리눅스 ELF 파일 변형 바이러스의 분석 및 탐지

곽효승[0], 김두현, 김판구
조선대학교 전자계산학과
E-mail : basic@stmail.chosun.ac.kr, {mindul, pkkim}@mina.chosun.ac.kr

Analysis and Detection of Linux ELF File Viruses

Hyo-seong Kwak, Doo-hyun Kim, Pan-Koo Kim
Dept. of Computer Science, Chosun University

요 약

본 논문에서는 리눅스의 대표적인 파일 포맷인 ELF 파일의 구조에 대해서 알아보며 리눅스 ELF 파일 변형 바이러스의 패턴을 추출하기 위해서 몇몇의 ELF 파일 변형 바이러스를 분석해보고 이것을 바탕으로 패턴을 추출하였다. 바이러스를 탐지하기 위한 방법으로는 전체 검색법과 특정위치 검색법이 있는데 본 논문에서는 특정위치 검색법을 사용하여 ELF 파일 변형 바이러스를 탐지하는 시스템을 구성하였다.

본 논문에서 제안한 바이러스 탐지시스템은 리눅스 ELF 파일 변형 바이러스 탐색을 위해 대상 ELF 파일을 열어서 ELF 파일 여부를 확인하고, 그 파일의 헤더와 프로그램 헤더 부분을 읽어들인 다음 특정 위치를 추적해 바이러스 코드 패턴의 유무를 판별하도록 하였다. 시스템 구현결과 리눅스 ELF 파일 변형 바이러스도 기존의 Windows 계열 파일 변형 바이러스와 비슷한 행위를 수행함을 확인하였다.

1. 서론

컴퓨터 사용자가 증가함에 따라서 공개 운영체제인 리눅스를 사용하는 사용자가 급증하고 있다. 리눅스 운영체제의 사용자가 증가한 만큼 그에 따른 피해도 많아지고 있다. 이전에 DOS 또는 윈도우즈에서 바이러스를 제작하던 제작자들이 리눅스의 사용이 많아지면서 리눅스의 바이러스를 만들어 내고 있다. 최근의 리눅스 바이러스를 보면 기생형 바이러스의 형태를 띄고 있다. 기생형 바이러스는 그들의 헤더와 코드를 일반 파일에 고의로 덧붙여서 그 파일이 실행됨에 따라서 바이러스 코드를 거쳐서 바이러스가 특정 조건에 맞으면 바이러스의 작동이 시작되게된다. 리눅스에서 바이러스의 피해를 줄이기 위해서는 바이러스가 작동하기 전에 바이러스를 탐지해 제거해야 하는데, 본 논문에서는 리눅스의 대표적인 파일포맷인 ELF 파일에 기생하는 ELF(Executable and Linkable Format) 파일 바이러스에 대하여 분석하고 이 바이러스를 탐지하기 위한 방법을 제안한다.

본 논문의 구성은 2장에서 ELF 파일 시스템과 ELF 파일 바이러스에 대하여 서술하고, 3장에서는 제안된 바이러스 탐지 방안을 4장에서는 결론 및 향후과제를 제시한다.

2. ELF 파일의 구조와 ELF 파일 바이러스 분석

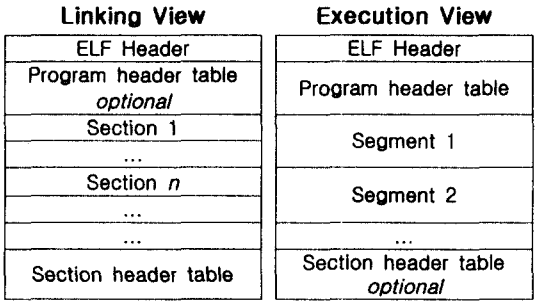
2.1 ELF 파일 구조

공개 운영체제인 리눅스는 여러 가지 형태의 버전이 있는데, 그 중에 전 세계적으로 배포된 RedHat과 Suse가 있다. 우리나라에서는 Redhat과 Redhat 한글 버전이 가장 많이 사용되고 있다.

ELF는 원래 UNIX 시스템 연구소에서 Application Binary Interface(ABI)의 한 부분으로 개발되고 공개되었다. Tool Interface Standards committee(TIS)는 ELF 표준을 다양한 운영체제를 위해서, 32 비트 인텔 아키텍처 환경에서 동작하는 이식 가능한 목적파일로 선택하였다.

ELF 표준은 다양한 운영체제에 걸쳐서 사용될 수 있는 이진 인터페이스를 프로그래머에게 제공함으로써, 소프트웨어 개발에 연계성을 주기 위해 만들어졌는데 서로 다른 여러 인터페이스의 구현을 방지하고, 프로그램을 다시 짜고 재 컴파일 해야 할 필요를 줄이도록 하기 위함이다.

목적파일은 프로그램의 링킹단계와 프로그램의 실행단계에서 사용된다. 편리함과 효율성을 위해, 목적파일의 구조는 [그림 2.1]처럼 두 가지 형태의 파일 구조를 나타내어 서로 다른 용도로 쓰임을 알려준다.

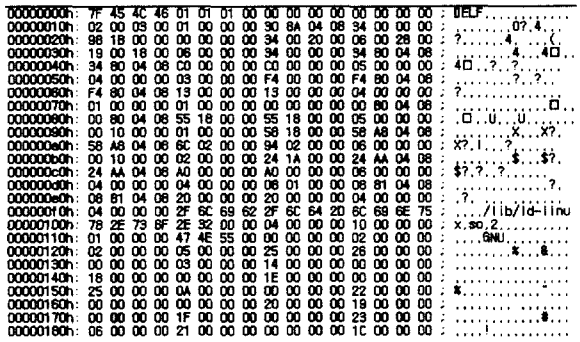


[그림 2.1] ELF 목적 파일 구조

ELF 헤더는 파일의 처음 부분에 있으며 전체 파일 구성의 이미지를 알려준다. 섹션(Section)은 링킹의 관점에서 필요한 명령어, 데이터, 심볼테이블, 재배치 정보 등의 목적파일 정보를 가지고 있다.

프로그램 헤더 테이블이 존재할 수 있는데, 그 때는 시스템이 어떻게 프로그램을 실행시키는지를 알려준다. 프로그램을 실행시키는데 필요한 파일은 반드시 프로그램 헤더 테이블을 가지고 있어야 한다. 반면 재배치 목적파일은 프로그램 헤더 테이블이 필요 없다. 섹션 헤더 테이블은 목적파일에 들어있는 섹션들이 어떻게 구성되어 있는지에 대한 정보를 포함하고 있다. 모든 섹션들은 이 섹션 헤더 테이블에 하나씩의 엔트리를 가지고 있으며, 각 엔트리는 섹션의 이름, 크기 등에 대한 정보를 가지고 있다. 링킹단계에서 사용되는 목적파일들은 반드시 이 섹션 헤더 테이블을 가지고 있어야 하며, 다른 종류의 목적파일에는 섹션 헤더 테이블이 없을 수도 있다.

[그림 2.2]는 샘플 분석을 위하여 임의 파일을 선택하였다. 'kill'의 경우 깨끗한 HANCOM LINUX 버전 2.0 실행파일이다. 16진수 에디터로 그 파일의 첫머리를 열어본 것을 보이고 있다.



[그림 2.2] 깨끗한 한컴 리눅스 실행파일의 첫머리

ELF 헤더는 몇 가지의 목적파일 제어구조의 실제 크기를 담고 있으며, 따라서 이러한 제어구조는 쉽게 커질 수 있다. 만일 목적파일의 구조가 변한다면, 목적파일을 처리하는 프로그램은 예상했던 크기보다 크거나 작은 제어구조를 보게 될 것이다. 따라서 이러한 경우 프로그램은 여분의 정보들을 무시하게 된다. 빠진 정보들은 목적파일의 문맥에 의존하며 목적파일의 확장이 정의된 경우에 표시되게 된다.

ELF 헤더는 ELF 파일을 나타내는 부분과 ELF 파일의 형태, 어떤 기계에서 사용되는 파일인지, 목적 파일의 버전, 최초 제어를 이동시켜 실제 프로그램이 실행되는 가상의 주소 값, 목적파일 내의 프로그램 헤더 테이블의 시작위치, 목적 파일내의 섹션 테이블 시작위치, ELF 헤더의 크기, 프로그램 헤더 테이블이 몇 개나 있는가, 프로그램 헤더 테이블에 들어 있는 각 엔트리의 크기의 바이트 단위, 섹션 헤더 테이블에 들어 있는 각 엔트리의 크기의 바이트 단위, 섹션 이름 문자열 테이블을 나타내는 관련된 섹션 헤더 테이블의 엔트리 인덱스 등의 정보를 포함하고 있다.

ELF 파일은 처음 4바이트가 ELF 파일인지를 검사할 수 있는 문자열을 가지고 있다. 이 문자열은 "0x7fELF"로 구성되어 있다. 이 문자열이 있으면 ELF 파일로 인식하고, 없으면 ELF 파일이 아닌 것으로 인식한다.

2.2 ELF 파일 변형 바이러스 분석

2.2.1 ELF 바이러스 감염

ELF 바이러스의 감염은 ELF 파일의 'text segment'와 'data segment' 부분에서 이루어진다. 'text segment'는 ELF 파일에서 'text segment' 내에 있는 코드를 읽고 수행할 수는 있지만 수정되지 않도록 메모리를 보호한다. 'data segment'는 그 내용을 읽어 들일 수도 있고 때에 따라서는 'data segment' 내에 필요한 내용을 넣을 수도 있다. [그림 2.3]는 ELF 파일의 구조적인 모양을 보여준 것이다.

#1	[TTTTTTTTTTTTTTTT]	<- Part of the text segment
#2	[TTTTTTTTTTTTTTTT]	<- Part of the text segment
#3	[PPPPDDDDDDDDDDDD]	<- Part of the data segment
#4	[DDDDDDDDDDDDPPPP]	<- Part of the data segment

[...] A complete page
T Text
D Data
P padding

[그림 2.3] ELF 파일의 구조적 모양

Text segment는 page 1, 2로 구성되어 있고, Data segment page 3, 4로 구성되어 있다. Text segment는 프로그램이 코드로 구성되어 있다. Data segment는 Text segment에 의해서 수행되어지는 코드들에 의해서 사용할 변수나 사용되어지는 문자열을 저장하는 공간으로 프로그램이 수행되면서 불려다 쓰여진다. ELF 파일 바이러스의 구성은 [그림 2.4]와 같다.

#1	[VTTTTTTTTTTTTVPP]	<- Text segment
#2	[PPPDDDDDDDDDDPPPP]	<- Data segment

V 바이러스 코드
T Text
D Data
P Padding

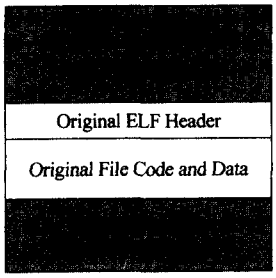
[그림 2.4] ELF 바이러스 형태

바이러스의 대부분은 기생형 바이러스로 text segment의 뒷부분에 바이러스에 관련된 코드가 덧붙여진다. text segment의 앞부분에서는 바이러스가 있는 위치로 점프하는 명령이 있어서 이 위치로 이동한 다음 바이러스가 위치한 코드부터는 바이러스가 이 파일에 감염되어 있는지를 검사한 다음 감염되어 있지 않으면 감염시키고 또, 메모리에 상주시켜서 사용자가 다른 파일을 열 때 그 파일에 감염시킨다. 이는 윈도우계열에서 존재하는 파일 변형 바이러스와 똑 같은 행위방식을 보이고 있다.

2.2.2 몇몇 ELF 파일 바이러스의 분석

2.2.2.1 Bliss Virus

1997년에 발견된 리눅스 이진 바이러스인 Bliss는 상대적으로 단순한 형태의 바이러스이다. 즉, 원본 파일에 바이러스의 헤더와 코드를 덧붙이는 바이러스이다. 감염된 파일들은 2개의 ELF 헤더를 갖게 된다. 첫 번째는 바이러스로부터, 두 번째는 정상적인 파일로부터 갖게 된다. 그래서, 감염된 파일의 두 번째 헤더(원본 ELF 헤더)는 오프셋 48AC(hex)에서 시작한다. 그러므로 진단과 치료가 비교적 쉬운 바이러스라고 할 수 있다.

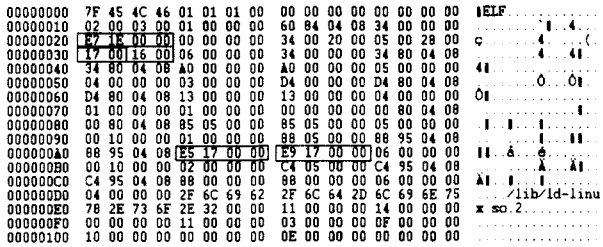


[그림 2.5] Bliss.B 바이러스 감염 예

2.2.2.2 Silv virus

Silv Virus는 ELF 파일의 Entry point의 위치를 변형 시켜서 바이러스를 삽입하는 형태를 취한다. 이 바이러스는 정상적인 ELF 파일의 중간 부분에 삽입하는 것으로 ELF 파일이 최초로 시작되는 부분의 위치를 바꾸어서 바이러스가 존재하는 부분에서부터 수행되도록 파일의 포인터를 이동시킨다. 이 바이러스는 하나의 섹션(.data)을 더 삽입하고, 이 섹션을 부르는 형태를 취한다. Silv Virus는 Entry Point는 바꾸지 않지만, Entry point에 의해서 시작되는 위치는 바뀌게 된다.

[그림 2.6]을 보면 Sliv Virus가 감염된 파일의 헤더부분을 나타낸 것이다. 섹션 헤더 부분은 00001EE7 번지에서 시작하고 섹션헤더가 16에서 17로 늘어났다. 섹션의 문자열 테이블을 가지고 있는 곳은 16번째 섹션에서 가지고 있다. 첫 실행될 프로그램 헤더의 파일의 크기와 메모리에 적재될 크기가 변형되었다.



[그림 2.6] Silv virus의 예

3. ELF 파일 변형 바이러스 탐지 및 치료

리눅스의 ELF 파일은 바이너리 형태로 되어 있고, 이 파일의 구성은 ELF 헤더, 프로그램 헤더, 섹션 헤더 등으로 구성되어 있다. ELF 파일을 다루기 위해서는 이러한 구조들을 알고 있어야 한다. 리눅스에서 ELF 파일 바이러스를 탐지하기 위해서는 먼저 ELF 헤더를 읽어오고, 읽어온 ELF 헤더에서 프로그램 헤더부분을 찾아서 읽게되고 불러온 여러개의 헤더 중에서 최초로 수행하게되는 프로그램 헤더를 불러와서 바이러스를 탐색한다.

바이러스 탐색 방법으로는 전체 검색법과 특정위치 검색법을 사용하고 있다. 전체 바이러스 탐색 기법은 파일 전체를 바이러스 패턴에 의해서 검색하기 때문에 바이러스 탐지 효율이 우수할 수는 있으나 단점으로는 검색 속도가 느리고 또 바이러스가 아닌 파일을 바이러스로 오인 할 수 있다. 특정위치 검색법은 전체 검색기법에 비해 속도가 빠르다는 장점은 있지만 알려지지 않은 바이러스를 찾아내는데는 다소 어려움이 있다. 본 논문에서는 특정 위치 검사방법을 이용하여 바이러스 패턴을 ELF 파일의 내용과 비교하여 바이러스를 탐지하고자 한다.

3.1 리눅스 ELF 파일 변형 바이러스의 탐지

리눅스 ELF 파일 변형 바이러스를 탐지하기 위해서는 우선 바이러스의 패턴을 분석 추출하여야 할 것이다. 추출한 패턴을 바이러스와 비교하기 위해 [그림 3.1]과 같은 구조체 형태로 저장한다.

```
struct pattern{
        unsigned int iOffset;
        unsigned char cLen, sVirStr[10];
} BvPattern;
```

[그림 3.1] 바이러스 패턴 구조체

ELF 파일 변형 바이러스의 탐지를 위해 ELF 파일을 오픈한다. 오픈한 ELF 파일의 헤더부분을 읽어오고, 읽어온 부분 중 프로그램 헤더에 해당되어지는 부분의 위치를 읽어들여 프로그램 헤더를 읽는다. 읽은 프로그램 헤더 중 처음으로 수행되어질 프로그램 헤더를 찾아 실행되는 첫 부분의 일정길이를 읽어 바이러스로 추출된 패턴과 비교하여 바이러스를 탐지한다. [그림 3.2]는 바이러스 탐지의 절차를 기술한 것이다.

1. 바이러스의 패턴 조사
2. ELF 파일을 오픈
3. ELF 파일의 헤더부분을 읽음
4. ELF 파일의 프로그램 헤더를 읽음
5. 프로그램 헤더를 찾아 일정 길이를 읽음
6. ELF 파일 변형 바이러스의 패턴과 비교
7. ELF 파일을 닫음

[3.2] 바이러스 탐지 절차

3.2 리눅스 ELF 파일 변형 바이러스의 치료

리눅스 ELF 파일 변형 바이러스를 치료하기 위해서는 ELF 파일을 열고 ELF 파일의 헤더 부분들을 읽어들인다. 바이러스 코드가 시작되는 부분의 위치를 계산하여 바이러스를 제거한다. 원래의 ELF 파일의 헤더를 모두 복구시킨다. 파일 포인터를 파일에 바이러스 길이를 뺀 위치에 파일 포인터를 위치시켜서 파일을 복구시킨다. 파일의 길이를 원래대로 복구시키기 ELF 헤더에 있는 파일의 크기 부분을 복구시킨다. [그림 3.3]은 바이러스의 치료 절차를 기술한 것이다.

1. ELF 파일을 오픈
2. ELF 파일의 헤더 영역을 읽어들임
3. 바이러스 프로그램이 시작되는 위치를 계산
4. 원래의 헤더 영역을 복구시킴
5. 파일 포인터를 바이러스의 크기를 뺀 위치에 위치시키고, 파일을 복구시킴
6. 프로그램의 길이를 원래대로 복구시킴
7. 프로그램을 닫음

[그림 3.3] 바이러스 치료 절차

4. 결론 및 향후과제

본 논문에서는 리눅스의 대표적인 이진 파일 형식인 ELF와 ELF 파일을 감염시키는 ELF 파일 바이러스에 대하여 알아보고 그 탐지 방법에 대하여 고찰하였다. 제안된 바이러스 탐지방법을 이용하여 ELF 파일 변형 바이러스를 탐지할 수 있다.

기존의 윈도우즈의 파일 변형 바이러스의 동작은 바이러스가 감염되면서 특정 텍스트가 변형되는 형태로 다른 파일을 감염시키게된다. 리눅스 ELF 파일 변형 바이러스 또한 윈도우즈의 감염 형태를 따르고 있고 이 같은 경우는 패턴을 이용하여 바이러스 탐지 할 수 있음을 보였다.

앞으로는 리눅스 스크립트 바이러스와 리눅스를 E-mail 서버로 많이 이용하고 있는 시점에서 E-mail에 첨부된 바이러스를 탐지할 수 있는 시스템에 대한 연구가 필요하다고 생각된다.

5. 참고문헌

[1] 김두현 김판구, *리눅스상에서의 ELF 파일 바이러스 진단 및 차단 시스템에 관한 연구*, ITRC Forum 2001(ITRC)

[2] Jakub Kaminski, *LINUX MALWARE HAS THE NEXT BATTLEFIELD BEEN DECIDED?*, Virus Bulletin 2000.

[3] Marius Van Oers, *LINUX VIRUSES - ELF FILE FORMAT*, Virus Bulletin 2000.

[4] Andrew P. Kosoresow and Steven A. Hofmeyr, *Intrusion Detection via System Call Traces*, IEEE Software, September/October 1997.

[5] 김두현 정종근 정일용, 파일 전송 프로토콜을 경유하는 실시간 바이러스 차단 시스템 설계, 한국 정보처리학회 2000년 추계 학술 발표대회.

[6] Peter V Radatti, *Computer Viruses in Heterogeneous Unix Networks?*, Proceedings of the Fifth Virus Bulletin Conference, 1995.

[7] Peter V Radatti, *The Plausibility of UNIX Virus Attacks?*, Cybersoft *Inc.*, 1996.

[8] Keith Haviland, Dina Gray, Ben Salama, *UNIX System Programming*, 2/E, 1999

[9] Kurt Wall, *Linux Programming by Example*, QUE 2000

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