

# Advanced trojan in Grub

CoolQ



X'con 2005

# Contents

- ❖ Overview
- ❖ Boot Process with Grub
- ❖ Possibility to load specified file
- ❖ Hacking techniques
- ❖ Usage
- ❖ Detection
- ❖ Note: The test environment is based on Linux/Ext2/3



# Overview

- ◆ In 1989, the 1<sup>st</sup> trojan horse appeared
  - ◆ Modify utmp, wtmp and lastlog, evade commands such as who, last, w
- ◆ LRK4/LRK5
  - ◆ Replace user-mode applications, such as ps, ls, netstat .....
- ◆ Knark/adore/adore-ng
  - ◆ LKM trojan, apply to Linux 2.2/2.4/2.6
- ◆ SuckIT
  - ◆ Via /dev/kmem
- ◆ Module injection
- ◆ Static kernel patching



# What remains untouched?

## Boot Loader !

Ø Grub

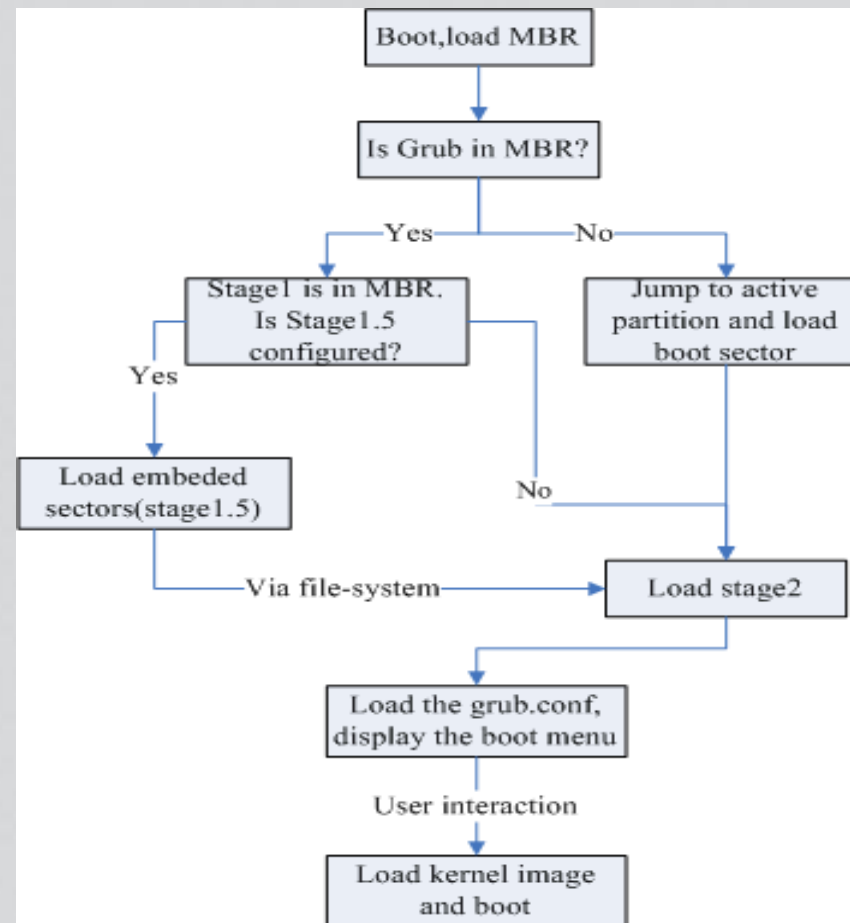
Ø Lilo

Ø ...





# Boot process



# stage1

- ◆ stage1.S
- ◆ 512 Bytes in size
- ◆ Located in MBR or boot sector of partition
- ◆ Its task is
  - ◆ Load specified sector(stage2\_sector) to
    - ◆ 0200:0000  $\beta$ if stage1.5 is configured
    - ◆ 0800:0000  $\beta$ if stage1.5 is not configured



# stage1.5 and stage2

## File list

-rw-r--r--	1	root	root	82		Feb 5	11:24	device.map
-rw-r--r--	1	root	root	10848	Feb 5		11:24	e2fs_stage1_5
-rw-r--r--	1	root	root	9744		Feb 5	11:24	fat_stage1_5
-rw-r--r--	1	root	root	8864		Feb 5	11:24	ffs_stage1_5
-rw-----	1	root	root	800		Jun 6	14:53	grub.conf
-rw-----	1	root	root	800		Jun 6	14:53	menu.lst
-rw-r--r--	1	root	root	9248		Feb 5	11:24	minix_stage1_5
-rw-r--r--	1	root	root	12512	Feb 5		11:24	reiserfs_stage1_5
-rw-r--r--	1	root	root	54044	Sep 5		20:01	splash.xpm.gz
-rw-r--r--	1	root	root	108328	May 23		14:21	stage2
-rwxr-xr-x	1	root	root	512		May 22	13:31	stage1
-rw-r--r--	1	root	root	8512		Feb 5	11:24	vstafs_stage1_5





# How to compile?

## e2fs\_stage1\_5:

```
gcc -o e2fs_stage1_5.exec -nostdlib -WI,-N -WI,-Ttext -WI,2000
e2fs_stage1_5_exec-start.o e2fs_stage1_5_exec-asm.o
e2fs_stage1_5_exec-common.o e2fs_stage1_5_exec-char_io.o
e2fs_stage1_5_exec-disk_io.o e2fs_stage1_5_exec-stage1_5.o
e2fs_stage1_5_exec-fsys_ext2fs.o e2fs_stage1_5_exec-bios.o
objcopy -O binary e2fs_stage1_5.exec e2fs_stage1_5
```





# How to compile?(Cont.)

## Stage2

---

```
gcc -o pre_stage2.exec -nostdlib -WI,-N -WI,-Ttext -WI,8200
pre_stage2_exec-asm.o pre_stage2_exec-bios.o pre_stage2_exec-boot.o
pre_stage2_exec-builtins.o pre_stage2_exec-common.o
pre_stage2_exec-char_io.o pre_stage2_exec-cmdline.o
pre_stage2_exec-disk_io.o pre_stage2_exec-gunzip.o
pre_stage2_exec-fsys_ext2fs.o pre_stage2_exec-fsys_fat.o
pre_stage2_exec-fsys_ffs.o pre_stage2_exec-fsys_minix.o
pre_stage2_exec-fsys_reiserfs.o pre_stage2_exec-fsys_vstafs.o
pre_stage2_exec-hercules.o pre_stage2_exec-serial.o
pre_stage2_exec-smp-imps.o pre_stage2_exec-stage2.o pre_stage2_exec-md5.o
```

```
objcopy -O binary pre_stage2.exec pre_stage2
cat start pre_stage2 > stage2
```



# File layout

## ◆ e2fs\_stage1\_5

[start.S] [asm.S] [common.c] [char\_io.c] [disk\_io.c]  
[stage1\_5.c] [fsys\_ext2fs.c] [bios.c]

## ◆ stage2

[start.S] [asm.S] [bios.c] [boot.c] [builtins.c] [common.c]  
[char\_io.c] [cmdline.c][disk\_io.c] [gunzip.c] [fsys\_ext2fs.c]  
[fsys\_fat.c] [fsys\_ffs.c]  
[fsys\_minix.c] [fsys\_reiserfs.c] [fsys\_vstafs.c] [hercules.c]  
[serial.c]  
[smp-imps.c] [stage2.c] [md5.c]

◆ **start.S is the sector that stage1 loads, 512B in size**



# Sector list of start.S

```
blocklist_default_start:
    .long 2                /* this is the sector start parameter, in logical
                          sectors from the start of the disk, sector 0 */

blocklist_default_len:
#ifdef STAGE1_5           /* this is the number of sectors to read */
    .word 0
/* the command "install" will fill this up */
#else
    .word (STAGE2_SIZE + 511) >> 9
#endif
blocklist_default_seg:
#ifdef STAGE1_5
    .word 0x220
#else
    .word 0x820           /* this is the segment of the
                          starting address to load the data into */
#endif
firstlist:               /* this label has to
                          be after the list data!!! */
```





# An example

```
# hexdump -x -n 512 /boot/grub/stage2
```

...

```
00001d0 [ 0000  0000  0000  0000 ][ 0000  0000  0000  0000 ]  
00001e0 [ 62c7  0026  0064  1600 ][ 62af  0026  0010  1400 ]  
00001f0 [ 6287  0026  0020  1000 ][ 61d0  0026  003f  0820 ]
```

We should interpret(backwards) it as: (8 bytes a time)

- ❖ load 0x3f sectors(start with No.0x2661d0) to 0x0820:0000
- ❖ load 0x20 sectors(start with No.0x266287) to 0x1000:0000
- ❖ load 0x10 sectors(start with No.0x2662af) to 0x1400:00
- ❖ load 0x64 sectors(start with No.0x2662c7) to 0x1600:0000

With the help of this list, stage1.5 can load **itself** without using file-system of OS





## The connection between stage1.5 and stage2

- ◆ If stage1.5 is configured, stage1 loads the 1<sup>st</sup> sector of stage1.5(start.S). Start.S uses its sector list to load the rest part of stage1.5. Then, stage1.5 uses its mini file-system to load stage2
- ◆ If stage1.5 is not configured, stage1 loads the 1<sup>st</sup> sector of stage2(start.S). Start.S uses its sector list to load the rest part of stage2.
- ◆ So, If you rename /boot/grub/stage2 to stage2.bak, when stage1.5 is configured, boot fails; while when not, boot remains OK.



# Grub utils

```
# grub
grub > find /grub/stage2
grub > find /boot/grub/stage2
(hd0,0)
grub > root (hd0,0)
grub > setup (hd0)
grub > setup (hd0,0)
Checking if "/boot/grub/stage1" exists... yes
Checking if "/boot/grub/stage2" exists... yes
Checking if "/boot/grub/e2fs_stage1_t" exists... yes
Running "embed /boot/grub/e2fs_stage1_5 (hd0)"... 22 sectors are
embedded succeeded.
Running "install /boot/grub/stage1 d (hd0) (hd0)1+22 p
(hd0,0)/boot/grub/stage2 /boot/grub/grub.conf"... succeeded
Done
```

- ⊠ If you have separate boot partition
- ⊠ If you don't have separate boot partition
- <= This is the output of 'find' command
- ⊠ Set root of boot partition
- ⊠ If you want to install grub in MBR
- ⊠ If you want to install grub in boot sector

<= If you want to install grub in boot sector, this step fails



# Possibility to load specified file

Grub uses its own mini file-system to read files of ext2/ext3

```
/* preconditions:  ext2fs_mount already executed, therefore supblk in buffer
 *
 *                known as SUPERBLOCK
 * returns:       0 if error, nonzero iff we were able to find the file
 *
 *                successfully
 * postconditions: on a nonzero return, buffer known as INODE contains the
 *
 *                inode of the file we were trying to look up
 * side effects:  messes up GROUP_DESC buffer area
 */
```

```
int ext2fs_dir (char *dirname) {
    int current_ino = EXT2_ROOT_INO;    /*start at the root */
    int updir_ino = current_ino;       /* the parent of the current directory */
    ...
}
```





# kernel=/boot/vmlinuz-2.6.11 ro root=/dev/hda1

- ◇ grub\_open ( ) à ext2fs\_dir("kernel=/boot/vmlinuz-2.6.11 ro root=/dev/hda1")
- ◇ INODE à i\_blocks[ ]
- ◇ ext2fs\_dir internals
  - ◇ /boot/vmlinuz-2.6.11 ro root=/dev/hda1  
^ inode = EXT2\_ROOT\_INO, put inode info of '/' to INODE
  - ◇ /boot/vmlinuz-2.6.11 ro root=/dev/hda1  
^ find 'boot' entry in '/', put inode info of '/boot' to INODE
  - ◇ /boot/vmlinuz-2.6.11 ro root=/dev/hda1  
^ find 'vmlinuz-2.6.11' entry in '/boot', put inode info of vmlinuz-2.6.11 to INODE
  - ◇ /boot/vmlinuz-2.6.11 ro root=/dev/hda1  
^ Now the pointer is space, INODE contains regular file, function returns 1(success), INODE contains inode info of vmlinuz-2.6.11





# What if ...

- ❖ `/boot/vmlinuz-2.6.11 ro root=/dev/hda1`  
^ `inode = EXT2_ROOT_INO`
- ❖ `boot/vmlinuz-2.6.11 ro root=/dev/hda1`  
^ change `'/'` to `0x0`, change `EXT2_ROOT_INO` to `inode of file_fake`
- ❖ `boot/vmlinuz-2.6.11 ro root=/dev/hda1`  
^ read `inode` info of `file_fake` to `INODE`, the `pointers` points to `0x0`, `INODE` contains regular file, return `1(success)`
- ❖ Result: `inode` info of `fake_file` is fetched to `INODE`, `grub` considers `file_fake` as `vmlinuz-2.6.11`



# Side effects?

- ✦ We have modified the parameter of `ext2fs_dir`, the latter part “`ro root=/dev/hda1`” is passed to kernel as boot parameters, do we need to change it back when returning from `ext2fs_dir`?



# kernel=...

```
static int
kernel_func (char *arg, int flags)
{
    ...
    /* Copy the command-line to MB_CMDLINE. */
    grub_memmove (mb_cmdline, arg, len + 1);
    kernel_type = load_image (arg, mb_cmdline, suggested_type, load_flags);
    ...
}
1) strcmp(mb_cmdline, arg) == 0 && mb_cmdline != arg
2) In load_image function, mb_cmdline and arg are unrelated
```

**So, no need to change 0x0 à ‘/’**





# Hacking techniques

- ◆ how to load file\_fake
- ◆ how to locate ext2fs\_dir
- ◆ how to hack grub
- ◆ how to make things sneaky





# how to load file\_fake

- 1) JMP at the beginning of ext2fs\_dir
- 2) Change the 1<sup>st</sup> char of ext2fs\_dir's parameter to 0x0
- 3) current\_ino = EXT2\_ROOT\_INO  
=>  
current\_ino = INODE\_OF\_FILE\_FAKE  
(In certain case)
- 4) JMP back



How to implement `current_ino = INODE_OF_FILE_FAKE`

```
int ext2fs_dir (char *dirname) {
    int current_ino = EXT2_ROOT_INO;          /*start at the root */
    int updir_ino = current_ino;             /* the parent of the current directory */
    ..
```

```
c7 85 e4 fb ff ff 02    movl    $0x2,0xffffbbe4(%ebp)
00 00 00
c7 85 e0 fb ff ff 02    movl    $0x2,0xffffbe0(%ebp)
00 00 00
c7 85 d8 fb ff ff 00    movl    $0x0,0xffffbd8(%ebp)
00 00 00
```

Optimized result might be

"movl \$2, %reg"

"movl %reg, 0xffffXXXX(\$esp)"

"movl %reg, 0xffffYYYY(\$esp)"

Other cases? Low in possibility

xor %eax, %eax; inc %eax; inc %eax

xor %eax, %eax; movb \$0x2, %al



ext2fs\_dir

# Our method

```
push %ebp
jmp embed
mov %esp, %ebp
push %edi
push %esi
sub $0x42c, %esp
mov $2, 0xffffbe4(%esp)
mov $2, 0xffffbe0(%esp)
back:
```

**embed**

```
Save registers
Compare strings
If match, goto 1
else goto 2
1: restore registers
jmp change_inode
2: restore registers
jmp not_change_inode
```

**not\_**  
**change\_inode**

```
push %ebp
mov %esp, %ebp
mush %edi
push %esi
sub $0x42c, %esp
mov $2, 0xffffbe4(%esp)
mov $2, 0xffffbe0(%esp)
jmp back
```

**change\_inode**

```
push %ebp
mov %esp, %ebp
mush %edi
push %esi
sub $0x42c, %esp
mov $?, 0xffffbe4(%esp)
mov $?, 0xffffbe0(%esp)
jmp back
```

INODE\_OF\_ FAKE\_FILE





## How to locate ext2fs\_dir()

- ❖ Because ext2fs\_dir is generated by objcopy, all ELF infos are stripped, NO SYMBOL TABLE! So we have to use other hacks to locate this function.



# 1<sup>st</sup> try

```
#define long2(n) ffz(~(n))
static __inline__ unsigned long
ffz (unsigned long word)
{
    __asm__ ("bsfl %1, %0"
            : "=r" (word)
            : "r" (~word));
    return word;
}
group_desc = group_id >> log2 (EXT2_DESC_PER_BLOCK (SUPERBLOCK));
```

ffz is declared as `__inline__`, so the find result is hard to predict, MAYBE inline, MAYBE not, so we give it up!



## 2<sup>nd</sup> try

### ◆ SUPERBLOCK->s\_inodes\_per\_group

```
group_id = (current_ino - 1) / (SUPERBLOCK->s_inodes_per_group);
```

```
#define RAW_ADDR(x) (x)
```

```
#define FSYS_BUF RAW_ADDR(0x68000)
```

```
#define SUPERBLOCK ((struct ext2_super_block *) (FSYS_BUF))
```

```
struct ext2_super_block{
```

```
...
```

```
__u32 s_inodes_per_group /* # Inodes per group */
```

```
...
```

SUPERBLOCK->s\_inodes\_per\_group is at 0x68028, search backward for the beginning of function

### ◆ Question

◆ How to locate RET? Search backward for 0xc3?

◆ How to locate the beginning of ext2fs\_dir? Function align(4/8/16 bytes, junk codes vary)

◆ Conclusion: practical but not reliable





# 3<sup>rd</sup> try

❖ At last, we noticed `fsys_table`

```
struct fsys_entry fsys_table[NUM_FSYS + 1] =  
{  
    ...  
    # ifdef FSYS_FAT  
        {"fat", fat_mount, fat_read, fat_dir, 0, 0},  
    # endif  
    # ifdef FSYS_EXT2FS  
        {"ext2fs", ext2fs_mount, ext2fs_read, ext2fs_dir, 0, 0},  
    # endif  
    # ifdef FSYS_MINIX  
        {"minix", minix_mount, minix_read, minix_dir, 0, 0},  
    # endif
```

`fsys_table` is called like this:

```
((*(fsys_table[fsys_type].mount_func)) () != 1)
```



# Our method

- ❖ Search stage2 for string "ext2fs", get its offset, then convert it to memory address(stage2 starts from 0800:0000) addr\_1.
- ❖ Search stage2 for addr\_1, get its offset, then get next 5 integers (A, B, C, D, E),  $A < B$  ?  $B < C$  ?  $C < \text{addr\_1}$  ?  $D == 0$  ?  $E == 0$  ? If any one is "No", goto 1 and continue search
- ❖ Then C is memory address of ext2fs\_dir, convert it to file offset. OK, that's it



# How to hack grub

◆ With the help of above, things are much easizer. But at the beginning of `ext2fs_dir`, where should we jump to?

◆ The tail of `stage2`? It will change the size of `stage2` (more...)

◆ `fat_mount`(It's right after `ext2fs_dir`)?

◆ **NO!**

```
root_func()->open_device()->attemp_mount()
for (fsys_type = 0; fsys_type < NUM_FSYS
    && (*(fsys_table[fsys_type].mount_func)) () != 1; fsys_type++);
```

Fat is ahead of `ext2fs`, so `fat_mount` will run before `ext2fs_mount`.

◆ At last, we choose `minix_dir`





# how to make things sneaky

- ❖ Drawback of the above method: the checksum of stage2 changes
- ❖ Countermeasure: let stage1 loads stage2\_fake
- ❖ Notice:
  - ❖ Refill the sector list of stage2\_fake
  - ❖ If stage1.5 is not configured, let stage1 load stage2\_fake directly(), 修改stage1直接调用 stage2\_fake(stage2\_sector should be the sector number of stage2\_fake), this may change MBR



# how to make things sneaky(Cont.)

- ◆ If stage1.5 is configured,
  - ◆ Modify stage1 to bypass stage1.5, load stage2 directly (modify stage2\_sector,stage2\_address,stage2\_segment)
    - ◆ Drawbacks: MBR and boot messages change
  - ◆ Use the same techniques to modify the file-system of stage1.5, refill the sector list of stage1.5
- ◆ You can hide stage2\_fake and file\_fake as well
- ◆ Wanna anti-FSCK? No problem...



# Usage

- ❖ Combined with static kernel patching
  - 1)cp kernel.orig kernel.fake
  - 2)Static kernel patch with kernel.fake
  - 3)cp stage2 stage2.fake
  - 4)hack\_grub stage2.fake kernel.orig  
inode\_of\_kernel.fake
  - 5)Hide kernel.fake and stage2.fake  
(Optional)





# Usage(Cont.)

- ❖ Combined with module injection
  - 1)cp initrd.img.orig initrd.img.fake
  - 2)Do module injection with initrd.img.fake, e.g. ext3.[k]o
  - 3)cp stage2 stage2.fake
  - 4)hack\_grub stage2.fake initrd.img inode\_of\_initrd.img.fake
  - 5)Hide initrd.img.fake and stage2.fake (Optional)
- ❖ Use fake grub.conf
- ❖ More...



# Detection

- 1) Keep an eye on MBR and the following 63 sectors, also primary boot sectors.
- 2) If not 1), then
  - a) If stage1.5 is configured, compare sectors from 3 (absolute address, MBR is sector No. 1) with /boot/grub/e2fs\_stage1\_5
  - b) if stage1.5 is not configured, see if stage2\_sector points to real /boot/grub/stage2 file
- 3) check the file consistency of e2fs\_stage1\_5 and stage2
- 4) If not 3), things are more difficult (Hey, are you a qualified sysadmin?)
  - a) If you're suspicious about kernel, dump the kernel and make a byte-to-byte with kernel on disk.
  - b) If you're suspicious about module, that's a hard challenge, maybe you can dump it and disassemble it?



# What about Lilo?

- ❖ Lilo doesn't have built-in file-system, so, no need to patch mini built-in file-system like Grub.
- ❖ Lilo relies on /boot/bootsect.b and /boot/map.b
- ❖ Lazy way: lilo -C fake\_config
- ❖ More details? Depends on yourself...





# Thanks to ...

- ✦ madsys & grip2 for help me solve some hard-to-crack things
- ✦ airsupply and other guys for stage2 samples
- ✦ zhtq for some comments about paper-writing



# References

- ◆ Design and Implementation of the Second Extended Filesystem
- ◆ Static Kernel Patching
- ◆ Infecting Loadable Kernel Modules
- ◆ module injection in 2.6 kernel
- ◆ Ways to hide files in ext2/3 filesystem
- ◆ Ways to find 2.6 kernel rootkits



# Questions & Answers

