제18회 한국정보처리학회 추계학술발표대회 논문집 제9권 제2호 (2002.11)

리눅스 상에서 가상 실행기를 이용한 윈도우 파일 바이러스 탐지기법

*오근탁, **이성태, *김용국, ***이영신, *이윤배
*조선대학교 전자계산학과, **서울사이버대학교 IT학부, ***서강정보대학 정보통신과
e-mail : osc9744@chollian.net

The Design of Virtual Emulator for Detecting Windows File Virus on Linux

*Guan-Tak Oh, **Sung-Tae Lee, *Young-Gug Kim, ***Young-Shin Lee, *Yun-Bae Lee
*Dept. of Computer Science, Chosun University
**Dept. of Information Technology, Seoul Cyber University
***Dept. of Information & Communication, SeoKang College

요 약

파일 내에서 바이러스의 패턴을 탐색하는 현재의 백신 프로그램으로는 매일 수없이 제작되는 바이러스에 시기 적절하게 대응하지 못하는 어려움이 있다. 바이러스에 감염된 파일을 사후 처리하는 이러한 방식으로는 늘어나는 바이러스 문제를 궁극적으로 해결하지 못한다.

따라서, 본 논문에서는 이러한 바이러스의 행위를 탐지할 수 있는 가상 실행기를 제안한다. 제안된 시스템은 대표적인 서버 운영체제인 리눅스 상에서 동작할 수 있도록 설계한다. 이를 이용함으로써 리눅스가 설치된 파일서버에 저장된 각종 윈도우 파일의 바이러스 감염여부를 판단할 수 있다. 또한, 제안된 시스템은 리눅스 뿐만 아니라 다른 Unix 계열 플랫폼에서도 동작할 수 있다는 장점이 있다.

1. 서론

컴퓨터 바이러스(이하 바이러스)는 인터넷이 급속히 확산됨에 따라 그 피해 또한 기하급수적으로 늘어만 가고 있다. 최근에는 외국에서 발견된 바이러스가 몇 시간 뒤면 국내에서도 발견되고 있는 게 현실이다. 이제 바이러스 문제는 국경을 초월한 전 인류의 문제로 대두되고 있다[9].

바이러스에 감염된 파일을 검색해 바이러스의 패턴을 탐지하는 현재의 백신 프로그램으로는 수없이 제작되는 바이러스에 신속하게 대응하지 못하는 어려움이 있다. 감염된 파일을 사후 처리하는 방식으로는 늘어만 가는 바이러스 문제를 궁극적으로 해결하지 못한다[11,12].

이를 해결하기 위해서 바이러스 코드의 행위를 추적해 볼 수 있는 가상 실행기를 이용한 방법이 필요하다. 이에 본 논문에서는 바이러스 실행코드와 실행상의 특징을 이용하여 가상 메모리 상에서 바이러스의 행위를 탐지하기 위한 새로운 가상 실행기를 제안한다. 제안된 시스템은 파일서버 운영체제로 많이 사용되고 있는 리눅스(Linux)에서 동작하도록 설계하였다.

2. 가상 실행기의 정의와 특징

일반적으로 가상 실행기는 어떤 하드웨어나 소프트웨어의 기능을 다른 종류의 하드웨어나 소프트웨어로 모방하여 실현시키기 위한 장치나 프로그램을 말한다. 이러한 가상 실행기가 필요한 이유는 시스템에 영향을 주지 않으면서도 프로그램을 가상으로 동작시켜 볼 수 있다는 장점 때문이다.

이러한 가상 실행기는 하나의 독립된 프로그램으로서 동작하게 된다. 가상 실행기의 구성요소를 살펴보면 다음과 같다.

프로그램 코드와 데이터가 실행되려면 CPU와 메모리가 필요하듯이 가상 실행기를 구성하려면 기본적으로 실행 프로그램을 적재하고 실행할 가상의 메모리 공간이 필요하다. 또한, 메모리에 적재된 프로그램을 실행시킬 수 있는 디코더(decoder)가 필요하다. 즉, 실행 프로그램은 기계어로 쓰여진 프로그램이므로 파

일로부터 읽혀진 기계어를 해석하고 해당 동작을 수행할 수 있는 프로그램이 바로 디코더이다[4].

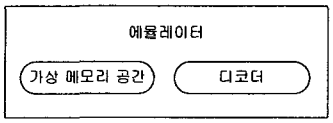

그림 1 가상 실행기 구성요소

본 논문에서 제안한 가상 실행기는 커널 위에서 동작하고, 바이러스는 그 가상 실행기 위에서 동작하게 된다. 또한, 시스템에 실제 피해를 주지 않으면서도 바이러스가 어떤 목적으로 제작되었고, 어떤 행위들을 하는 지 감시할 수 있다.

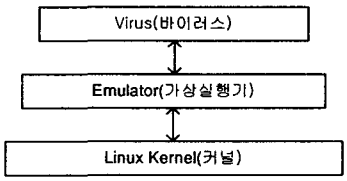

그림 2 가상 실행기와 바이러스의 관계

3. 가상 실행기 설계

본 장에서는 바이러스의 행위를 모니터링 할 수 있는 가상 실행기에 대하여 논하고, 인텔 CPU에서 동작할 수 있도록 설계한다[8,10].

3.1 16비트 가상 실행기 설계 및 구현

(1) 가상 메모리 공간 설계

본 가상 실행기에서는 가상 메모리 개념을 도입, 4KB의 배열을 한 페이지로 하여 모두 16 페이지(64KB)를 준비하였다. 그리고, 한 페이지마다 사용횟수를 기록하는 변수를 두어서 메모리 참조가 한번씩 일어날 때마다 1씩 증가하게 하여 나중에 스왑(swap)이 필요할 때 조회 할 수 있도록 하였다. 1MB의 메모리 대신 매핑 테이블을 메모리 참조가 일어날 경우, 먼저 참조되는 주소를 매핑 테이블의 배열 요소로 변환하고 그 위치의 페이지가 메모리에 존재하는지를 판단한다. 존재하면 페이지 번호를 얻고 참조되는 주소에서 페이지 내의 오프셋을 산출하여 해당 페이지의 해당 주소에 접근하게 한다.

이러한 매핑 테이블의 구조를 C언어로 표현하면 아래와 같다.

```
struct Dinfo {
        WORD numCache;
        BOOL in_memory;
} MappingTable[16];
```

여기에서 numCache는 캐쉬페이지 번호를 나타내고 in_memory는 페이지가 메모리에 있는가의 유무를 기록한다.

다음은 실제 주소를 가지고 매핑 테이블을 찾아가는 예이다.

실제 주소가 012345h라면 4096(=01000h)로 나누어서 배열번호 012345h/01000h = 12h = 18을 얻고, MappingTable[18]에 접근한다. 그리고, MappingTable[18].in_memory가 TRUE 값을 가진다고 가정하면, BufferCache[MappingTable[18].numCache].memSpace 배열에 접근하고 여기서 memSpace 내의 배열요소는 012345h를 01000h로 나눈 나머지가 된다. 즉, memSpace[345h]를 접근하면 되는 것이다.

아래 (그림 3)은 매핑테이블 및 페이지 배열만 가상 실행기 프로그램 내에 유지하면서 가상의 1MB 공간의 메모리를 접근하는 방식을 그림으로 표현한 것이다.

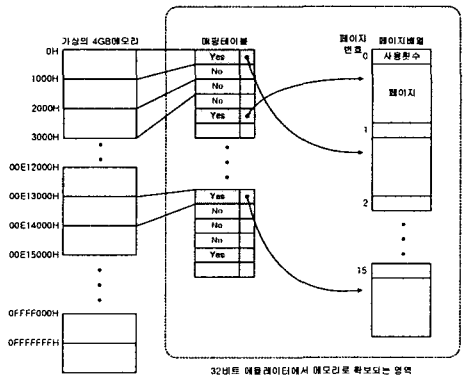

그림 3 가상 실행기용 가상 메모리

(2) 가상 실행기 설계

가상 실행기의 구조는 CPU의 구성방식을 본 따서 설계하였다. 버스 대신 주어진 주소에 대하여 해당 페이지 내의 메모리를 매핑 시켜주는 가상메모리 처리기를 두었다. 가상메모리의 구조는 앞에서 설명한 바 있다. 가상메모리로부터 명령어를 가져와서 큐에 적재하는 fetch() 함수를 두었다. fetch() 함수 실행의 결과로 명령어가 Instruction Queue에 적재되면 디코더가 이를 입력으로 받아 명령을 수행한다.

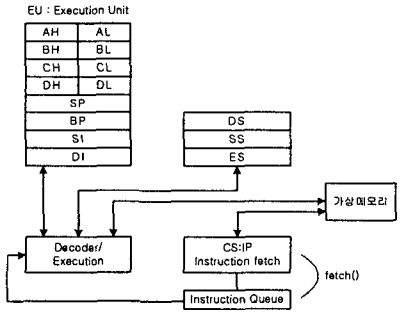

그림 4 가상 실행기의 구조

실행 프로그램은 기계어로 표현되어 있고 이진수로 구성되어 있다. 입력되는 이진값들은 OP Code의 종류, Operand 종류 및 크기의 정보가 들어있다. 인텔 8086의 인스트럭션 포맷은 아래 (그림 5)와 같다.

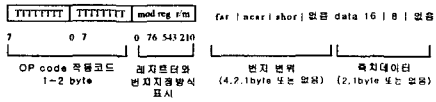

그림 5 OP Code의 구조

명령어 스트링의 첫 번째 바이트에서 명령어의 종류가 결정됨을 알 수 있다. 그래서, 각각의 기능을 함수로 정의하고 이 함수들의 포인터 배열을 만든 다음, 명령어 큐의 첫 번째 바이트의 값을 배열의 인덱스로 처리한다. 그러면, 입력된 OP Code에 대하여 비교, 판단을 거치지 않고 해당되는 함수를 바로 수행할 수가 있다.

각각의 명령어를 수행하는 함수를 동일한 파라미터 구조로 만든다. 여기서 ipstr은 명령어 큐의 맨 꼭대기 주소이다. 그리고, 명령 수행 후 디코딩에 사용된 인스트럭션의 크기를 정하고 그 값을 isize에 돌려주어 다음 실행주소(CS:IP)를 결정하는 데 사용한다.

이렇게 작성된 각 함수들의 함수 포인터를 저장하는 구조를 만들고 함수 포인터의 배열을 구성한다.

이상에서 설명한 바와 같이 가상 실행기를 수행하는 방법을 C언어로 표현하면 아래와 같다.

```
while(1)
{
  opcode=fetchcode();
  // 명령어 큐에서 실행할 명령을 가져온다
  isize =
lsetLists[*opcode].Operation(opcode);
  // *opcode는 맨 앞 OP code이므로
  // 이를 함수배열의 인덱스로 사용한다
}
```

4. 결론 및 향후 연구과제

파일 내에서 바이러스의 패턴을 탐색하는 현재의 백신 프로그램으로는 매일 수없이 제작되는 바이러스에 시기 적절하게 대응하지 못하는 어려움이 있다. 바이러스에 감염된 파일을 사후 처리하는 이러한 방식으로는 늘어나는 바이러스 문제를 궁극적으로 해결하지 못한다. 본 논문에서는 바이러스 제작자가 의도적으로 바이러스 암호 해제 코드의 반복 실행 수를 늘려 놓았다 하더라도 WINDOWS 파일 바이러스를 탐지해 내는 새로운 알고리즘을 이용한 에뮬레이터를 제안하였다.

제안된 시스템은 대표적인 서비 운영체제인 리눅스 상에서 동작할 수 있도록 설계하였다. 제안된 시스템은 LINUX 뿐만 아니라 다른 UNIX 계열 플랫폼에서도 동작 할 수 있다는 장점이 있다. 이를 이용함으로써 플랫폼에 구애받지 않으면서도 더욱 빠르게 WINDOWS 파일 바이러스들을 탐지해낼 수 있다. 하지만, 날이 갈수록 고급기술에 의한 바이러스 제작이 늘어가는 현실 속에서 바이러스 창궐 후 분석하고 대처 방안을 세우기에는 시간적으로 기술적으로 한계가 많다. 본 연구의 결과로 최근 급격히 증가하고 있는 WINDOWS 파일 바이러스 등에 대한 대응책과 함께 컴퓨터 바이러스 오진율을 낮추는 계기가 되길 바란다.

참고문헌

[1] Eugene Kaspersky, Vadim Bogdanov, "Strange - A New Way to Hide", Virus Bulletin, pp. 12-13, April 1993.

[2] Jan Hruska, *Computer Viruses and Anti-virus Warefare*, Ellis Horwood, 1990.

[3] Keith Haviland, Dina Gray, Ben Salama, "UNIX System Programming, 2/E",1999.

[4] Koray Oner, Luiz A. Barroso, Sasan Iman, Jaeheon Jeong, Krishnan Ramamurthy and Michel Dubois, "The design of RPM : an FPGA-based multiprocessor emulator", Proc. of the third International ACM symposium on Field-programmable gate arrays, pp. 60-66, 1995.

[5] Kurt Wall, "Linux Programming by Example", QUE, 2000.

[6] Marko Helenius, "Automatic and Controlled Virus Code Execution System", Proc. of eicar Conference'95, EICAR, 1995.

[7] Peter Szor, "Attacks on Win32 - Part II", Proc. of VB2000, p111, 2000.

[8] Richard Stones,"Beginning Linux Programming, 2E", 정보문화사, 2000.

[9] Vesselin Bontchev, "Future Trends in Virus Writing", Anti-Virus Papers Online, Virus Bulletin, 2001.

[10] W.Richard Stevens, "Advanced Programming in the UNIX Environment", Addison Wesley 1992.

[11] 권석철, 주염흠, 김판구, "컴퓨터 바이러스 완전소탕", 크라운 출판사,1997.

[12] 안철수,"바이러스 분석과 백신 제작", 정보시대,1995.