# 커널 기반 Call gate 오용 탐지에 관한 연구

유동훈*, 김민수**, 김동국***, 노봉남****
*전남대학교 정보보호협동과정
**목포대학교 정보보호학과
***전남대학교 전자컴퓨터공학부
****전남대학교 시스템보안연구센터
e-mail: szoahc@hotmail.com

# A Study on Detecting Kernel Based Call Gate Abuse

Dong-Hoon You*, Min-soo Kim**, Dong-Kook Kim***, Bong-Nam Noh****
*Interdisciplinary Program of Information Security, Chonnam National University, Gwangju, Republic of Korea
**Dept. of Information Security, Mokpo National Univ
***Dept. of Electronics and Computer Engineering, Chonnam National University, Gwangju, Republic of Korea
****System Security Research Center, Chonnam National University, Gwangju, Republic of Korea

요 약

Call gate 오용으로 인한 커널 공격 취약점은 커널 보안 문제점 중 하나로서 시스템 관리자들을 위협하고 있다. 이로 인해 근본적으로 커널 공격을 방어할 수 있는 대책이 시급하나 아직까지 효과적으로 Call gate 오용을 탐지할 수 있는 방법은 알려진 바가 없다. 본 논문에서는 적재가능커널모듈(loadable kernel module)을 이용하여 Call gate 오용을 통한 커널 공격을 탐지할 수 있는 방법을 기술하고자 한다.

## 1. 서론

오래 전부터 하드웨어 설계자와 운영체제 설계자들은 32비트 보호모드가 설계되기까지 운영체제 커널(kernel)을 보호하기 위해 많은 노력을 기울여왔다. 그 결과 사용자 영역과 커널 영역을 나누기 위한 세그먼테이션 메커니즘, 특권레벨(privilege level) 개념과 같은 보호 관련 메커니즘이 등장하게 되었다. 이러한 기술의 등장으로 시스템 코드가 위치한 메모리에 침범이 발생할 확률은 현저히 낮아졌으며 악의적인 사용자 어플리케이션이 커널을 망가뜨리는 일이 없도록 보다 더 안전해진 시스템이 설계되었다.

하지만 권한 상승과 관련된 몇 가지 기능으로 인해 커널은 여전히 취약점을 내포하고 있으며 그 문제점 중 하나가 본 논문에서 설명하는 Call gate 기능을 악용한 커널 공격 기법이다. 이러한 Call gate 기능을 악용한 초기 커널 공격 기법은 디스크립터 테이블에 시스템 세그먼트를 추가하는 매우 단순한 형태의 익스플로잇(exploit) 코드[1]로 존재하였으나 현존하는 커널에서는 대부분 취약점이 수정되었다. 하지만 최근까지 발견되고 있는 커널 결함[2]을 이용한 메모리 변조 취약점으로 인해 시스템 관리자 권한을 획득하는 수단으로 여전히 Call gate 기능이 악용되고 있다. 시스템 내부에 침입한 공격자는 Call gate 생성을 통해 커널 전체를 통제할 수 있는 특권레벨 권한을 얻을 수 있으며 시스템 관리자 권한을 획득할 수 있다.

이러한 문제점을 근본적으로 해결하기 위해서는 개별적인 취약점 패치가 아닌 커널 레벨의 보안 메커니즘 설계가 필요하며 보안 모듈 설계를 통한 다양한 커널 기반 0day 취약점 공격에 대응할 수 있는 방안을 모색하게 되었다. 따라서 본 논문에서는 Call gate 기능이 공격자에 의해 악용되는 것을 검사하기 위한 공격 시도 탐지 방법을 제시하고자 한다. 2장에서는 Call gate 메커니즘과 기존의 알려진 커널 공격 기법에 대해서 기술하고 3장에서는 본 논문에서 제안하는 Call gate 오용 탐지 메커니즘에 대하여 논할 것이며 마지막 4장에서는 결론과 향후 연구 방향에 대해서 언급하고자 한다.

## 2. 관련 연구

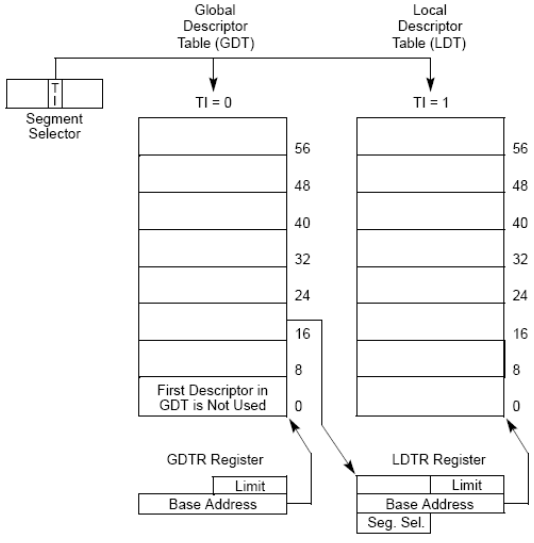
### 2.1. Call gate 오용 커널 공격 기법

Call gate 오용 커널 공격 기법은 어플리케이션 특권레벨의 사용자가 시스템 세그먼트(segment)인 Call gate 기능을 악용하여 커널 특권레벨로 권한을 상승시킬 수 있는 여러 종류의 해킹 기법을 의미한다. 이 장에서는 이러한 해킹 기법들의 메커니즘을 설명하기 위해 프로세서 아키텍처와 운영체제에 대한 배경 지식, 하드웨어적으로 제

공되는 콜 게이트 메커니즘에 대해 설명한다[3,4,5].

### 2.1.1. GDT, LDT 세그먼테이션 메커니즘

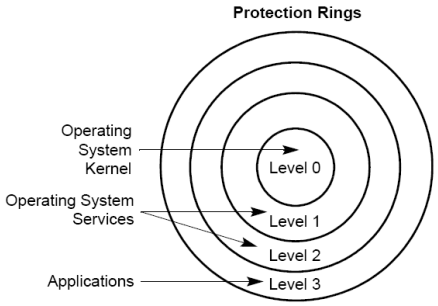
세그먼테이션(segmentation)은 디스크립터(descriptor)를 이용하여 메모리의 속성을 지정하는 메커니즘이다. 세그먼테이션 메커니즘을 이해하기 위해서는 세그먼트 레지스터(register)에 대한 지식이 필요하다. 세그먼트 레지스터는 크게 16비트 셀렉터(selector) 레지스터와 64비트 디스크립터 레지스터로 나뉜다. 먼저 셀렉터의 13비트를 차지하는 index는 메모리 내의 디스크립터 테이블을 지시하도록 설계되어 있는데 여기서 디스크립터 테이블은 시스템 전역에 해당하는 GDT(Global Descriptor Table)와 프로세스마다 개별적으로 정의되는 LDT(Local Descriptor Table)로 구분된다. 이렇게 지시된 디스크립터는 검증 과정 이후 디스크립터 레지스터로 로드된다. 세그먼트의 속성은 각각의 세그먼트 디스크립터에 의해 정의되며 디스크립터 내의 S 필드에 의해 코드, 데이터 세그먼트 또는 시스템 세그먼트로 나뉘게 된다. GDT와 LDT의 위치는 GDTR 레지스터와 LDTR 레지스터라는 특수한 레지스터에 의해 지정되며 기본적으로 사용되는 GDT와 달리 LDT는 각 프로세스 특성에 따라 사용 여부가 결정된다. (그림 1)에서의 LDT의 동작 구조를 살펴보면 먼저 LDTR 레지스터 내의 셀렉터 레지스터를 통해 GDT의 특정 시스템 세그먼트 디스크립터를 지시하는데 해당 디스크립터 속성을 참고하여 LDT 디스크립터의 시작 위치를 얻게 된다. GDT에 접근할 것인지 또는 LDT에 접근할 것인지는 세그먼트 셀렉터 레지스터 내의 1비트를 차지하는 TI 필드를 통해 정해진다.




(그림 1) GDT와 LDT의 구조

### 2.1.2. Privilege Level (Ring) 메커니즘

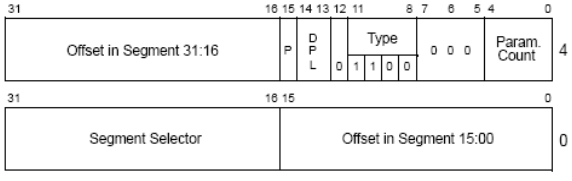
보호모드의 가장 큰 변화는 세그먼테이션 과정을 통해 특권레벨이라는 개념으로 커널과 사용자의 메모리 영역을 분리하여 읽기, 쓰기, 실행 차원의 보호를 제공한 것이다. 각 특권레벨은 (그림 2)와 같이 동그라미로 표현되기 때문에 Ring이라고 불리며 현존하는 대부분의 운영체제는 커널 특권레벨인 Ring0와 사용자 어플리케이션 특권레벨인 Ring3만을 사용하고 있다. 특권레벨의 종류는 크게 3가지로 나뉘며 현재 태스크의 특권레벨로서 코드 세그먼트 셀렉터에 명시된 CPL과 디스크립터 생성 시 명시된 DPL, 세그먼트 셀렉터 내에 명시되어 특정 디스크립터 접근 시 쓰이는 RPL 등이 있다. 여기서 RPL은 특권레벨 Ring3 어플리케이션이 특권레벨 Ring0 권한인 커널 영역에 접근하는 것을 차단하기 위해 제공하는 특권레벨로서 낮은 레벨이 높은 레벨의 코드를 실행하려고 하면 GP(General Protection) 에러를 발생시켜 접근하지 못하도록 차단하는 역할을 한다.



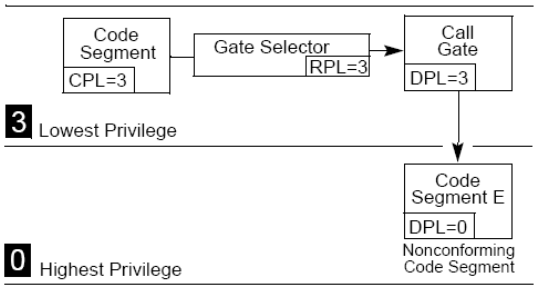

(그림 2) Privilege Level (Ring) 구조

### 2.1.3. Call gate 메커니즘

Call gate는 낮은 레벨 프로그램이 높은 레벨 루틴을 잠시 동안 사용하는 메커니즘으로서 항상 정해진 루틴에 의해 움직인다. Call gate도 다른 세그먼트 디스크립터와 함께 GDT 또는 LDT에 세그먼트 디스크립터에 포함되며 (그림 3)과 같은 하나의 시스템 세그먼트 정의라고 할 수 있다. (그림 4)와 같이 특권레벨이 Ring3인 일반 어플리케이션의 CPL 값이 3이라고 가정하면 특권레벨 Ring0에서 동작하는 동안 CPL 값은 0이 되며 특권레벨 Ring3로 돌아온 후 CPL 값은 다시 3이 되는 동작 과정을 거친다.




(그림 3) Call gate 구조




(그림 4) Call gate 동작 과정

## 2.2. Call gate 오용 커널 공격 사례

고전 유닉스와 리눅스 커널에서는 표1과 같이 LDT에 임의의 Call gate를 생성하여 특권레벨 Ring0 권한을 악용당하는 사례가 많았다. 특히 2001년 Argus 해킹 대회 사례의 경우 자사 제품의 문제가 아닌 커널 취약점으로 인해 발생했던 대표적인 사건이라고 볼 수 있다[6]. 현존하는 대부분의 커널은 LDT에 시스템 세그먼트를 생성하지 못하도록 수정되어졌으며 특권레벨 Ring3 어플리케이션이 특권레벨 Ring0 권한을 악용하지 못하도록 보호한다. 하지만 커널 내에 특정 메모리를 변조할 수 있는 취약점이 존재하는 경우 디스크립터 테이블에 악의적인 Call gate를 생성하는 행위는 여전히 해결할 수 없는 고질적인 문제점으로 남아있다.

<표 1> 실제 Call gate를 악용한 커널 공격 취약점 사례

| Call gate를 악용한 커널 공격 취약점 사례 | 해당 운영체제 |
|---|---|
| sysarch() 함수로 인한 LDT Call gate 오용 취약점 | NetBSD 1.4 1.4.x 1.5 / OpenBSD 2.6-2.8 |
| sysi86() 함수로 인한 LDT Call gate 오용 취약점 | Solaris 2.7 2.8 / SCO Openserver 5.0.4 5.0.6 / SCO Unixware 7.0.1 |
| setcontext() 함수로 인한 LDT Call gate 오용 취약점 | SCO Openserver 5.0.4 5.0.6 |
| do_brk() 함수로 인한 vma 메모리 변조 취약점 | Linux kernel 2.4.x |
| binfmt_elf uselib() 함수로 인한 vma insert race 취약점 | Linux kernel 2.4.x 2.6.x |

## 2.3. Call gate 오용 커널 공격을 탐지하기 위한 연구

Call gate 오용 커널 공격을 탐지하기 위한 관련 연구는 아직까지 발표된 사례가 없다. 단지 메모리 변조 취약점이 발견된 커널 코드에 대해서 적재가능커널모듈(LKM: Loadable Kernel Module, 이하 LKM)을 통한 임시 패치와 정기적인 커널 패치만 제공되는 실정이다. 반면 커널 기반 취약점을 발견하기 위한 노력은 계속되고 있으며 여러 해외 해커 그룹에 의해 꾸준히 연구되어 지속적으로 발표되고 있는 실정이다[7].

# 3. 제안하는 Call gate 오용 탐지 방법

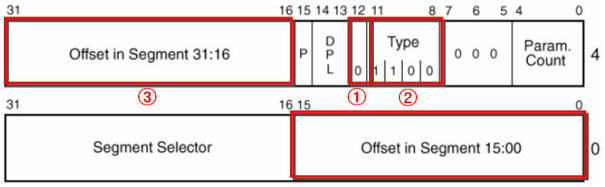
## 3.1. Call gate 오용 탐지 메커니즘

Call gate를 악용하는 커널 공격 익스플로잇 코드는 시스템 세그먼트인 Call gate 디스크립터를 생성하고 PCB 내의 id를 변경하여 관리자 권한 획득 후 시스템 특정 명령을 실행하는 일련의 패턴을 갖는다. 따라서 이러한 패턴을 검사하면 공격 시도 여부를 확인할 수 있다. 먼저, 익스플로잇 코드 내에서 반드시 한 번 이상 호출되는 주요 시스템 콜 함수를 가로챈 후[8] GDT와 LDT 내에 게이트 디스크립터가 생성되었는지 검사함으로써 Call gate 생성 여부를 확인할 수 있다. 만일 Call gate가 존재한다면 특권레벨 Ring0 권한으로 호출하는 코드 세그먼트가 어디에 위치해있는지 검사하여 사용자의 코드 세그먼트에 있는 경우 악의적인 익스플로잇 코드에 의한 공격으로 판단할 수 있다. 그 이유는 비정상적인 Call gate의 경우 오프셋(offset) 주소의 위치가 항상 특권레벨 Ring3 사용자의 코드 세그먼트 영역에 위치하는 특징을 가지기 때문이다. 이렇게 익스플로잇 코드에 의한 공격으로 판단되면 로그를 기록하여 악의적인 공격자가 관리자 권한의 쉘(shell)을 실행하려 했던 사실을 기록한다.

## 3.2. Call gate 오용 탐지 메커니즘의 적용

이 절에서는 본 논문에서 제안하는 메커니즘을 리눅스 시스템에 적용시켜 설명한다. 탐지 메커니즘을 Linux 시스템에 적용하기 위해서는 LKM 형태의 커널 기반 모듈로 작성해야만 한다. 현재 널리 공개되어 있는 커널 익스플로잇 코드[9]들은 공격 시 Call gate로 PCB를 변조한 후 관리자 권한 쉘을 실행시키는 execve() 함수를 호출하기 때문에 execve() 함수를 가로채면 LDT 디스크립터 내에 생성된 Call gate를 검출할 수 있다. Call gate의 존재 유무를 확인하기 위해서는 (그림 5)와 같이 먼저 LDT 시작 위치부터 존재하고 있는 디스크립터 1번 위치의 S 필드 값이 0인지 검사하여 시스템 세그먼트인지 알 수 있으며 2번 위치의 Type 필드 값이 16진수 0xC인지 검사하여 386 Call gate 형식인지 확인할 수 있다. 그러나 execve() 함수를 가로채 디스크립터 테이블 전체에 대한 검사를 수행할 경우 시스템 퍼포먼스에 큰 영향을 미칠 우려가 있으므로 반드시 PCB 내의 context.ldt 값을 우선적으로 검사하여 LDT를 사용 중인 프로세스만을 대상으로 검사하도록 한다. 마지막으로 특권레벨 Ring0 권한으로 실행할 코드 위치는 3번의 오프셋 주소가 PAGE_OFFSET 이하에 있는지 검사하여 판단한다. 이는 일반적으로 특권레벨 Ring0 권한으로 호출되는 오프셋 주소가 커널 영역에 자

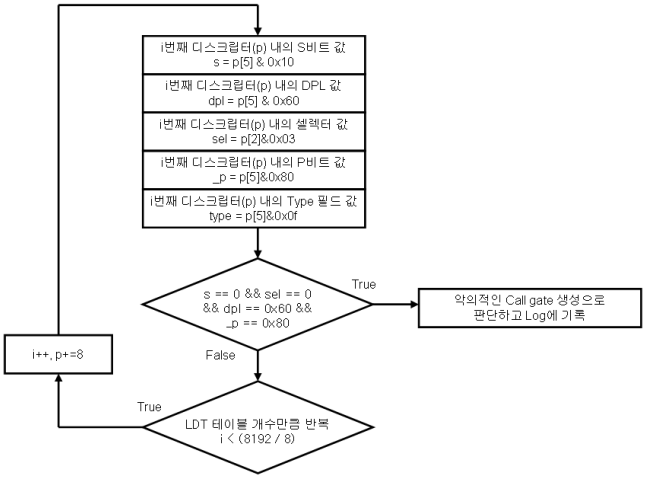
리 잡는 특성을 이용한 것으로 실행될 코드의 위치가 특권레벨 Ring3 권한인 사용자 영역의 메모리로 확인될 경우 악의적인 익스플로잇 코드에 의해 생성된 Call gate 디스크립터임을 판단할 수 있다.




(그림 5) Call gate 내의 검사 대상 필드

### 3.3. Call gate 오용 탐지 메커니즘의 구성

(그림 6)은 악의적인 Call gate가 생성되었는지 여부를 검사하는 탐지 모듈의 알고리즘 구성도이다. 먼저 8192개의 디스크립터 크기만큼 비교 문을 반복하여 디스크립터 내의 각 필드 값을 얻은 후 Call gate인지 비교하는 과정을 수행한다. 검사 과정에서 악의적인 Call gate를 발견한 경우에는 관리자에게 탐지된 공격 사실을 알리기 위해 공격 시도에 대한 상세 내용을 로그에 기록한다.




(그림 6) Call gate 존재 여부 검사 및 탐지 알고리즘

## 4. 결론 및 향후 연구

각 운영체제 커널은 특권레벨 권한과 관련된 Call gate 기능을 기본적으로 제공한다. 이러한 Call gate 기능은 초기 운영체제 커널과 마찬가지로 설계 단계에서 보안을 고려하지 않고 개발되었기 때문에 커널의 보안과 시스템 관리자들을 위협하는 요소로 작용하고 있다. 또한 커널 기반 취약점을 발견하기 위한 연구와 공격 코드 개발은 해외 해커들에 의해 활발하게 진행되고 있지만 Call gate 오용을 막기 위한 근본적인 대책과 보안에 관한 연구는 거의 전무한 상태이다. 커널 0day 취약점의 공격 사례가 증가할수록 커널 보안의 필요성은 점차 부각될 것이며 Call gate 기능의 보안성을 높일 수 있는 다양한 기술들이 요구될 것이다. 본 논문에서는 Call gate 오용 커널 공격을 효과적으로 탐지하기 위한 메커니즘과 함께 커널 모드에서 LDT 디스크립터를 확인하여 악의적인 Call gate 생성 여부를 검사하는 커널 모듈을 소개하였다. 하지만 현재까지의 연구 수준으로는 LDT에 Call gate를 생성하는 익스플로잇 코드에 대해 탐지만할 수 있는 기술적 한계가 존재한다.

향후 연구 과제로는 LDT 뿐만 아니라 GDT에 악의적인 Call gate를 추가하는 공격에 대한 탐지 및 차단을 통해 공격을 실시간으로 방지하는 기술에 대한 연구가 필요하다. 따라서 GDT와 LDT 변조 여부를 동시에 검사하고 악의적인 Call gate를 탐지하였을 때 공격을 신속하게 차단하는 보안 메커니즘에 대해 지속적으로 연구하고 구현해나갈 계획이다.

### 참고문헌

[1] LSD Research Group, "Kernel Level Vulnerabilities", 11-22, 2001.

[2] iSEC, Linux kernel vmsplice unchecked user-pointer dereference, 02-08, 2008.

[3] Intel, "IA-32 Intel® Architecture Software Developer's Manual Volume #3", 06, 2005.

[4] Sergio Sáez, Juan Carlos Pérez, "The i386 processor architecture", E.S.O, 2003.

[5] 김범준, "만들면서 배우는 OS 커널의 구조와 원리", 한빛미디어, 03-28, 2005.

[6] Michelle Delio, "Hackers Win Security Challenge", WIRED, 04-23, 2001.

[7] milw0rm, http://www.milw0rm.com/

[8] halfilife, "Abuse of the Linux Kernel for Fun and Profit", Phrack 50-5, 04-09, 1997.

[9] iSEC, "Linux Kernel do_brk() Vulnerability", 12-01, 2003.