# 리눅스 운영체제에서 DLM을 이용한 USB 디바이스 커널 하드닝에 관한 연구

## A Study of USB Device Kernel Hardening Using DLM, in Linux Operating System

장승주, 최은석
동의대학교 컴퓨터공학과

Jang Seung-Ju, Choi Eun-Seok
Dept. of Computer Engineering, Dong-eui Univ.

요약

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 리눅스 운영체제에서 동적 모듈(DLM)을 이용하여 USB 디바이스를 연결할 경우 USB 디바이스 사용에 대해 발생할 수 있는 커널 PANIC 현상을 줄이는 커널 하드닝 기법에 대해서 연구 한다.

Abstract

Computer system to operate normally without interruption, is one of the important issues. Likewise, a computer system to operate without interruption, failure to endure a variety of techniques, development and commercial use is arranged. Most guards will take a lot of technique failure endure. In this paper on the Linux operating system, dynamic module (DLM) to connect using the USB device to use USB devices can cause the symptoms to reduce the kernel PANIC hadeuning technique for studying the kernel.

## Ⅰ. 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 커널 하드닝 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.

커널 하드닝 운영체제에서는 현재 프로세스가 PANIC으로 들어갈 경우 시스템을 정지시키지 않고, 현재 프로세스를 검사하여 복구가 가능한지 복구가 불가능한지를 판단하여 복구가 불가능 하다고 판단하면 PANIC 상

태가 되고, 복구가 가능하다고 판단되는 경우에 한하여 복구를 하여 시스템이 정지되지 않는다.

## Ⅱ. 관련연구

### 1. Fault Tolerant(비상 안전 능력)

비상 안전 시스템(Fault Tolerant System)은 고장이나 프로그램에 버그가 있더라도 시스템 전체에 장애가 발생하지 않게 구성한 시스템을 의미하며. 예를 들면 항상 두 대의 시스템이 서로 감시하면서 동일 처리를 하다가, 한 대가 고장이 나면 다른 한 대가 처리를 계속할 수 있는 이중 시스템의 방식이 있다. 이때 동작중인 시스템을 Active 상태라 하고, 동일 동작을 수행하면서 대기중인 시스템을 Stanby 상태라 하며, Active 시스템이 문제가 발생하게 되면 자동 절제 되도록 설계되어져 있다. 실제로 On-line 기능이 중요하게 요구되는 교환기, 네트워크 시스템 등이 예라 할 수 있다.

Fault의 종류로는 기기에 대한 부분과 소프트웨어에 대한 부분으로 나눌 수 있다. 소프트웨어에 대한 Fault의 종류로는 먼저 무한루프(endless loop)가 있다. 무한루프는 프로그램이 어떤 처리 루틴을 반복 실행하여 그 부분을 벗어나지 못하는 상태를 말한다.

다음으로 교착상태(deadlock)이 있다. 교착상태는 다중 테스크 처리에 있어서 상대방이 점유하고 있는 자원을 요구하는 처리가 동시에 발생하게 되면 어떤 자원도 해제 되지 못하여 처리가 진행되지 않거나 지연되는 상태를 말한다.

### 2. DLM(Dynamic Linking Module)

리눅스 운영체제는 커널 버전 1.2부터 등장한 리눅스(Linux®) 커널 동적 적재 모듈(DLM)은 리눅스 커널에서 가장 중요한 기술 혁신 중 하나입니다. 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 DLM이라고 한다.

동적으로 변경이 가능하다는 표현은 새로운 기능의 모듈을 커널에 추가/제거를 할 수 있다는 의미다. DLM을 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. 마운트

윈도우즈 운영체제 환경에서는 윈도우즈에서 인식할 수 있는 파일시스템만 사용하기 때문에 마운트라는 절차 없이 연결된 드라이브의 특성에 따라 파일시스템을 고정하여 해당 데이터를 인식한다.

하지만 리눅스는 다양한 파일시스템을 인식할 수 있다. 어떤 장비의 파일시스템이 시스템에 연결되면 트리 구조로 이루어진 전체 파일시스템에서 지정된 디렉터리에 연결되어 하부 경로를 생성한다. 이렇게 데이터 입/출력을 위해 파일시스템을 특정 디렉터리에 붙여 인식하는 작업을 마운트라 한다.

각기 다른 세 개의 파일시스템이 존재할 경우 'home', 'usr' 디렉터리 아래에 각기 다른 두 개의 파일시스템을 마운트하여 연결할 수 있다.

리눅스에서는 이와 같은 마운트라는 개념으로 인하여 운영체제가 동작되는 중에 다양한 파일시스템을 수시로 연결하고 해제할 수 있다. 또, 파일시스템을 지정하여 마운트 함으로 인해 다양한 파일시스템을 사용할 수 있다.

다양한 파일시스템을 사용할 수 있는 장점이 있는 반면 블록 장치를 연결하고자 할 때, 어떤 파일시스템을 사용하는지 어느 디렉터리에 붙여 사용할 것인지에 대

해서 지정한 후 마운트 해야 하므로 번거롭게 느껴질 수 있다.

## Ⅲ. 커널 하드닝 기법 설계

리눅스 시스템에서 커널에 로드 될 수 있는 모듈에 커널 하드닝 기법 코드를 추가한다. 모듈이 커널에 로드된 상태에서 USB 디바이스를 연결하면 커널 하드닝 기법이 동작하여 시스템의 안전을 지키게 설계한다.

리눅스 커널에서 USB 디바이스를 연결하면 동작하는 모듈을 찾아 커널 하드닝 기법 코드를 추가 한다.

커널 하드닝 기법으로 추가할 USB_ASSERT() 매크로는 인자값으로 들어오는 'expr3' 의 값에 의해서 한번 분기를 하게 된다. 분기된 각각은 서로 다른 동작을 하고 마지막엔 USB_ASSERT() 매크로가 종료된다.

USB_ASSERT() 매크로를 이용하여 /.../linux-2.4/drivers/usb/usb-uhci.c 소스코드와 동일한 usb-uhci-deu.c라는 소스코드를 만든다. 여기에 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 ***/
#define USB_ASSERT(expr1, expr2, expr3) ₩
  if(expr3 == 1) { ₩
    printk( "₩nerror USB_ASSERT3.₩n"
            __FILE__ ":%d: Assertion "
            #expr1 " failed!₩n",__LINE__); ₩
  } ₩
  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 ***/
```

표 1.은 USB_ASSERT() 매크로의 소스 코드 이다. '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 않는 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 들어있는 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.

**표 2. USB_ASSERT_INW() 소스 코드**

```
#define USB_ASSERT_INW(expr1,
                       expr2, expr3) ₩
        if(!expr1){ ₩
                expr1 = inw(expr2); ₩
                if(expr1 == NULL){ ₩
                        expr3 = 1; ₩
                }else{ ₩
                        expr3 = 0; ₩
                } ₩
        }
```

표 2.는 USB_ASSERT_INW() 매크로의 소스 코드 이다. USB_ASSERT_INW() 매크로에서 '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 커널 하디능 기법이 구현된 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)
{
... 중략 ...
//------- USB kernel hardening ----------
/// USB_ASSERT TEST
info("PANIC TEST 10 ---------------!₩n");
t = 0;
printk("before kernel hardening ------ t=%d₩n", t);
USB_ASSERT(t, t=5, 2);
printk("after kernel recovery ------ t=%d₩n", t);
/// USB_ASSERT TEST END
... 중략 ...
}
```

표 3.은 USB_ASSERT() 매크로를 테스트하기 위한 코드를 보여 준다.

오류를 잃을 킬 변수 't' 를 선언하고, '//------- USB kernel hardening ----------' 부분 아래에 사용된 USB_ASSERT() 매크로 함수에 중요한 변수로 사용된다.

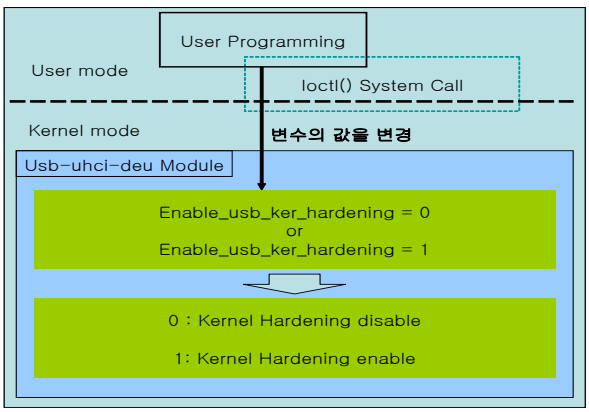
변수 't' 에 '0' 이라는 잘못된 값을 입력하고, USB_ASSERT() 매크로의 첫 번째 인자값으로 넣는다. USB_ASSERT() 매크로의 두 번째 인자값으로 정상적인 값인 '5' 를 넣는다. 정상적으로 USB_ASSERT() 매크로가 동작하게 되면 USB_ASSERT() 매크로 이후에 't' 값은 '5' 로 변경된다. 이를 확인하기 위하여 USB_ASSERT() 매크로 수행 전/후에 'printk()' 함수를 사용하여 't' 변수의 값을 확인 할 수 있도록 하였다.

**표 4. USB_ASSERT_INW() 매크로를 사용한 코드**

```
int P_flag = 0;
USB_ASSERT_INW(status, io_addr + USBSTS, P_flag);
if (P_flag == 1){
        printk("status failed!!!!!₩n");
}
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.

## IV. USB 디바이스를 이용한 테스트

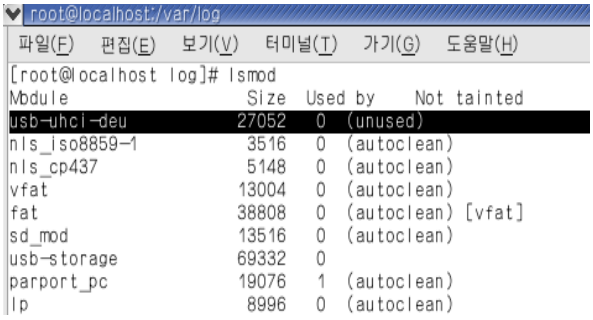



▶▶ 그림 1. ioctl() 시스템 콜을 사용한 변수 변경

그림 1.은 ioctl() 시스템 콜로 값을 전달하는 개념도이다. 사용자 프로그램인 응용프로그램에서 ioctl()을 통하여 커널로 진입하고, 'usb-uhci-deu' 모듈의 변수인 'enable_usb_ker_hardeing' 의 값을 '0' 또는 '1' 로 변경을 할 수 있다.

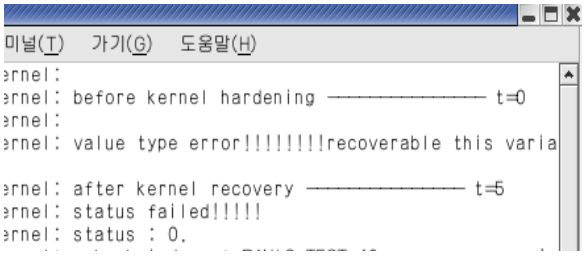
'enable_usb_ker_hardeing' 의 값이 '0' 인 경우 로드된 'usb-uhci-deu' 모듈에서 커널 하드닝 기법을 사용하지 않고 기존에 사용하던 기능들이 동작하도록 프로그램 되어있다.

'enable_usb_ker_hardeing' 의 값이 '1' 인 경우에는 로드된 'usb-uhci-deu' 모듈에서 커널 하드닝 기법을 사용하도록 프로그램 되어있다.



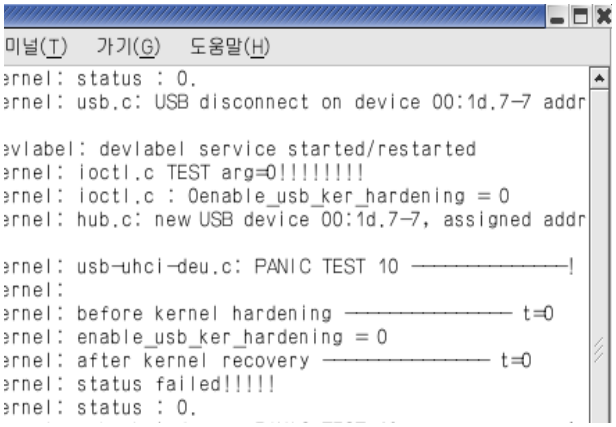

▶▶ 그림 2. 새로 만든 'usb-uhci-deu' 모듈이 로드되어 있는 화면

그림 2.는 리눅스 시스템의 기존 'usb-uhci' 모듈을 커널에서 제거 하고, 커널 하드닝 기법을 추가하여 새로 만든 'usb-uhci-deu' 모듈을 커널에 로드한 화면이다. 'usb-uhci-deu' 모듈은 커널 하드닝 기법이 추가된 모듈로 'usb-uhci' 모듈의 기능을 그대로 가지고 있다. 그렇기 때문에 새로 만든 'usb-uhci-deu' 모듈을 커널에 로드 하더라도 기존에 사용하던 USB 디바이스 접근 방법 등이 변하지 않는다.




▶▶ 그림 3. /var/log/messages 파일 내용(커널 하드닝 적용)

그림 3.은 테스트한 내용이 커널 메시지 형태로 저장된 messages 파일의 내용이다. 앞서 작성한 소스 코드들은 현재 변화되고 있는 상태나 상황을 커널 메시지로 출력할 수 있게 프로그램 하였다. '/var/log/messages' 파일에서는 커널 메시지로 출력되는 출력문들을 시간 별로 기록을 하여 사용자가 커널의 장애사항이나 시스템의 작동 상황을 알 수 있게 해준다. 출력된 메시지에서 't' 값이 '0' 에서 '5' 로 변화된 것을 볼 수 있다.




▶▶ 그림 4. /var/log/message 파일 내용 (커널 하드닝 비 적용)

그림 4.는 커널 하드닝 기법을 사용하지 않도록 설정을 하였을 때의 커널 메시지를 보여준다. 출력된 메시지에서 't' 값이 '0' 으로 변하지 않는 것을 볼 수 있다.

## V. 결론

컴퓨터 시스템을 중단 없이 동작시키는 비상 안전 기법중 하나인 커널 하드닝은 적은 비용으로 구현이 가능하다. 특히 리눅스 운영체제의 경우 커널 소스코드가 오픈되어 있어 커널 단계에서의 커널 하드닝 기법의 적용이 가능하다.

현대 사회에서 많이 사용되고 있는 USB 디바이스에 대해 커널 하드닝 기법을 적용한다. 리눅스 커널의 경우 외부의 모듈을 커널이 동작하는 도중에 로드 할 수 있는 커널 확장 기능의 한 가지인 DLM을 지원한다.

DLM을 이용하여 커널의 기능을 확장하고, 확장되는 기능에 커널 하드닝 기법을 추가한다.

커널의 동작을 보여 주는 커널 메시지를 이용하여 USB 디바이스에 대한 커널 하드닝 기법을 테스트해 보았다.

인위적인 오류를 정상적으로 복구하여 시스템에 영향을 주지 않고 정상적으로 시스템이 동작함을 볼 수 있다.

## ■ 참 고 문 헌 ■

[1] 장승주, “리눅스 운영체제에서 주소값 오류시 스텍 복구를 통한 커널 하드닝 기능 구현”, 한국해양정보통신학회, pp.173-180, 2007

[2] 박승규, “레드햇 리눅스 9”, 한빛미디어, 2003

[3] Peter Jay Salzman, “Linux Kernel Module Programming Guide”, 2005-05-26

[4] M. Tim Jones, “Access the Linux Kernel using the /proc filesystem”, Emulex Corp., 2008-06-17

[5] 권용덕, 박두성, “Linux 현장 실무자가 공개하는 리눅스 서버 관리”, 이비커뮤니케이션(주), 2002

[6] 서자룡, “서자룡의 Linux 리눅스 9 Plus 그대로 따라하기”, 혜지원, 2005

[7] M. Tim Jones, “리눅스 시스템 호출을 활용한 커널 명령”, Emulex Corp., 2008-06-17