논문 2009-3-27

리눅스 커널 백도어 침입에 대한 차단 기법 연구

A Study on Intrusion Protection Techniques against Linux Kernel Backdoor

김진택*, 고정호*, 홍민석*, 손철웅*, 박범*, 이도원*, 이극**

Jin-Taek Kim, Jeong-Ho Kho, Min-Seok Hong, Choul-Woong Son, Beom Park, Do-Won Lee, Geuk Lee

요 약 기존의 백도어는 애플리케이션 모드인 유저 모드에서 작동을 하였기 때문에 tripwire나 MD5와 같은 시스템 파일 무결성 검사로 백도어의 유무를 확인할 수가 있었다. 그러나 커널 모듈을 이용한 백도어는 기존의 애플리케이션 모드에서 작동하는 시스템 파일 무결성 검사로는 백도어의 유무를 확인할 수 없다. 이런 LKM 커널 백도어를 방지하기 위해 여러 가지 프로그램들이 제시가 되었지만 시스템 콜 테이블 변경 유무만을 검사하므로 방지에 한계가 있다. 본 논문에서는 이러한 LKM 커널 백도어에 대한 공격의 위험성을 인식하여 기존의 무결성 검사만으로 커널 백도어 침입을 차단할 수 없었던 커다란 한계성에 대한 대안을 제시하고자 한다.

Abstract As the existing backdoor worked at user mode, which is application mode, it was possible to check the existence of backdoor by the integrity check of system file. However, for the backdoor using kernel module, it is impossible to check its existence by the integrity check of system file. Even various programs were presented to protect this LKM Kernel backdoor, there is limitation in protection as they examine the changes on the system Call Table. This study, recognizing the danger of invasion through such LKM Kernel backdoor, will provide alternative for the limitation which the existing integrity check couldn't prevent intrusion through Kernel backdoor.

Key Words : 커널, 시스템콜, LKM

Ⅰ. 서 론

나날이 발전하는 컴퓨터, 네트워킹 기술과 인터넷 문화의 빠른 보급으로 인해 지식 정보화 사회로의 이전이 급속하게 변하고 있다. 우리나라는 다른 나라들에 뒤지지 않을 만큼의 네트워크 기반 시설을 갖추었으며, 다양하고 많은 사용자 층이 있다. 이렇게 수많은 사용자들은 인터넷을 통하여 수많은 정보의 공유와 사회·문화·경제 교류의 확대, 편리한 생활 제공 등으로 다방면에서 삶의 질을 높이고 긍정적인 효과를 얻고 있다. 그러나 설계상의 개방적인 특성과 TCP/IP 프로토콜의 근본적인 문제점 및 정보시스템 자체의 취약성 등으로 인해서 악의적인 불법 침입에 의한 접근 정보의 오용 및 도청, 위조 및 변조 행위들이 이루어지고 있다. 이러한 문제점들은 개인뿐만 아니라 나아가서는 기업과 정부의 정보 시스템의 마비까지 발생하는 사태 등 큰 사회 문제로까지 발전될 가능성이 있으며, 더 나아가서는 전쟁을 위한 도구로까지 이용될 수 있다.

최근 국내에서 사용 증가가 많이 늘어난 공개 운영 체제인 리눅스를 대상으로 하는 침입 사례가 늘어나는 추세이다. 리눅스 시스템은 시스템 콜을 통해서 작동한다.

*정회원, 한남대학교
**정회원, 한남대학교(교신저자)
접수일자 2009.04.15, 수정완료.2009.06.05

파일의 생성, 삭제할 때도, 사용자 권한에 대한 명령도 시스템 콜에 의해서 작동을 한다. 이에 따라 리눅스 시스템 침입 시에도 시스템 콜을 통해서 이루어 진다고 볼 수 있다.[1] 그래서 침입자들은 시스템 콜 테이블을 수정하기 위해서 여러 가지 방법을 사용한다. 그 중 하나가 수정이 된 커널을 패스워드 없이도 시스템 재차 접근할 수 있도록 비밀리에 LKM을 사용하여 백도어를 설치하는 것이다. 기존의 백도어들은 주로 시스템의 주요 시스템 파일들 ls, ps, netstat 등을 바꾸는 기법을 사용하였으나, 이러한 기법은 tripwire, md5 같은 무결성 검사 툴로 쉽게 탐지하여 대응할 수 있었다. 그러나 이 LKM 백도어는 한 차원 높은 커널 레벨수준의 백도어로서 기존의 탐지 툴이나 방법을 이용하여 커널 백도어를 쉽게 찾아낸다는 것이 불가능하다.

따라서 본 논문에서는 커널 LKM 백도어 차단과 탐지에 대하여 연구한다. 커널 LKM 백도어에 관련된 연구를 통하여 차단과 탐지에 관한 기법을 찾고 커널 변경을 통한 시스템의 큰 피해를 막을 수 있다.

Ⅱ. 관련 연구

1. 커널(Kernel)

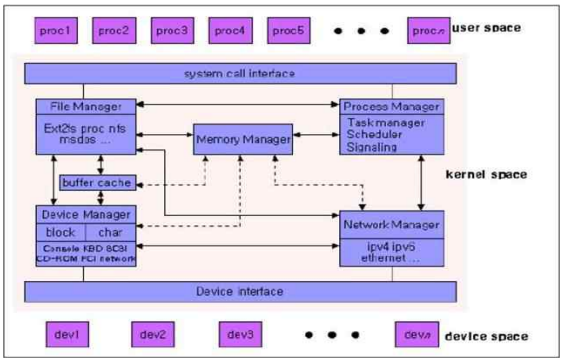

그림 1. 리눅스 커널
Fig. 1. Linux Kernel

리눅스 커널이란 운영체제의 핵심을 이루는 요소로서 컴퓨터 내의 자원을 사용자 프로그램이 사용할 수 있도록 관리하는 프로그램이며 그 기능은 프로세스 관리, 파일 시스템, 메모리 관리 및 네트워크 등이 있다. 대부분의 사용자는 C 라이브러리를 이용하여 시스템 콜을 사용한다. 프로세스들은 open, close, read, write, stat, chown, chmod 등의 파일에 관련된 시스템 콜을 이용하여 파일 서브 시스템을 제어한다. 파일 시스템은 커널과 2차 기억장치 사이의 데이터의 흐름을 버퍼를 통해서 엑세스한다. 버퍼 캐쉬는 블록 장치 드라이버와 상호 작용하여 커널과 데이터를 주고 받는 것을 제어한다. 블록 입출력 장치는 랜덤 엑세스 기억장치를 말한다. 캐시 기능을 사용하지 않는 디바이스 드라이버는 문자 디바이스 드라이버로서 블록 디바이스 이외의 디바이스 드라이버가 해당이 된다. 프로세서 제어 서브 시스템에서 사용하는 시스템 콜은 fork, exec, exit, wait, brk, signal, kill, setpgrp, setuid 등이 있으며, 이들은 프로세스 간의 통신, 스케줄링, 메모리 관리제어를 한다. 커널 구조 중에는 하드웨어 제어 루틴이 있는데, 컴퓨터간의 통신 및 하드웨어의 인터럽트를 처리한다.[2] 결국 리눅스 시스템은 시스템 콜을 통해서 작동한다는 것을 알 수 있다. 시스템 콜을 사용해서 파일의 내용을 읽고, 쓰며, 프로세스를 제어한다.

2. 시스템 콜

시스템 콜이란 커널 영역과 사용자 영역을 중재해 주는 인터페이스에 해당하는 것이다. 다중 사용자나 혹은 단일사용자에 의한 실수로부터 커널 영역을 보호하기 위해 커널은 구별된 메모리공간을 사용한다. 따라서 사용자 영역에서 하드웨어를 컨트롤하기 위해선 직접 하드웨어에 접촉할 수가 없고, 커널에서 제공한 인터페이스인 시스템 콜을 이용해야 하는 것이다. 예를 들자면 특정프로그램이 파일을 편집하기 위해서 파일을 열었다면, 파일시스템에서 해당하는 파일의 inode정보를 읽어오고 파일의 서술자를 반환하기 위한 시스템 콜인 sys_open()을 내부적으로 사용한 것이 된다.[3] 다음 그림 2는 /usr/include/sys/syscall.h 파일에 들어있는 시스템 콜 목록의 일부이다.

```
#ifndef _SYS_SYSCALL_H
#define _SYS_SYSCALL_H
#define SYS_setup 0 /* Used only by init, to get system going. */
#define SYS_exit  1
#define SYS_fork  2
#define SYS_read  3
#define SYS_write 4
#define SYS_open        5
#define SYS_close       6
#define SYS_waitpid     7
#define SYS_creat       8
#define SYS_link  9
#define SYS_unlink      10
#define SYS_execve      11
#define SYS_chdir 12
#define SYS_time  13
#define SYS_prev_mknod  14
#define SYS_chmod       15
#define SYS_chown       16
#define SYS_break       17
#define SYS_oldstat     18
#define SYS_lseek       19
#define SYS_getpid      20
#define SYS_mount       21
#define SYS_umount      22
#define SYS_setuid      23
```

그림 2. 시스템 콜
Fig. 2. System Call

3. LKM(Loadable Kernel Modules)

운영체제의 핵심을 이루는 커널도 새로운 디바이스를 추가하는 경우 등의 이유로 모듈이 새롭게 추가될 필요성이 있을 때가 있다. 이러한 이유로 LKM(Loadable Kernel Module)을 지원 가능하게 되었는데. 그러한 커널의 대표적인 경우로 맥 OS X의 커널인 다윈이 있다. 이 커널의 경우는 커널의 모든 부분이 모듈로 이루어져 있어서 필요에 따라 로드 및 언로드가 가능하다. 이러한 커널을 마이크로 커널이라고 한다. 반면 전통적인 유닉스(리눅스 역시 포함된다.)의 경우는 모놀리틱 커널이라고 하여 새로운 모듈을 적재하는 것이 불가능하다. 하지만 같은 모놀리틱 커널이라고 하더라도 커널의 확장을 위해 일부 모듈의 로딩이 가능하도록 수정되어 있는 경우가 대다수다. 이때 사용되는 모듈을 LKM 모듈이라고 한다.[4]

이러한 LKM은 시스템이 부팅된 후 언제라도 커널에 동적으로 링크를 시킬 수 있으며, 로드된 모듈은 다른 보통 커널 코드처럼 커널의 한 부분이 된다. 이러한 LKM은 리눅스, FreeBSD, 솔라리스 등 많은 유닉스 시스템에서 그 기능이 제공되고 있다. 모듈은 커널 코드와 똑같은 권한과 책임을 진다. 다르게 말하면, 유닉스 커널 모듈은 모든 커널 코드나 디바이스 드라이버처럼 커널을 망가뜨릴 수도 있다.

Ⅲ. LKM 커널 백도어

1. LKM을 이용한 커널 백도어

영역에 설치된 백도어는 커널의 확장을 위해 개발된 LKM의 특징을 악용한 것이다. LKM 백도어의 기본적인 개념은 침입자가 자신만의 커널 모듈을 만들어 그것을 커널 영역에 insmod하여 그 모듈로 하여금 정상적인 시스템 콜을 가로채서 특정 프로세스를 숨기는 등의 정상적인 작동을 변조하는 것이다.

아래 그림3과 같이 전통적인 루트킷이 설치되어있는 시스템의 경우 제대로 된 체크섬 값이 저장된 파일과 무결성이 확보된 tripwire 같은 체크섬 검사 프로그램이 있다면 login, ps와 같은 주요 프로그램들에 대한 체크섬 값을 계산해서 그 이상 유무를 알아낼 수 있고 더 나아가 그 해당 프로세스를 종료하고 파일만 제거해 버린다면 간단히 제거도 가능하다. 그러나 커널 백도어의 문제점은 사용자 영역의 어플리케이션을 전혀 변경하지 않고서도 정상적인 프로그램을 백도어처럼 작동하도록 만들어 버린다는 것에 있다.[5] 어플리케이션 프로그램이 변하지 않았으므로 그 프로그램에 대한 체크섬 값을 가지고는 전혀 그 이상 징후를 발견할 수 없다.

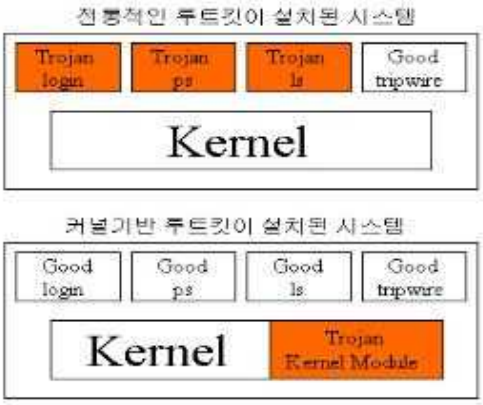

그림 3. 커널 백도어
Fig. 3. Kernel Backdoor

따라서 커널에 삽입된 백도어의 경우 그 발견과 제거가 쉽지 않다. 더군다나 커널 모듈의 경우 재부팅을 하기 전까진 스스로 제거 되지 않으며, 대부분의 커널 백도어의 경우 커널을 제거하지 못하도록 그와 관련된 시스템 콜까지 변경하고 있으므로 제거가 더욱 어렵다. 만약 24시간 켜져 있어야 하는 상용서버의 경우 그 문제가 상당히 심각할 것이고 이를 해결하기 위해선 잠시 서비스를 중단하고 재부팅과 함께, 혹시 있을지도 모를 시작 스크립트 속에 숨어있는 커널 백도어를 커널에 loading하는 스크립트를 찾아내 제거해야 한다.

2. 커널 백도어의 시스템 콜 가로채기

시스템 콜은 자신이 생성된 커널 영역에서가 아니라 사용자 영역에서 호출되어야 하기 때문에 시스템 콜의 인터페이스가 export되어 있다. 앞서서 본 것과 같이 시스템 콜들은 각자 자신의 번호를 가지고 있으며 자신이 호출되기 위해서 자신의 주소를 sys_call_table에 저장하고 있다. 그렇기에 사용자 영역에서 시스템 콜을 호출하려면 sys_call_table[]에 해당하는 시스템 콜 이름을 넣고 그 시스템 콜의 주소를 얻어와 호출하는 것이다. 예를 들어 sys_open()함수를 호출하기 위해선 다음과 같은 방식으로 시스템 콜을 호출할 수 있다.

openfunc = sys_calll_table[sys_open]

이러한 시스템 콜의 성질을 이용하여 시스템 콜을 가로채는 것은 아래 그림4와 같은 코드로 간단히 해결할 수 있다.

```
#define MODULE
#define __KERNEL__

#include <linux/module.h>
#include <linux/kernel.h>
#include <asm/unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <asm/fcntl.h>
#include <asm/errno.h>
#include <linux/types.h>
#include <linux/dirent.h>
#include <sys/mman.h>
#include <linux/string.h>
#include <linux/fs.h>
#include <linux/malloc.h>

extern void* sys_call_table[];        /*sys_call_table is exported, so we
                                       can access it*/

int (*orig_mkdir)(const char *path); /*the original systemcall*/


int hacked_mkdir(const char *path)
{
 return 0;                            /*everything is ok, but he new systemcall
                                       does nothing*/
}

int init_module(void)                /*module setup*/
{
 orig_mkdir=sys_call_table[SYS_mkdir];
 sys_call_table[SYS_mkdir]=hacked_mkdir;
 return 0;
}

void cleanup_module(void)             /*module shutdown*/
{
 sys_call_table[SYS_mkdir]=orig_mkdir; /*set mkdir syscall to the origal
                 one*/
}
```

그림 4. 시스템 콜 가로채기
Fig. 4. Interrupting the system call

3. 커널 심볼 테이블

지금까지는 시스템 콜 테이블에 들어 있는 시스템 콜을 가로채서 백도어가 심어져 있는 시스템 콜로 대체해 온 것들이다. 이를 위해서 백도어가 심어져 있는 적재 가능한 커널 모듈을 만들어 루트 권한으로 커널을 적재하고 시스템 콜을 대체하였다. 커널 모듈을 관리하는 명령어 중에 lsmod명령이 있다. 이 명령은 현재 적재되어 있는 모듈을 보여주는 역할을 한다. 그렇기에 우리가 백도어를 심으려고 적재한 모듈은 lsmo d명령 한 번에 그 모습이 드러나고 만다. 따라서 그것을 감추어 줄 수 있는 다른 방법이 필요하다.

커널 심볼 테이블이란 시스템 콜 테이블과 비슷하게 적재된 커널들을 등록하여 놓는 배열이다. 이러한 커널 심볼 테이블에 들어있는 모듈들의 이름은 /proc/ksyms 디렉토리 혹은 /proc/module/ 아래에 파일처럼 나열되어 있는 것을 발견할 수 있다.

4. 커널 모듈 감추기

이곳에 올라와 있는 모듈들의 이름을 보고 경험 있는 관리자라면 예상외의 모듈이 올라와 있음을 알게 될 것이고 그것은 곧 제거되고 말 것이다. 따라서 백도어를 만드는 모듈들을 숨길 필요성이 있는데 간단하게 그 방법을 알아보고자 한다.

커널 심볼 테이블에서 커널들을 숨기는 방법은 다양하다. 그 중에서 가장 기초적인 것은 대라면 모듈을 초기화 하는 과정에서 모듈의 이름을 0또는 NULL로 초기화 하는 것이다. 다음의 간단한 모듈의 초기화 부분을 보자.

```
/*from Phrack // or whatever register it is in
*(char*)mp->name=0;
mp->size=0; mp->ref=0;
```

위와 같이 한다면 모듈을 쿼리한 경우라도 무시가 되므로 보이지 않음은 물론 제거 역시 되지 않는다. 다른 방법, 그러니 좀 더 나은 방법을 대라고 한다면, lsmod, rmmod의 시스템 콜을 추척해 보면 공통적으로 발견되는 시스템 콜 즉 sys_query_module()을 가로채는 것이 있다. 이 시스템 콜을 가로채는 것은 sys_write()콜을 가로채는 것과 같다. Nsys_query_module을 가로채서 심겨진 모듈과 관련된 문자열을 처리 요청을 모조리 걸러내는 것이다. lsmod과 rmmod가 공통으로 사용하는 시스템 콜이므로 이것을 이용해서 시스템 콜을 가로채면 /proc/module에서 확인할 수도 그리고 제거할 수도 없다.[6]

Ⅳ. LKM 커널 백도어 침입 차단 기법

LKM기반의 백도어를 막는 가장 기본적인 방법은 컴파일의 모듈 로딩기능을 삭제한 완전한 모놀리틱 커널로 컴파일하는 것이다. 그러나 이것은 단순하고 확실한 방법이기는 하나 나중에 디바이스를 추가하는 등의 필요에 의해 커널이 확장되어야 할 순간이 온다면, 전체 커널을 다시 컴파일해야 하고 시스템은 재부팅시켜야 하는 맹점이 따른다. 실제로 서버들은 어지간한 일이 아니고선 재부팅을 할 수 없는 상태인 경우가 많다. 만약 이러한 상황이 된다면, 그 서버관리자로선 무척 난감할 일이 아닐 수가 없을 것이다. 따라서 이후에 설명할 몇 가지 예시들은 커널의 모듈기능을 제거하지 않은 상태에서 어떻게

하면 커널 백도어가 설치되지 않도록 막을 수 있을 것인가 하고 백도어가 설치되어 있다면 어떻게 이를 발견할 수 있을 것인가 하는 부분에 대한 것이 될 것이다.

1. 로그파일로 로딩된 커널 모듈 이름 남기기

로딩된 커널의 경우는 그 정보가 /proc/module/아래에 보이게 되지만 그 정보는 앞서서 언급한 여러 가지 방법에 의해 가려지기도 하고 삭제되기도 한다. 따라서 별도의 로그파일을 유지하는 것이 커널 모듈의 관리에 있어서 매우 유익하다. 로딩된 커널 모듈의 이름을 로그파일로 유지 하려면 sys_create_module()이라는 시스템 콜을 약간 변조하면 된다. 이것은 모듈이 로딩 될 때마다 호출되도록 되어 있으므로 이 시스템 콜에 로딩되는 모듈의 이름을 기록하는 코드를 넣는다면 따로 지정된 파일에 기록이 저장되어 침입자가 자신의 모듈을 보이지 않게 한다 하더라도 그 존재를 숨길 수 없게된다.

```
#define MODULE
#define __KERNEL__

#include <linux/module.h>
//~~~중략
#include <linux/malloc.h>

extern void* sys_call_table[];


int (*orig_create_module)(char*, unsigned long);


int hacked_create_module(char *name, unsigned long size)
{
 char *kernel_name;
 char hide[]="ourtool";
 int ret;

 kernel_name = (char*) kmalloc(256, GFP_KERNEL);
 memcpy_fromfs(kernel_name, name, 255);

 /*이 곳에서 syslog를 통해 로그파일을 기록하는 코드를 집어넣거나 아니면 다른 로그파일에
기록할수 있는 코드삽입하는게 가능하다. */
 printk("<1> SYS_CREATE_MODULE : %s\n", kernel_name);

 ret=orig_create_module(name, size);
 return ret;
}


int init_module(void)                  /*module setup*/
{
 orig_create_module=sys_call_table[SYS_create_module];
 sys_call_table[SYS_create_module]=hacked_create_module;
 return 0;
}

void cleanup_module(void)              /*module shutdown*/
{
 sys_call_table[SYS_create_module]=orig_create_module;
}
```

그림 5. 로그파일로 로딩된 커널 모듈 이름 남기기
Fig. 5. Logging loaded kernel modules' names to the log file

2. 패스워드를 통한 모듈인증

이 기법 역시 모듈을 커널에 올리는 sys_create_module()을 변환하는 것이다. 그러나 앞서의 방법과 달리 로그파일 유지가 아니라. 관리자나 혹은 침입자가 새로운 모듈을 커널에 올릴 때 패스워드를 통한 인증절차를 받도록 하는 것이다. 이때 미리 정의된 패스워드를 사용작가 알고 있어야 모듈을 로딩할 수 있다. 문제는 관리자가 입력한 패스워드가 중간에 도청당하지 않고 안전하게 커널에 올라가 있는 인증모듈에 보낼 수 있을 것 인가 하는 것이다. 간단하게 구현 한다면 관리자가 환경파일에 패스워드를 입력하고 커널을 올릴 수 있으나 이는 다른 침입자로 하여금 패스워드를 쉽게 얻을 수 있는 방법이므로 그 안정성은 매우 낮다.

```
#define MODULE
#define __KERNEL__

#include <linux/module.h>
//~ 헤더파일 중략
#include <sys/stat.h>


extern void* sys_call_table[];

/*if lock_mod=1 THEN ALLOW LOADING A MODULE*/
int lock_mod=0;

int __NR_myexecve;

/*intercept create_module(...) and stat(...) systemcalls*/
int (*orig_create_module)(char*, unsigned long);
int (*orig_stat) (const char *, struct old_stat*);

char *strncpy_fromfs(char *dest, const char *src, int n)
{
 char *tmp = src;
 int compt = 0;

 do {
  dest[compt++] = __get_user(tmp++, 1);
 }
 while ((dest[compt - 1] != '\0') && (compt != n));

 return dest;
}

int hacked_stat(const char *filename, struct old_stat *buf)
{
 char *name;
 int ret;
 char *password = "password"; /*yeah, a great password*/

 name     = (char *) kmalloc(255, GFP_KERNEL);

 (void) strncpy_fromfs(name, filename, 255);

 /*do we have our password ?*/
 if (strstr(name, password)!=NULL)
 {
  /*allow loading a module for one time*/
  lock_mod=1;
  kfree(name);
  return 0;
 }
 else
 {
  kfree(name);
  ret = orig_stat(filename, buf);
 }
 return ret;
}
int hacked_create_module(char *name, unsigned long size)
{
 char *kernel_name;
 char hide[]="ourtool";
 int ret;

 if (lock_mod==1)
 {
  lock_mod=0;
  ret=orig_create_module(name, size);
  return ret;
 }
 else
 {
  printk("<1>MOD-POL : Permission denied !\n");
  return 0;
 }
 return ret;
}


int init_module(void)                  /*module setup*/
{
 __NR_myexecve = 200;

 while (__NR_myexecve != 0 && sys_call_table[__NR_myexecve] != 0)
  __NR_myexecve--;

 sys_call_table[__NR_myexecve]=sys_call_table[SYS_execve];


 orig_stat=sys_call_table[SYS_prev_stat];
 sys_call_table[SYS_prev_stat]=hacked_stat;

 orig_create_module=sys_call_table[SYS_create_module];
 sys_call_table[SYS_create_module]=hacked_create_module;

 printk("<1>MOD-POL LOADED...\n");
 return 0;
}

void cleanup_module(void)              /*module shutdown*/
{
 sys_call_table[SYS_prev_stat]=orig_stat;

 sys_call_table[SYS_create_module]=orig_create_module;
}
```

그림 6. 패스워드를 통한 모듈인증
Fig. 6. Module verification by a password

V. 결 론

기존의 백도어는 애플리케이션 모드인 유저 모드에서 작동을 하였기 때문에 tripwire나 MD5와 같은 시스템 파일 무결성 검사로 백도어의 유무를 확인할 수가 있었다. 그러나 커널 모듈을 이용한 백도어는 기존의 애플리케이션 모드에서 작동하는 시스템 파일 무결성 검사로는 백도어의 유무를 확인할 수 없다. 이런 LKM 커널 백도어를 방지하기 위해 여러 가지 프로그램들이 제시가 되었지만 시스템 콜 테이블 변경 유무만을 검사하므로 방지에 한계가 있다. 본 논문에서는 이러한 LKM 커널 백도어에 대한 공격의 위험성을 인식하여 기존의 무결성 검사만으로 커널 백도어 침입을 차단할 수 없었던 커다란 한계성에 대한 대안을 제시하고 있다. 그러나 커널의 변경을 통한 침입 차단은 수정의 어려움과 사용의 제약이 따르기 때문에 향후 다양한 방법의 백도어 차단 연구가 필요하다.

참 고 문 헌

[1] Maurice J. Bach, “UNIX 운영체제의 설계”, pp 47-50, 대익사, 1992.

[2] Aessandro rubini, “Linux Device Drivers”, 2nd Editon, O’Reilly, 2001.

[3] Daniel.P.Bover, “Understanding the Linux Kernel”, 2000.

[4] Pragmatic, “Complete Linux Loadable Kernel Modules”, 1999.

[5] 정현철, “커널 기반 루트킷 분석 보고서”, 2000.

[6] http://packetstorm.decepticons.org/groups/thc/LKM_HACKING.htm

※ 본 연구는 지식 경제부 지역혁신센터사업인 민군겸용보안공학연구센터와 2009년도 2단계 두뇌한국(BK)21 사업 그리고 2009년 한남대학교 학술연구조비 지원으로 수행되었음

저자 소개

김 진 택(정회원)

• 2008년도 한남대학교 컴퓨터공학과 졸업.
• 2009년도 현재 한남대학교 일반대학원 석사 3학기 과정.
<주관심분야 : 보안, 운영체제, 네트워크>

고 정 호(정회원)

• 2007년도 한남대학교 컴퓨터공학과 졸업
• 2009년도 현재 한남대학교 일반대학원 석사 4학기 과정.
<주관심분야 : 보안, 운영체제, 네트워크>

홍 민 석(정회원)

• 2008년도 한남대학교 컴퓨터공학과 졸업.
• 2009년도 현재 한남대학교 일반대학원 석사 2학기 과정.
<주관심분야 : 보안, 운영체제, 네트워크>

손 철 웅(정회원)

• 2009년도 한남대학교 컴퓨터공학과 졸업.
• 2009년도 현재 한남대학교 일반대학원 석사 1학기 과정.
<주관심분야 : 보안, 운영체제, 네트워크>

박 범(정회원)

• 2009년도 한남대학교 컴퓨터공학과 졸업.
• 2009년도 현재 한남대학교 일반대학원 석사 1학기 과정.
<주관심분야 : 보안, 운영체제, 네트워크>

이 극(정회원)

• 1978년도 경북대학교 전자계산기과 졸업.
• 1986년도 서울대학교 컴퓨터 공학 석사 졸업.
• 1993년도 서울대학교 컴퓨터 공학 박사 졸업.
• 2009년도 현재 한남대학교 컴퓨터공학과 교수.
• 민군겸용보안공학연구센터(SERC) 소장.
<주관심분야 : 보안, 통신, 인공지능, 네트워크>

이 도 원(정회원)

• 2004년도 원광대학교 전기전자 및 정보통신 공학부 졸업.
• 2007년도 원광대학교 일반대학교 컴퓨터 공학 석사 졸업.
• 2009년도 현재 한남대학교 일반대학원 컴퓨터 공학 박사 4학기 과정.
<주관심분야 : 보안, 웹서비스, 네트워크>