리눅스에서 USB를 이용한 커널 하드닝에 관한 연구

장승주, 최은석
동의대학교 컴퓨터공학과
e-mail: sjjang@deu.ac.kr

A Study of Kernel Hardening using USB Device on Linux

Seung-Ju Jang, Eun-Seok Choi
Dept. of Computer Engineering, Dong-Eui University

요 약

본 논문은 적은 비용으로 시스템 정지 현상(PANIC)을 줄일 수 있는 Kernel Hardening 기법에 대해서 연구한다. 최근 USB의 사용이 증가함에 따라 USB의 사용에 의한 시스템 정지 현상이 자주 발생하고 있다. 본 논문에서는 이러한 컴퓨터 시스템의 정지 현상을 줄이고자 USB 디바이스를 사용하여 리눅스 커널에서의 Kernel Hardening 기법에 대해 연구한다. USB와 관련된 커널 모듈을 수정하고 수정된 모듈이 정상적으로 동작하는 지 테스트를 수행하여 정상적으로 동작함을 확인하는 실험을 수행한다.

1. 서론

컴퓨터 시스템을 중단 없이 정상적으로 동작 시키는 것은 중요한 문제 중의 하나이다. 이와 같이 컴퓨터 시스템이 중단 없이 동작하도록 하기 위하여 여러 가지 고장 감내 기법들이 개발 및 상용화되어 사용되고 있다. 대부분의 고장 감내 기법은 많은 경비가 소요된다. 본 논문은 적은 비용으로 시스템 정지 현상(PANIC)을 줄일 수 있는 Kernel Hardening 기법에 대해서 연구 한다.

운영체제에서 잘못된 원인으로 인해서 시스템이 정지되는 현상은 언제나 발생한다. 프로그래머의 실수나 관리자의 실수로 인하여 시스템이 정지 되는 현상이 발생하게 되는데 이 때, 시스템의 잦은 정지로 인하여 큰 피해를 받을 수 있다.

이런 시스템이 회사의 웹 서버, 데이터베이스 서버인 경우 서비스에 지장이 생길 수 있다. 이 때 사용자의 실수에 의해 시스템이 정지될 상황에 사용자의 실수로 시스템 정지를 유발 시키는 프로세스만 정지 시킬 수 있다면 시스템은 정지 되지 않고 정상적으로 작동할 수 있어 시스템이 보다 안정적으로 사용할 수 있을 것이다.

Kernel Hardening 운영체제에서는 현재 프로세스가 PANIC으로 들어갈 경우 시스템을 정지시키지 않고, 현재 프로세스를 검사하여 복구가 가능한지 복구가 불가능한지를 판단하여 복구가 불가능 하다고 판단하면 PANIC 상태가 되고, 복구가 가능하다고 판단되는 경우에 한하여 복구를 하여 시스템이 정지되지 않는다.

복구 가능성 여부를 판단할 경우 첫 번째로 value type인지 address type인지를 검사하도록 하여 type에 맞게 분기 하도록 한다. value type인 경우 데이터만 올바르게 복구하도록 하면 정상적으로 동작함으로 복구가 가능한 경우라 할 수 있다. 하지만 address type인 경우는 올바른 주소를 찾기 힘들다. 이는 복구 가능성이 보이지 않아 복구가 불가능하다고 볼 수 있다.

본 논문은 적은 비용으로 시스템 정지 현상(PANIC)을 줄일 수 있는 Kernel Hardening 기법에 대해서 연구한다. 최근 USB의 사용이 증가함에 따라 USB의 사용에 의한 시스템 정지 현상이 자주

본 논문은 2008년도 '중기청 산학컨소시엄과제'의 연구비 지원에 의하여 이루어졌음.

발생하고있다. 본 논문에서는 이러한 컴퓨터 시스템의 정지 현상을 줄이고자 USB 디바이스를 사용하여 리눅스 커널에서의 Kernel Hardening 기법에 대해 연구한다. USB와 관련된 커널 모듈을 수정하고 수정된 모듈이 정상적으로 동작하는 지 테스트를 수행하여 정상적으로 동작함을 확인하는 실험을 수행한다.

본 논문은 2장에서 DLM에 대해 설명하고, 3장에서 USB를 이용한 Kernel Hardening 기능을 설계한다. 4장에서 USB를 이용한 Kernel Hardening 기능을 간단하게 테스트 하고, 5장에서 결론을 맺는다.

2. DLM(Dynamic Linking Module)

커널 버전 1.2부터 등장한 리눅스(Linux®) 커널 동적 적재 모듈(DLM)은 리눅스 커널에서 가장 중요한 기술 혁신 중 하나이다. 동적 모듈은 커널을 동적으로 확장 가능하게 만든 기술이다.

리눅스 커널은 모놀리틱 커널로 알려진 구조를 따른다. 모놀리틱 커널은 운영체제 기능 대부분이 커널에 들어있으며 특권 모드로 돌아감을 의미한다. 모놀리틱 커널과는 달리 마이크로 커널은 IPC(Inter-Process Communication), 스케줄링, 기본 입출력, 메모리 관리와 같은 기본 기능만 제공하며 드라이버, 네트워크 스택, 파일 시스템과 같은 나머지 기능을 특권 공간 외부로 밀어내 프로그래머가 직접 관리하는 응용 프로그램을 만들어 주어야 한다. 리눅스는 무척 정적인 커널이라고 생각할지도 모르겠지만, 실제로는 정반대이다. 리눅스는 LKM (리눅스 커널 모듈) 활용을 통해 동적으로 변경이 가능하다. 이를 DLM(Dynamic Linking Module)이라고 한다.

동적으로 변경이 가능하다는 표현은 새로운 기능의 모듈을 커널에 추가/제거를 할 수 있다는 의미다. DML을 사용하면 커널의 크기를 줄인다는 장점이 있다. 필요한 구성 요소만 메모리에 올리면 되기 때문이다. 이는 임베디드 시스템에서 아주 중요한 기능이다.

리눅스가 동적으로 변경할 수 있는 유일한 모놀리틱 커널은 처음은 아니다. BSD(Berkeley Software Distribution), 썬 솔라리스, OpenVMS와 같은 옛날 커널, 기타 마이크로소프트 윈도(Microsoft® Windows®)나 애플 맥 OS X 같은 기타 유명한 운영체제에서도 동적 모듈을 지원한다.

3. 커널 모듈 설계

USB_ASSERT() 매크로를 이용하여 /.../linux-2.4/drivers/usb/usb-uhci-deu.c 소스코드에 panic이 발생할 수 있는 상황을 만들고 이를 복구할 수 있게 프로그램 하였다.

USB_ASSERT() 매크로는 3개의 인자값으로 구성되어 있다.

(표 1) USB_ASSERT() 매크로 소스 코드

```
/*** USB_ASSERT() TEST ***/
// expr1 :
// expr2 : value
// expr3 ; 1->address type or 2->value type
#define USB_ASSERT(expr1, expr2, expr3) ₩
  if(expr3 == 1) { ₩
    printk( "₩nerror USB_ASSERT3.₩n" __FILE__
":%d: Assertion " #expr1 " failed!₩n",__LINE__); ₩
  } ₩
/*  panic(#expr1); } */ ₩
  else if(expr3 == 2) { ₩
      printk( "₩nvalue type error!!!!!!!!"); ₩
      printk( "recoverable this variable₩n"); ₩
      expr1 = expr2; ₩
  } else { ₩
    printk( "₩naddress type error!!!!!!!!"); ₩
    printk( "unrecoverable this variable₩n"); ₩
  }
/*** USB_ASSERT() END ***/
```

*'expr1'*은 PANIC을 발생시킬 변수에 해당된다. *'expr1'* 변수가 가질 수 있는 변수Type은 value일 수도 있고, address일 수도 있다. PANIC을 유도하는 변수의 Type이 value일 경우 *'expr2'*에 들어가는 인자값 역시 변수Type이 value이나, address일 경우 정해진 변수Type이 없다. 이를 구분하는 인자가 *'expr3'*이다. *'expr3'*의 값에 의하여 변수의 Type이 value 인지 address 인지 구분된다.

위 프로그램의 구조는 변수의 Type을 구분하는 *'expr3'*의 값이 '1'이면 *'expr1'*의 변수의 Type이 address로 잘못된 *'expr1'*의 값이 들어 올 경우 복구가 불가능하다. 하지만 *'expr3'*의 값이 '2'이면 *'expr1'*의 변수 Type이 value로 잘못된 *'expr1'*의 값이 들어와도 바로잡아 줄 수 있는 *'expr2'*의 값이 있기 때문에 복구가 가능하여 PANIC이 발생하지 않

는 구조이다.

```
printk( "₩nerror USB_ASSERT3.₩n" __FILE__ ":%d:
Assertion " #expr1 " failed!₩n",__LINE__);
```

위 소스 코드는 오류가 발생하였을 경우 커널에 출력해주는 메시지이다. *'__FILE__'*은 현재 오류가 발생한 파일의 이름을 의미한다. *'__LINE__'*은 오류가 발생한 파일에서의 줄을 의미한다. 위의 소스코드는 현재 파일인 usb-uhci-deu.c 파일에의 printk() 함수가 들어 있는 줄의 라인번호를 출력해주어 오류가 발생하였을 때, 어떤 파일의 몇 번째 라인에서 오류가 발생하였는지를 쉽게 찾을 수 있도록 해준다.

USB_ASSERT() 매크로를 이용하여 usb-uhci-deu.c 파일에 inw() 함수의 오류를 방지하기 위하여 USB_ASSERT_INW() 매크로를 만들어 추가하였다.

USB_ASSERT_INW() 매크로 역시 USB_ASSERT() 매크로와 동일하게 3개의 인자값으로 구성되어 있다.

(표 2) USB_ASSERT_INW() 소스 코드

```
/*** USB_ASSERT_INW() ***/
// expr1 : variable
// expr2 : function INW() parameter
// expr3 : panic flag
#define USB_ASSERT_INW(expr1, expr2, expr3) ₩
        if(!expr1){ ₩
                expr1 = inw(expr2); ₩
                if(expr1 == NULL){ ₩
                        expr3 = 1; ₩
                }else{ ₩
                        expr3 = 0; ₩
                } ₩
        }
/*** USB_ASSERT_INW() END ***/
```

*'expr1'*은 확인하고자 하는 변수이고, *'expr2'*는 INW() 함수에 들어갈 인자이다. *'expr3'*는 *'expr1'*의 값이 정상적이면 '0'을 비정상적이면 '1'값을 가지게 되는 변수이다. 즉, *'expr3'*은 INW()함수의 결과 값이 비정상적이면 '1'값을 가지게 된다.

usb-uhci-deu.c 파일에 추가한 USB_ASSERT() 매크로가 정상적으로 동작하는지 테스트하기 위하여 USB를 컴퓨터 시스템에 꽂으면 동작하는 uhci_interrupt() 함수에 구현하였다.

(표 3) USB_ASSERT() 매크로 테스트 코드

```
_static void uhci_interrupt (int irq, void *__uhci,
struct pt_regs *regs)
{
        uhci_t *s = __uhci;
        unsigned int io_addr = s->io_addr;
        unsigned short status;
        struct list_head *p, *p2;
        int restarts, work_done;
int t;
//
//----- USB kernel hardening -------
//
//panic("11111111111111111111111111      usb_uhci
₩n");
/// USB_ASSERT TEST
info("PANIC TEST 10 ---------------!₩n");
//printk("s address : 0x%x", s);
//s = 0;
//USB_ASSERT(s, s, 1);
t = 0;
printk("before kernel hardening ----------------
t=%d₩n", t);
//info("------ t=%d₩n", t);
USB_ASSERT(t, t=5, 2);
//info("PANIC failed ---------------!₩n");
printk("after kernel recovery ----------------
t=%d₩n", t);
/// USB_ASSERT TEST END
}
```

(표 3)은 USB_ASSERT() 매크로의 테스트 코드이다. 오류를 잃을 킬 변수 *'t'*를 선언하고, '//---------- USB kernel hardening ----------'부분 아래에 사용된 USB_ASSERT() 매크로 함수에 중요한 변수로 사용된다.

변수 *'t'*에 '0'이라는 잘못된 값을 입력하고, USB_ASSERT() 매크로의 첫 번째 인자값으로 넣는다. USB_ASSERT() 매크로의 두 번째 인자값으로 정상적인 값인 '5'를 넣는다. 정상적으로 USB_ASSERT() 매크로가 동작하게 되면 USB_ASSERT() 매크로 이후에 *'t'*값은 '5'로 변경된다. 이를 확인하기 위하여 USB_ASSERT() 매크로 수행 전/후에 'printk()' 함수를 사용하여 *'t'* 변수의 값을 확인 할 수 있도록 하였다.

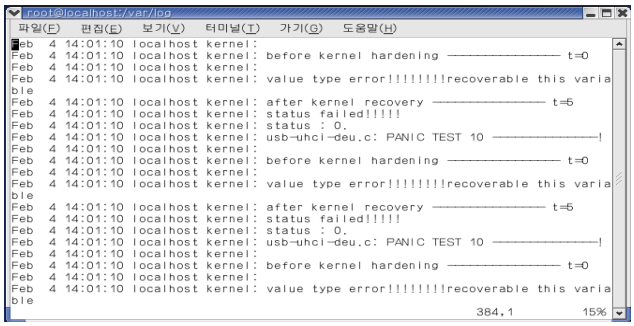
(표 4) USB_ASSERT_INW() 매크로를 사용한 코드

```
//printk("57@");
/*
 * Read the interrupt status, and write it back to
clear the
 * interrupt cause
 */
//          status = inw (io_addr + USBSTS);
int P_flag = 0;
USB_ASSERT_INW(status,    io_addr    +    USBSTS,
P_flag);
if (P_flag == 1){
          printk("status failed!!!!!₩n");
//          panic(status);
}
printk("status : %d.₩n", status);
```

USB_ASSERT_INW() 매크로의 인자값으로 *'status'*, *'io_addr + USBSTS'*, *'P_flag'*가 순서대로 들어간다. *'status'*는 첫 번째 인자값으로 INW() 함수의 결과가 반영되는 변수이다. *'P_flag'*는 INW() 함수의 결과에 따라 '0'과 '1'로 마크된다. *'P_flag'* 변수의 값에 의해 panic() 함수를 호출할 것인지가 결정된다.

4. 실험

리눅스 시스템에서 3장에서 설계한 프로그램을 커널 소스 코드에 적용시키고, 커널 빌드를 한다. 새로 만들어진 커널 이미지를 적용시켜 재부팅을 하여 수정된 프로그램이 반영되게 한다.



(그림 1) /var/log/messages 내용

(그림 1)은 USB를 리눅스 시스템에 연결한 후 커널 메시지가 기록되는 /var/log/messages 파일을 vi 편집기로 확인한 화면이다.

처음 '0'으로 설정되어 있던 *'t'*값이 '5'로 변경된 것을 'before kernel hardening ---------- t=0' 과 'after kernel recovery ---------- t=5'를 통하여 정상적으로 동작됨을 확인 할 수 있다.

3장에서 테스트를 위해 설계한 모듈이 정상적으로 동작되는 것을 확인할 수 있다.

5. 결론

본 논문은 적은 비용으로 시스템 정지 현상(PANIC)을 줄일 수 있는 Kernel Hardening 기법에 대해서 연구한다. 최근 USB의 사용이 증가함에 따라 USB의 사용에 의한 시스템 정지 현상이 자주 발생하고있다. 본 논문에서는 이러한 컴퓨터 시스템의 정지 현상을 줄이고자 USB 디바이스를 사용하여 리눅스 커널에서의 Kernel Hardening 기법에 대해 연구한다.

이러한 시스템의 정지 현상을 줄이기 위한 연구가 Kernel Hardening 기법이다. Kernel Hardening 기법을 적용하여 USB를 사용할 때, 좀 더 안정적으로 시스템을 활용할 수 있는 방법이 본 논문이다.

USB 디바이스에 대한 Kernel Hardening 기법을 적용을 설계를 하고, 설계한 내용을 바탕으로 간단한 테스트를 해보았다.

테스트 결과 잘 못된 값인 '0'으로 설정되어 있던 변수 *'t'*가 Kernel Hardening 이후 정상적인 값 '5'로 변경됨을 볼 수 있었다.

본 논문의 연구가 완료가 되면 안정적으로 USB를 활용한 프로그램을 개발 할 수 있게 된다.

참고문헌

[1] The Linux Online, http://www.linux.org
[2] Linux Kernel, http://faqs.org/docs/kernel_2_4/
[3] Uresh Vahalia, "UNIX Internals", 홍릉과학출판사, 2001
[4] Linux Kernel Programming, ADDION WESLEY, Beck, pp2~5, 2002
[5] Linux programming bible, 권수호, 글로벌, 2002
[6] Kernel Projects for Linux, Gary Nutt, Addison Wesley, 2001