정적 분석을 이용한 알려지지 않은 악성 스크립트 감지

이 성 욱[†]·배 병 우[††]·이 형 준[†††]·조 은 선[††††]·홍 만 표[†††††]

요 약

정적 휴리스틱 분석은 알려지지 않은 악성 코드를 감지하는데 널리 사용되는 기법으로, 악성 코드에 보편적으로 존재하는 코드 조각들을 탐색하여 대상 코드의 악성 여부를 판단한다. 그러나, 스크립트로 작성된 악성 코드에서는 정형화된 코드 조각들을 찾아내기 어려우므로, 특정한 메소드 호출들의 존재만을 검사하는 것이 보편적이다. 이러한 감지 방식은 높은 감지 오류율을 보이게 되는데, 이는 많은 메소드들이 일반 스크립트에서도 빈번하게 사용될 수 있는 것들임에 기인한다. 따라서, 현재 정적 휴리스틱 분석은 일반 스크립트에서 거의 사용되지 않는 특별한 메소드 호출들로 이루어진 악성 행위만을 감지하는데 제한적으로 사용되고 있다. 본 논문에서는 메소드 호출 뿐 아니라 이에 관련된 파라미터와 리턴 값까지 고려하여 악성 행위 패턴을 정확하게 감지함으로써 이러한 단점을 극복할 수 있는 정적 분석 기법을 제안하고 그 구현을 제시한다. 또한, 구현된 시스템 상에서의 실험을 통해, 높은 긍정 오류 때문에 기존 기법의 적용이 어려웠던 악성 행위가 제안된 기법으로 감지될 수 있음을 보인다.

Detection of Unknown Malicious Scripts Using Static Analysis

Seong-uck Lee[†] · Byungwoo Bae[††] · Hyong-joon Lee[†††]
Eun-Sun Cho[††††] · Manpyo Hong[†††††]

ABSTRACT

Analyzing the code using static heuristics is a widely used technique for detecting unknown malicious codes. It decides the maliciousness of a code by searching for some fragments that had been frequently found in known malicious codes. However, in script codes, it tries to search for sequences of method calls, not code fragments, because finding such fragments is much difficult. This technique makes many false alarms because such method calls can be also used in normal scripts. Thus, static heuristics for scripts are used only to detect malicious behavior consisting of specific method calls which is seldom used in normal scripts. In this paper, we suggest a static analysis that can detect malicious behavior more accurately, by concerning not only the method calls but also parameters and return values. The result of experiments show that malicious behaviors, which were difficult to detect by previous works, due to high false positive, will be detected by our method.

키워드 : 악성 스크립트(malicious script), 악성 코드(malicious code), 컴퓨터 바이러스(computer virus), 정적 분석(static analysis)

1. 서 론

악성 스크립트는 스크립트 언어로 작성된 악성 코드를 말한다. 최초의 악성 스크립트는 1994년 12월에 최초로 발견되었으며, 대부분 인터넷 웜의 형태로 메일이나 IRC(Internet Relay Chat) 같은 매체를 통해 전파되고 있다[1]. 악성 코드의 작성에 주로 이용되는 스크립트 언어는 비주얼 베이직 스크립트(Visual Basic Script)와 자바 스크립트(JavaScript)이며, 비주얼 베이직 스크립트로 작성된 것들이 대부분이다.

스크립트 언어는 비교적 간단하여 초보자도 쉽게 익힐 수 있어서 컴퓨터에 관해 전문적인 지식이 없는 사람도 쉽게 악성 스크립트 코드를 생성할 수 있으며, 최근에는 자동으로 악성 스크립트를 생성시켜 주는 생성기까지 인터넷을 통해 유포되고 있는 실정이다. 스크립트의 이러한 성격 때문에 2000년 후반까지 스크립트를 기반으로 하여 구현된 매크로 및 스크립트 바이러스가 전체 악성 코드의 절반 가량을 차지하였으며, 많은 피해를 가져왔다[2]. 현재는 악성 스크립트의 발생 빈도가 이전에 비해 줄었으나, 계속하여 발생하고 있으며, 작성이 용이하다는 특징으로 인해 그 수가 폭발적으로 증가할 가능성이 여전히 존재한다.

이러한 악성 스크립트의 감지에는 이진 형태의 악성 코드와 마찬가지로 시그너쳐(signature) 기반의 스캐닝(scan-

※ 본 연구는 한국전자통신연구원 2002년 위탁연구에 의해 지원되었음.
† 준 회 원 : 아주대학교 대학원 컴퓨터공학과
†† 준 회 원 : (주)트라이튼테크 연구원
††† 준 회 원 : 아주대학교 정보통신전문대학원
†††† 정 회 원 : 충북대학교 전기전자및컴퓨터학부 교수
††††† 종신회원 : 아주대학교 정보및컴퓨터공학부 교수
논문접수 : 2002년 2월 8일, 심사완료 : 2002년 7월 25일

ning)을 통한 방법이 널리 사용되고 있다. 그러나, 이 기법은 분석을 통해 시그너처를 추출한 악성 코드만을 감지할 수 있으므로, 알려지지 않은 새로운 악성 스크립트의 감지를 위해서는 휴리스틱 분석이 주로 이용된다[3]. 휴리스틱 분석은 대상 코드를 스캐닝하여 악성 코드에 보편적으로 존재하는 코드 조각들을 탐색하는 정적 휴리스틱 분석과, 에뮬레이션을 수행하여 나타나는 행위 패턴을 분석함으로써 악성 여부를 판단하는 동적 휴리스틱 분석으로 나누어질 수 있다. 실제에 있어, 에뮬레이션을 통한 악성 행위의 감지는 많은 시간과 시스템 자원을 소모하므로 정적 휴리스틱 기법이 가장 보편적으로 이용된다.

그러나, 이진 악성 코드와 달리 소스 코드 형태로 존재하는 악성 스크립트에서 악성 행위를 수행하는 정형화된 코드 블록을 찾아내는 데에는 많은 어려움이 따르게 된다. 따라서, 악성 스크립트를 대상으로 하는 정적 휴리스틱 분석은 메소드 호출 또는 어트리뷰트와 같은 특정 단어들의 존재나 출현 빈도를 검사하는 방식을 취하고 있다[4]. 이러한 악성 스크립트 감지 방식의 가장 큰 문제점은 높은 감지 오류율이다. 즉, 악성 행위에 사용되는 메소드들 중 상당수는 실제 일반 스크립트에서도 빈번하게 사용될 수 있는 것들이므로, 실제로 악성 행위가 아님에도 불구하고 이를 악성 코드로 간주하는 긍정 오류(false positive)가 빈번하게 발생할 여지를 가지게 된다. 따라서, 현재 정적 휴리스틱 분석은 긍정 오류가 높을 것으로 예상되는 악성 행위의 감지를 포기하고, 일반 스크립트에서 거의 사용되지 않는 특별한 메소드 호출들로 이루어진 일부 악성 행위만을 제한적으로 감지하는데 그치고 있다.

본 논문에서는 메소드 호출 뿐 아니라 이에 관련된 파라미터와 리턴 값까지 고려하여 악성 행위 패턴을 정확하게 감지함으로써 이러한 단점을 극복할 수 있는 정적 분석 기법을 제안하고 그 구현을 제시한다. 또한, 구현된 시스템 상에서의 실험을 통해, 높은 긍정 오류 때문에 기존 기법의 적용이 어려웠던 악성 행위가 제안된 기법으로 감지될 수 있음을 보인다. 이를 위해 본 논문의 2장에서는 기존의 관련 연구들과 문제점을 제시하고, 3장에서는 제안하는 악성 스크립트 감지 기법을 설명한다. 4장에서는 스크립트 샘플에 이를 적용한 실험 결과를 제시하고, 5장에는 결론과 향후 연구를 기술한다.

2. 관련 연구

2.1 악성 스크립트 구조

악성 스크립트 코드가 수행하는 대표적인 악성 행위는 로컬 시스템 또는 네트웍을 대상으로 하는 자기복제이며, 그 외에도 시스템 레지스트리 또는 다른 기존 파일을 변형하는 등의 악성 행위를 수행한다. 악성 스크립트가 수행하는 악성 행위를 정리하면 <표 1>과 같다.

〈표 1〉 악성 스크립트가 수행하는 악성 행위

구 분	악성 행위
자기복제	로컬 시스템에 자기복제
	메일을 통한 자기복제
	IRC 프로그램을 이용한 자기복제
	네트웍 공유 폴더를 통한 자기복제
시스템 정보 변경	레지스트리 변경
파일 변형	데이터 파일 변형
	어플리케이션 설정 변형

여기에서 메일을 통한 자기 복제는 일반적으로 마이크로소프트 아웃룩(outlook)의 주소록을 참조하여 메일에 자신의 파일을 첨부하여 발송하는 방법으로 이루어지며, IRC를 통한 복제는 IRC 클라이언트 프로그램의 스크립트 파일을 수정하여 채팅 시 다른 사용자들에게 자동적으로 전송이 이루어지도록 하는 방법이 이용된다. 시스템 정보 변경은 레지스트리 변경을 통해 시스템 재시작 시에 해당 스크립트가 자동으로 실행되도록 할 목적으로 이루어진다[3]. 악성 코드의 가장 기본적인 특징은 자신과 동일한 이미지를 생성하거나 다른 파일에 기생한 형태로 자신을 전파시키는 자기복제 능력이다. 따라서, 악성 스크립트 감지를 위해 탐색되는 주된 패턴은 자기복제이며, 상대적으로 데이터 파일 수정이나 삭제와 같은 악성 행위는 부가적인 탐지 대상 행위가 된다.

사실, 비주얼 베이직 스크립트나 자바스크립트의 기본 구성 요소만으로는 이러한 악성 행위 수행에 필요한 시스템 자원에 접근할 수 없다. 따라서, 이러한 자원에 접근하기 위해서는 <표 2>에 제시된 COM 또는 ActiveX 객체를 이용한다.

〈표 2〉 악성 행위에 이용될 수 있는 객체

객 체	용 도
Scripting.Filesystem	파일 입출력 관련
WScript.Shell	윈도우 시스템 정보
WScript.Network	네트웍 드라이브 이용
Outlook.Application	메일 전송 관련

"Scripting.filesystem" 객체는 로컬 파일 시스템에 자기복제를 수행하는데 이용된다. 이 객체는 주로 파일 입출력과 관련된 메소드를 지원하며, 이를 사용하여 파일 복사, 파일 생성, 파일 삭제 등의 행위를 하는 스크립트 코드를 작성할 수 있다.

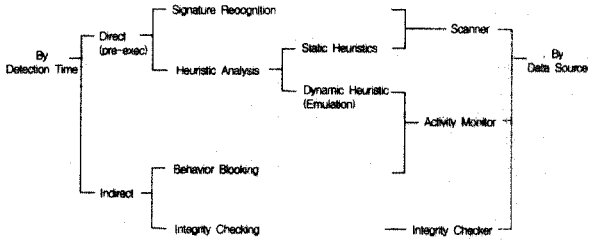
"WScript.Shell" 객체는 윈도우 시스템 정보를 수정하거나 새로운 프로세스를 구동시키기 위해서 이용된다. 이 객체는 윈도우 시스템 레지스트리 정보를 조작할 수 있는 메소드와 새로운 프로세스를 구동시키는 메소드, 기타 환경 설정 값을 조작할 수 있는 메소드를 지원한다. 악성 스크립

트에서는 이 객체에서 지원하는 레지스트리 관련 메소드를 사용하여 시스템 시작과 같은 특정한 시점에 자신의 스크립트가 자동으로 실행되게 하며, 새로운 프로세스를 구동하는 메소드를 사용하여 트로이 목마(trojan horse)와 같은 악성 프로그램을 수행시키기도 한다.

"Outlook.Application" 객체는 전자 메일을 통한 전파에 이용된다. 악성 스크립트에서는 이 객체의 메소드와 어트리뷰트들을 사용하여 주소록을 읽고 자신이 첨부된 새로운 메일을 생성 및 발송한다.

2.2 기존의 악성 코드 감지 기법

악성 스크립트의 감지에는 이진 코드를 위한 기법들을 그대로 이용하거나, 소스 프로그램 형태인 스크립트에 적합하도록 다소간의 변형을 가하여 적용하는 것이 일반적이다[5]. 기존의 악성 코드 감지 기법을 정리하면 (그림 1)과 같다.



(그림 1) 기존의 악성 코드 감지 기법

감지 시점에 의한 분류는 실행 전에 해당 코드를 분석하여 악성 여부를 판단하는 직접적 방식(direct method)과, 실행 중 또는 후에 나타나는 악성 행위 및 결과를 관측하여 판단하는 간접적 방식(indirect method)으로 악성 코드 감지 방식을 크게 구분한 것이다[3]. 이와는 다른 관점에서의 전통적인 분류는 악성 여부를 판단하는 근거에 의한 것으로, 코드의 스캐닝을 통해 특정 패턴을 검색하는 스캐너, 에뮬레이션 또는 실제 실행을 통해 대상 코드의 행위 패턴을 감시하는 행위 감시기, 그리고 화일의 변형을 검사하는 무결성 검사기로 악성 코드 감지 기법을 분류하였다[6].

스캐닝(scanning)을 통한 시그너쳐 인지(signature recognition)는 가장 보편적으로 사용되고 있는 악성 코드 감지 방식이다[7]. 이 방식은 하나의 악성 코드에만 존재하는 특별한 문자열들을 탐색함으로써 해당 코드의 악성 여부를 진단하므로, 진단 속도가 빠르고 악성 코드의 종류를 명확하게 구분할 수 있다는 장점을 가지고 있다. 그러나, 알려지지 않은 악성 코드에 대해서는 전혀 대응할 수 없으므로, 안티바이러스 업체에서 해당 악성코드의 시그너쳐와 치료 방법을 포함하는 새로운 악성 코드 데이터베이스를 배포하기 전까지 많은 사용자들이 악성 코드에 그대로 노출될 수 밖에 없다. 특히, 악성 스크립트들은 대부분 전자우편과 IRC, 네트웍 공유 등을 통해 주로 전파되므로 전파 속도가 빨라 그 피해가 큰 것이 현실이다[8].

휴리스틱 분석(heuristic analysis)은 기본적으로 새로운 악성 코드의 출현은 빈번하게 이루어지나, 새로운 악성 행위 기법의 출현은 매우 드물게 이루어진다는 데에 착안한 것이다[3, 4, 9]. 일반적인 프로그램에 있어서, 특정 기능을 수행하기 위한 새로운 기법의 개발은 몇몇 선도적인 프로그래머 또는 학자들에 의해 이루어지며, 대부분의 프로그래머들은 이렇게 알려진 기법을 이용하여 프로그램들을 작성하게 된다. 악성 코드 또한 프로그램이므로 선도적인 역할을 수행하는 일부 악성 코드 제작자들에 의해 새로운 악성 행위의 기법이 공개되고, 그 뒤로 이를 이용한 다수의 악성 코드들이 출현하게 된다. 따라서, 이미 알려진 악성 행위의 기법에 대한 휴리스틱을 이용하여 주어진 코드를 분석함으로써, 이미 알려진 악성 행위를 포함하고 있는 많은 새로운 악성 코드를 감지할 수 있다.

휴리스틱 분석 기법은 악성 코드 내부에 존재하는 코드 형태에 대한 정적 휴리스틱을 이용하는 것과, 에뮬레이션을 통해 얻어지는 실행 중 발생 행위에 대한 동적 휴리스틱을 이용하는 방법으로 나누어진다. 정적 휴리스틱 분석은 악성 행위에 자주 이용되는 코드 조각들을 데이터베이스화 하여 두고 대상 코드를 스캔하여 그 존재 여부나 출현 빈도를 탐색하여 악성 코드를 감지하는 방식이다. 이 방식은 속도가 비교적 빠르고 높은 감지율을 보이나, 긍정 오류가 다소 높다는 단점을 가지고 있다[10]. 동적 휴리스틱 분석은 가상 기계를 구현한 에뮬레이터 상에서 해당 코드를 수행하면서 프로그램 수행 중에 발생하는 시스템 호출과 시스템 자원(resource)들에 발생하는 변화를 감시함으로써 악성 행위를 감지하는 방식이다[9]. 그러나, 이를 위해서는 완전한 가상 기계를 구현하여야 하며, 한번의 에뮬레이션만으로는 모든 프로그램 흐름을 추적할 수 없다는 단점을 가지고 있다. 특히, 스크립트 코드를 위한 에뮬레이터는 하드웨어, 운영체제 뿐 아니라 관련된 시스템 객체(object) 및 제반환경을 모두 포함하여야 하므로 구현이 매우 어렵고, 부하 또한 큰 것으로 알려져 있다[11].

행위 차단(behavior blocking) 기법은 실제로 대상 시스템에서 코드를 실행시킨다는 점 외에는 동적 휴리스틱을 이용한 감지 방법과 유사한 것으로 생각될 수 있다. 그러나, 에뮬레이션은 아무런 부작용(side effect) 없이 긴 시간 동안의 행위 감시를 통해 대상 코드의 악성 여부를 판별할 수 있는데 반해, 악성 코드를 실제 시스템에서 실행하고 동일한 감시를 수행한다면 악성 행위가 실제로 일어나게 되므로, 디스크 포맷이나 시스템 파일 변형 등과 같이 악성 코드가 실행할 가능성이 높은 각각의 행위가 감지되면 이를 즉각 차단하여야 한다. 따라서, 실질적으로 에뮬레이션에서와 같은 긴 시간 동안의 행위 패턴 감시가 어렵고, 각각의 위험 행위가 발생할 때마다 경고가 주어지므로 매우 높은 긍정 오류(false positive)를 보이게 된다.

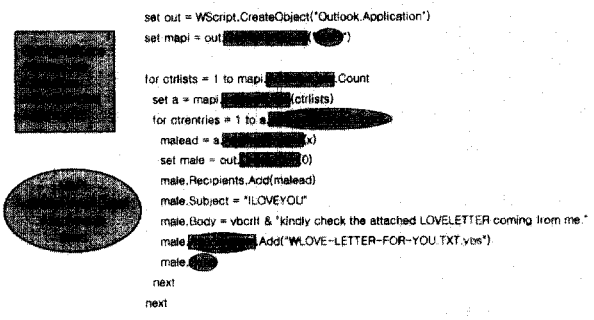
무결성 검사(integrity checking)는 로컬 디스크에 존재하는 파일들 전체 또는 일부에 대하여 파일 정보 및 체크섬, 또는 해쉬 값을 기록하여 두었다가 일정 시간이 지난 후 파일들이 변형되었는가를 검사하는 간접적인 악성 코드 대응 방식이다[12]. 이 방식은 지정된 파일의 변형만을 감지하므로 적법한 내용의 변화가 예상되는 파일에 사용할 경우 매우 높은 긍정 오류를 발생시킨다는 단점을 가지고 있다. 따라서, 서버 상에서 악성 코드 또는 시스템 침입(intrusion)에 의한 변형을 감지할 목적으로 일부 시스템 파일들에 대해서 적용하는 것이 일반적이다.

2.3 기존 정적 휴리스틱 분석 기법의 문제점

상술한 바와 같은 행위 감시와 무결성 검사 기법의 단점들로 인해, 악성 코드 감지 기법 중에서 악성 스크립트의 감지에 가장 현실적인 대안으로 받아들여지고 있는 것은 정적 휴리스틱 분석 기법이며, 스크립트 형태의 특수성을 고려하여 메소드 호출 또는 어트리뷰트와 같은 특정 단어들의 존재나 출현 빈도를 검사하는 방식으로 이용되고 있다.

이 때 검색의 대상이 되는 메소드와 어트리뷰트들은 주로 자기 복제를 수행하는 코드에 나타날 수 있는 것들이다. 주어진 코드가 일반적인 목적을 위해 작성된 정상 코드인지 또는 악성 행위를 위해 작성된 악성 코드인지를 구분하는 것은 프로그래머의 의도(intention)를 파악해내는 문제로 간주될 수 있다. 이러한 판단의 기준으로 가장 보편적으로 사용되는 것은 해당 코드의 자기복제 수행 여부이다. 악성 코드는 가능한 많은 시스템에 전파되어 악성 행위를 수행하려 하는 본질로 인해 자기 복제 루틴을 포함하게 되나, 정상적인 프로그램들은 이 같은 자기 복제를 수행하지 않으므로 가장 근본적인 판단 기준으로 이를 이용할 수 있다. 즉, 주어진 코드의 악성 여부 판별은 자기 복제 행위의 수행 여부를 정확히 판별함으로써 달성될 수 있다.

그러나, 자기 복제 행위에 사용되는 메소드들 각각은 실제 일반 스크립트에서도 빈번하게 사용될 수 있는 것들이므로, 단순한 메소드 존재 유무를 통한 판단만으로는 긍정 오류의 발생 확률이 높다. (그림 2)는 기존 안티바이러스들이 채용하고 있는 정적 휴리스틱 분석의 예이다.



(그림 2) 기존 안티바이러스에서 정적 휴리스틱 분석의 예

그림의 우측에 제시한 것은 러브레터(love letter) 웜의 일부로서 메일을 통해 자기 자신을 발송하는 부분이다. 그러나, 실제로 메일을 통해 자기복제를 수행하는가를 판단하는 것이 아니라 좌측에 열거된 메소드와 어트리뷰트의 존재 여부만을 검색하여 악성 여부를 가리게 된다.

따라서, 좌측 상단의 5개 단어 또는 하단의 4개 단어를 담고 있는 모든 스크립트는 악성 스크립트로 간주된다. 따라서, 주소록에 접근하고 메일을 생성하여 전송하는 일반(legitimate) 스크립트를 악성으로 진단하는 긍정 오류가 발생하게 된다. 그러나, 메일을 전송하는 스크립트가 주소록까지 접근하는 경우는 많지 않으므로, 이것은 상대적으로 긍정 오류의 여지가 적은 경우로 볼 수 있다. 더욱 문제가 되는 것은 (그림 3)과 같은 경우이다.

```
Set fso = CreateObject ( "Scripting.FileSystemObject" )
set file = fso.OpenTextFile ( WScript.ScriptFullname, 1 )
vbscopy = file.ReadAll
file.close
folderlist ( "c : W" )

sub folderlist ( folderspec )
dim f, f1, sf
set f = fso.GetFolder ( folderspec )
set sf = f.SubFolders
for each f1 in sf
  infectfiles ( f1.path )
  folderlist ( f1.path )
next
end sub

Sub infectfiles ( folderspec )
dim f, f1, fc, ap
set f = fso.Getfolder ( folderspec )
set fc = f. Files
for each f1 in fc
  if fso.GetExtensionName ( f1.path ) = "vbs" then
    set ap = fso. OpenTextFile ( f1.path, 2, true )
    ap.write vbscopy
    ap.close
  end if
next
end sub
```

(그림 3) 시스템 내에서 자기 복제를 수행하는 스크립트 코드의 예

제시된 스크립트 코드는 시스템 내의 모든 VBS 파일에 자신의 내용을 겹쳐씀으로써(overwrite) 로컬 시스템 내의 자기복제를 수행한다. 이 코드는 시스템에 존재하는 모든 VBS 파일을 자신과 같은 악성 스크립트로 만드는 악성 행위를 수행함에도 불구하고 파일을 열고, 폴더의 리스트를 얻는 것과 같이 많은 스크립트에서 사용하는 메소드들로만 이루어져 있으므로, 특정 단어의 존재 유무만을 탐색하는 것만으로는 악성 여부를 결정할 수 없게 된다.

따라서, 대부분의 안티바이러스 시스템은 긍정 오류가 높을 것으로 예상되는 악성 행위는 감지를 포기하고, 일반 스크립트에서 거의 사용되지 않는 특별한 메소드 호출들로 이

루어진 일부 악성 행위의 감지에 이 기법을 제한적으로 이용하고 있는 것이 현실이다. 결국, 실제 악성 스크립트들이 알려진 모든 악성 행위를 포함하지는 않으므로, 일반적으로 빈번하게 사용되는 메소드 호출만을 사용하는 악성 스크립트가 출현하였을 때 악성 행위를 탐지하고 악성 여부를 판정하는 것이 매우 어렵게 된다.

3. 정적 분석을 통한 악성 스크립트 감지

3.1 제안 기법 개요

상술한 기존 정적 휴리스틱 기법의 단점은 각각의 위험한 메소드 호출이 아니라 악성 행위를 구성하는 메소드 시퀀스들을 정의하고 악성 행위를 정확하게 감지함으로써 극복될 수 있다. (그림 4)는 (그림 2)에서 제시되었던 자기복제 코드 패턴에서 일부 주요 문장만을 발췌한 것이다.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set dirsystem = fso.getSpecialFolder(1)
Set c = fso.GetFile(WScript.ScriptFullName)
c.Copy(dirsystem&"\LOVE-LETTER-FOR-YOU.TXT.VBS")
set out = WScript.CreateObject("Outlook.Application")
set male = out.CreateItem(0)
male.Attatchments.Add(dirsystem&"\LOVE-LETTER-FOR-YOU.TXT.VBS")
male.send
```

(그림 4) 메일을 통한 자기 복제를 수행하는 비주얼 베이직 스크립트의 일부

제시된 예에서 확인 할 수 있듯, 다수의 메소드 호출이 하나의 악성 행위를 구성하기 위해서는 반드시 그것들의 파라미터와 리턴 값 사이에 특별한 관계가 존재하여야 한다. 예를 들어, 4행의 Copy 메소드는 현재 실행 중인 스크립트를 "LOVE-LETTER-FOR-YOU.TXT.VBS"라는 이름으로 복사하고, 7행의 "Attachments.Add" 메소드는 그 파일을 새로 만들어진 메일 객체에 첨부함으로써 메일을 통한 자기 복제를 달성한다. 그러나, 메소드 호출의 존재유무만을 검사하는 방식을 사용하게 되면, A라는 이름으로 스크립트 파일을 생성하고 B라는 이름의 파일을 첨부하는 관계없는 메소드 호출이 존재하여도 이를 악성 코드로 간주하므로 높은 긍정 오류를 보이게 되는 것이다. 즉, (그림 4)의 4행은 동일하지만 7행에서 메일에 첨부하는 파일이 "MYPIC.JPG"라는 전혀 관계없는 파일이었다면 이것은 메일을 통한 자기 복제로 판단되지 않아야 한다. 또한, 다른 변수들의 검사도 같은 맥락에서 이해될 수 있는데, 3행의 "c"는 해당 스크립트 자신의 파일 핸들을 가지게 되고 4행의 copy 메소드 호출을 통해 로컬 복사본을 생성하게 된다. 그러나, 만약 copy 메소드의 호출이 "d.copy …"와 같이 전혀 관계없는 다른 파일 객체의 메소드로 되어 있는 스크립트가 주어졌다면, 이것은 명백히 무관한 다른 파일의 복사본을 만드는 것일 뿐 자기 복제가 아니라고 할 수 있다.

기존의 정적 휴리스틱 분석은 단지 자기 복제 행위에 사용될 수 있는 메소드 호출 시퀀스의 존재 여부만을 가지고 자기 복제 행위를 수행하는 코드가 있는지를 판단한다. 예를 들어, 주소록에 있는 각각의 대상에게 자신의 사진을 메일로 전송하는 스크립트가 주어졌을 때, 기존의 정적 휴리스틱 분석은 주소록 검색과 메일 전송을 수행하는 메소드 시퀀스가 발견되었으므로 이것을 악성 코드라고 진단한다. 그러나, 제안하는 기법은 악성 행위를 구성하는 메소드 시퀀스의 파라미터와 리턴 값들까지 참조하므로 메일에 첨부된 파일이 자기 자신 또는 자신의 복제본이 아니라면 이것을 악성 행위로 간주하지 않게 된다. (그림 4)와 같은 예에서, 제안하는 기법은 메소드 호출의 존재 뿐 아니라, 사용된 파일명, "fso", "c", "out", "male" 등 모든 관계 있는 값들이 일치하는가를 검사함으로써, 단순한 문자열 탐색보다 정확한 감지 결과를 얻을 수 있다. 즉, 제안하는 기법은 기본적으로 악성 행위에 대한 휴리스틱을 이용한다는 점에서 기존 기법과 유사하지만, 컴파일러 최적화 또는 소프트웨어 엔지니어링 분야에서 프로그램의 분석에 이용되던 코드 정적 분석 기법과 유사한 정밀한 분석을 수행한다는 차이점을 가지고 있다.

실제에 있어 이러한 악성 행위는 단순히 일련의 메소드 시퀀스로만 정의할 수 없으며, 다양한 메소드 또는 메소드 시퀀스들의 조합으로 이루어진다. 따라서, 제안하는 기법에서는 악성 행위가 단위 행위들의 조합으로 이루어지며, 각각의 단위 행위는 더욱 작은 단위 행위 또는 하나 이상의 메소드 호출들로 이루어진다고 모델링하고, 각 단위 행위와 메소드 호출 문장을 하나의 규칙(rule)으로 표현한다.

3.2 행위 패턴 규칙

악성 행위 패턴 규칙은 스크립트 코드에서 탐지될 문장 형태를 정의하는 매칭 규칙(matching rule)과 매치된 패턴 간의 관계를 정의하는 관계 규칙(relation rule)으로 구분되며, 이들의 형식을 BNF로 표기한 것이 (그림 5)이다.

```
< Match_Rule >      ::= < match_ruld_id > " : " < pattern >
< pattern >         ::= { < variable > | < string > | " * " |
                          < char > }1+
< Relation_Rule >   ::= < relation_rule_id > " : " [ < precond > ]
                          [ < cond > ] < action >
< cond >            ::= "cond" < rule_variable > [ "==" | "<" ]
                          < rule_variable >
< precond >         ::= "precond" < relation_rule_id > { ","
                          < relation_rule_id > }
< action >          ::= "action" [ < assignment > | alert ]1+
```

```
< assignment >        : : = < variable > "=" < rule_variable >
< rule_variable >     : : = (< relation_rule_id > "." < variable >) |
                            (< match_rule_id > "." < variable >)
< variable >          : : = "$" < digit >
< relation_rule_id >  : : = "R" [ < digit > | < alpha >]
< match_rule_id >     : : = "M" [ < digit > | < alpha >]
```

(그림 5) 규칙 표기 형식

"<Match_Rule>"은 매칭 규칙이며, 규칙을 나타내는 식별자(identifier)와 탐지할 패턴으로 구성된다. 식별자는 "M"으로 시작하며 규칙 종류와 번호가 덧붙여진다. 탐지할 패턴은 악성 행위를 구성하는 문장 패턴으로, 감지 대상이 되는 스크립트 언어와 동일한 문법을 가진다. 단, 각 메소드가 사용하는 인자와 리턴 값을 규칙 변수(variable)로 바꾸어 넣어 다른 규칙이 이것을 이용할 수 있도록 한다. "<Relation_Rule>"은 관계 규칙을 의미하며, 매칭 규칙을 만족하는 문장에 사용된 규칙 변수의 관계를 분석하여 악성 행위를 찾는데 이용된다. 관계 규칙은 기 만족 조건(precond), 조건식(cond), 동작부(action)로 구분된다. 조건식에는 해당 규칙이 만족되기 위한 조건이 기술되며, 기 만족 조건에는 조건식 만족 이전에 만족되어야 하는 규칙이 기술된다. 즉, 하나의 규칙은 기 만족 조건에 기술된 규칙이 이미 만족되었고 조건식에 기술된 내용이 참일 때 만족되며, 이 때 동작부의 내용이 실행된다.

한편, 앞에서 밝힌 바와 같이 악성 스크립트에 존재하는 악성 행위들은 다양한 형태를 가지고 있으나, 악성 코드의 본질상 가장 핵심이 되는 악성 행위는 자기복제라 할 수 있다. 따라서, 자기복제 행위를 대상으로 악성 행위 패턴 규칙의 기술 사례를 제시한다.

로컬 시스템상의 자기복제는 가장 기본이 되는 악성 행위이며 로컬 디스크에 자신과 동일한 내용의 스크립트를 생성한다. (그림 6)은 로컬 복제 행위를 나타낸 규칙이다.

```
ML1       : $1.copyfile wscript.scriptfullname, $2 [, *]
RLOCAL    : precond ML1
            action $1 = ML1.$2
                 Alert local copy
```

(그림 6) 로컬 복제 행위 감지를 위한 규칙의 예

실제 정적 분석의 진행 중에 스크립트에서 "ML1"에 기술된 형태의 문장을 발견하면, 해당 규칙이 만족되었음을 기록하기 위해 규칙의 인스턴스(instance)를 생성하고, 여기에 "$1"과 "$2"에 해당하는 문자열을 저장한다. 또한, 뒤이은 관계 분석 단계에서 "RLOCAL"이 "ML1"의 만족과 동시에 자동적으로 만족되는 규칙임이 밝혀지고, "ML1"의 "$2"값이 보관된다. 그림의 "M1"에서 "[ ]"로 표기된 부분의 내용은 해당 부분이 옵션(option)이므로 존재하지 않을 경우도 있음을 나타내며, 정확한 인자 분석을 위해 괄호 안의 형태가 나타날 경우 해당 부분을 무시한다. 결국, 이 같은 과정을 거쳐 정의된 로컬 복제 행위 패턴이 탐지되며, 다른 규칙에서 이 정보를 이용할 수 있도록 복사된 파일명을 규칙 변수 "RLOCAL.$1"에 저장한다.

메일을 통한 자기복제는 로컬 시스템에 복제된 파일 또는 자신의 원본 파일을 메일에 첨부하여 전송하는 행위이다. (그림 7)은 로컬 복제본의 첨부 및 발송을 탐지하기 위한 규칙으로, 복제된 파일을 메일에 첨부하는 부분과 메일을 전송하는 부분으로 이루어져 있음을 알 수 있다. "MA1"과 "MS1"은 각각 메일에 파일을 첨부하는 행위와 메일을 전송하는 코드를 나타내며, "RATTACH"는 "MA1"과 로컬 복제 행위 탐지 규칙 "RLOCAL"의 파일명이 일치될 경우 만족된다. "RSEND"는 메일에 파일을 첨부하는 행위 "RATTACH"와 메일 전송 행위인 "MS1"이 존재하고, 메일을 전송하는 객체와 파일을 첨부하는 객체가 동일한 경우에 만족된다.

```
MA1       : $1.Attachments.Add $2
RATTACH   : cond RLOCAL.$1 == MA1.$2
            action $1 = MA1.$1
MS1       : $1.Send
RSEND     : cond RATTACH.$1 == MS1.$1
            action Alert spread by Mail
```

(그림 7) 메일을 통한 자기복제를 감지하는 규칙의 예

전 세계적으로 가장 많이 사용되는 채팅 프로그램 중 하나인 IRC 프로그램은 대부분 자신의 실행 환경과 이벤트(event)를 지정하는 설정 파일이 존재한다. 많은 악성 스크립트들은 이와 같은 IRC 프로그램의 설정 파일을 수정하여, 채팅 중에 대화 상대에게 로컬 복사본이나 자신의 원본 파일을 자동 전송하도록 한다. (그림 8)은 IRC를 통한 전파 행위를 감지하기 위한 규칙의 예이다.

```
MI1       : * send $nick $1
RIRC      : cond RLOCAL.$1 < MI1.$1
            action Alert spread by IRC
```

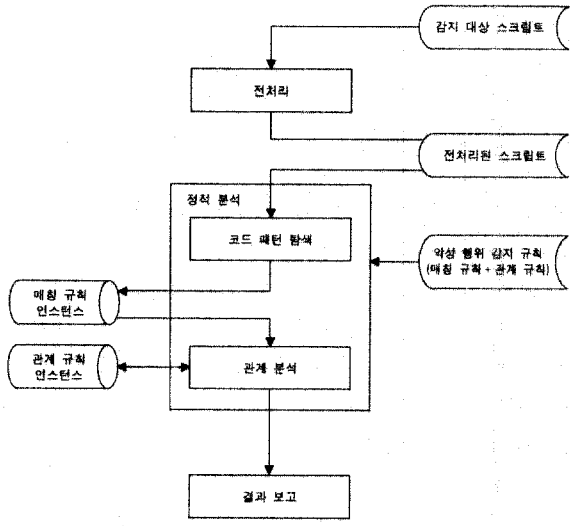
(그림 8) IRC를 통한 자기복제를 감지하는 규칙의 예

"<" 연산자는 우측의 규칙변수가 담고 있는 문자열이 좌측 규칙변수의 문자열을 포함하고 있는가를 검사한다. 따라서, 이 예에서는 스크립트의 "send $nick" 뒤에 존재하는 문자열에 로컬 복제본의 파일명이 나타나는가를 검사하게 된다.

3.3 악성 스크립트 감지를 위한 정적 분석 시스템 구현

제안된 시스템은 전처리, 코드 패턴 탐색, 관계 분석 과정을 거쳐 스크립트의 악성 여부를 판정하고 결과를 보고한다. 이러한 작업 수행 과정을 도시하면 (그림 9)와 같다.

전처리 과정은 주어진 스크립트를 정적 분석에 적합한 형태로 변환하는 과정이다. 많은 악성 스크립트들은 안티바이러스가 자신을 감지하는 것을 어렵게 하기 위하여 암호화된 형태로 존재하거나, "chr( )" 함수를 이용하여 일부 문자열을 아스키 코드 형태로 인코딩하는 방법을 사용하고 있다. 이러한 문제는 기존의 정적 휴리스틱 분석을 위한 전처리 과정과 동일하게 휴리스틱과 부분적 에뮬레이션을 이용하여 대응할 수 있다[3].



(그림 9) 정적 분석 과정

코드 패턴 탐색 과정에서는 스크립트 코드에서 매칭 규칙과 부합되는 코드 패턴을 찾고, 각 패턴에서 사용된 함수의 인자들을 추출하여 규칙 변수에 저장함으로써, 매칭 규칙의 인스턴스를 생성한다. 즉, 코드 패턴 탐색 과정이 종료된 후에는 주어진 매칭 규칙의 집합에 부합되는 스크립트 문장 각각에 대응되는 매칭 규칙 인스턴스가 얻어지게 된다.

관계 분석은 앞 단계에서 얻어진 매칭 규칙의 인스턴스 집합에서 관계 규칙을 만족하는 것들을 찾아내는 과정이다. 코드 패턴 탐색 과정과 마찬가지로 각각의 관계 규칙이 만족되면 관계 규칙의 인스턴스가 만들어지며, 이 때 해당하는 관계 규칙에 연관된 다른 관계 규칙의 만족 여부를 계속적으로 검사해 나간다는 점이 앞의 단계와 다르다.

결과 보고는 관계 분석을 통해 탐지된 악성 행위와 해당 코드의 악성 여부를 사용자에게 보고하는 단계를 지칭한다. 대부분의 악성 스크립트는 다른 프로그램에 기생하지 않고 독립적인 프로그램으로 존재하는 웜 형태이므로, 해당 스크립트 파일을 삭제하여 악성 행위에 대응할 수 있다.

4. 실험 및 결과 분석

기존의 정적 휴리스틱 분석은 특정 악성 행위에 필수적인 메소드 또는 어트리뷰트의 존재 유무를 검사하여 해당 악성 행위를 탐지한다. 이러한 감지 방식은 일반적으로 잘 사용되지 않는 메소드들의 조합으로 구성된 악성 행위의 감지에는 유용하게 이용될 수 있으나, 일반적인 스크립트에서 보편적으로 사용되는 메소드 호출만으로 구성된 악성 행위를 감지하려고 시도하면 높은 긍정 오류율을 보이게 된다. 이러한 이유에서 정적 휴리스틱 분석은 긍정 오류가 비교적 낮을 것으로 예상되는 악성 행위의 감지에만 제한적으로 이용되고 있다. 따라서, 정적 휴리스틱 분석을 통해 감지를 시도하기 어려운 악성 행위들만을 수행하는 악성 코드가 주어지면, 이를 감지하는 것이 매우 어렵게 된다.

제안된 기법은 기존의 정적 휴리스틱 분석 기법과 같이 단순한 문자열 탐색에 의존하지 않고, 악성 행위를 구성하는 일련의 코드를 정확하게 탐지함으로써, 기존 기법에서 감지하기 어려웠던 악성 행위를 정확하게 감지할 수 있다. 따라서, 실험은 기존 안티바이러스와 제안된 기법에 대한 단순한 감지율 비교 뿐 아니라, 기존 안티바이러스가 감지하지 못하는 자기 복제 행위를 찾아내고 제안된 기법으로 이들에 대한 감지를 시도하는 방식으로 이루어졌다. 실험에 사용된 샘플은 비주얼 베이직 스크립트로 작성된 것들로 인터넷에서 수집된 50개의 악성 스크립트와 비주얼 베이직 스크립트 웜 생성기로 만들어진 50개였다.

악성 코드의 감지 정확도를 검증하기 위한 실험에서 가장 문제가 되는 것은 충분한 수의 악성 코드 샘플을 수집하는 것이다. 이것이 어려운 이유는 특정 기간 내에 인터넷에 유포되는 악성 코드의 수는 알려진 악성 코드의 수에 비해 현저히 적기 때문이다. 일반적으로 현재까지 알려진 악성코드는 변종까지 포함할 때 수만 종에 이르는 것으로 집계되고 있으나, 와일드 리스트(wild list)에 따르면 특정 시기에 활동하는 악성 코드는 200종 내외이며, 악성 스크립트는 그 중의 10% 내외인 것으로 나타나고 있다[13]. 따라서, 가급적 많은 수의 샘플을 확보하고, 알려지지 않은 악성 스크립트에 대한 감지율을 반영하기 위하여 웜 생성기를 통해 50종의 새로운 악성 스크립트를 만들어 내고 이를 실험에 사용하였다. 그러나, 웜 생성기를 통해 생성된 스크립트 코드는 대개 일정한 패턴을 가지고 있으므로, 수작업에 의해 주석 삭제, 변수명과 문자열 상수의 변경, 그리고 동작에 문제가 없는 문장의 순서를 바꾸는 등의 변형을 통해 새로운 악성 코드를 만들어 냈다. 이러한 샘플 집합에 대해 3개의 기존 안티바이러스와 제안된 기법이 보여준 감지 결과는 <표 3>과 같다.

괄호 안의 수치는 정적 휴리스틱 기법에 의해 감지된 수를 의미한다. 기존 안티바이러스들은 대부분의 기존 악성 스크립트를 시그너처로 감지하며 휴리스틱 분석을 사용한 예가 적었다. 또한, 새롭게 만들어진 악성 스크립트가 감지된 수가 많지 않음을 볼 때, 앞에서 밝힌 제약 사항으로 인

〈표 3〉 기존 안티바이러스와 제안된 기법의 악성 코드 감지

구 분	A	B	C	제안된 기법
수집된 악성 스크립트	34 (0)	47 (4)	50 (0)	50 (50)
웜 생성기로 생성 + 수작업 변형	0 (0)	15 (15)	50 (0)	50 (50)
계	34 (0)	62 (19)	100 (0)	100 (100)

해 정적 휴리스틱 기법을 적극적으로 활용하고 있지 못함을 알 수 있다. 특이한 점은 안티바이러스 "C"의 경우로, 수작업으로 변형된 악성 스크립트를 여전히 웜 생성기에 의해 만들어진 것으로 진단하고 있었다. 이것은 웜 생성기를 통해 만들어진 다양한 패턴에 대한 방대한 시그너처를 보유하고 그 매칭에 다소간의 융통성을 두는 감지 방식을 사용하고 있는 것으로 유추될 수 있다. 결국, 대부분의 안티바이러스는 감지 정확도가 비교적 높으며 해당 악성 코드의 정확한 명칭까지 알려주는 시그너처 기반의 감지 방식을 먼저 수행하고, 이를 통해 감지되지 않는 경우에만 알려지지 않은 악성 코드를 감지할 수 있는 정적 휴리스틱 기법을 적용하는 것이 일반적이다.

기존 안티바이러스들이 시그너처 기반의 감지를 수행하지 않고 휴리스틱 분석을 이용하도록 하는 가장 손쉬운 방법은 악성 스크립트에 약간의 변형을 가하는 것이다. 일반적으로 스크립트를 위한 시그너처는 특정 부분의 문자열 하나만을 취하는 것이 아니라 코드 전체에 걸쳐 많은 양의 문자열을 지정하는 것이 보편적이다. 따라서, 프로그램에 사용된 변수 이름이나 문자열 상수를 변경하거나 동작에 문제가 없는 문장의 순서를 바꾸는 것만으로도 시그너처에 의한 감지를 피할 수 있게 된다. 그러나, 이러한 실험만으로는 특정 악성 행위의 감지 여부를 판단할 수 없으므로, 각각의 악성 스크립트를 분리하여 하나의 악성 행위만을 수행하는 여러 개의 스크립트로 만들고 이에 대한 실험을 수행하였다. 즉, 로컬 시스템의 복사본 생성, 메일, 그리고 IRC를 통해 자기 복제를 수행하는 악성 스크립트가 주어졌을 때, 각각의 자기 복제를 수행하는 3개의 스크립트 코드로 분리하여 검사를 실시하면, 시그너처에 의한 감지는 거의 이루어지지 않고 휴리스틱 분석에 의해 악성 여부를 판단하게 된다. 이렇게 분리된 각각의 코드에 대한 감지 결과를 정리하면 <표 4>와 같다.

〈표 4〉 기존 안티바이러스와 제안된 기법의 악성 행위 감지

자기 복제 행위	전체	A	B	C	제안된 기법
로컬 시스템에 복사본 생성	95	0	0	0	76
메일 전송	35	0	35	14	30
IRC 이용	46	0	0	0	44
네트웍 공유 폴더	2	0	0	0	2

이 표에서 "전체"는 분리된 악성 코드의 실제 개수를 의미한다. 하나의 악성 스크립트는 하나 이상의 자기 복제 행위를 포함하는 것이 일반적이므로 이들의 총합이 샘플 악성 스크립트의 수 보다 큰 것이 정상이다. 이상의 실험에서, 기존 안티바이러스들은 알려지지 않은 악성 스크립트를 위한 정적 휴리스틱 분석을 수행할 때, 메일 전송을 통한 자기 복제 행위를 주로 감지하며, 상대적으로 긍정 오류의 여지가 높은 행위들은 적극적으로 감지하지 않고 있음을 확인할 수 있다. 따라서, 기존 기법으로 감지가 어려운 다른 수단만을 이용하는 새로운 악성 스크립트는 감지할 수 없게 된다.

긍정 오류는 정상 스크립트를 악성으로 간주하는 오류이다. 하나의 프로그램 언어로 작성될 수 있는 프로그램의 수는 무한하므로, 무작위적으로 수집된 정상 코드에 대한 실험으로는 긍정 오류를 올바르게 측정할 수 없다. 따라서, 긍정 오류의 실험을 위해 사용된 스크립트들은 인터넷의 비주얼 베이직 스크립트 관련 사이트로부터 수집된 것들 중, <표 2>에 제시된 객체들을 포함하는 것들 50개가 선별되었다. 그러나, 이러한 일반적인 코드들만으로는 긍정 오류의 발생 빈도가 지나치게 낮아 비교가 정확하게 이루어지지 않을 가능성이 높으므로, 기존의 악성 코드 10개를 수정하여 악성 행위를 수행하지 않도록 한 뒤 추가하였다. 이렇게 수정된 스크립트 코드는 악성 행위의 구성에 사용되는 메소드들을 다수 포함하고 있으나, 실제로는 아무런 악성 행위를 수행하지 않게 된다. 또한, 기존 안티바이러스들이 시그너처로 사용할 것으로 예상되는 문자열들을 삭제하여, 모든 감지가 휴리스틱 분석에 의해서만 이루어지도록 하였다. 이렇게 구성된 60개의 정상 스크립트 집합에 대한 긍정 오류 발생은 <표 5>와 같다.

〈표 5〉 기존 안티바이러스와 제안된 기법의 긍정 오류 발생

안티바이러스	긍정 오류
A	0
B	6
C	9
키워드 검색	10
제안된 기법	0

여기에서 "키워드 검색"은 기존의 정적 휴리스틱 분석 기법을 의미하며, 가장 높은 긍정 오류를 보임을 확인할 수 있다. 안티바이러스 "A"는 <표4>에 나타난 것과 같이 휴리스틱 분석을 거의 수행하지 않으므로 긍정 오류가 나타나지 않았고, 안티바이러스 "B", "C"는 메일 전송 행위에 관련된 정적 휴리스틱 분석에서 긍정 오류가 나타나는 것으로 추정할 수 있다. 반면, 제안된 기법은 정확한 자기 복제 행위가 존재하지 않으면 이를 악성으로 간주하지 않으므로 주어진 모든 스크립트를 정상 코드로 보고하였다.

이와 같이, 제안된 기법은 단순한 키워드(keyword)의 검

색에 의존하지 않으며, 악성 행위에 사용되는 모든 메소드 호출이 존재하여도 이들의 인자와 리턴 값들이 정확하게 정해진 관계를 만족하지 않으면 이것을 악성 행위로 간주하지 않는다. 이로 인해 <표 4>와 같이 악성 행위가 실제로 존재함에도 불구하고 이를 감지하지 못하는 부정 오류가 발생할 뿐, 긍정 오류의 확률은 극히 낮아지게 된다. 즉, 이러한 정적 분석 방식은 실행 시간 중의 메소드 호출 감시[14]를 통해 악성 행위를 탐지하는 기존의 행위 감시 기법과 동일한 수준의 긍정 오류율을 가지게 되며, 단지 실행 전에 수행되는 정적 분석의 한계로 인해 그 값이 실행 중에 일치함을 결정할 수 없는 데이터가 존재할 경우 부정 오류를 발생하게 된다.

5. 결 론

정적 휴리스틱 분석은 알려지지 않은 악성 코드를 감지하는데 가장 널리 사용되는 기법이다. 그러나, 이진 파일 형태로 존재하는 악성 코드와는 다른 스크립트 코드의 특수성으로 인해, 악성 스크립트를 대상으로 하는 정적 휴리스틱 분석은 특정한 메소드 호출들의 존재만을 검사하는데 그치고 있다. 이러한 감지 방식은 일반 스크립트에서도 빈번하게 사용될 수 있는 메소드 호출만으로 구성된 악성 행위는 감지하기 어려우므로, 실제로는 일반 스크립트에서 거의 사용되지 않는 특별한 메소드 호출들로 이루어진 악성 행위만을 감지하는데 제한적으로 사용되고 있다. 본 논문에서는 메소드 호출 뿐 아니라 이에 관련된 파라미터와 리턴 값까지 고려하여 악성 행위 패턴을 정확하게 감지함으로써 이러한 단점을 극복할 수 있는 정적 분석 기법을 제안하였다. 이를 구현한 시스템 상에서의 실험 결과는 높은 긍정 오류 때문에 기존 기법의 적용이 어려웠던 악성 행위가 제안된 기법으로 감지될 수 있음을 보여주고 있다.

참 고 문 헌

[1] Alex Shipp, "Heuristic Detection of Viruses within Email," virus bulletin conference, 2001.

[2] CERTCC-KR, "2000년 5월 바이러스 통계", http://www.certcc.or.kr/statistics/virus/virus-200005.html, 2000.

[3] Francisco Fernandez, "Heuristic Engines," 11th International Virus Bulletin Conference, 2001.

[4] Igor Muttik, "Stripping down an AV Engine," Virus Bulletin Conference, 2000.

[5] Vesselin Bontchev, "Macro Virus Identification Problems," 7th International Virus Bulletin Conference, 1997.

[6] Eugene H. Spafford, "Computer Viruses as Artificial Life," Journal of Artificial Life, MIT Press, 1994.

[7] Sandeep Kumar, Eugene H. Spafford, "A Generic Virus Scanner in C++," Purdue University Technical Report CSD-TR-92-062, 1992.

[8] Mark Kennedy, "Script-Based Mobile Threats," Symantec White Paper, 2000.

[9] Baudouin Le Charlier, Morton Swimmer, Abdelaziz Mounji, "Dynamic detection and classification of computer viruses using general behaviour patterns," Fifth International Virus Bulletin Conference, Boston, pp.20-22, September, 1995.

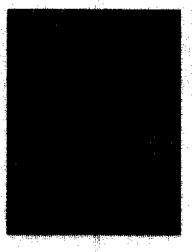
[10] Symantec AntiVirus Research Center, "Understanding Heuristics," Symantec White Paper, 1998.

[11] Gabor Szappanos, "VBA Emulator Engine Design," Virus Bulletin Conference, 2001.

[12] Gene H. Kim, Eugene H. Spafford, "The Design and Implementation of Tripwire : A File System Integrity Checker," ACM Conference on Computer and Communications Security, 1994.

[13] The WildList Organization International, "PC Viruses In-the-Wild - February, 2002," http://www.wildlist.org/WildList/200202.htm, 2002.

[14] Tim Hollebeek and Dur Berrier, "Interception, Wrapping and Analysis Framework for Win32 Scripts," Cigital Labs.



이 성 욱

e-mail : suleeip@yahoo.co.kr

1994년 아주대학교 컴퓨터공학과 졸업 (학사)

1996년 아주대학교 교통공학과(공학석사)

1996년~1997년 기아정보시스템 지능형 교통시스템팀

1997년~현재 아주대학교 컴퓨터공학과 박사과정

관심분야 : 병렬처리, 컴퓨터 보안

배 병 우

e-mail : vessel35@chollian.net

1999년 아주대학교 컴퓨터공학과 졸업 (학사)

2002년 아주대학교 정보통신전문대학원 정보통신공학과(공학석사)

2002년~현재 (주)트라이튼테크 연구원

관심분야 : 컴퓨터 보안, KM/EDMS

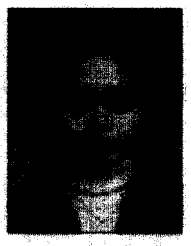
이 형 준

e-mail : prime@doit.ajou.ac.kr

2002년 아주대학교 정보 및 컴퓨터 공학부 졸업(학사)

2002년~현재 아주대학교 정보통신전문대학원 석사과정

관심분야 : 컴퓨터 보안



조 은 선

e-mail : eschough@chungbuk.ac.kr

1991년 서울대학교 자연과학대학 계산통계학과 졸업(학사)

1993년 서울대학교 자연과학대학 전산과학과(석사)

1998년 서울대학교 자연과학대학 전산과학과(박사)

1999년~2000년 한국과학기술원 전산학과 연구원

2000년~2002년 아주대학교 정보통신전문대학원 조교수대우

2002년~현재 충북대학교 전기전자및컴퓨터학부 조교수

홍 만 표

e-mail : mphong@ajou.ac.kr

1981년 서울대학교 자연과학대학 계산통계학과 졸업(학사)

1983년 서울대학교 자연과학대학 계산통계학과(석사)

1991년 서울대학교 자연과학대학 계산통계학과(박사)

1983년~1985년 울산공과대학 전자계산학과 전임강사

1993년~1994년 미네소타대학 전자공학과 교환교수

2000년~2001년 조지와싱턴대학교 컴퓨터과학과 교환교수

1985년~현재 아주대학교 정보및컴퓨터공학부 교수

관심분야 : 병렬처리, 컴퓨터 보안