Linux 운영체제에서 Kernel Hardening 설계 및 구현

문지훈*,김기환*,장승주*,정성인**
*동의대학교 컴퓨터공학과
**한국 전자 통신 연구원

e-mail : ki-hany@hanmail.net
{mjh,sjjang}@dongeui.ac.kr, sijung@etri.re.kr

Design and Implementation of the Kernel Hardening in the Linux Operating System

Ji-Hoon Moon*, Ki-Hwan Kim*, Seung-Ju Jang*, Seung-In Jung**
*Dept. of Computer Engineering, Dong-Eui University
**ETRI

요 약

본 논문에서는 Linux 운영체제에서의 kernel hardening 을 설계 및 구현한다. 커널 내에서 panic 이 발생할 경우 복구가 가능한 경우에는 정상적인 동작이 될 수 있도록 한다. 이렇게 함으로써 Linux Kernel Hardening 기능은 안정적인 커널의 동작을 보장한다. 본 논문에서 Linux Kernel Hardening 을 보장하기 위하여 커널 내 ASSERT() 함수를 중심으로 설계 및 구현을 한다.

1. 서론

현재 리눅스 운영체제의 사용이 증가되어 가고 있으며 회사의 웹 서버, 파일 서버, 데이터베이스 서버 등으로 활용되고 있다[1,5,7]. 리눅스 운영체제는 리눅스 커널 소스 코드를 수정하여 파일시스템, 디바이스 드라이버 등을 커널에 추가할 수 있다[6,8,9]. 운영체제 커널 소스를 수정하거나 커널 모듈 프로그램을 이용할 경우 커널 프로그램을 잘못 작성한 경우 시스템에 큰 영향을 미칠 수 있다. 시스템의 영향으로 인하여 시스템이 다운되는 현상이 발생하기도 하고 심한 경우는 운영체제를 다시 설치해야 하는 경우도 발생하게 된다. 만약 커널 프로그램 소스 코드를 잘못 작성하여 운영체제가 정지되는 현상을 막아준다면 큰 재난을 막을 수 있을 것이다.

리눅스 운영체제에서 커널 하드닝의 기본 기능은 현재 프로세스를 kill 시킬 경우 시스템이 정상적으로 동작할 수 있다는 것이다[4]. 하지만 모든 프로세스를 kill 시킬 경우 시스템에 문제가 발생할 수 있다. 이를 위해서 현재 프로세스에 대해서 복구가 가능한지 복구가 불가능한지를 판단하여 복구가 가능할 경우에 대해서만 시스템이 정상적으로 동작할 수 있도록 하고, 만약 복구가 불가능하다고 판단된 경우 시스템이 정지 될 수 있도록 해준다.

본 논문에서는 리눅스 운영체제에서 커널 하드닝 기법을 이용하여 현재 프로세스가 panic 으로 들어갈려고 할 경우 현재의 프로세스가 복구 가능한지를 판단하여 복구가 가능한 경우 복구하여 시스템이 정지되지 않도록 설계 하였다.

논문의 구성은 2 장에서 관련 연구를 살펴보고, 3 장에서 커널 하드닝 설계 , 4 장에서 커널 하드닝 구현을 마지막으로 5 장에서 결론에 대해서 논의한다

2. 관련연구

리눅스 운영체제에서 커널 하드닝에 관한 연구는 몬타비스타 회사에 의해서 연구되고 있으며, 몬타비스타는 커널 하드닝 기능이 내장되어 있는 리눅스 운영체

본 논문은 2003 년도 한국전자통신연구원 연구비 지원에 의하여 수행되었음.

제인 CGE 버전을 상업적으로 판매하고 있다. 몬타비스타의 CGE 버전은 커널 하드닝 기능을 3 가지 영역으로 분류하고 있으며 [표 1]은 CGE 버전의 커널 하드닝 기능을 나타낸다[2].

[표 1] CGE 버전의 커널 하드닝 기능

code reviews	high quality 소프트웨어 공학 프로세스의 standard part(코드 재검토)
panic removal	standard linux code 를 검사한 후 시스템을 중지 시킬 것인지 아니면 process 를 kill 시킬 것인지를 결정
fault injection testing	software fault 인 경우 error 에 대해서 kernel 이 복구할 수 있는 능력이 있는지 없는가에 대해서 검사한다.

위와 같이 몬타비스타의 커널 하드닝 기능은 code reviews 를 통해서 코드를 재검토한 후 특정 프로세스가 panic 루틴으로 들어왔을 때 standard linux code 의 검사를 통해 panic 루틴으로 들어온 프로세스를 kill 하여 시스템이 정상적으로 수행 되도록 하는 것이 아니라 현재 프로세스가 시스템에 영향을 주는 프로세스인 경우에 한해서 kill 하도록 한다.

몬타비스타의 CGE 버전은 리눅스 커널 2.4.17 을 기반으로 하고 있으며 커널 하드닝은 High Availability 에 기반을 두고 있다. 임의의 fault 설정에 따른 panic 에 적절히 대처할 수 있는 기법으로 code path 시험을 통한, 즉 에러 코드에 대한 모순되지 않은 과정을 피해보고자 시도해보는 과정이라고 할 수 있다. 이를 fault injection 이라고 하는데, 이러한 실험적 방법을 통하여 구현하게 되는 커널 리소스를 통해 보다 안정된 코드를 생성한다. high availability 는 다중 프로세스에서 프로세스의 기능성을 분배함으로써 컴퓨터 시스템의 up-time 을 강화하는 기술을 말한다[2,3].

3. 설계

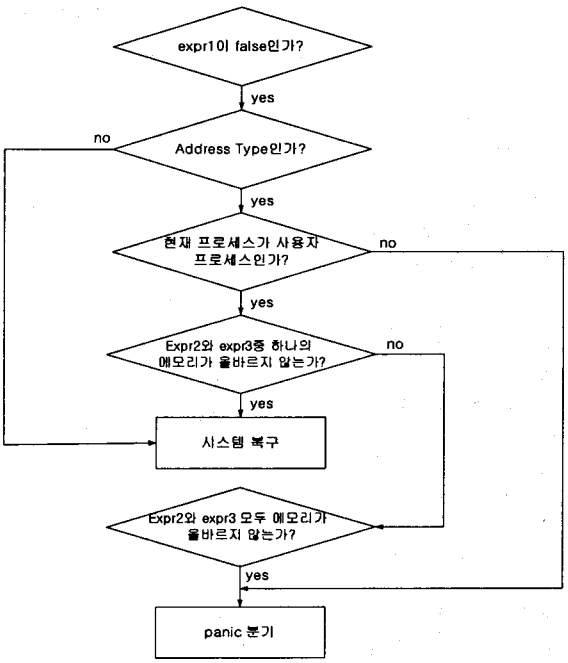
3.1 커널 하드닝의 기능

커널 하드닝 기능은 시스템에 문제가 발생하여 시스템이 정지 되어야 할 경우 현재 수행중인 프로세스를 검사하여 복구가 가능한 프로세스인지를 검사하여 복구가 가능한 프로세스인지 판단하게 된다. 복구가 가능하다고 판단된 프로세스인 경우 현재 프로세스를 kill 하여 시스템이 정상적으로 동작할 수 있도록 하고 복구가 불가능하다고 판단된 프로세스인 경우 kill 하지 않고 panic 루틴으로 들어가도록 해준다. 현재 프로세스만 kill 하여 시스템이 복구될 수 있다면 많은 이점을 가지고 있다.

3.2 커널 하드닝 설계

커널 하드닝에서는 ASSERT 함수를 이용하여 panic 으로 보내기 전에 복구가 가능한지를 판단하여 만약 복구가 가능하다고 판단된 경우에 복구하도록 하며 복구 불가능한 경우에 복구 하지 않고 바로 panic 으로 보내게 된다. 복구 하는 경우와 복구하지 않는 경우의 판단은 ASSERT 매크로 함수에 의해서 판단되는데 ASSERT(expr1,expr2,expr3,expr4)에서 expr1 의 표현식이 false 인 경우 리눅스 운영체제에서는 panic 루틴을 수행하도록 되어 있는데 panic 으로 바로 보내지 않고, 복구할 수 있는지를 검사하여 복구할 수 있는 경우인지를 검사한다.

expr1 의 type 에는 address type 가 value type 이 있는데, value type 인 경우라면 메모리 주소값은 정상이지만, 메모리 주소의 데이터의 값이 일치하지 않는 경우이므로 메모리 주소의 데이터값만 올바르게 수정하여 주면 된다. address type 인 경우에는 복구할 수 있는 경우와 복구할 수 없는 경우가 존재하는데 address type 의 복구 판단하는 과정은 다음과 같다. expr2 는 expr1 의 표현식의 왼쪽값을 가지고 있으며, expr3 는 expr1 의 표현식의 오른쪽 값을 가지고 있다. expr2,expr3 가 정상적인 메모리인지를 검사하여 올바른 메모리인 경우라면 무조건 복구하도록 해주며, 만약 한 개의 표현식만 잘못된 메모리인 경우라면 복구를 시도하도록 한다. 현재 프로세스가 사용자 프로세스인지, 시스템 프로세스인지를 검사하여 사용자 프로세스인 경우라면 복구하도록 해 주지만 시스템 프로세스인 경우라면 복구할 경우 시스템에 더 큰 영향을 미치므로 복구하지 않고 panic 루틴을 수행하도록 한다. expr2 이 잘못된 경우라면 expr3 값을 expr2 에 대입하고, expr3 가 잘못된 경우라면 expr2 값을 expr3 에 대입하도록 하여 복구할 수 있도록 해준다. [그림 1]은 ASSERT 함수를 이용한 복구 과정을 나타낸다.



[그림 1] ASSERT 함수를 이용한 복구 과정

4. 구현

4.1 개발 환경

리눅스 운영체제에서 커널 하드닝 기능을 위한 개발 환경으로 Intel CPU 450MHz 프로세스와 RAM 128Mbyte 을 사용 하였으며, 메인보드 캐시는 256KB 인 시스템에서 리눅스 ReadHat 9.0 기반의 운영체제를 사용하였다. 리눅스 커널 버전은 2.4.20 을 사용하였다. 개발을 위해서 GNU Tool 을 사용하였다.

4.2 복구 판단 과정

시스템에 오류가 발생하여 panic 루틴으로 바로 들어가지 않고 복구할 수 있는지를 검사하여 복구할 수 있는 경우에 시스템은 보다 안정적으로 동작 할 수 있을 것이다.

메모리 type 은 address type 가 value type 이 있는데, value type 인 경우 메모리 주소는 정상이지만, 데이터만 정확하지 않는 경우이므로, 데이터 값만 올바르게 변경하게 되면 시스템은 정상적으로 동작할 수 있을 것이다. address type 인 경우라면 메모리 주소가 일치하지 않거나 존재하지 않는 경우이므로, 복구할 수 있는 경우도 있고, 복구할 수 없는 경우도 존재한다. 현재 프로세스가 사용자 프로세스인지 시스템 프로세스인지를 검사하여 사용자 프로세스인 경우에는 복구할 있도록 하며, 만약 시스템 프로세스인 경우라면 panic 루틴을 수행하도록 해준다.

expr 의 수식의 왼쪽 수식과, 오른쪽 수식의 메모리가 올바른지를 판단하여 모두 올바른 경우라면 무조건 복구할 수 있도록 하며, 만약 하나의 메모리만 존재하지 않는 경우라면 복구할 수 있도록 한다.

4.2 ASSERT 함수의 수정

리눅스 운영체제에서 오류로 인하여 시스템이 정지되기 전에 ASSERT 함수에서 판단하여 false 인 경우라면 panic 루틴을 수행한 후 die 함수를 수행한 후 시스템이 정지되게 된다. [그림 2]는 일반적인 ASSERT 함수를 나타낸다.

```
#define ASSERT(expr) do {₩
        if(expr) {₩
            // 정상적인 경우
        }else{₩
            // 비정상적인 경우
        }}while(0)
```

[그림 2] 일반적인 ASSERT 함수

[그림 2]와 같이 expr 인자값을 판단하여 false 인 경우에 무조건 panic 루틴을 수행하도록 되어 있다. 하지만 현재 프로세스가 사용자 프로세인 경우 현재 프로세스만 kill 한다면 시스템은 정상적으로 동작할 수 있을 것이다. 원래의 ASSERT 함수를 수정하여 커널 하드닝 기능을 구현하였다. [그림 3]은 변경된 ASSERT 함수를 나타낸다.

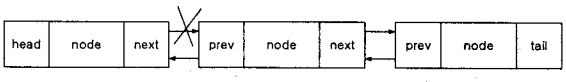
```
#define ASSERT(expr1,expr2, expr3,expr4) do {₩
        if(expr4 == 2) {₩ // Address Type 인 경우
                if(!expr1) {₩
                        if(panic_or_not(expr2, expr3) == 1){₩
                                if(사용자 프로세스인가?){₩
                                        //시스템 복구₩
                                }else{₩
                                        //panic 수행 ₩
                                }₩
                        }₩
                //panic 수행₩
                }₩
        }₩
        else if(expr4 == 1) {₩ // Value Type
                        if(!expr1) {₩
                                //시스템 복구₩
                        }₩
        }₩
        }while(0)
```

[그림 3] ASSERT 함수의 수정

[그림 3]은 ASSERT 함수에 커널 하드닝 기능을 추가한 부분을 나타내며 , 기존의 ASSERT 함수의 인자값이 하나에서 4 개로 추가 되었다. expr1 은 기존 ASSERT 함수의 expr 이며, expr2 는 expr 표현식의 왼쪽 수식을 나타내며, expr3 은 expr 표현식의 오른쪽 수식을 나타낸다. 소스에서 panic_or_not 함수는 정상적인 메모리 주소인지 비정상적인 메모리 주소인지를 판별하는 함수로써 둘중 하나의 메모리 주소가 올바르지 않는 경우 1 이 리턴되고, 모두 정상적인 메모리 주소라면 0 이 리턴된다. expr4 는 메모리 type 을 나타나는 것으로써, 1 인 경우 value type 을 나타내며, 2 인 경우 address type 의 메모리를 나타낸다.

4.3 특정 노드가 끊어진 double linked list 의 복구

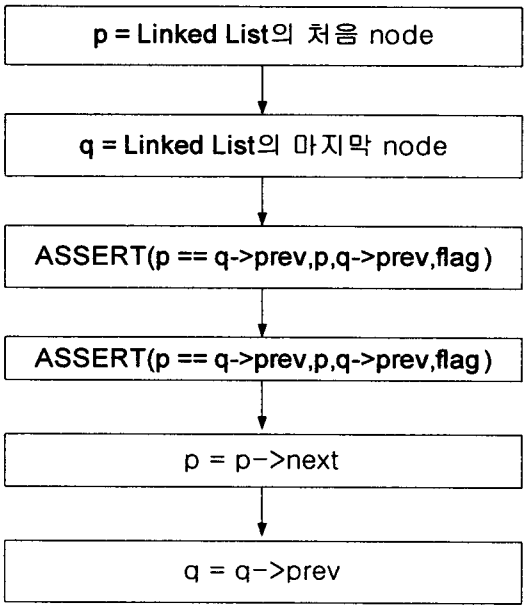
리눅스 커널에서 커널 하드닝을 구현하기 위해서 리눅스 커널에 double linked list 를 구현한 후 특정 링크를 끊어 다음 노드로 이동하지 못하게 하여 panic 루틴을 강제적으로 수행하도록 하였다. 이렇게 함으로써 본 논문에서 제안하는 기능을 쉽게 실험 할 수 있다. [그림 4]는 double linked list 를 나타낸다.



[그림 4] double linked list

[그림 4]는 이중 링크드 리스트를 나타내며, 현재 노드에서 다음 노드로 이동하지 못하도록 링크를 끊을 경우 커널에서 링크드 리스트를 작성한 경우 panic 루틴을 수행한 후 시스템이 정지 될 것이다. 커널 하드닝에서 현재 노드의 next 의 주소값은 다음 노드의 주소값과 일치하여야 하며 다음 노드의 prev 는 현재 노

드의 주소값과 일치하여야 한다. 만약 이 조건을 만족하지 못할 경우 커널 하드닝 기능이 없는 리눅스 운영체제인 경우 panic 루틴이 실행되어 시스템이 정지되는 현상이 발생할 것이다. [그림 5]은 double linked list 에서 링크가 끊어진 경우 커널 하드닝 기능을 이용한 복구 방법을 나타낸다.



[그림 5] double linked list 에서 끊어진 링크 복구

[그림 5]은 double linked list 에서 노드가 끊어졌을 경우 ASSERT 함수를 통해서 시스템을 복구하는 기능을 나타낸 것이다.

p 는 링크드 리스트의 처음을 나타내며, q 는 링크드 리스트의 마지막을 나타낸다. p->next 필드는 다음 노드의 address 와 같아야 정상적인 링크드 리스트의 구성이 되며, 현재 노드의 주소와 다음 노드의 address 와 다음 노드의 prev 필드의 값이 일치해야 정상적으로 동작하는 링크드 리스트이다. ASSERT 함수에서는 이 링크드 리스트의 next 필드와 다음 노드의 address 가 일치하는지를 검사하여 같지 않을 경우 현재 프로세스를 kill 하여 시스템이 정상적으로 동작하도록 하며, 현재 노드의 다음 노드의 address 와 prev 필드의 값이 일치하지 않은 경우 ASSERT 함수를 통해서 현재 프로세스를 kill 하여 시스템이 정상적으로 동작할 수 있도록 한다.

만약 커널 하드닝 기능이 없을 경우 링크드 리스트의 2 번째 노드의 next 의 값을 잘못된 주소 값을 넣었을 경우 사용자 프로그램에서는 segmentation fault 를 커널 프로그래밍에서는 ASSERT 함수에서 false 값이 리턴되므로 panic 루틴을 수행한 후 , die 함수를 수행한 후 시스템이 비정상적으로 종료되게 될 것이다 [10, 11].

5. 결론

본 논문에서는 리눅스 운영체제에서 잘못된 결과로 인하여 panic 루틴을 수행하기 전에 복구할 수 있는 경우 복구하여 시스템이 정상적으로 동작할 수 있도록 커널 하드닝을 설계 및 구현 하였다. 커널 하드닝은 ASSERT 함수를 이용하여 복구할 수 있는지 없는지를 판단하고 address type 인지 value type 인가를 검사하여 value type 인 경우는 무조건 복구 하도록 하였고, address type 인 조건에 의하여 복구할 수 있는 경우와 그렇지 않는 경우로 나누어서 복구할 수 있는 경우는 현재 프로세스를 kill 하여 시스템이 정상적으로 동작할 수 있도록 하였고, 복구할 수 없는 경우는 현재 프로세스가 panic 루틴으로 수행하도록 하여 시스템이 정지되도록 하였다.

참고문헌

[1] Barrie Sosinsky and Carol Tanielu, Mastering Solaris 8, SYBEX pp6 ~ 9 , 2001

[2] http://www.mvista.com/cge/index.html , 2002

[3] Montavista™ Linux® Carrier Grade Edition[WHITE PAPER] ,Montavista Software Inc. John Mehaffey, April 8, 2002

[4] SEQUOIA "kernel Hardening Guidelines", Tim Udalll, 1994

[5] Linux Kernel Programming , ADDISON WESLEY, Beck , pp2 ~ 5 , 2002

[6] Linux Kernel Internals, Michael Beck,Mirko Dziadzka, Ulrich Kunitz and Harald Bohme,Addison-Wesley,1997

[7] Operating System Concepts(6th), SILBERSCHATZ&GALVIN&GAGNE, JOHNWILEY&SONGS INC. , 2002

[8] Linux programming bible, 권수호, 글로벌, pp20 ~ 22, 2002

[9] Kernel Projects for Linux,Gary Nutt, Addison Wesley Longman,2001

[10] Linux Device Driver(2nd), A.Rubini&J.Corbet, O'Relly , 2001

[11] Understanding the Linux Kernel, BOVET & CESATI, OREILLY, p216~p222, "Page Fault Exception Handler", 2001