악성코드 사전 탐지기술

연재순서

1. 행위기반 봇넷 탐지기술
2. 알려지지 않은 스파이웨어 탐지기술
3. VoIP공격 탐지기술
4. 알려지지 않은 신종 인터넷 웜 탐지기술
5. **악성코드 사전 탐지기술**
6. IP 스푸핑 탐지기술

악성코드가 실행되기 이전에 탐지하고 정확도를 높이는 방법으로 실행 파일을 바이너리 코드 레벨에서 분석하여 악성코드를 탐지하는 정적분석(static analysis) 기법은 하나의 대안으로 이용될 수 있다. 이번 호에서는 이 정적분석 기법을 활용한 코드 그래프(Code Graph)를 소개하고 이를 이용한 악성코드 사전 탐지 기술에 대해 알아본다.

글 · 정규창 고려대 컴퓨터보안연구실 연구원, 이희조 고려대 교수

일반 컴퓨터 사용자가 어떤 응용 프로그램을 처음 사용하고자 할 때, 그 실행 결과를 프로그램을 직접 실행시켜 보기 전까지는 예측하기 어렵다. 컴퓨터 바이러스나 인터넷 웜과 같은 악성코드는 일단 한번 실행이 되면 사용자의 컴퓨터를 감염시켜 시스템에 치명적인 악영향을 끼친다. 사용자가 어떤 프로그램을 악성코드가 포함이 되어있는 상태로 실행 시키게 된다면 그 시스템은 사용자도 모르는 사이에 감염되어 심각한 문제를 일으키게 된다.

1985년 컴퓨터 바이러스가 처음 나타난 이래 악성코드의 종류와 수는 기하급수적으로 증가하고 있다. 또한 컴퓨터와 인터넷 기술의 급격한 발달과 더불어 악성코드의 기능들이 점점 고도화 되고 지능화 되어가고 있다. 플로피 디스켓을 주된 전파 매체로 하여 장난기 섞인 메시지 출력과 파일 조작 등이 고작이었던 초기의 바이러스와 달리 요즘의 악성코드들은 인터넷이라는 빠르고 강력한 전송 매체를 이용하여 전파되며 중요한 사용자 정보를 몰래 빼가거나 시스템을 파괴시키며 또 다른 인터넷 공격에 이용하기도 한다.

이러한 악성코드들을 탐지하고 제어하기 위한 다양하고 많은 연구들이 진행되고 있다. 전통적인 방법이며 현재에도 가장 많이 사용되고 있는 악성코드 탐지 방법은 시그너처를 이용한 패턴 매칭 기법이다. 이 기법은 악성코드 각각의 코드 정보를 시그너처화 하여 데이터베이스에 모두 저장한 다음 이 정보들을 하나씩 비교해가며 악성코드를 탐지하는 방법이다.

이 기법은 정확도는 높지만 나날이 늘어가는 악성코드들을 데이터베이스로 유지하기 힘들고 패턴 매칭에 걸리는 시간 또한 길어지기 때문에 새로운 탐지 방법이 요구되어져 왔다. 악성코드의 행위에 기반 한 탐지 방법은 그 대안으로 주목을 받고 있다. 악성코드가 시스템에서 동작할 때 발생하는 특징적인 행위를 보고 탐지하는 이 기법은 데이터베이스 유지의 부담을 줄이고 알려지지 않은 악성코드 탐지에는 유용하지만 일단 악성코드가 실행이 되어야 하므로 시스템이 감염되는 위험에 항상 노출되어 있고 시그너쳐를 이용한 방법보다 정확도가 떨어진다는 문제점이 있다.

악성코드가 실행되기 이전에 탐지하고 정확도를 높이는 방법으로 실행 파일을 바이너리 코드 레벨에서 분석하여 악성코드를 탐지하는 정적분석(static analysis) 기법은 하나의 대안으로 이용 될 수 있다.

프로그램의 바이너리 명령어 풀어보기

악성코드는 그 목적이 시스템에서 악성행위를 하는 것에 있기 때문에 스스로를 은폐시켜 발견되기 어렵게 하고 발견 되더라도 자신의 코드가 분석되는 것을 어렵게 한다. 여기서 악성코드가 자신을 숨기고 코드를 복잡하게 하는 기술에 대해서 살펴보자.

실행 파일은 프로그램이 어떤 방법과 절차로 실행된다는 정보를 담고 있는 것으로 건물의 설계도면이나 생명체의 DNA와 같은 것이다. 이 파일에는 프로그램 실행을 위한 모든 정보들이 포함되어 있으므로 상황에 따라서는 특별히 보호해야 할 필요가 있다. 은밀하게 사용되고 있는 정품을 크랙한 버전의 프로그램이나 각종 프로그램 크랙 툴들은 대부분 이 정보에 조작을 가하여 만들어진다.

코드 혼란 기법(code obfuscation)은 실행 코드를 보호하는 기법으로 암호화나 압축 등을 이용해 실행 코드를 복잡화 하는 것이다. 코드 혼란 기법에는 코드 재배치(code reordering), 정크 명령 삽입(junk insertion), 실행압축(packing) 3개의 주요한 방법이 있다. 코드 재배치는 코드의 실행 순서를 유지 시킨 상태에서 점프 명령어들을 추가해 문법적인 순서를 변경시켜 복잡화 하는 방법이다.

정크 명령어 삽입은 프로그램 행위에 영향을 주지 않는 한에서 불규칙적으로 아무런 의미가 없는 더미 명령어들을 삽입하는 방법이다. 실행 압축은 실행 코드의 순서를 변경하여 암호화나 압축된 형태로 실행 코드를 변환하는 것이다. Zip과 같은 알고리즘으로 파일을 압축 하는 것과 다른 점이 있다면 암호화나 압축된 형태를 원래의 형태로 되돌리는 루틴(실행압축해제, Unpacking)을 실행 코드 자체에 담고 있다는 점이다.

이 실행압축해제 루틴을 이용해 실행 파일이 실행 될 때 스스로 원래의 형태로 되돌려 실행 할 수 있는 것이다. 자신의 코드가 분석 되는 것을 어렵게 하기 위해 대부분의 악성코드들은 코드 혼란 기법을 사용하고 있다. 특히 실행압축 기법을 활용해서 실행 파일을 보호하고 있기 때문에 악성코드 분석을 위해서는 반드시 실행압축해제가 필요하다. 널리 사용되는 실행압축해제 기법에는 디버거를 이용해서 직접 실행압축해제 하는 방법, 공개된 실행압축해제 툴들을 이용해서 하는 방법, 가상머신에서 악성코드를 실행시켜 원본 파일을 얻는 방법 등이 있다. 코드 그래프에서는 공개된 실행압축해제 툴을 이용해서 몇몇의 악성코드 원본을 얻어서 실험에 사용했고 다른 방법들로 확장해 사용이 가능하다.

의미 있는 명령어 추리기

코드 그래프는 실행 파일에서 특징을 잡아내 악성코드를 탐지한다. 앞서 설명 한 것처럼 실행 파일에는 프로그램의 실행에 관한 모든 정보가 포함되어 있기 때문에 이 정보를 분석하면 프로그램의 특징들을 알아 낼 수 있다.

하지만 실행 코드 내에 분석해야할 명령어들의 종류와 개수가 너무 많기 때문에 명령어 전체를 분석하기에는 연산량이 많고 분석 시간도 오래 걸린다. 코드 그래프에서는 프로그램의 특징 추출에 꼭 필요한 4개의 그룹에 해당하는 명령어들만을 분석함으로써 전체를 분석하는 것보다 훨씬 효율적이고 빠른 속도로 바이너리를 분석한다. 그림 1은 분석에 필요한 4개의 그룹에 해당하는 명령어들을 보여주고 있다.

GROUP INSTRUCTION		그룹 명령어
SCALL	시스템 콜 호출	CALL(system-call)
PCALL	내부 함수 호출	CALL(procedure)
JMP	점프	JMP
CJMP	분기문	JA, JAE, JB, JBE, JC, JCXZ, JECXZ, JRCXZ, JE, JG, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC,JNE, JNG, JNGE, JNL, JNLE, JNO, JNP, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ, JA, JAE, JB, JBE, JC, JE, JZ, JG, JGE, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC, JNE, JNG, JNGE, JNL, JNLE, JNO, JNP, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ

그림 1. 바이너리 명령어 그룹화

또한 코드 그래프에서는 시스템에 실질적으로 영향을 미칠 수 있는 시스템 콜들이 호출되는 부분을 중심으로 분석한다. 시스템 콜이란 파일 사용이나 네트워크 접근과 같이 운영체제가 처리를 해주어야 하는 부분들을 응용 프로그램이 사용 할 수 있도록 인터페이스로 정의해 둔 것이다. 사실 응용 프로그램은 프로그램의 목적에 맞게 운영체제가 제공하는 시스템 콜들을 적절한 순서와 조합으로 호출하도록 정의해 놓은 것이다. 운영체제 종류와 버전마다 조금씩 다르지만 일반적으로 운영체제가 제공하는 시스템 콜의 종류와 수는 수백에서 수천여개에 이른다. 시스템 콜들 중에는 단순한 메모리 복사와 같은 시스템에 영향을 미치지 않는 시스템 콜과 같은 기능을 가지지만 확장된 버전의 시스템 콜들이 있기 때문에 이러한 시스템 콜들은 적절하게 그룹화 하여 사용한다.

코드 그래프 구조

한글과컴퓨터에서 만든 한글과 Microsoft에서 만든 MS Word는 프로그램은 다르지만 분명 같은 기능을 하는 워드프로세서이다. 이 두 프로그램은 많은 부분에서 같은 기능들을 수행하므로 프로그램의 실행 코드 역시 비슷한 부분이 많을 것이라고 예측 해 볼 수 있다.

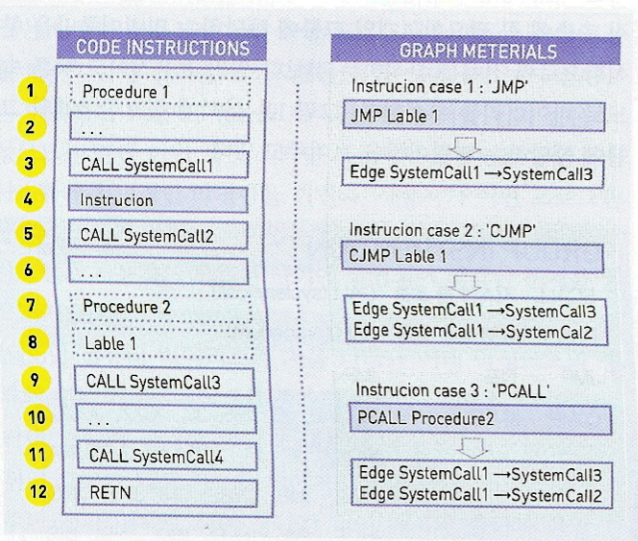

그림 2. 알고리즘 동작 예제

코드 그래프는 여기에서 동기를 얻어 개발 되었다. 코드 그래프는 실행 코드를 분석하여 프로그램을 실행 시켜보기 전에 그 특징을 추출하는 시스템이다. 프로그램이 어떠한 기능들을 포함하고 있는지, 어떠한 목적을 가지고 만들어진 프로그램인지, 어떠한 위험성을 내포하고 있는지 알아내는 일종의 미리보기 시스템이다.

이 시스템은 실행 코드에서 프로그램의 주된 특징을 표현 할 수 있는 시스템 콜 위주로 분석해 그 결과와 특징을 그래프로 나타낸다. 이 그래프를 코드 그래프라고 한다. 코드 그래프는 먼저 실행 코드를 입력받아 분석에 필요한 명령어들을 추출하고 시스템 콜이 호출되는 순서에 따라서 그래프로 변환한다. 코드 그래프는 그림 3과 같이 정의 되어 있다.

Code graph G = (V, E)
V = 노드들의 집합, E = 에지들의 집합
노드는 프로그램에서 사용된 시스템 콜
에지는 두 시스템 콜 간의 호출 관계에서 먼저 호출된 시스템 콜과 나중에 호출된 시스템 콜의 호출 순서 •

그림 3. 코드 그래프 정의

그래프 변환 알고리즘에 의해서 실행 코드는 위에 정의한 위상 그래프 형태로 변환 된다. 그림 2는 실행 코드의 명령어들을 코드 그래프로 변환 시키는 알고리즘의 예를 나타내고 있다. 시스템 콜 호출을 그래프로 변환하는 과정에 필요한 명령어들만을 추출해 그래프 형태로 변환한다. 이 과정은 두 번 이상의 검사 과정을 거치는 일반적인 디버깅 툴과 컴파일러와는 달리 단 한 번의 검사에 의해서 변환 되므로 훨씬 빠른 속도로 변환이 이루어진다.

코드 그래프는 프로그램이 실행되기 이전에 각 프로그램의 실행 코드를 그래프로 변환해 특징을 검사한다. 이때 프로그램의 복잡도에 따라 그래프의 복잡도가 변화하므로 분석하기가 어려워 질 수 있고 그래프에서 프로그램의 특징을 추출하기 어려우며 그래프 자체를 이해하기 난해할 수 있다. 따라서 프로그램의 복잡도와 상관없이 그래프를 이해하기 쉬운 형태로 표현해야 하는데 이 때 다음과 같은 조건을 만족해야 한다.

* 위상 그래프에서 표현하고 있는 모든 정보를 포함해야 한다.
* 그래프 자체가 사람이 이해하기 쉬운 형태이어야 한다.
* 각 그래프들 간의 유사도를 알아내기 쉬워야 한다.
* 각 그래프들 간의 차이점을 알아내기 쉬워야 한다.

코드 그래프는 매트릭스에서 이 조건들을 모두 만족하는 해답을 얻었다. 프로그램에서 그래프의 데이터를 저장하기 위한 구조로 매트릭스를 주로 이용하는데 이 기본 매트릭스를 코드 그래프만의 특징을 저장할 수 있는 형태로 확장하여 이용한다. 시스템 콜의 종류와 각 시스템 콜 간의 호출 유무에 따른 특징을 매트릭스 요소마다 색을 달리하여 유사도와 차이점을 쉽게 알 수 있도록 한다.

검사 및 실험

컴퓨터 앞에 앉아 인터넷 웹 서핑이나 온라인 게임과 같은 작업을 할 때 사용자의 명령을 받거나 내부적으로 실행 중인 다른 프로그램의 명령을 받아서 관련된 프로그램들이 구동 된다. 이 때 그 프로그램을 실행 시키는 이벤트를 받아서 프로그램이 실행되기 전에 코드 그래프를 이용한 검사가 먼저 이루어진다. 다행히 이미 수천, 수만 종류의 악성코드들이 알려져 있기 때문에 악성코드들의 코드 그래프로 그려 보고 그 특징들을 알아낼 수 있다. 이 정보들을 이용해 새롭게 시작 하려는 프로그램이 악성코드와 얼마나 닮아 있는지와 얼마만큼 차이가 있는지를 비교할 수 있고 악성코드 실행되는 것을 사전에 탐지하고 막을 수 있다.

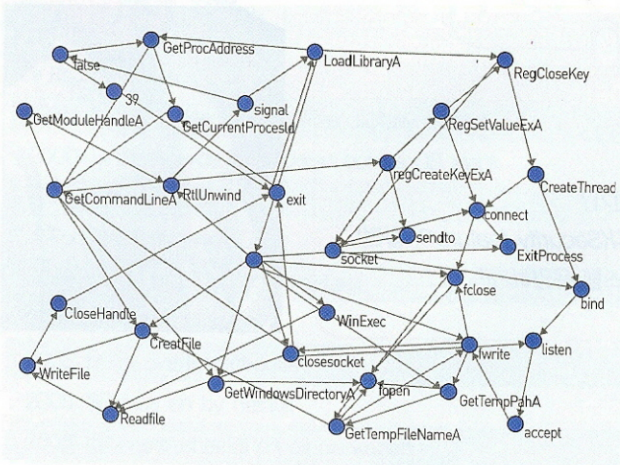

그림 4. 실제 악성코드의 코드 그래프

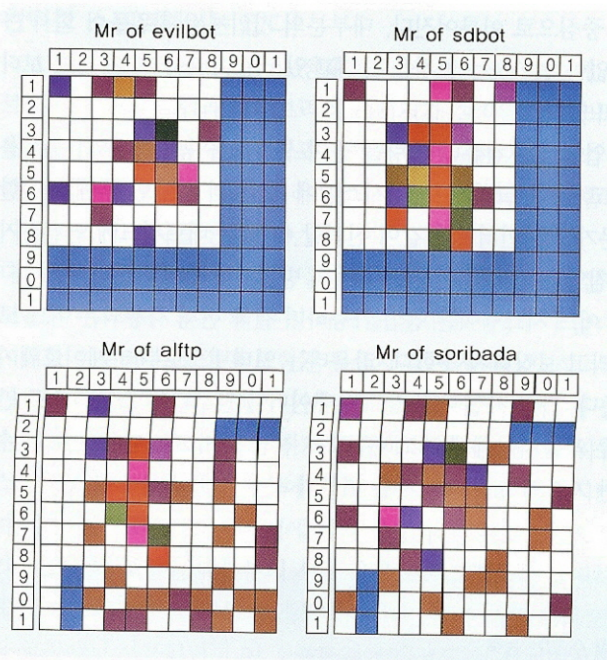

그림 5 매트릭스로 표현된 코드 그래프

실험에는 실제 악성코드들과 악성코드와 행동 패턴이 비슷한 p2p와 같은 파일 전송 프로그램들을 사용했다. 그림 4는 실제 악성코드를 코드 그래프의 위상 그래프로 표현한 것이다. 그래프 자체만으로는 어떤 의미가 있는지 알기 어려우므로 이를 매트릭스 형태로 표현했고 그림 5가 그 예를 보여준다.

위쪽 두 개의 매트릭스는 악성코드를 표현한 것이고 아래쪽 두 개는 악성코드와 행동 패턴이 비슷한 정상 프로그램을 나타낸 것이다. 악성코드에서는 특정 시스템 콜이 집중적으로 사용됐다는 것을 쉽게 확인 할 수가 있고 악성코드 간에는 유사성이 많음을 알 수 있다. 반면 정상 프로그램은 악성코드와 행동 패턴이 비슷하더라도 프로그램 내부적으로 구현된 내용은 차이점이 많음을 확인 할 수 있다.

사전 탐지기술 응용

일반 컴퓨터 사용자는 처음 접하는 프로그램의 위험성을 프로그램을 실행시켜 시스템에 특별한 문제점이 발생하기 전까지는 알기 어렵다. 코드 그래프는 이런 위험성을 미리 파악하여 악성코드로부터 시스템을 보호 할 수 있다. 또한, 프로그램에 어떠한 기능들이 포함이 되어 있는지, 혹은 어떤 목적을 가지고 만들어진 프로그램인지를 미리 엿볼 수 있기 때문에 이러한 기능들에 응용이 될 수 있다.

프로그램의 정보와 특징들을 사용자들에게 실행 전에 보여주고 표현 해 준다면 악성 코드로부터 시스템을 원천적으로 보호할 수 있고 깨끗하고 안전한 인터넷을 유지 할 수 있을 것이다.