19-1-R03

악성 행위 방지를 위한 프로그램 분석[1)]
(Program analyses for malicious behavior detection)

조 은 선
충북대학교 전기전자컴퓨터공학부 컴퓨터공학전공
eschough@cbnu.ac.kr

요 약

인터넷의 대중화와 컴퓨터 적용 도메인의 빠른 확장에 따라 침해사고의 피해가 커지고 이에 따라 컴퓨터의 보안은 매우 중요한 주제로 등장하였다. 최근 들어 여러 가지 공격 기법 중에도 피해 컴퓨터에서 명시적/암묵적으로 수행되는 프로그램의 악성 행위에 인한 것이 사고 원인의 많은 부분을 차지하고 있다.

본고에서는 프로그램의 대한 정적/동적 판별을 하여 악성 행위에 대해 대응하는 방법에 살펴본다. 특히 악성 프로그램 수행을 사전에 예방하도록 하는 정적 분석 기법을 중심으로 하여 관련 기술들을 소개한다.

1. 서 론

인터넷의 발달과 컴퓨터의 기능 확장은 컴퓨터가 보다 많은 부분에 적용되도록 하게 하여 인류 생활을 편리하게 해주고 있다. 그러나 컴퓨터 및 네트워크 침해사고, 개인 정보 침해 등 그 역기능의 폐해도 심각해졌다.

그 중에서도 피해 컴퓨터에서 명시적/암묵적으로 수행되는 웜/바이러스, 트로이목마와 같은 악성 행위를 수행하는 경우가 주요한 원인 중 하나이다. 한국정보보호진흥원의 2004년 통계에 의하면 구성설정 오류, E-mail 관련 행위, 취약성 정보 수집 등과 함께 악성 프로그램 수행이 해킹의 주요 공격 기법으로 이용 된다. 또한 올해 8월까지의 웜/바이러스로 인한 사고의 건수가 11,097건으로 작년도 총계를 이미 넘어서 있다.

이에 따라 컴퓨터를 침해 사고로부터 지키는 기술에 대한 연구도 많이 진행되어 오고 있다. 특히 악성 프로그램의 수행을 예방, 탐지하는 기술도 다양하게 등장하고 있다. 본고에서는 이러한 기술들을 프로그램에 대한 정적 분석을 중심으로 간단히 정리하여 소개하고자 한다. 특히, 정적 분석 자체에 대한 주제 보다는, 정적 분석이 실제 유용한 도구로 사용되기 위해 어떠한

1) 이 논문은 2004학년도 충북대학교 학술연구지원사업의 연구비지원에 의하여 연구되었음

부분들을 고려해야하는지에 촛점을 맞추어 관련 기술들에 대해 전개하였다.

다음절에서는 악성 코드와 그 대응 기법에 대한 전반적인 분류 방식을 소개한다. 3 ~ 5절에서는 분석 대상, 분석 파일의 종류, 분석의 실효성 등의 주제로 관련 기법들을 살펴본다. 정보 유출이나 취약점에 대한 정적 분석 기법들은 이미 많이 알려져 있으므로 상대적으로 적게 다루었다.

2. 악성 코드와 대응 기법

2.1. 악성 코드

악성 프로그램은 악성 행위를 하는 프로그램을 말하며, 악성 코드는 악성 행위를 하는 프로그램의 한 조각 (fragment)을 일컫는다[1]. 여기서 악성 행위란 기밀성, 시스템의 무결성/기능, 자료 및 제어 흐름 등에 사용자가 의도하지 않은 방향으로 영향을 주는 것을 말한다.

악성 프로그램은 사용자가 믿고 컴퓨터에 설치하였지만 악성 행위를 수행하거나 (트로이 목마), 사용자의 개입이 거의 없이 다른 방식으로 자동 설치된다 (바이러스나 웜). 이 때, 대상 프로그램이 존재하여 그것이 수행 될 때 악성 행위가 함께 수행되는 경우도 있고, 특정 일, 특정 시간, 특정 파일 접근 횟수 등에 따라 수행이 촉발 되는 경우도 있어 판별에 어려움이 있다.

또한 악성행위가 수행 전 실행 코드에 내재되는 경우도 있으나 수행중인 프로그램의 취약성을 틈타 악성 코드가 주입되는 경우도 있다.

악성 행위 판단 기술이란 주어진 실행 코드가 이러한 악성 코드의 성질을 가지는지 여부를 판별하는 기술을 말한다.

2.2. 대응 기법의 분류 기준

전통적으로 안티바이러스 소프트웨어에서 수행되는 대응 기법은 간단한 정적 분석의 일종이다. 악성 코드의 바이너리가 다른 정상 프로그램들의 바이너리와 달리 가지는 '진단 문자열'을 찾아내고, 이 진단 문자열의 포함 여부로 프로그램을 스캔하여 악성 행위 여부를 수행 전에 알아내는 것이다. 따라서 악성 행위의 수행이 원천적으로 차단되는 효과를 가진다.

이러한 방식의 문제는 진단 문자열을 찾아내는 것이 통상 매우 어렵다는 것과, 피해가 발생 한 후에야 진단 문자열을 찾기 시작할 수 있다는 점이다. 따라서 신종 악성 코드의 등장이 가속화 되고 있는 상황에 비추어 볼 때 이러한 대응 기법은 미래에는 적합하지 않을 수 있다.

이것을 극복 및 보완하기 위해 알려지지 않은 불특정 악성 코드에 대해 체계적인 분석을 통해 수행을 차단할 수 있는 다양한 대응 기법들이 존재하고 있다. 하나의 분석 기법이 모든 프로그램에 대해 모든 악성 행위를 검출해 내는 것은 어렵기 때문에, 각 연구 결과들은 독특한 주안점들을 가지고 있고 다음과 같이 분류가 가능하다.

분석 대상 : 어떤 행위를 악성 행위로 볼 것인지에 따라 대응 방법이 분류 될 수 있다. 즉, 시스템을 파괴 하거나, 시스템의 자료 유출 및 제어의 흐름을 바꾸는 일 등이 분석 대상이 될 수 있다.

그러나 이러한 행위들도 사용자가 의도한 경우는 악성 행위에서 배제시켜야 하기 때문에 기계적으로 판별하기에 매우 어려움이 있다. 또한 악성 행위에 대한 범주에 대해서도 프로그램의 응용 범위

마다 특성이 있으므로 도메인 지식이 필요한 경우도 있다. 상대적으로 분석 대상이 명확한, 취약성이나 정보 유출 등에 대한 정적 분석 기법들이 가장 활발히 연구되고 있는 것도 이 때문이다[2]. 본고에서는 취약성 및 정보 유출외의 경우의 악성 행위에 촛점을 맞추어 대응 기법을 소개한다.

분석 파일의 종류 : 일반적으로 상위 언어(high level language)를 가지고 기술된 프로그램들의 경우 취약점 분석이 수월하고, 의미 구조를 간직하고 있어 악성 행위 대응을 위한 정적 분석 기법을 고안하는 데에 상대적으로 용이하다. 그러나 대부분의 악성코드는 기계어로 된 바이너리 코드 형태로 배포되기 때문에 외부로부터 들어오는 악성 코드에 대한 판별에서 실효성을 가지려면 바이너리 코드에 대한 정적 분석이 필요하다. 최근들어 몇 가지 연구가 시도되고 있지만 기술적으로 해결되어야할 문제들이 많이 남아있다[3].

Java의 경우 중간 언어 프로그램인 바이트 코드 형태로 배포되므로 Java 기반의 스마트카드에 배포되는 악성 코드 정적 분석에 대한 관련 연구가 비교적 활발히 되어 있는 편이다[4,5,6].

분석 기법의 상세성과 실효성 : 속도와 실효성을 중시하여 정확한 프로그램 성질 분석보다는 몇 가지 가시적 특성만을 판별하게 되며 결과적으로 거짓 오류 신호(false alarm) 가 많이 나타난다. 따라서 분석의 상세성과 엄밀성을 보이기 위한 연구가 이론적인 기반에서 연구되어 오게 된다. 이러한 분석 기법들은 실효성과 연결되는 것이 수월하지 않다는 제한점이 있으나, 이를 확장시키거나 분야에 맞게 적응시키는 과정을 통하여 극복이 될 수가 있다.

이와 다소 상반되는 측면으로 분석 자체의 엄밀함 보다는 운용의 아이디어로 실효성을 꾀하여 속도 및 탐지율 등의 효과를 얻는 분석 기법들도 존재한다. 주로 분석 기법의 적용 구조에 대한 연구로서 동적인 검사와 어떤 방식으로 결합하는지 위주가 되며, 대상 도메인의 성격에 영향을 받을 수도 있다.

여기서는 위와 같은 관점을 중심으로 기존의 대응 기법들을 분류하여 설명한다. 먼저 3장에서는 분석 대상을 정의 하는 방법 및 이러한 분석 대상에 따른 분석 기법을 간단히 소개한다. 4장에서는 분석 파일이 소스 코드인지 실행 코드인지에 따른 분석 기법을 간략히 소개하며 특히 실행 코드인 경우를 중점적으로 소개한다. 5장에서는 순수한 정적 분석 기법 연구 외에 여러 분석 기법들에 대한 효과적인 운용을 통한 실효성 있는 판별 기법들에 대해 역시 간단히 소개 한다.

3. 분석 대상

3.1. 악성 행위 수집 분류

앞서 언급했던 바와 같이 '악성 행위'를 정의하는 것은 까다로운 문제이다. 이를 위해 가장 보편적으로 사용되는 방법은 알려진 악성 코드를 수집한 후 분류함으로써 악성행위를 규명하는 것이다.

현재까지는 분류된 코드에서 공통 검색 스트링을 추출에 의존하고 있으나, 코드 카테고리가 가지는 의미 구조를 주어진 코드

와 비교하는 것도 가능하다. 특히, 안티바이러스 업계에서는 바이러스 및 웜에 대해 분류하고 동종의 새로운 형태의 바이러스에 대해 효율적으로 탐지할 수 있도록 명명하는 것에 촛점을 두고 있다. 일본의 정보처리진흥사업협회(IPA)[6]는 기존의 보안 도구에서는 검출할 수 없는 다양한 악성 코드를 수집하여 39개의 카테고리와 명칭, 위험도, 작성연월일, 작성자, 개요 등으로 분류하게 된다.

인공지능 기법을 써서 분류 및 분석을 자동화 한 경우도 있다. IBM ANN(Artificial Neural Network)에서는 신경망을 통해 실행 코드들을 악성 또는 정상 실행 코드로 자동으로 분류하는 작업을 수행하였다[7]. 또한 미 Columbia 대의 MEF (Malicious Email Filter) 그룹에서는 2001년경에 데이터 마이닝 기법을 실시하여 행위 패턴에 대한 통계 자료를 도출하는 연구를 하였다. 결과적으로 이메일 첨부 실행 파일에 적용하였으며, 특정 함수의 호출 등 약간의 코드의 의미 구조를 반영하여 악성 여부를 판단하는 데에 도움을 주게 된다[8].

3.2. 악성 행위 판별 기준 모델링

캐나다 MaliCOTS[1] 팀은 악성 행위의 기본 성질을 접근 제어 위반, 정보 유출가능성, 가용성 제한으로 보고 있다. 이 세 가지 성질은 그 정의가 명확하게 되어 있고 코드로 옮겨 규명하기도 직관적이기 때문에 분석 대상으로 비교적 많이 연구되어 왔다. 일례로 Java의 동적 접근 제어를 정적 분석으로 수행하려는 연구들이 있으며[9], 연속적인 행위를 인지하여 접근 제어를 시도하는 정적 분석 기법이 제안되기도 하였다[10]. 버퍼 오버플로우 취약성과 정보 흐름 및 정보 유출 등에 관한 연구 역시 전통적으로 많이 되어 있다.

그러나 이 외의 성질의 경우 모델링이나 판별 기준이 어렵고 판별도 애매하기 때문에 상대적으로 많지 않은 연구가 수행되고 있다.

미국 CERTCC 에서는 생존성 보장 연구의 일환으로 실제 사용되는 소프트웨어의 행위를 추상적으로 뽑아내는 도구인 FX (Function eXtractor) 를 개발하여 기존의 악성행위를 정의하는데에 도움을 주게 된다[11].

순차 행위 모델링을 사용하여 악성 행위를 묘사하기도 한다. 호스트 보호 정책을 표현하기 위해 먼저 사용되었던 방법으로 특정 단위 행위들이 특정 순서로 배열되어 나타나는 경우 악성 코드가 수행된 것으로 판별하는 방식이다. FSA(finite state automata), PDA(push down automata)를 주로 사용하여 시간에 따른 프로그램 성질 묘사가 가능하나, 단위 조건으로 표현하는 것에 비해 검증 시간이 길어진다는 단점도 가진다. 극복하기 위한 연구로 뉴욕주립대 (SUNY)에서는 FSA 기반으로 명세 모델과 매치 모델을 별도로 정의한 후 이를 전환시켜 효과적인 표현 기법과 빠른 매치를 동시에 제공하고 있다[12].

MIT에서는 시간 논리식을 근간으로 하는 TTL 이라는 언어를 사용하여 악성 행위를 판별하는 예를 2001년에 발표한 바가 있다[13]. 이로써 각 이벤트의 지속 시간을 모델에서 고려를 하게 되는 장점이 있으나 판별의 정확성이나 속도 부분에는 해결해야할 문제들이 남아있다.

4. 분석 파일의 종류

4.1. 소스 코드의 정적 분석

소스 코드에 대한 정적 분석 기법들은 버퍼 오버플로우 등의 취약점 분석, 정보 유출 분석 분야에서 활발하게 이루어져왔다. 기존의 자료들이 풍부하므로 여기서는 생략한다.

기타 자원의 소모에 대한 예측하는 분석의 경우 서비스 거부 공격에 대응할 수 있으며, 소프트웨어 공학의 정형 기법 분야에서 연구되는 설계 명세에 따라 수행하는지 검증하는 기술을 사용자의 의도와 다른 수행을 검출하는 데에 응용할 수 있다. 후자는 설계 단계에서 프로그램의 명세를 정형화 시킨 후, 완성된 프로그램이 이 명세에 충실한 지를 모델 체킹 기법이나 일반적인 프로그램 분석 기법을 적용하여 검증하게 된다.

4.2 실행 코드 분석

4.2.1. 실행 코드 분석을 위한 선행 기술들

COTS 제품 및 이진 코드 형태로 다운로드 되는 실행 코드 모듈이 증가하면서 이들의 신뢰성에 대한 검증의 필요성이 증가하고 있다. 그러나 실행코드 분석을 위해서는 소스 코드에 비해 상대적으로 많은 기술들이 선행 연구되어야 하므로 실제적인 연구는 아직 많이 진전되어 있지 않은 편이다.

현재 실행 코드에 대한 분석을 시도해 볼 수 있는 배경에는 다음과 같은 기술들의 발달이 있었다.

역 어셈블 기법 : 소스코드에서 적용되어 오던 기술을 이진 코드에 적용하기위해 이진 코드에서 어셈블리어 코드로 변환하는 역 어셈블 기능은 필수적이다. 주소값을 직접 다루는 부분들로 인한 제어 흐름의 모호성으로, 기술적으로 간단한 사항은 아니나, 현재 많은 연구가 높은 정확도를 보여주는 실험결과를 발표. 기타 각종 상용 제품들도 출시되어 있는 상태이다.

역 어셈블은 코드의 순서대로 번역하는 Linear Sweep 알고리즘과 프로그램 수행 순서를 따라가며 번역하는 recursive traversal 알고리즘이 존재한다[14]. 컴퓨터가 추후 64비트 체제로 전환되면 새로운 역 어셈블 문제가 도입될 가능성이 있다.

특정 하드웨어에 적용되는 역 어셈블 기법은 정확도가 높은 편이다. 미국 워싱턴 대학의 Etch에 이르러 Win32/IntelPE 코드 역 어셈블이 가능해지고[15], EEL을 확장한 싱가포르 대학의 LEEL[16]은 Linux/x86 코드를 역 어셈블하고 있다. 또한 미국 SUNY 대학의 RAD[17]에서 버퍼 오버플로우 취약성을 방지하기 위한 instrumentation에 사용된 역 어셈블러도 높은 정확도를 보이고 있다.

다양한 하드웨어 플랫폼에 적용되는 연구도 존재하는데, 호주 Queensland 대학의 UQBT(A Relocatable Retargetable Binary Translator)는 플랫폼 독립적인 relocatable binary translation을 지원하고 있다. 제어를 따라가며 역 어셈블 하는 방법과 동적인 수행에 의해 도움을 받는 방법을 병행하여 역 어셈블한다[18][19].

상용제품인 Fenris[20]와 IDA Pro[21]는 각각 동적 또는 정적 분석 방식으로 역 어셈블 하는 것으로, 특히 IDA Pro는

Wisconsin 대학의 WISA나 Cigital사의 악성 행위 및 취약성 분석 연구용으로 활용된다.

실행 코드 흐름 분석 기법 : 흐름 분석은 코드의 의미를 파악할 수 있는 가장 기본적인 요소의 하나로서 이진 코드에 대한 악성 행위 정적/동적 분석에 중요한 기초가 된다. MS의 PPRC(Programmer Productivity Research Center)의 Athena [22]는 기계 코드 단계에서 자료 흐름 분석(data flow analysis), 제어 흐름 분석(control flow analysis), 호출 관계 분석(call-graph analysis) 등을 통하여 프로그램 분석과 테스트 데이터를 모으는 작업을 함하며, 서브시스템의 분석을 확장하는 연구도 병행하고 있다.

상용 제품인 Fenris는 테스트 기반의 동적 분석을 이용하여 역 어셈블러와 간단한 흐름 분석 결과를 만들어낸다[20]. 타 UQBT[19] 및 대부분의 역 어셈블러들도 흐름 분석 결과를 시각적으로 보여주는데, 이는 역 어셈블을 위해 recursive traversal 알고리즘 등에서 기본적인 흐름 분석이 생성되기 때문이다. 또한 덴마크 스페인 등지의 유럽에서는 alias 분석 및 data dependency 분석 등 기타 이진 코드 분석이 단편적으로 이루어지고 있다[23,24].

4.2.2. 실행 코드 분석 시도

이러한 기술 들을 바탕으로 실행 코드를 분석하려는 시도가 몇몇 연구자들에 의해 있었다.

캐나다 Laval 대학의 MaliCOTS는 이진 코드를 추상화 시키고, 의심스러운 부분을 프로그램 슬라이싱(slicing)으로 추려낸 후, 일부에 대해 알려진 악성 행위 명세를 적용하여 정적으로 악성 여부를 분석 한다[25].

미국 Wisconsin 대학에서는 바이러스가 안티바이러스를 우회하기 위해, 암호화하거나 무의미한 연산자를 사용하여 모호하게 하는 부분을 특질로 삼아 기계 코드를 분석하고 있다. 직접 행위를 분석해 내지 않고, 현존하는 악성 코드의 특질을 기준으로 코드의 일반 패턴을 분석한다[26].

1990년대 후반 카네기멜런 대학을 중심으로 프로토콜 안전성이나 네트워크의 생존성(survivability) 등을 검증하는 데에 model checking을 주로 사용하였고, 여기에 병렬성, 실시간성, temporal logic등을 기반으로 확장된 것들이 제안된 바가 있다.

이탈리아의 Padova 대학에서는 model checking과 정적 분석과의 접목 및 트로이 목마 등에 대비한 프로그램의 성질을 추출하는 것에 대한 연구를 수행하였다[27].

실행 코드 형태로 되어있는 COTS 제품들을 검증하기 위해 변수 복원 기법 등 원천적으로 상위 언어 프로그램이 의도한 바를 복원하는 것이 가장 효율적인 의미 파악 방법일 것이다. 그러나 아직 많은 부분들이 기술적으로 해결되지 못하고 있다[28].

5. 분석 기법의 실효성

정적 분석과 여러 가지 다른 기법을 혼합 적용함으로써 오류율을 줄이거나 속도를 향상시키는 연구가 비교적 활발히 진행되고 있다.

5.1 정상 행위에 대한 검증

정상 행위에 대한 속성 (property) 을 기술하고 이로부터 벗어나는 지를 판단하면 알려지지 않은 악성 행위에 대해서도 판단을 할 수 있다는 장점이 있다. 행위가 잘 알려져 있는 이진 코드의 경우에는 해당 행위를 그대로 property로 정의할 수 있으나 대부분 그렇지 못하다. 그래도 악성 행위를 규명하는 것 보다는 어렵지 않은 일로 여겨지고 있다.

속성 추출 : 미 Santa Babara UC 대학의 MAPbox[29]와 같은 미리 분류된 코드의 종류에 따른 행위 모델을 속성으로 하여 분석하였다. Cigital 사에서 개발한 PEAT는 악성 행위 자체가 아니라 프로그램의 프로파일을 통해 정상 행위를 정의하고, 이 정상 행위에서 벗어나는 지를 검사 한다[30]. 이를 위해 여러 코드를 각기 영역 (region) 별로 나누어 명령어 수행 빈도 및 컴파일러에 따른 명령 패턴 등의 통계자료 도출하고 있다. 뉴욕주립대의 MCC[12]는 데이터 마이닝을 이용한 분석을 통하여 효율적인 접근 제어 오토마타를 추출해내고 있다.

또한 소스 코드로 부터 속성을 추출하는 방식이 있는데, 실행 코드를 검증하는 경우에는 개발자 측으로부터 소스 코드에 대한 분석 결과를 가져오게 되며 가장 많은 의미 정보를 포함한다고 할 수 있다. 다양한 certifying 컴파일러를 사용하여 개발자가 의도하고 있는 속성을 추출하여 배포 시 첨부하는 연구가 비교적 활발하다. 이때의 속성은 자원 사용 방식이나 접근 제어 사항을 나타내는 오토마타 형태가 일반적이며 정적 분석으로 추출하는 경우가 있으며 많은 경우 타입으로 나타내어 배포와 검증에서 추가 부담을 덜고자 하고 있다.

메모리 자원 사용을 위한 Cornell 대학의 TAL(Typed Assembly Language)[31] 이나, Princeton 대학의 D. Walker[10] 와 같이 종속 (dependent) 타입으로 자원 접근 오토마타 형식을 표현하는 방법이 등장하게 되었다. 프랑스 연구소 INRIA도 자바 보안 모듈에서 정의되고 있는 스택 조사 (stack-inspection) 모니터링 기법을 타입으로 나타내어 첨부함으로써 수행 중 모니터링 오버헤드를 줄이는 방안을 개발하였다[32]. 단, 자바 보안 모델의 한계에 의해 단편적인 접근 제어를 다루고 있고, 실행 오토마타를 다루지는 않고 있다. 비슷한 예로 이동 에이전트에서 필요한 정보보호 기술 중의 한 부분으로 신뢰할 수 없는 코드의 분석이 포함되며, 이를 위해 자원 접근 소모에 대한 리스트를 각 에이전트가 가지고 다니는 예가 다수 있다. 이탈리아 KLAIM의 경우 에이전트들이 capability based type[33]이라는 첨부자료를 가지고 각 호스트들을 방문하게 된다.

실행 오토마타 자체를 첨부한 예는 U.C Berkeley 의 D. Wagner 등의 연구 결과가 있다. 여기서는 잘 알려진 서버 프로그램에 대해 정적분석을 하여 오토마타를 PDA (Push Down Automata) 형태로 추출하여, 실제 수행될 때 이를 벗어나면 취약성을 통한 공격 코드가 수행되는 것으로 판별하는 방식이다[34,35].

속성 검증 : 타입 형태로 존재하는 속성은 타입의 올바름을 검사함으로써 검증 될 수 있다. 반면 CMU의 Fox project 의 결과물 중 하나인 PCC (Proof Carrying

Code) 처럼, 타입 자체가 속성을 나타내지는 않으나 타입의 올바름을 검사함으로써 속성을 검증하는 기법도 가능 하다[36]. 이 경우는 타입 자체를 검증하는 것이 목표라기보다 타입 검증이 수학적으로 코드에 대한 특정 성질의 증명 과정과 동일하다는 특수한 정리로 부터 기인한다. 이로써 타입 이외의 일반적인 성질을 코드에 첨부 가능하게 되었다. Fox project 후속의 ConCert project[37]가 진행 중이며 PCC를 적용하여 나타낼 수 있는 첨부 자료의 범위를 확장하는 연구를 진행 중에 있다. 그러나 현재까지는 정적 분석 과정을 통한 검증 보다는 미국의 MCC[12]나 일본의 SoftwarePot[38]과 같이 테스트나 수행 중 동적 모니터링에 의해 첨부된 속성을 검증하는 것이 더 일반적이다.

5.2 테스트과의 연동

Testing과 emulation 등의 동적인 분석 기술은 정적으로 코드로부터 알아낼 수 없는 사용자 입력에 기반한 계산이나 지연 바인딩 등 동적인 자료들에 대해 알 수 있는 장점이 있으므로 정적 분석과 보완 관계를 가지게 된다. 따라서 정적 분석과 동적 분석 기술을 결합하는 방식은 학계나 업계 모두에서 관심을 가지고 시도하였다.

일차적인 접근 방법은 정적 분석을 거친 후 그 결과를 토대로 동적 분석을 함으로써 동적 분석의 수행 속도를 높이는 방안들이 이용되고 있다.

이것은 COTS 제품의 Black Box Test를 위해 주로 사용되는 방법인데, 미 국방성의 지원을 받는 Cigital 사의 PEAT[30]은 code segment에서 시작하는 경우와 바로 다음 명령으로 호출되어 넘어가는 경우(바이러스에서 많이 이용)들을 정적 분석으로 판별하고, 사용될 DLL을 파악하여 악성 코드로 확인된 경우를 걸러낸 후 동적 분석을 꾀한다[39]. 비슷한 예로 캐나다 Laval 대학의 MaliCOTS에서는 정적 분석으로 코드를 slicing 한 후 의심스러운 부분에 대해 동적 분석을 수행함으로써 동적 분석의 부하를 줄이고 있다[1].

또한 프로그램의 제어 흐름을 파악한 후 각 path에 대해 White Box 테스트를 수행할 수 있는데, 소스 코드와 마찬가지로 바이너리 형태의 코드도 역 컴파일 된 상태에서 이를 행할 수 있다. 이 때 미리 정확한 테스트 케이스를 만들어내는 과정에 정적 분석을 사용하게 된다. UC Davis의 TASPEC[40]은 Property기반의 테스트 도구로서 테스트 케이스를 포함하여 사용자의 테스트를 돕는 각종 자료를 생성하는데 이때 정적 분석 결과가 포함되게 된다.

미 Symantec 사에서는 다형성 바이러스 등에 대한 동적인 테스트의 부하를 줄이기 위해 정적 분석의 최적화와 기법을 도입하는 것을 시도하고 있다.

기타 에뮬레이션을 활용한 테스트 기법[41] 에서는 정적 분석 기법의 도움이 유효할 것으로 여겨지나, 연구가 있었는지 아직 알려진 바가 없다.

5.3 코드 변환 기술

코드의 수행 과정을 모니터링 하여 악성 행위를 시도하는지를 판단하는 기법은 많이 시동되어 왔다[42][43]. Java와 같이 자원 접근 메소드 호출시 마다 접근 권한을 검사하기도 하며[44], Solaris, Linux, MS 등의 운영체제는 각각 시스템 호출과 수행 코드간의 인터페이스 부분을 가로채어(hooking) 검사 코드를 수행하는 경우가

일반적이다[45]. 그러나 이러한 기법들은 시스템 (커널이나 VM) 상의 부하가 걸리게 되므로 성능을 위해서는 바람직하지 못하다는 지적이 있다.

이를 극복하기 위해 정적 분석을 하여 적절한 위치를 찾아 검사 코드로 둘러싸기(wrapping)를 하거나 코드 주입(instrumentation)을 통하여 검증 코드를 프로그램 안에 삽입시키는 방식이 등장하였다. 이것은 정적분석을 통하여 코드를 변경시킴으로써 코드 자체에 악성 행위에 대한 방지 장치를 하는 기법이 중심이 된다. 특히 컴파일 단계에서 판별되지 않는 사항에 대해 수행 중에 검사할 수 있고, 수행 중 모니터링을 하는 것에 비해 커널모드와 사용자모드 교차의 부하가 현저히 낮다는 장점이 있다.

MIT의 Naccio[46]에서는 다단계로 코딩된 정책을 적용한 코드 주입 기법을 Win32 이진 프로그램과 Java 프로그램에 대해 수행하고 있다. 미 Wisconsin 대학의 WISA [28]도 안전한 코드로의 이진 코드 변환기 Brew를 연구 중에 있다. 또한 버퍼 오버플로우 취약성을 방지하기 위해 RAD와 같이 미리 코드에 버퍼 넘침을 점검하는 코드를 삽입하기도 한다[14]. 난이도 관계로 다소 제한적이긴 하나, 순차 행위를 모델링하여 이를 점검하는 코드를 삽입하는 기법도 연구중에 있다[47].

6. 정리

정적 분석은 다른 분야 뿐 아니라 컴퓨터 보안을 위해서도 다양하게 사용되고 있다. 특히 악성 행위를 수행 전에 미리 판별하기 위해 다양한 대상을 위해 형태로 활용되고 있다.

본고에서는 악성 행위를 판별하기 위한 정적 분석 연구들에 대해 개괄적으로 소개하였다. 정적 분석 자체에 대한 부분 보다는 실제 유용한 도구로 사용되기 위해 어떠한 부분들을 고려해야하는지에 초점을 맞추어 전개하였다. 정보 유출이나 취약점에 대해서는 이미 많이 알려져 있다고 가정하고 상대적으로 거의 다루지 않았다.

악성 행위는 그 정의부터가 모호하여 이것을 규명하는 연구가 먼저 깊이 이루어져야하는데 이때도 기존의 악성코드를 분석하는 과정이 필요하다. 악성 코드에 대한 방어로는 실행 파일을 정적 분석하는 것이 가장 실용적이나 기술적으로 한계가 많으므로 여러 다른 기법들과 효과적으로 연계하여 우수한 결과를 낼 수 있을 것으로 본다.

참 고 문 헌

[1] The MaliCOTS Project, Defence Research and Development Canada, http://www.drdc-rddc.dnd.ca/researchtech/malicots/tech_e.asp

[2] A. Sabelfeld, A. Myers, Language-Based Information-Flow Security

[3] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. In 12th USENIX Security Symposium, Washington, DC, August 2003

[4] Java bytecode verification: algorithms and formalizations. Journal of Automated Reasoning 30(3-4):235-269, 2003.

[5] M.Bartoletti, Pierpaolo, Degano, Gian Luigi Ferrari, Static analysis for eager stack inspection,

[6] 일본 정보처리진흥사업협회 (IPA),

http://www.ipa.go.jp/SPC/report/02fy-pro/html/security

[7] IBM Patent covers artificial intelligence virus effort, Computer World, http://www.computerworld.com/news/1997/story/0,11280,20715,00.html,

[8] The Malicious Email Filter (MEF) Group, Columbia University, http://www1.cs.columbia.edu/ids/mef/

[9] L. Koved, M. Pistoia, A. Kershenbaum, "Access Rights Analysis for Java", OOPSLA, 2002.

[10] David Walker, "A type system for expressive security policies', Proc. of 27th symposium Principles of Programming Languages. pp. 254--267, 2000.

[11] Pleszkoch, M. & Linger, R. "Improving Network System Security with Function Extraction Technology for Automated Calculation of Program Behavior." Proceedings of the 37th Hawaii International Conference on System Sciences (HICSS-37). Hawaii, January 5-8, 2004. Los Alamitos, CA: IEEE Computer Society Press, 2004.

[12] R. Sekar, V.N. Venkatakrishnan, Samik Basu, Sandeep Bhatkar and Dan DuVarney, "Model-Carrying Code: A Practical Approach for Safe Execution of Untrusted Applications", Proc. of ACM Symposium on Operating Systems Principles, October 2003

[13] Jon Doyle, Isaac Kohane, William Long, Howard Shrobe, and Peter Szolovits, "Event Recognition Beyond Signature and Anomaly," Proc. of 2001 IEEE Workshop on Information Assurance and Security, June 2001

[14] Manish Prasad and Tzi-cker Chiueh, "A Binary Rewriting Defense against Stack based Buffer Overflow Attacks", Proc. of Usenix Annual Technical Conference, June 2003.

[15] T. Romer et. al , Embra: Fast and Flexible Machine Simulation. The proceedings of ACM SIGMETRICS '96: Conference on Measurement and Modeling of Computer Systems, Philadelphia, 1996.

[16] Lu Xun. A linux executable editing library. Masters Thesis, 1999.

[17] Manish Prasad and Tzi-cker Chiueh, "A Binary Rewriting Defense against Stack based Buffer Overflow Attacks", Proc. of Usenix Annual Technical Conference, June 2003.

[18] C Cifuentes, M Van Emmerik, N Ramsey and B Lewis, Experience in the Design, Implementation and Use of a Retargetable Static Binary Translation Framework, Sun Microsystems Laboratories, Technical Report TR-2002-105, January 2002.

[19] C Cifuentes and M Van Emmerik, "UQBT: Adaptable Binary Translation at Low Cost", Computer, Vol 33, No 3, March 2000, IEEE Computer Society Press, pp 60-66

[20] M. Zalewski, Fenris, http://razor.bindview.com/tools/fenris/, BindView Co.

[21] The IDAPro Disassembler and Debugger, http://www.datarescue.com/idabase/

[22] Microsoft Binary Technologies Group, http://research.microsoft.com/bit/#Nirvana, Microsoft Research

[23] Samya Debray, Robert Muth, and Matthew Weippert. "Alias analysis of executable code", Proceedings of the 25th ACM

Symposium on Principles of Programming Languages, January 1998.

[24] M. Fernandez and R. Espasa. Speculative alias analysis for executable code. Technical Report UPC-DAC-2002-27, Computer Architecture Department, Universitat Politecnica de Catalunya, Barcelona, 2002

[25] F. Besson, T. Jensen, D. Le Métayer, T. Thorn: Model-checking security properties of control-flow graphs, Journal of Computer Security, 9:217-250, 2001.

[26] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. In 12th USENIX Security Symposium, Washington, DC, August 2003.

[27] Lenore D. Zuck, Paul C. Attie, Agostino Cortesi, Supratik Mukhopadhyay (Eds.): Verification, Model Checking, and Abstract Interpretation, 4th International Conference, VMCAI 2003, New York, NY, USA, January 9-11, 2002, Proceedings. Lecture Notes in Computer Science 2575 Springer 2003

[28] WIsconsin Safety Analyzer, http://www.cs.wisc. edu/wisa

[29] A.Acharya and M. Raje, "Mapbox: Using parameterized behavior classes to confine applications", Proceedings of the USENIX Security Symposium, pages 1-17, August 2000.

[30] M. Weber et. al, "A Toolkit for Detecting and Analyzing Malicious Software", Proc. of 18th annual computer security application conference, 2002.

[31] Typed Assembly Language, Cornell University, http://www.cs.cornell.edu/talc/

[32] F. Pottier, C. Skalka, S. Smith, A systematic approach to static access control, ACM Transactions on Programming Languages and Systems (TOPLAS), 27(2),2005

[33] Rocco De Nicola, GianLuigi Ferrari, and Rosario Pugliese, "Programming access control: The KLAIM experience", Proc. of Conference on Concurrency Theory, LNCS 1877. Springer Verlag, 2000.

[34] D. Wagner and D. Dean, Intrusion Detection via Static Analysis, in proc of 2001 IEEE Symposium on Security and Privacy, 2001

[35] Lap Chung Lam and Tzi-cker Chiueh, Automatic Extraction of Accurate Application-Specific Sandboxing Policy, 7th International Symposium on Recent Advances in Intrusion Detection (RAID), Sophia Antipolis, French Riviera, France, September 15-17, 2004

[36] The Fox Project, Proof Carrying Code, http://www-2.cs.cmu.edu/~fox/pcc.html, CMU

[37] The ConCert Project Certified Code for Grid Computing, http://www-2.cs.cmu.edu/~concert/, CMU

[38] SoftwarePot, http://www.osss.is.tsukuba.ac.jp/pot/

[39] J. Voas, A Defensive Approach to Certifying COTS software, rst corp. Technical report: RSTR-002-97-002.01, 1997

[40] G. Fink et.al, An Interface Language Between Specifications and Testing, Technical Report CSE-95-15, University of California, Davis, 1995

[41] IBM Antivirus Research, http://www.research. ibm.com/ antivirus/SciPapers.htm

[42] Calvin Ko, Manfred Ruschitzka, and Karl

Levitt, "Execution monitoring of security-critical programs in distributed systems: a specification based approach", Porc. of IEEE Symposium on Security and Privacy May 1997.

[43] Tancmothy Fraser, Lee Badger, and Mark Feldman. Hardening COTS software with generic software wrappers. In IEEE Symposium on Security and Privacy, 1999.

[44] A. Gordon and C. Fournet. Stack inspection: Theory and variant. In Proceedings of POPL '01

[45] I. Goldberg, D. Wagner, R. Thomas, and Eric Brewer, A Secure Environment for Untrusted helper Applications (Confining the Wily Hacker), Proceedings of the Sixth USENIX UNIX Security Symposium, San Jose, Califorina, July 1996

[46] MIT Naccio Project, http://naccio.lcs.mit.edu/

[47] Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: Enforcement mechanisms for run-time security policies. International Journal of Information Security, 2003

조 은 선

1987년~1991년 서울대학교 계산통계학과 (학사)
1991년~1993년 서울대학교 계산통계학과 전산과학전공 (석사)
1993년~1998년 서울대학교 계산통계학과 전산과학전공 (박사)
1999년~2000년 한국과학기술원 연구원
2000년~2002년 아주대학교 조교수대우
2002년~현재 충북대학교 전기전자컴퓨터전공 조교수
관심분야 : 컴퓨터 보안, 프로그램 분석