

# 高级木马之Grub篇

CoolQ



X'con 2005

# 主要内容

- ◆ 前言
- ◆ **Grub**的启动过程
- ◆ 装载指定文件的可能性
- ◆ 技术细节
- ◆ 应用
- ◆ 检测
- ◆ 注意，本文的环境是Linux/Ext2/3



# 前言

- ◆ **1989年出现了第一个木马**
  - ◆ 修改utmp,wtmp和lastlog,躲避who,w,last命令
- ◆ **LRK4/LRK5**
  - ◆ 替换用户态程序,例如ps,ls,netstat...
- ◆ **knark/adore/adore-ng**
  - ◆ 内核模块形式,针对2.2/2.4/2.6
- ◆ **SuckIT**
  - ◆ 通过/dev/kmem
- ◆ 模块注入技术
- ◆ 静态内核补丁



还有什么地方没有考虑？

## Boot Loader!

Ø Grub

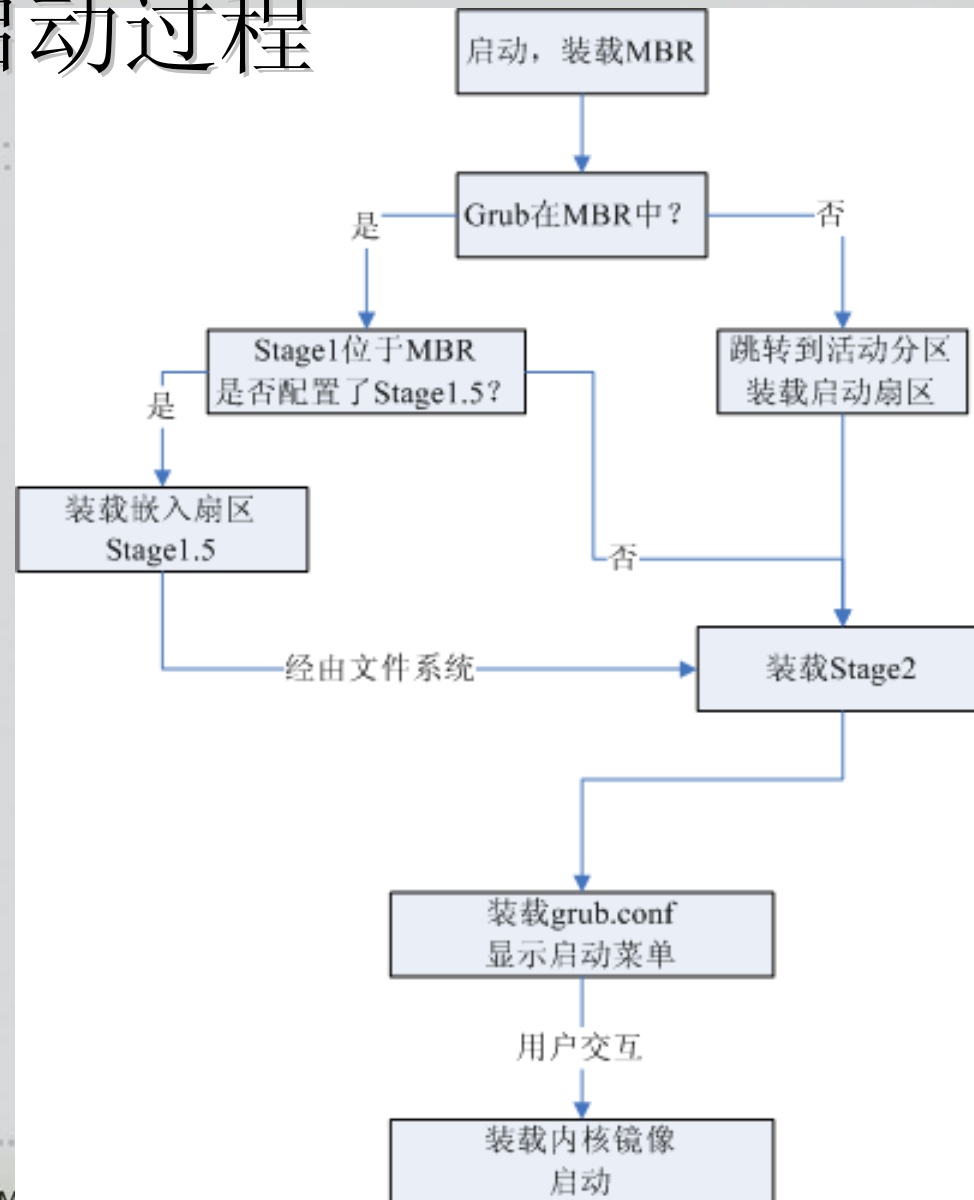
Ø Lilo

Ø . . .



# Grub的启动过程

综述



# stage1

- ◆ stage1.S
- ◆ 大小512字节
- ◆ 位于**MBR**或者分区的启动扇区里
- ◆ 具体的工作
  - ◆ 将指定扇区（**stage2\_sector**）装载到
    - ◆ 0200:0000 **β**如果配置stage1.5
    - ◆ 0800:0000 **β**如果没配置stage1.5



## stage1.5和stage2

## 文件列表

```

-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

```



# 编译过程

e2fs\_stage1\_5:

```
gcc -o e2fs_stage1_5.exec -nostdlib -Wl,-N -Wl,-Ttext -  
-Wl,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
```



# 编译过程(续)

## Stage2

```
gcc -o pre_stage2.exec -nostdlib -Wl,-N -Wl,-Ttext -Wl,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-stage
pre_stage2_exec-md5.o
objcopy -O binary pre_stage2.exec pre_stage2
cat start pre_stage2 > stage2
```



# 文件结构

## ❖ 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]

❖ 其中start.S就是stage1装载的扇区，512B



# start.S的扇区列表

```

blocklist_default_start:
    .long 2                /* this is the sector start parameter, in
    logical                sectors from the start of the disk,
                            sector 0 */
    .word 0                /* this is the number of sectors
                            to read */
#ifdef STAGE1_5
    .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

```

# 一个例子

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

```
...
```

```
00001d0 [ 0000  0000  0000  0000 ] [ 0000  0000  0000  0000
00001e0          [ 62c7  0026  0064  1600 ] [ 62af  0026  001
  1400 ]
00001f0          [ 6287  0026  0020  1000 ] [ 61d0  0026  003
  0820 ]
```

我们应该从后往前看，每次8个字节

- 将从0x2661d0扇区开始的0x3f个扇区装载到从0820:0000开始的内存中
- 将从0x266287扇区开始的0x20个扇区装载到从1000:0000开始的内存中
- 将从0x2662af扇区开始的0x10个扇区装载到从0x1400:0000开始的内存中
- 将从0x2662c7扇区开始的0x64个扇区装载到从0x1600:0000开始的内存中

有了这个列表，stage1.5就不需要依靠文件系统来装载自己了

## stage1.5与stage2的关系

- ❖ 如果配置了stage1.5，stage1首先将stage1.5的第一个扇区(start.S)读入内存，依靠start.s的扇区列表将stage1.5全部读入内存，然后stage1.5会用自己的文件系统将stage2读入内存。此时stage2的start.S的扇区列表不被使用。
- ❖ 如果没有配置stage1.5，stage1会将stage2的第一个扇区(start.S)读入内存，然后依靠扇区列表将stage2全部读入内存
- ❖ 因此，如果把/boot/grub/stage2改名为stage2.bak，在配置了stage1.5的情况下，启动会失败；如果没有配置，启动仍然不受影响。



# Grub工具

```
# 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
```

ⓑ 如果你有单独的启动分区

ⓑ 如果你没有有单独的启动分区

<= 这是'find'命令的结果

ⓑ 设置启动分区的根

ⓑ 如果你想把grub装在mbr中

ⓑ 如果你想把grub装载启动扇区里

<= 如果你想把grub安装到启动扇区, 这一步会失败



# 装载指定文件的可能性

Grub使用自带的小型文件系统来读取ext2/ext3上的文件

```
/* 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的内部实现

◆ /boot/vmlinuz-2.6.11 ro root=/dev/hda1

^ inode = EXT2\_ROOT\_INO, 将 / 的inode信息保存到INODE中

◆ /boot/vmlinuz-2.6.11 ro root=/dev/hda1

^ 在 / 中查找 'boot' 的条目, 并将 '/boot' 的inode信息放到INODE中

◆ /boot/vmlinuz-2.6.11 ro root=/dev/hda1

^ 在 '/boot' 中查找 vmlinuz-2.6.11 的条目, 并将 vmlinuz-2.6.11 的inode信息存放到INODE中

◆ /boot/vmlinuz-2.6.11 ro root=/dev/hda1

^ 现在指针指向了空格, INODE里是常规文件, 函数返回1(成功), INODE中包含了 vmlinuz-2.6.11 的inode信息



## 如果...

- ❖ `/boot/vmlinuz-2.6.11 ro root=/dev/hda1`  
^ `inode = EXT2_ROOT_INO`
- ❖ `boot/vmlinuz-2.6.11 ro root=/dev/hda1`  
^ 把/改成0x00, 把EXT2\_ROOT\_INO改成file\_fake的inode号
- ❖ `boot/vmlinuz-2.6.11 ro root=/dev/hda1`  
^ 把file\_fake的inode信息读到INODE, 指针的内容为0x0, INODE又是常规文件, 返回1 (成功)。
- ❖ 结果:fake\_file的inode信息被读入INODE中, grub认为file\_fake就是vmlinuz-2.6.11



## 副作用？

- ❖ 我们修改了 `ext2fs_dir` 的参数，后面的“`root=/dev/hda1`”还要传给内核作为启动参数，我们是否需要在函数返回的时候再改回来呢？



# 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) load\_image函数中, mb\_cmdline和arg互不相关

因此, 不需要将0x0 à `/'



## 技术细节

- ❖ 如何装载file\_fake
- ❖ 如何定位ext2fs\_dir函数
- ❖ 如何hack grub
- ❖ 如何变得更隐蔽



## 如何装载file\_fake

- 1) 在ext2fs\_dir开头用JMP跳出
- 2) 将ext2fs\_dir参数的第一个字符变成0x0
- 3) `current_ino = EXT2_ROOT_INO`  
=>  
`current_ino =`  
`INODE_OF_FILE_FAKE`  
(在某些条件下)
- 4) 跳回



如何实现 `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    $0x2,0xffffffe4(%ebp)
00 00 00
c7 85 e0 fb ff ff 02    movl    $0x2,0xffffffe0(%ebp)
00 00 00
c7 85 d8 fb ff ff 00    movl    $0x0,0xfffffd8(%ebp)
00 00 00
```

优化的情况

“`movl $2, %reg`”

“`movl %reg, 0xffffXXXX($esp)`”

“`movl %reg, 0xffffYYYY($esp)`”

其它的情况？可能性较低

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

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



## 我们的方法

ext2fs\_dir

```

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

```

embed

保存环境  
比较字符串  
如果符合，转1  
否则转2  
1: 恢复环境  
jmp change\_inode  
2: 恢复环境  
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

INODE\_OF\_  
FAKE\_FILE

```

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

```



## 如何定位ext2fs\_dir函数

- ✦ 由于stage2是由objcopy生成的，因此所有的ELF信息都被去除了，没有符号表，定位ext2fs\_dir就必须另找他法。



# 尝试一

```
#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是\_\_inline\_\_，因此最后的编译结果难以预测，可能展开，也可能不展开，放弃！



## 尝试二

### ◆ 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位于0x68028, 往回寻找函数的开始

### ◆ 问题

- ◆ 如何寻找RET? 单纯往回寻找0xc3?
- ◆ 如何确定ext2fs\_dir开始? 函数的对齐(4/8/16字节, 指令不定)

◆ 结论: 可行但不可靠



## 尝试三

❖ 最后，我们注意到了 `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`是这样被调用的:

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



## 我们的方法

- 1) 在stage2中搜索“ext2fs”字符串，获得它在stage2中的偏移量，并转换成在内存中的地址（stage2从0800:0000开始），记作addr\_1
- 2) 在stage2中搜索addr\_1，获得它的偏移量，并继续读取5个int，记作（A， B， C， D， E），然后判断  $A < B$ ?  $B < C$ ?  $C < \text{addr\_1}$ ?  $D == 0$ ?  $E == 0$ ? 如果任何一个不成立，回到1) 继续搜索
- 3) 最后C就是我们想要的ext2fs\_dir地址



# 如何hack grub

- ◆ 有了前面的内容，我们就方便多了。但在ext2fs\_dir一开始的JMP到底应该跳到什么位置呢？
  - ◆ stage2文件的尾部？会改变stage2文件的大小(more...)
  - ◆ fat\_mount(正好在ext2fs\_dir的后面)？

◆ **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在ext2fs前面，因此fat\_mount在ext2fs\_mount前运行

- ◆ 最后，我们选择了minix\_dir



## 如何变得更隐蔽

- ◆ 上述方法的缺点：改变了stage2的校验和
- ◆ 对策：让stage1装载stage2\_fake
- ◆ 注意事项：
  - ◆ 重写stage2\_fake的扇区列表
  - ◆ 如果没有配置stage1.5，修改stage1直接调用stage2\_fake(stage2\_sector填入stage2\_fake的扇区地址)，这样有可能修改MBR



## 如何变得更隐蔽(Cont.)

- ◆ 如果配置了stage1.5
  - ◆ 直接修改stage1跳过stage1.5, 装载stage2(修改stage2\_sector,stage2\_address,stage2\_segment)
    - ◆ 问题: MBR改变 / 启动信息改变
  - ◆ 利用相似的技术修改stage1.5的文件系统, 重写stage1.5的扇区列表
- ◆ 你可以继续隐藏stage2\_fake和file\_fake
- ◆ Wanna anti-FSCK? No problem...



# 应用

## ❖ 与静态补丁相结合

- 1) `cp kernel.orig kernel.fake`
- 2) 对`kernel.fake`打静态补丁
- 3) `cp stage2 stage2.fake`
- 4) `hack_grub stage2.fake kernel.orig`  
`inode_of_kernel.fake`
- 5) 隐藏`kernel.fake`和`stage2.fake` (可选)



## 应用(Cont.)

### ◆ 与模块注射相结合

- 1) `cp initrd.img.orig initrd.img.fake`
- 2) 对 `initrd.img.fake` 进行模块注入, 例如. `ext3.[k]o`
- 3) `cp stage2 stage2.fake`
- 4) `hack_grub stage2.fake initrd.img`  
`inode_of_initrd.img.fake`
- 5) 隐藏 `initrd.img.fake` 和 `stage2.fake` (可选)

### ◆ 使用假的 `grub.conf`

### ◆ More...



# 检测

- 1) 注意**MBR**以及后边的**63**个扇区，还要关注每个分区的启动分区
- 2) 如果没有**1)**，那么
  - a) 如果配置了**stage1.5**，从第三个扇区开始（绝对扇区，第一个扇区是**MBR**）与**/boot/grub/e2fs\_stage1\_5**做比较
  - b) 如果没有配置**stage1.5**，看**stage2\_sector**是否指向真实的**/boot/grub/stage2**文件
- 3) 检查**/boot/grub/stage2**文件和**/boot/grub/e2fs\_stage1\_5**的文件完整性
- 4) 如果**3)**也无法完成，就比较困难了（不合格的管理员）
  - a) 如果怀疑内核有问题，可以与磁盘上的内核文件进行逐字节的比较
  - b) 如果怀疑模块有问题，可以把它**dump**出来并进行反汇编



## Lilo呢？

- ❖ Lilo没有文件系统，因此不需要像Grub那样给内建的mini-FS打补丁
- ❖ Lilo主要依靠/boot/bootsect.b和/boot/map.b
- ❖ 一个懒的方法：`lilo -C fake_config`
- ❖ 更多的细节？ Depends on yourself...



# 感谢

- ❖ 本文的许多地方得到了**madsys**和**grip2**的帮助
- ❖ 测试使用的多个发行版的**stage2**文件由**airsupply**及热心网友提供
- ❖ 全文由**zhtq**帮助审稿



## 参考

- ❖ 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

