Linux 상에서 메모리 덤프 데이터 분석 기술 조사*

이길주, 김일희, 허 영*, 박용수, 임을규

한양대학교 정보통신학부

국가보안기술 연구소*

Survey on Dumping and Analysing Techniques of Memory Contents in the Linux

Gil-Ju Lee, Il-Hee Kim, Young Heo*, Yong-Su Park, Eul-Gyu Im

Collage of Information and Communications, Hanyang University.

NSRI (National Security Research Institute)*

요 약

리눅스 메모리 덤프데이터 분석기술은 메모리에 저장되는 데이터를 찾는데 목적이 있고, 현재 수행중인 프로그램이나, 리눅스 커널의 오류 발생 원인을 찾는데 이용이 된다. 이 논문에서는 리눅스 메모리 덤프 방법과 메모리 덤프 데이터 분석 기술에 대하여 기존의 방법들을 조사한 결과를 설명한다. 메모리 덤프데이터 분석기술은 메모리에 저장되는 기본적인 데이터를 조사를 할 수 있고, 수행중인 프로그램을 분석하고, 이미 만들어진 시스템을 역으로 추적하여 애초의 문서나 설계기법 등의 자료를 얻어낼 수 있다.

I. 서론

메모리에는 저장장치에 저장이 되지 않는 개인정보나 비밀번호와 같은 중요한 정보뿐만 아니라 실행 파일의 어셈블리코드, 프로그램의 알고리즘 및 구조에 관련된 내용이 저장된다.

메모리 덤프데이터 분석 기술은 메모리에 저장되어 있는 데이터의 분석 및 추출, 현재 수행중인 프로그램의 데이터 추출, 동작원리 분석, 프로그램의 구조파악에 필요하다. 실행파일 자체만으로는 프로그램의 알고리즘이나 구조를 파악하기 어렵기 때문에 메모리에 저장되는 데이터를 이용하여 수행중인 프로세스의 결과에 대하여 역으로 추적을 하여 처음 프로그램 제작시의 알고리즘이나 설계에 관련하여 정보를 얻을 수 있다. 그리고 프로그램이나 커널의 오류가 발생했을 때 문제점에 대한 적절한 해결 방안을 찾아내기 위해 필요하다.

윈도우즈는 기본적으로 물리적 메모리 덤프 유틸리티, 커널 크래시 덤프, 메모리 분석 도구가 제공된다. 하지만 리눅스는 기본적인 분석도구는 지원되지만 물리적인 메모리 덤프는 제공되지 않기 때문에 물리적 메모리 덤프를 위해서는 유틸리티를 직접 제작하거나, 오픈소스를 이용하여 메모리 덤프를 받아 분석하여 메모리에 어떤 종류의 데이터가 저장되고, 물리적 메모리 프레임에 어떤 데이터가 저장되는지 확인해야 한다.

본 논문의 구성은 다음과 같다. 2장에서는 리눅스 메모리 모델을 설명하였다. 3장에서는 사용자 모드 덤프와 물리적 메모리 덤프의 두 가지 방식을 설명하였다. 4장에서는 덤프 메모리 데이터 분석에 대하여 설명하였다. 5장에서는 결론을 서술하고, 6장에서는 향후 연구방향에 대하여 서술하였다.

* 본 연구는 국가보안기술연구소의 2006 위탁과제 (과제번호 : 06052) 연구 결과로 수행 되었습니다.

II. 메모리 모델

리눅스 운영체제의 메모리 모델에는 크게 물리적 메모리(Physical Memory)와 가상 메모리(Virtual Memory)로 구분된다.

1. 물리적 메모리(Physical Memory)

물리적 메모리는 실질적인 시스템에 장착되어 있는 실제 메모리이다. 물리적 메모리는 각각의 Frame형식으로 구성되어 있어 가상 메모리에서 Page Table이라는 변환 테이블을 이용하여 물리적 메모리 각각의 Frame에 Mapping 된다.

2. 가상 메모리(Virtual Memory)

가상 메모리는 각 프로세스가 사용할 수 있는 물리적 메모리의 크기보다 더 큰 메모리를 사용하기 위한 데이터 공간의 확장이다. 프로세스에서 가상 메모리는 4GBytes의 공간을 사용한다. 그 중 상위 3GBytes는 사용자 영역으로 수행 가능한 프로그램이 사용하도록 할당되고, 하위 1GBytes는 리눅스 커널이 사용하도록 할당된다.

메모리에서 사용되지 않는 부분은 스왑 영역으로 저장해서 메모리의 여유 공간을 확보하는 것이 스왑(Swap)의 역할이다.

III. 메모리 덤프(dump)

메모리 덤프 방식에는 크게 두 가지 방식이 존재한다. 사용자 모드(Application) 메모리 덤프 방식과 시스템 전체 메모리 덤프 방식이다. 사용자 모드 메모리 덤프는 현재 수행중인 프로세스의 메모리를 분석할 때 사용 하고, 물리적 메모리 덤프는 시스템이 크래시 되었거나, 사용자가 임의의 전체 메모리 분석이 필요 할 때 사용 한다.

1. 사용자 모드 메모리 덤프

사용자 모드 메모리 덤프(Dump)는 현재 수행중인 프로세스의 메모리를 덤프 받는 것이다.

사용자 모드 메모리 덤프를 위해서는 리눅스 메모리에 접근할 수 있는 특별한 이미지 파일 /proc/kcore를 덤프 하거나, ptrace와 PID를 이용한 메모리 프로그램을 제작하여 덤프 받는 방법이 있다. 덤프를 위하여 리눅스의 'dd' 명령을 이용하여 /proc/kcore 파일을 덤프 받는다. ptrace와 PID를 이용하여 제작한 프로그램을 이용하여 덤프 받을 경우 물리적 메모리의 내용이 아닌 현재 수행중인 프로그램의 어셈블리 코드를 덤프 받는다.

또한 가상 메모리 주소의 사용을 확인하기 위하여 /proc/<PID>/maps 파일을 참고하면 현재 수행중인 파일의 가상 메모리 주소 확인이 가능하고, 각각의 메모리 영역이 어떻게 사용되는지 확인이 가능하다.

/proc/kcore를 덤프 하는 경우 물리적 메모리가 1G 이상이 되는 경우에는 데이터가 최대 896M까지 덤프 되어진다. 이는 실제 메모리의 896M 이상 부분은 메모리 확장을 위하여 남겨놓은 부분으로 덤프가 되지 않는다.

2. 물리적 메모리 덤프

물리적 메모리 덤프(크래시 덤프)를 위해서는 /dev/mem, /dev/kmem 파일을 사용자 모드 메모리 덤프와 같이 'dd' 명령을 이용하여 덤프를 받거나, 리눅스 시스템이 크래시(crash)되었을 때 전체 메모리 덤프를 받을 수 있다.

사용자가 /dev/mem, /dev/kmem 파일을 직접 접근하여 덤프하면 시스템에 치명적인 영향이 있을 수 있으므로, 크래시 덤프 프로그램을 이용하여 전체 메모리 덤프를 받아야 한다.

물리적 메모리 덤프에서 크래시 덤프를 이용하면 메모리의 커널 영역, 사용자 영역 및 DMA(Direct Memory Access) 부분은 덤프 가능하지만 SWAP 영역은 덤프가 되지 않는다. SWAP영역은 LKCD에서는 메모리의 내용을 임시 저장하는 영역으로 사용된다.

리눅스는 다음과 같은 프로그램을 이용하여 크래시 덤프를 지원한다.

1) LKCD(Linux Kernel Crash Dump)

LKCD는 커널 2.4.x와 2.6.x 버전에 설치 할 수 있다. 실제 구축했던 환경은 Fedora core2와 커널 2.6.9 소스, lkcd-6.1.0_2.6.9.patch를 이용하여 커널 패치 적용 후, 커널 환경설정 중 Kernel hacking 부분에서 LKCD 관련된 부분을 수정하고 커널 컴파일 하여 덤프 받을 수 있는 환경을 구축한다.

덤프 받을 수 있는 환경구축이 끝나면 lkcdutil을 설치해야 한다. lkcdutil은 리눅스에서 크래시가 일어나거나, 사용자가 임의적으로 크래시 상황을 만들었을 때 자동적으로 메모리 덤프를 받아주게 하는 프로그램이다. lkcdutil은 리눅스 시스템에 크래시 상황이 일어났을 때 재 부팅 후 덤프 파일을 생성한다. 덤프 데이터를 효율적으로 분석할 수 있는 유틸리티가 포함되어 있다. LKCD는 libelf, ksymoops라는 라이브러리를 추가적으로 필요로 하게 된다. libelf는 ELF파일에 관련된 라이브러리를 추가적으로 제공하여 주고, ksymoops 라이브러리는 CPU 레지스터 내용과 같이 시스템 실패와 관련된 세부 내역과 코드 명령어와 스택 값을 커널 심볼로 변환해서 역 추적 정보를 제공하는 라이브러리이다.

2) Netdump (Network dump)

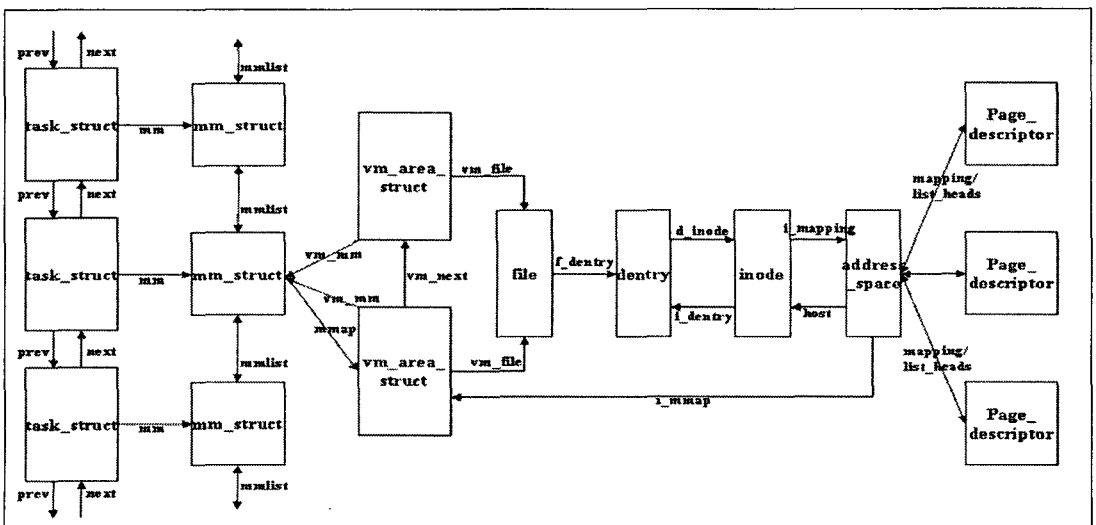
Netdump는 LKCD 유틸리티에 포함되어 있는 프로그램으로, Client-Server 방식으로 동작하는 네트워크 크래시 덤프 프로그램이다. 리눅스 크래시 상황이 일어났을 때 네트워크를 통해 서버에 덤프 파일을 생성한다.

Netdump를 사용하기 위해서는 서버의 저장할 수 있는 충분한 공간을 유지하고, 클라이언트에서 보내는 데이터를 저장하는 환경설정을 한다. 클라이언트에서는 크래시 상황이 일어났을 때 네트워크를 통하여 메모리 덤프데이터, 현재의 환경설정, 시스템 관련 정보 등을 서버에 보낼 수 있는 환경을 설정해야 한다.

3) DiskDump

DiskDump는 Netdump와 유사한 기능을 하지만 크래시 상황의 데이터 자신의 로컬 파일 시스템에 데이터를 저장한다. 커널 크래시가 발생 당시의 시스템 관련 정보와 메모리 덤프데이터, 현재 메모리 데이터를 디스크의 할당 영역에 저장을 한다. 시스템 재시작 후 diskdump init이라는 스크립트가 할당 영역에 저장된 정보를 토대로 덤프 파일을 생성한다.

4) mcore



<그림 1> Relations between data structures of each process

mcore는 시스템 패닉이 일어났을 때 크래시 덤프를 생성한다. 다른 덤프 방식과 다른 점이 있다면 mcore는 시스템의 크래시가 발생한 것이 아닌 시스템 패닉과 관련되어 덤프 파일을 생성한다. 시스템 패닉이 언제 발생되었고, 왜 발생되었는지에 관련된 정보를 덤프 받는다.

IV. 메모리 덤프 파일 분석

사용자 모드 메모리 덤프 받았거나 크래시 메모리 덤프를 받았을 경우 분석이 필요할 때 gdb, lcrash(Linux crash), crash 등의 유틸리티를 이용하여 덤프 데이터를 분석할 수 있다.

1. gdb

gdb는 기본적으로 프로그램 내부를 디버깅하는데 목적을 두고 있다. C, C++, Java등의 여러 언어로 작성한 프로그램을 디버깅하는데 이용된다.

gdb를 이용하여 덤프 데이터를 분석할 경우 리눅스의 커널 이미지와 덤프 이미지를 이용하여 분석한다. 덤프 당시의 수행 중이었던 프로세스의 분석, 스택 추적, 메모리 주소의 데이터 등을 분석할 수 있다.

2. lcrash, crash

lcrash와 crash는 분석하는 방법이나 분석되는 종류에서 거의 유사하기 때문에 여기서는 lcrash만 설명한다.

lcrash를 이용하여 분석할 경우 기본적인 프로세스의 분석, 스택 추적, 메모리 주소의 데이터 분석은 gdb와 유사하다. lcrash를 이용한 분석은 시스템의 기본적인 정보를 볼 수 있고, lcrash의 역 추적 명령은 크래시가 일어난 동안의 시스템 호출, 커널 스택, 크래시 종류, 시스템 log, 크래시를 일으킨 명령어를 보여줌으로서 덤프 데이터로부터 크래시를 일으킨 원인을 찾을 수 있도록 한다. lcrash는 가상주소에서 물리적주소로의 변환이 가능하다. 그리고 lcrash의 자체의 다양한 명령어를 이용하면 gdb의 메모리 주소를 통한 데이터 검색보다 손쉽게 더 많은 정보를 얻을 수 있다.

lcrash를 이용하여 분석하는 한 가지 예로 <그림1>을 볼 수 있다. <그림1>에서 각 프로세스는 task_struct로 표현이 된다. 이 그림은 각 프로세스의 데이터 구조체간 관계를 보여준다. <그림2>에서는 하나의 프로세스(bash)로부터 하위 file descriptor까지 추적하여 file descriptor의 내용을 확인한 것이다.

```
0xdb26da50      0   2768   2787      1     0x100   -  gnome-pty-helpe
0xdb9a1a50      0   2789   2787      1     0x100   -  bash
0xd3d34a50      0   3444   2787      1     0x100   -  bash
0xdb216a50      0   3464   2789      0     0x100   -  lcrash
0xc7a8fa50      0   3471      1      1     0x100   -  mozilla-bin
==========================================================================
71 active task structs found
>> print ((task_struct*)0xd3d34a50)->mm
(struct mm_struct * *) 0xd8d56d24
>> print ((mm_struct*)0xd8d56d24)->mmlist
struct list_head {
        next = 0xdc1b3d8c
        prev = 0xdb01cd8c
}

>> print ((mm_struct*)0xd8d56d24)->mmap
(struct vm_area_struct * *) 0xd36ea5b0
>> print ((vm_area_struct*)0xd36ea5b0)->vm_file
(struct file * *) 0xd8be1f54
>> print ((file*)0xd8be1f54)->f_dentry
(struct dentry * *) 0xdeae2f54
>> print ((dentry*)0xdeae2f54)->d_inode
(struct inode * *) 0xdeae3e44
>> print ((inode*)0xdeae3e44)->i_mapping
(struct address_space * *) 0xdeae3f24
>> print ((address_space*)0xdeae3f24)->private_list
struct list_head {
        next = 0xdeae3fa8
        prev = 0xdeae3fa8
}
>> dump 0xdeae3fa8 -B 100
0xdeae3fa8: a8 3f ae de a8 3f ae de 00 00 00 00 00 00 00 00 : .?...?..........
0xdeae3fb8: 00 00 00 00 bc 3f ae de bc 3f ae de 00 00 00 00 : .....?...?.....
0xdeae3fc8: 00 00 00 00 00 00 00 00 00 00 00 00 16 38 57 13 : .............8W
0xdeae3fd8: 00 00 00 00 00 00 00 00 00 00 00 00 7b f5 39 00 : ............{.9
0xdeae3fe8: 00 00 00 00 55 ff ff ff fc a7 72 df 00 00 00 00 : ....U.....r....
0xdeae3ff8: a5 c2 0f 17 42 a4 9b e0 00 00 00 00 00 00 00 00 : ....B...........
0xdeae4008: 00 00 00 00                                     : ....
>>
```

<그림 2> lcrash를 이용하여 분석

V. 결론

리눅스 메모리 데이터를 다양한 유틸리티를 이용하여 덤프 받는 방법과 덤프 된 메모리 데이터를 분석하는 방법에 관하여 논의하였다.

이러한 덤프데이터 분석은 다음과 같이 이용될 수 있다.

1. 리눅스 크래시 분석

리눅스 시스템에서 크래시 상황이 발생하였을 경우 왜 그러한 상황이 발생하게 되었는지 분석할 수 있다.

2. 사용자의 개인 정보 획득

사용자의 ID, Password등과 같은 정보는 저장장치에 저장이 되지 않고, 저장이 될지라도 암호화되기 때문에 메모리 분석을 통하여 획득할 수 있는 가능성이 존재한다.

3. 침해당한 시스템의 분석

해킹당한 시스템은 일반적으로 로그 파일이 지워진 경우가 많다. 로그 파일이 없는 경우 시스템에 발생한 문제점과 침입경로 확인이 힘들다. 메모리에 남아 있는 정보를 이용하여 크래커의 행동 양식이나, 소실된 데이터를 복구할 때 유용하게 사용된다.

4. 역 공학

기존에 만들어진 시스템을 역으로 추적하여 설계할 때의 문서나 설계기법 등의 자료를 메모리 분석을 통하여 얻어낼 수 있다.

VI. 향후 연구 과제

본 논문에서는 기존에 오픈소스로 제작되어 있는 메모리 덤프 유틸리티와 분석도구를 이용하여 메모리 덤프데이터를 분석하였다. 향후 메모리 덤프와 분석을 하는 도구를 자동화 된 툴로 제작을 하고, 리눅스 전체 메모리에 대한 Map을 만들어 분석할 것이다.

역 공학을 위하여 현재 수행중인 프로세스의 메모리 덤프를 분석하고, 재 사용이 가능하도록 알고리즘, 설계기법 추출을 위한 메모리 덤프 분석도구를 제작해야 할 것이다.

크래시 덤프 프로그램은 Swap영역은 Dump가 되지 않는다. Swap영역에 관하여 덤프 받을 수 있는 방법을 앞으로 연구해야 할 것이다.

[참고문헌]

[1] jasonuhl, LKCD(Linux Kernel Crash Dump), http://lkcd.sourceforge.net

[2] 한동훈, Linux Kernel Memory Model, http://network.hanbitbook.co.kr/view.php?bi_id=1113

[3] David Anderson, RedHat crash Utility, http://people.redhat.com/anderson

[4] indow, Diskdump, http://sourceforge.net/projects/lkdump/

[5] Dave Winchell, Memory Core Dump, http://oss.missioncriticallinux.com/projects/mcore

[6] GNU, gdb, http://www.gnu.org/software/gdb/gdb.html

[7] mariusz Burdach, Digital forensics of the physical memory, warsaw, march 2005.