제18회 한국정보처리학회 추계학술발표대회 논문집 제9권 제2호 (2002.11)

코드 삽입을 통한 악성 코드 감지 기법의 거시적 효과

이형준*, 김철민*, 이성욱*, 홍만표*
*아주대학교 정보통신전문대학원
e-mail:prime@doit.ajou.ac.kr, {ily, wobbler, mphong}@ajou.ac.kr

The Macroscopic Effect on Malicious Code Detection by Code Insertion

Hyung-Joon Lee*, Chol-Min Kim*,
Seong-Uck Lee*, Man-Pyo Hong*
*Graduated School of Information and Communication

요 약

현재 안티바이러스 시스템에서는 시그너쳐를 기반으로 하는 탐지 방법을 사용하거나 간단한 휴리스틱 검색을 이용 하지만, 이러한 방법은 알려지지 않은 새로운 악성 코드에 대해서는 취약하기 때문에 행위 감시 기반의 감지 방법이 추가적으로 사용된다. 그러나 행위 감시 기반의 안티바이러스 시스템을 대부분의 호스트에 설치하는 일은 많은 어려움이 있다. 이에 따라 안티바이러스 시스템이 설치되지 않은 호스트에서의 행위 감시를 위한 코드 삽입 기법이 제시 되었으나 아직 코드 삽입 기법이 거시적인 과점에서 전체 도메인에 미치는 영향에 대한 연구가 되어 있지 않다. 본 논문은 시뮬레이션을 통하여 코드 삽입 기법이 전체 도메인 상에서 악성 코드의 감지에 미치는 영향을 보여준다.

1. 서론

최근 엔터프라이즈 환경 등에서는 도메인 보호를 위해 서버용 안티바이러스 시스템을 사용하는 것이 보편적이다. 그러나, 이러한 기존 안티바이러스 시스템에서는 클라이언트용 안티바이러스 시스템과 같은 시그너쳐 기반 감지 방식을 주로 사용하고 있으며, 알려지지 않은 악성 코드를 위해 악성코드에 빈번하게 나타나는 코드 패턴을 검색하거나 실행 가능한 코드에 대한 필터링을 수행하는데 그치고 있다[1, 2].

이러한 기존 방법은 알려진 악성 코드의 변종에는 효과가 있으나, 새로운 악성코드에 대응하기 어려우며, 필터링은 필연적으로 높은 긍정 오류(False Positive)를 수반하게 된다. 따라서 새로운 악성코드의 적극적 감지를 위해서는 행위 감시를 통한 악성 코드 감지 방법이 효과적이다[3].

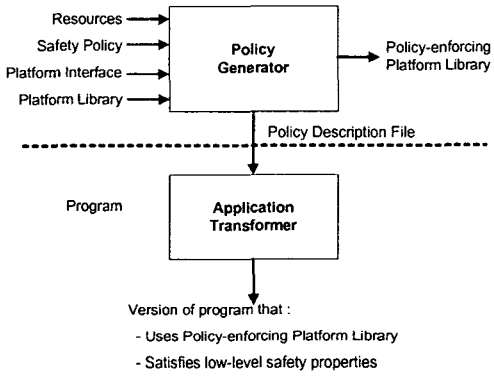
그러나, 행위 감시 기법은 실행중인 악성 코드의 행위를 감시하는 방법이므로 서버용 안티바이러스 시스템에는 적용할 수 없다. 또한 개개의 클라이언트에 설치되었을 경우에도, 운영체제 수준의 검사로 인해 시스템에 부담을 주게 되는 것이 사실이다. 특히 후자의 이유로 인해 안티바이러스 시스템의 보급률은 80-90%에 달하고 있음에도 불구하고 행위 감시 기법을 채용한 안티바이러스 시스템은 극소수에 그치고 있다[4].

이러한 문제를 해결하기 위해 외부로부터 유입되고 안전을 확신할 수 없는 코드에 자기 감시를 수행하는 코드를 삽입하는 방법이 제안되었다. 코드 변환을 통한 기법은 의심되는 코드에 악성 행위 감시를 위한 코드가 삽입되는 방법이기 때문에 안티바이러스 시스템이 설치되지 않은 호스트에서도 삽입된 코드에 의해서 전파 되는 악성 코드의 행위를 감시할 수 있다. 따라서 코드 변환 기법은 안티바이러스 시스템의 보급율이 낮은 상태에서도 전체 도메인 상에서 알려지지 않은 새로운 악성코드를 감지할 수 있어 전체 도메인을 보호할 수 있는 효과가 예상된다. 코드 삽입 기법의 시스템 구조와 특정 샘플 집합에 대한 미시적 감지 결과는 제시되었으나, 이들 기법의 거시적 효과에 대한 연구는 이루어지지 않고 있다. 본 논문에서는 시뮬레이션을 통해 메일을 통

해 전파되는 악성코드를 대상으로 코드 삽입 기법의 거시적 효과를 분석하고자 한다.

2. 관련연구

코드 변환 기법은 본래 코드 안전을 위해 제안되었다. 이 기법은 안전을 보장할 수 없는 코드에 대해서 미리 정의된 정책을 적용할 수 있도록 코드를 변환하며, 변환된 코드는 각각의 API가 호출될 때마다 해당 API 호출로 인해 접근하게 되는 시스템의 자원이 해당 사용자에게 허가되어 있는지를 검사한 후 해당 작업을 수행하게 된다. 이를 위해 제안된 아키텍쳐는 <그림1>과 같다[5].



< 그림 1 > 어플리케이션 변환 시스템 아키텍쳐

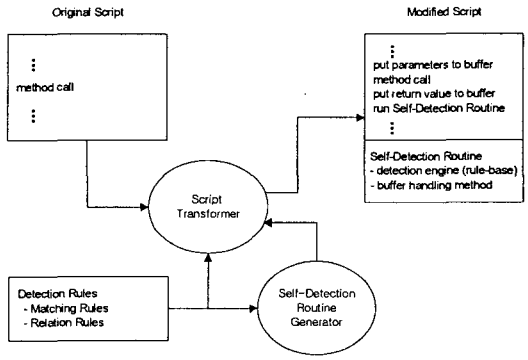
전체 시스템은 정책 생성기(Policy Generator)와 어플리케이션 변환기(Application Transformer)로 구성된다. 정책 생성기는 시스템 자원(Resource), 해당 자원 접근에 관한 보안 정책(Safety policy), 대상 플랫폼의 API 정보(Platform Interface), 그리고 API를 제공하는 라이브러리(Platform Libarary)를 입력으로 받아 정책 강행에 필요한 코드를 삽입한 새로운 플랫폼 라이브러리(Policy-enforcing Platform Library)와 상세한 코드 수정 지침이 기술된 정책 기술 파일(Policy Description Rule)을 생성한다.

대상 코드가 주어지면 어플리케이션 변환기는 정책 기술 파일을 참조하여 해당 코드의 특정 API 호출을 변형한 플랫폼 라이브러리에 대한 호출로 교체함으로써 실행 시에 사전 정의된 정책이 적용되도록 한다.

이 기법은 주어진 정책을 반영하는 API 집합을 자동적으로 생성하므로 지속적으로 변경되는 보안 정책을 도메인 관리자가 손쉽게 반영할 수 있어 특정 도메인에 유입되는 이동 코드에 대한 접근 제어 강행에 유용하게 사용될 수 있다.

그러나, 행위 감시를 통한 악성 행위의 탐지는 이와 달리 상호 관련 이 있는 이련의 API 호출 패턴을 인지하는 문제이므로 이러한 기법을 그대로 적용할 수 없다. 즉, 악성 행위의 감지는 개별적인 API의 수정만으로는 불가능하며, 악성 행위에 사용될 수 있는 API 호출들을 지속적으로 감시하다가 일련의 API 호출들이 악성행위의 그것과 일치하는지를 판단하는 진단 엔진을 필요로 한다. 따라서 스크립트 코드에서 어플리케이션 변환 기법을 이용하여 탐지 루틴을 스크립트 소스에 삽입하는 기법이 제안되었다[7].

제안된 코드 삽입 기법은 <그림2>와 같다.



< 그림 2 > 코드 삽입 기법 개념도

외부로부터 유입되거나 악성 여부가 의심되는 스크립트는 실행 전 임의의 시점에 스크립트 변환기(Script Transformer)를 거쳐 실행 중 자체 진단을 수행할 수 있는 형태로 변환된다. 그러나, 변환기는 본래부터 스크립트에 기술되어 있던 문장을 변경하지는 않으며, 추가적인 코드의 삽입만을 수행한다.

코드의 삽입은 특정 메소드를 대상으로 한다. <그림 3>은 메일을 통해 자기복제를 수행하는 비주얼 베이직 스크립트에서 대상이 되는 메소드들의 관계를 보여준다. 각 메소드들은 메일을 통해 전파되기 위해 반드시 필요하며 이러한 메소드들의 조합과 파라미터의 관계를 정적 분석을 통하여 악성 코드를 감지한다. 정적 분석에서 메소드들간의 관계를 정확히 판단할 수 없을 때, 해당 메소드의 전후에 코드

를 삽입한다. 삽입된 코드는 대상 메소드가 호출 될 때, 파라미터 값과 리턴 값을 비교하여 메소드간의 관계를 판단한다.

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

< 그림 3 > 메일을 통한 자기복제 스크립트

코드 삽입이 이루어진 이후에는 추후 실행 시 위험 메소드가 실행 될 때마다 당시까지의 메소드 호출 과정을 고려하여 악성 코드 여부를 검사하게 되므로 안티바이러스가 설치되지 않은 시스템에서도 지속적으로 삽입된 코드가 수행 된다.

시스템 전체의 모든 루틴을 가로채는 기존의 모니터링 시스템과 비교해 볼 때, 이러한 방법은 삽입된 코드 부분만 모니터링의 대상이 되기 때문에 모니터링으로 인한 부하가 현저히 감소하게 된다.

스크립트 코드가 아닌 이진 코드로 이루어진 경우에도 코드 삽입 기법을 사용할 수 있다. 이진 코드의 경우 스크립트와는 달리 의심되는 코드를 변경할 수 없기 때문에 대상 코드는 수정하지 않고 행위 감시를 위한 모니터링 코드를 추가하는 방법을 사용한다. 대상 코드가 실행되었을 때, 추가한 모니터링 코드가 먼저 실행되어 대상 코드의 행위를 감시한다.

윈도우즈 환경의 경우 모든 Win32 기반의 이진 코드는 PE(Portable Executable) 포맷을 따르고 있기 때문에 PE 파일의 헤더정보를 수정함으로써 행위 감시를 위해 삽입된 코드를 실행할 수 있다. PE 파일의 헤더 정보중 프로그램의 시작 주소부분을 추가한 모니터링 코드로 변경하고 모니터링 코드가 실행된 뒤에 실제 대상 코드를 실행한다[8].

3. 시뮬레이션

이 절에서는 코드 삽입 기법을 이용한 메일 서버가 인터넷에 보급되었을 때의 효과를 추정한다. 이를 위하여 가상 메일 환경을 만들어 바이러스의 전파를 시뮬레이션 해 보았다. 시뮬레이션은 SIMSCRIPT II.5를 이용하여 수행되었다[9].

가상 전자 메일 환경은 10000명의 메일 사용자가 있다는 가정에서 출발한다. 각 메일 사용자는 하루에 한번씩 메일을 확인한다고 가정하였다. 이에 따라 시뮬레이션의 한 스테이지는 시간적으로 하루를 의미한다.

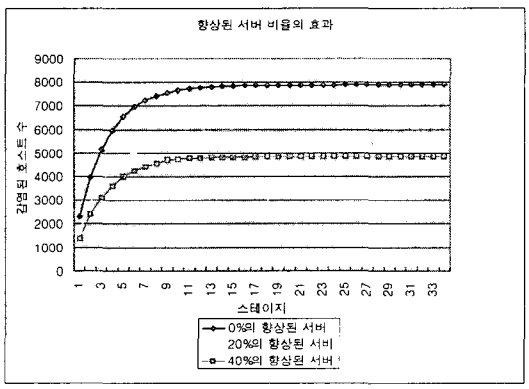
한편 가상 메일 환경의 사용자중 일부는 이미 바이러스를 탐지 할 수 있는 안티바이러스 시스템을 가지고 있다고 가정하였으며 이 도구를 이용하여 알려지지 않은 바이러스도 일부 탐지가 가능하다고 가정하였다.

가상 메일 환경에서 활동하는 바이러스는 알려지지 않은 바이러스로 한정하였다. 이를 통해 코드 삽입 기법이 알려지지 않은 바이러스의 감지율을 높여준다는 사실을 확인할 수 있었다.

가상 메일 환경내의 메일 서버 중 일부는 코드 삽입 기법을 적용한 향상된 서버이며 이 서버들은 외부로부터 유입되는 메일에 코드를 삽입 할 수 있는 기능을 가졌다고 가정한다.

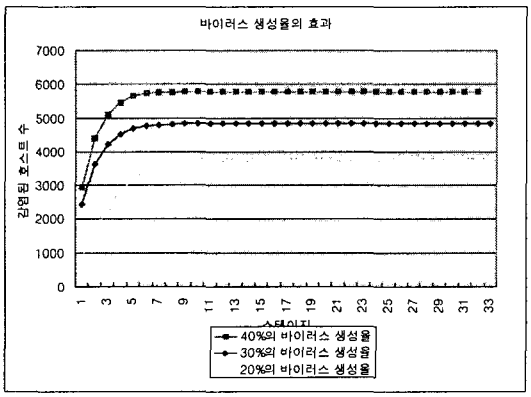
이러한 환경에서 시간(스테이지)에 따른 각 호스트의 바이러스 감염 추이는 <그림 4>, <그림 5>와 같다.

<그림 4>는 메일 사용자의 안티바이러스 설치 비율이 20% 일때, 향상된 서버가 있는 경우와 없는 경우의 바이러스 전파 속도 차를 보여준다. 향상된 서버의 비율이 0%인 경우에 바이러스는 스테이지가 증가함에 따라 급격하게 증가하지만 향상된 서버의 비율을 40%로 하였을 때 바이러스의 증가속도가 현저하게 감소 되었음을 확인할 수 있다.



< 그림 4 > 향상된 서버 비율의 효과

<그림 5> 는 향상된 서버의 비율을 40%로 고정하였을 때 바이러스의 발생율이 미치는 영향을 보여준다. 바이러스는 종류에 따라 급격한 발생율을 보이는 것이 있고 또 그렇지 않은 것이 있으므로 이러한 고찰이 필요하다. <그림 5>에 따르면 바이러스의 발생율을 30%에서 40%로 높이면 바이러스의 전파속도가 증가 하지만 여전히 <그림 4>와 비교해 볼 때 증가속도의 감소가 일어나고 있음을 확인할 수 있다.



< 그림 5 > 바이러스 생성율의 효과

4. 결론

알려지지 않은 악성 코드의 감지를 위해서는 코드의 행위를 감시하는 방법이 효과적이나, 실제 모니터링을 통한 안티바이러스 시스템의 현재 설치율을 고려하면 새로운 악성 코드의 감지는 많은 문제를 안고 있다.

코드 삽입을 이용한 악성 코드 감지 기법은 안티바이러스가 설치되지 않은 호스트에서도 모니터링을 할 수 있기 때문에 안티바이러스 시스템의 설치율이 낮은 상황에서도 거시적인 관점에서 악성 코드의 전파 속도가 감소되며 전체 도메인을 악성 코드로 부터 보호할 수 있다.

참고문헌

[1] Francisco Fernandez, *"Script-Based Mobile Threats"*, Symantec White Paper, 2000. 6.

[2] Sandeep Kumar, Eugene H. Spafford, *"A Generic Virus Scanner in C++"*, Purdue University Technical Report CSD-TR-92-062, 1992. 9.

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

[4] Igor Muttik, "Stripping down and AV Engine", Virus Bulletin Conference, 2000. 9.

[5] David Evans, Andrew Twyman, *"Flexible Policy-Directed Code Safety"*, IEEE Security and Privacy, CA, May 9-12, 1999.I

[6] 이성욱, 배병우, 이형준, 조은선, 홍만표, *"정적분석을 이용한 알려지지 않은 악성 스크립트 감지"*, 한국정보처리학회 논문지 (2002. 10. 게재 예정)

[7] 이성욱, 홍만표, *"코드 변환을 이용한 알려지지 않은 악성 스크립트 탐지"*, 한국정보과학회 논문지 (2002. 12. 게재 예정)

[8] Matt Pietrek, *"Microsoft Portable Executable and Common Object File Format Specification"* Microsoft Coporation, February 1999

[9] Edward C. Russell, "Building Simulation Models with SIMSCRIPT II.5", CACI Product Company

[10] Cholmin Kim, Seonguck Lee, Hyeongchol Jung, Yoosuk Jung, Manpyo Hong, "Macroscopic Treatment to E-mail Based Viruses", International Conference on Security and Management, Jun 24-27, 2002.