위 취약점이 모두 악용될 확률이 일정 수준 이상일 때 참이 된다. SAND는 두개의 단위 취약점이 순차적으로 악용될 때만 참이 되는 경우에 사용되는 연산자 이다.

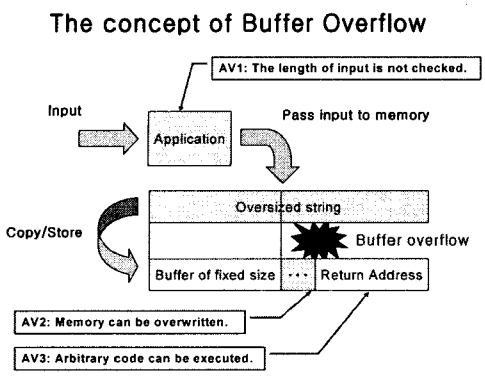
단계 4. 특화(Specialization)

단위 취약점 각각은 의미적으로 더 이상 나눌 수 없는 구성 요소 이지만, 적용 시에 특화된 형태로 나타낼 수 있다. 즉, 프로그램이 수용할 수 없는 긴 외부 입력에 대해서 외부 입력에 대한 세부적인 특성으로 표시될 수 있는 다음 4가지와 같이 나타낼 수 있다.

- 외부에서의 서비스 요청 파라미터(I1)
- 내부에서 보내는 서비스 요청 파라미터(I2)
- 특정 소프트웨어에서 열어볼 수 있는 콘텐츠의 헤더 및 메시지 내용(I3)
- 환경 변수(I4)

단계 5. 검증(Verification)

본 연구의 목적이 취약점 데이터베이스를 만드는 데에 있으며, 그 결과는 취약점 점검도구 및 침입탐지 시스템에서 활용을 목적으로 하기 때문에 그 결과의 신뢰성이 중요하다. 이로 인해 현재의 가장 마지막 단계의 검증은 사람에 의해서 진행 하고 있다.



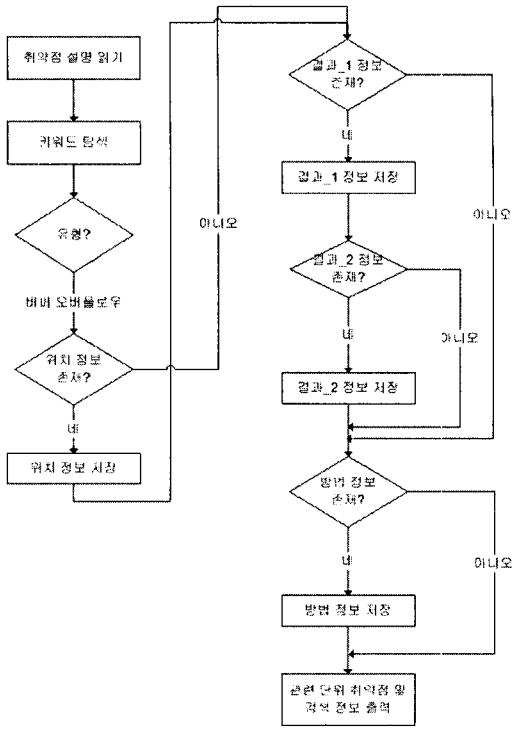
(그림 4) 버퍼 넘침 취약점의 단취 취약점 추출

3.2 취약점 자동 분석 방법연구

본 논문에서는 취약점 유형을 7가지로 분류하였다. 7가지 유형에는 버퍼 넘침(buffer overflow), 비정상적인 요청 처리 불가(failure to handle exceptional requests), 과도한 요청 처리 불가(failure to handle excessive request), 약한 인증(weak authentication), 크로스 사이트 스크립트(cross site script), 스푸핑(spoofing), 설정 오류(configuration error)이다.

이러한 취약성 유형 중에서 버퍼 넘침의 개념을 살펴보면 그림 4와 같다. 버퍼 넘침 유형은 3개의 단위 취약점으로 나누어진다. 먼저 애플리케이션이 입력 데이터의 길이를 확인하지 않는데, 이것이 첫 번째 단위 취약점(AV1)이 된다. 또한 메모리의 일부가 덮어쓰기가 되는 것(AV2)과 이것이 실행되는 것(AV3)이 각각 단위 취약점이 된다. 따라서 이렇게 3가지 단위 취약점의 세부 정보를 정의하기 위하여, 입력 데이터의 종류, 메모리 덮어쓰기가 발생하는 곳의 위치, 결과 등의 정보를 확인하여야 한다.

다음은 유형이 버퍼 넘침인 경우 상세한 처리 흐름도는 그림 5와 같다. 먼저 XML 취약점 데이터베이스를 파싱한 취약점 정보 중 설명(Description) 부분을 읽어 들인다. 취약점의 설명은 기본적으로



(그림 5) 단위취약점 자동 추출을 위한 알고리즘

자연어로 정리되어 있지만, 작성을 위한 일정 패턴이 존재하기 때문에 키워드 검색을 통해 유형을 결정하는 단계를 거치게 된다. 유형 결정 단계를 통해 본 연구에서 분류한 7가지 유형의 취약점 중 하나로 선정이 되면, 이를 기준으로 해당 취약점이 갖는 단위 취약점 및 이들의 관계에 대한 선정이 가능하다. 이렇게 7가지의 취약점 유형 중 하나를 정하는 것은 CVE 기반 취약점을 여러 개의 단위 취약점으로 나누는 과정이기 때문에 분할(Decomposition)이라고 명명하였다. 예를 들어, 버퍼 넘침의 경우, 위에서 설명한 것과 같은 3가지 단위 취약점으로 나누어지는 구조가 되며, 이렇게 버퍼 넘침 취약점이라는 것을 결정하는 것을 기반으로, 이후에 적합한 키워드를 선택한다. 키워드 선택이 끝나면, 취약점이 존재하는 위치, 취약점이 악용 되었을 때의 결과, 취약점을 악용하는 방법을 자동 검색하여, 이에 해당하는 단위 취약점을 지정하게

(표 3) 7개 취약점 유형 별 키워드 추출 방법

취약점 유형	키워드				
	유형	위치	결과_1	결과_2	방법
Buffer overflow	buffer overflow	stack/ heap	execute	[arbitrary] + (commands/code)	(via/through/using/with) + [(overly long/malformed/corrupt/a large number of)] + (input/string)
			(gain/obtain)	[(root/shell)] + (privileges/access)	
			(allow/give)	[arbitrary] + (command/execution)	
			(cause/enable)	(core dump/denial of service/crash/memory consumption/reset/hang/reboot/packet loss/memory exhaustion)	
Failure to handle exceptional requests	denial of service		cause	(system/service/kernel/program/response/application/process) + (crash/hang/reboot/ panic/failure/crash)	(via/through/using/with) + [(a large number of/overly long/malformed/malicious/ invalid/crafted/forged/spoofed)] + (input/method/packet/message/request)
				(CPU/memory/bandwidth/resource) + (consumption/ exhaustion)	
Failure to handle excessive request	denial of service		cause	(system/service/kernel/program/response/application/process) + (crash/hang/reboot/ panic/failure/crash)	(via/through/using/with) + [(large number of/malformed/repeated/flood)] + (input/connection/request/no rate limit)
				(CPU/memory/bandwidth/resource) + (consumption/ exhaustion)	
Weak authentication	authentication /identification		(bypass/execute/read/obtain/establish/gain)	(access/authentication/code/command/connection/data/identification/information/privilege/ right/rule)	without + (the user being aware/authentication/identification)
Cross site script	(cross site script/xss)		bypass	local security zone/zone restriction	(via/by) + attaching inline macro
			execute	Java script/code/script	
			gain	administrator level access/privilege	
			(add/delete/install/modify)	data	
			inject	script	
Spoofing	Spoofing		spoof		(via/by) + (modification/modifying) + (username/content-disposition /content-type)
			execute	arbitrary code	
Configuration error	configuration		bypass		[(default/specific)] + (permissions/configuration/ registry)
			execute	[arbitrary] + (scripts/code/command)	
			gain	(administrator level access/privilege)	
			share	[important] + (information/data)	
			(add/delete/install/modify)	data	

된다. 이렇게 단위 취약점을 지정하는 과정을 특화(Specialization)로 명명하였다. 이러한 과정에서 본 연구에서는 문자열 내의 영문장의 동사와 목적어 부분으로 분리하여 검색하여 검색 효율을 높이도록 하였다.

이러한 검색 및 추출을 통한 분할/특화가 갖는 한계점은 자연어의 특성상 정규화하기 힘든 경우가 많이 있고, 특히 설명 부분을 작성하는 분석자에 따라서 사용하는 단어, 문장이 달라질 수 있어서 완벽한 검색 및 단위 취약성 부여는 쉽지 않다는

(표 4) 키워드 표현에 대한 기호 설명

기호	설명
[ ]	[ ]로 묶여진 정보는 필수적이지 않은 생략 가능한 정보임
/	논리 연산자 OR 관계를 표시
+	논리 연산자 AND 관계를 표시
결과1, 결과2 관계	두 결과사이의 관계는 AND 관계임

점이다. 따라서 그러한 경우를 해결하기 위해 그림 3의 제 5단계에서 명시된 것처럼 사람의 개입이 일정 부분 필요하게 된다.

그림 5의 첫 번째 절차는 설명 정보를 읽어 들여서 키워드를 탐색하는 것이다. 이 키워드의 탐색은 단위취약점의 추출에 가장 중요한 요소로서 이 키워들을 사용하여 취약점의 유형, 위치정보, 결과정보 및 방법정보를 찾아낸다.

취약점의 유형은 7가지 취약점 중 하나를 의미하고, 본 그림에는 버퍼 넘침 유형이라는 것을 발견하였다. 위치 정보는 버퍼 넘침에만 해당하는 정보로, 스택 혹은 힙 중 어떤 부분에서 넘침이 발생했는지를 찾기 위한 것이다. 결과 정보는 2개로 나누어지는데 결과_1은 영어표현의 동사(verb)에 해당하여 결과로 발생하는 동작에 대한 추출이고, 결과_2는 해당 동사의 목적어로서 취약점을 악용한 공격자가 얻게 되는 결과에 해당한다. 따라서 결과_1 정보가 존재하면, 결과_2의 존재 여부를 묻게 되지만, 결과_1이 존재하지 않는 경우 결과_2의 존재를 묻지 않는다. 마지막으로, 방법 정보는 취약점을 악용하는 방법을 의미하며, 주로 via, through, using, with와 같은 단어 뒤에 나오는 내용을 담는다. 이렇게 추출된 정보를 기반으로 단위 취약점(AV)이 지정되고, 취약점 표현식(VX)으로 표현되게 된다.

표 3은 본 연구에서 개발된 취약점 정보 추출 기술의 핵심적인 내용으로 취약점의 유형 별로 위치/결과/방법 각각에서의 키워드를 보여주고 있다. 이 키워드들은 본 연구에서 개발된 SW 내부에 사용되는 값들이다. 각 키워드 들은 XML 형태의 취

(표 5) CVE 취약점 자동 맵팽을 위한 검색 예

CVE-ID	CVE-2002-0364
Description	①*Buffer overflow* in the chunked encoding transfer mechanism in IIS 4.0 and 5.0 allows attackers to ③*execute* ④*arbitrary code* ⑤*via the processing of HTR request sessions*, aka "②*Heap* Overrun in HTR Chunked Encoding Could Enable Web Server Compromise."

약점 설명 데이터를 파싱하여 얻어진 빈도에 의해서 도출된 결과로서, 취약점의 개수가 증가함에 따라, 표 3의 내용은 점점증가 될 것이다.

표 4는 표 3에서 나타난 기호들의 의미에 해당하며, 이를 통해 표 3의 탐색 키워드의 관계를 간결하게 표시 할 수 있었다.

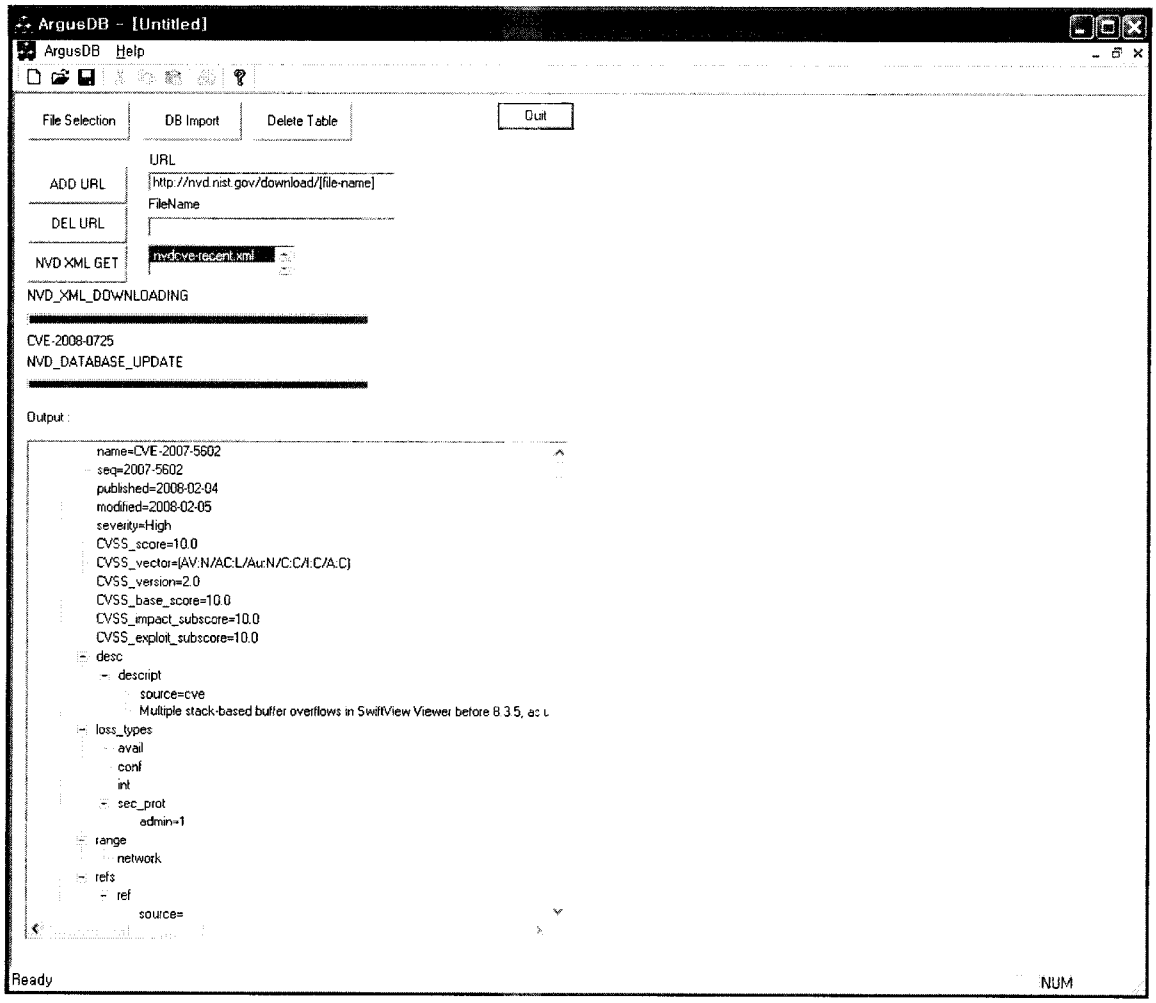
3.3 키워드 추출 예

실제로 CVE-2002-0364를 예로 들어서 검색 방법을 설명하면 표 5와 같다. 먼저 취약점 설명 부분에서 *buffer overflow*라는 키워드를 검색하여 취약점 유형을 버퍼 넘침으로 결정한다. 다음으로 위치 키워드를 검색하여 취약점 위치를 스택(stack)이나 힙(heap)으로 결정한다. 그리고 결과_1 키워드를 검색하여 *execute*를 찾은 후에 결과_2 키워드인 *arbitrary code*를 검색한다. 마지막으로 방법 키워드인 *via*를 검색하게 된다.

4. 구현 소프트웨어 및 실행 결과

앞에서 설명한 검색 알고리즘 및 키워드를 적용하여 C++ 언어를 사용하여 단위 취약점 식별자 부여 자동화 소프트웨어를 구현하였다.

그림 6은 개발된 프로그램의 사용자 인터페이스이다. NVD에서 제공하는 XML 파일을 파싱하여, 단위 취약점 기반 취약점 데이터베이스의 데이터를 생성하는 것이 본 소프트웨어의 역할이다. 특히, 취약점 정보를 가지고 있는 XML 파일은 CVE 취약

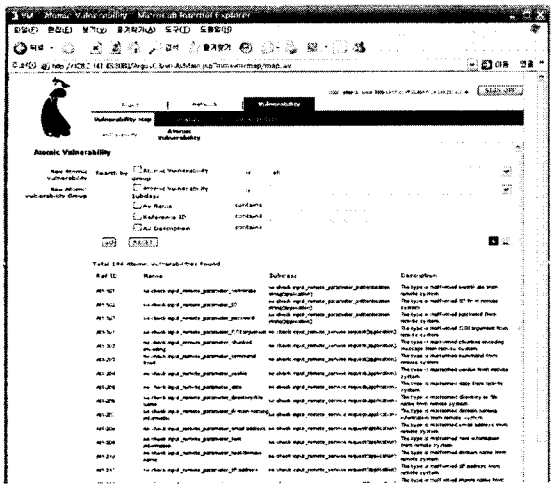


(그림 6) 단위 취약점 식별자 부여 자동화 소프트웨어 사용자 인터페이스

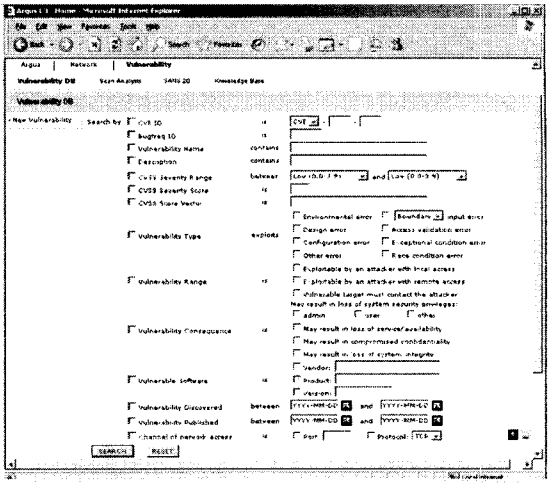
점 데이터베이스의 아카이브 형태로 존재하는 것이기 때문에 필요에 따라 취약점 데이터베이스를 언제든 구축할 수 있게 해준다.

사용자 인터페이스의 버튼들의 역할을 설명하면, XML 파일을 선택하기 위한 "File Selection" 버튼이 있고, 파싱된 결과를 데이터베이스에 저장하기 위한 "DB Import" 버튼이 있다. "Delete" 버튼은 데이터베이스의 테이블들을 초기화하기 위한 것으로, 취약점이 추가되는 과정에서 연결 관계가 깨진 경우 이를 바로잡아주기 위한 용도로 사용된다. 본 인터페이스는 특정 URL의 XML 파일을 읽어 올 수 있는 인터페이스를 제공하며, 이를 입력할 수 있는 창을 제공한다. 입력 후 "ADD URL" 버튼을 클릭하면 "NVD XML GET" 버튼 옆의 리스트 박스에 해당 URL이 추가된다.

이렇게 추가된 XML URL을 선택하여, "NVD XML GET" 버튼을 클릭하면 해당 XML 파일을 다운로드 하여, 파싱을 수행한다. 파싱이 완료 후 "DB Import" 버튼을 클릭하여, 데이터베이스에 결과를 데이터베이스에 저장하게 된다. 그림 6의 가장 하단의 창은 XML 데이터를 파싱한 결과를 트리구조로 보여주기 위한 것이다. 그림 6의 사용자



(그림 7) 취약점 데이터베이스의 단위 취약점 검색 결과



(그림 8) 단위 취약점 기반 취약점 데이터베이스의 검색 화면

인터페이스는 데이터베이스에 자동화된 분석 결과를 저장하기 위한 것이고, 입력된 데이터를 조회하기 위해서는 별도의 웹기반의 사용자 인터페이스를 구현하였다.

그림 7은 취약점 정보를 검색하기 위한 화면이다. 취약점 데이터베이스는 MySQL5.0으로 구축되었으며, 자바 기반의 코드로 인터페이스를 구성하였다.

해당 인터페이스는 취약점 데이터베이스 뿐 아니라, 네트워크의 취약점 관리, 네트워크 대역폭 측정 및 이상 징후 탐지/대응 등의 다양한 정보보호 기능을 제공하고 있다.

그림 7은 그중 취약점의 종류 및 공격 가능 위치, 공격의 위험도 등 CVE에서 제공해주는 다양한 데이터를 기반으로 검색하는 기능을 제공해 주는 화면이다.

그림 8은 단위 취약점의 리스트로서, 단위취약점 추출 자동화는 결국 이들 중 여러 개의 취약점을 선택하여, 단위취약점을 사용하여 CVE 취약점을 표현하기 위한 취약점 표현식을 구성하는 데 있다. 이렇게 표현된 취약점 표현식은 네트워크의 보안 관리를 좀 더 직관적으로 하기위한 기능을 제공해 준다.

5. 결론

본 논문에서는 단위 취약성 분석을 용이하게 수행 가능하도록 기존의 CVE 취약성 정보에서 키워드를 검색하여 해당하는 정보를 수집하고 단위 취약성을 자동으로 맵핑하여 주는 방법을 제안하였다. 특히, 단위 취약점을 추출 알고리즘을 개발하기 위해 5단계의 분석 절차와 이를 통해 분류 가능한 7가지의 취약점 종류에 대해서 제시하였다. 이러한 분석 절차 및 키워드들은 NVD XML을 기반으로 취약점 데이터베이스를 자동으로 구축하는 소프트웨어 형태로 결과를 보였다. 이러한 구현 결과는 취약점을 좀 더 세분화해서 보고자 하는 단위 취약점 기반 분석 방법을 자동화하기위한 연구로서 그 의미를 갖는다.

향후 연구 개발 방향으로는, 현재 키워드들이 소프트웨어 내부의 코드 형태로 들어 있는데, 이를 데이터베이스화 하여 추후 추가될 취약점의 분류 및 키워드들을 효율적으로 관리하기 위한 연구가 필요하다. 또한 정보보호 정책의 결정에 있어서 중요하게 고려되어야 할 것이 네트워크가 갖는 취약점이라고 할 수 있는데, 본 연구에서 자동 생성된 단위 취약점을 활용할 경우 기존 보안 정책이 갖는 한계를 극복할 수 있을 것으로 기대된다.

참 고 문 헌

[1] B. P. Zeigler, *Theory of Modeling and Simulation 2nd Edition*, Academic Press, 2000.

[2] B. P. Zeigler, *Multifacetted Modeling and Discrete Event Simulation*, Orlando, FL: Academic, 1984.

[3] B. P. Zeigler, *Object-Oriented Simulation with Hierarchical, Modular Models*, San Diego, CA, USA: Academic Press, 1990.

[4] HyungJong Kim, KyoungHee Ko, DongHoon Shin and HongGeun Kim, "Vulnerability Assessment Simulation for Information Infrastructure Protection," Proceedings of the Infrastructure Security Conference 2002, LNCS Vol. 2437, pp. 145-161, October, 2002.

[5] HyungJong Kim, "System Specification Network Modeling for Survivability Testing Simulation," Information Security and Cryptology ICISC 2002, LNCS Vol. 2587, pp. 90-106, November, 2002.

[6] D. Bell and L. LaPadula. "Secure Computer System: Unified Exposition and Multics Interpretation," Mitre Corporation ESD-TR-75-306, 1976.

[7] D. Ferraiolo and R. Kuhn. Role-Based Access Control. In Proc. of the NIST-NSA Nat. (USA) Comp. Security Conf., pp 554-563, 1992

[8] Original Paper: Biba, K. J. "Integrity Considerations for Secure Computer Systems", Technical Report MTR-3153, MITRE Corporation, Bedford, Massachusetts, April 1977

[9] M. Condell, C. Lynn, and J. Zao. "Security Policy Specification Language", Internet draft, Internet Engineering Task Force, July 1999

[10] N. Damianou, N. Dulay, E. Lupu and M. Sloman, "The Ponder Policy Specification Language", Proceedings of Workshop on Policies for Distributed Systems and Networks, Bristol, UK, 29-31 Jan. 2001, LNCS 1995, pp. 18-39

[11] Nancy R.Mead et. al., "Survivable Network Analysis Method", CMU/SEI-2000-TR-013, Sep. 2000

[12] Robert J. Ellison, David A. Fisher, Richard C. Linger, Howard F. Lipson, Thomas A. Longstaff, Nancy R. Mead "Survivability: Protecting Your Critical Systems," IEEE Internet Computing, November December, Vol 3, pp. 55-63, 1999

[13] F. Cohen, "Simulating Cyber Attacks, Defenses, and Consequences," Computer & Security, Vol.18, pp. 479-518, 1999

[14] M. Bishop, "Vulnerabilities Analysis," Proceedings of the Recent Advances in Intrusion Detection, pp. 125-136, September, 1999

[15] N. Ye and J. Giordano, "CACA - A Process Control Approach to Cyber Attack Detection," Communications of the ACM, Vol.44(8), pp. 76-82, 2001.

[16] TaeHo Cho and HyungJong Kim, "DEVS Simulation of Distributed Intrusion Detection System," Transactions of the Society for Computer Simulation International, vol. 18, no. 3, pp. 133-146, September, 2001.

[17] Jay Beale, *Snort 2.1 Intrusion Detection 2nd Edition*, Syngress, 2004.

[18] Renaud Deraison, *Nessus Network Auditing*, Syngress, 2004.

[19] NVD: National Vulnerability Database - http://nvd.nist.gov

[20] CVE: Common Vulnerabilities and Exposures - http://cve.mitre.org

[21] CVSS: Common Vulnerability Scoring System - http://www.first.org/cvss

[22] Common Weakness Enumeration - http://cwe.mitre.org

◐ 저 자 소 개 ◑

김 형 종(Hyung-Jong Kim)
1996년 성균관대학교 정보공학과(공학사)
1998년 성균관대학교 대학원 정보공학과2(공학석사)
2001년 성균관대학교 대학원 전기전자및컴퓨터학과(공학박사)
2001년~2007년 한국정보보호진흥원 수석연구원
2004년~2006년 미국 카네기멜론대학 Visiting Researcher
2007년~현재 서울여자대학교 컴퓨터학부 전임강사
관심분야 : 모델링 및 시뮬레이션, 네트워크 취약점 분석, VoIP 스팸
E-mail : hkim@swu.ac.kr