将需要 inline hook 的函数机器码先拷贝到一块内存中， 然后动态修正这块内存的 call,jmp 对应的 offset，然后给原始函数做个 inline hook, 跳到新函数去执行， 新函数做一些过滤判断后， 调用新分配的内存保存的机器码，想当于调用原始函数。这种技术在实战中没什么用， 拓展下思路还是不错的。

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/version.h>
#include <linux/string.h>
#include <linux/list.h>
#include <asm/processor.h>
#include <asm/uaccess.h>
#include <asm/unistd.h>
#include <asm/cacheflush.h>

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

#define OPCODE_LEN          38

unsigned char orig_opcode[5];
void (*new_fn)(char *, int n);
unsigned char *s;

void test(char *str, int n)
{
int i = 0;

for (;i < n ; i++) {
printk("%s\n", str);
}
}

void test1(char *str, int n)
{
int i;

if (n > 3) {
test("aa", 1);
}
test(str, n);
}
```

```
void new_test1(char *str, int n)
{
printk("in new_test1.\n");
new_fn(str,n);
}

void test2(char *str, int n)
{
test("aaa", 2);
}

void copy_opcode(void)
{
unsigned char *p;
unsigned char buf[4] = "\x00\x00\x00\x00";
unsigned int orig_func_addr = 0;
int orig_offset, new_offset;
int i;

printk("orig func addr: 0x%x\n", test);

p = (unsigned char *)test1;
for (i = 0; i < OPCODE_LEN; i++) {
printk("0x%02x", p[i]);
}

s = kmalloc(100, GFP_KERNEL);
memset(s, 0x0, 100);
printk("\ns: 0x%x\n", s);

for (i = 0; i < OPCODE_LEN; i++) {
s[i] = p[i];
if (p[i] == 0xe8 || p[i] == 0xe9) {
printk("\nfound 0x%x: 0x%x\n", p[i], &p[i]);

buf[0] = p[i + 1];
buf[1] = p[i + 2];
buf[2] = p[i + 3];
buf[3] = p[i + 4];

orig_offset = *(unsigned int *)buf;
printk("orig offset: 0x%x\n", orig_offset);

orig_func_addr = orig_offset + (unsigned int)&p[i] + 5;
printk("orig func addr: 0x%x\n", orig_func_addr);

new_offset = orig_func_addr - (unsigned int)&s[i] - 5;
printk("new_offset: 0x%x\n", new_offset);

i++;
s[i++] = (new_offset & 0x000000ff);
s[i++] = (new_offset & 0x0000ff00) >> 8;
s[i++] = (new_offset & 0x00ff0000) >> 16;
s[i++] = (new_offset & 0xff000000) >> 24;
}
}

printk("\n");
for (i = 0; i < OPCODE_LEN; i++) {
printk("0x%02x", s[i]);
}
printk("\n");

new_fn = s;
}

static int inline_hook_func(unsigned int old_func, unsigned int new_func,
unsigned char *old_opcode)
{
unsigned char *p;
unsigned int offset;

p = (unsigned char *)old_func;
memcpy(old_opcode, p, 5);

offset = (unsigned int)new_func - (unsigned int)old_func - 5;
*p++ = 0xe9;
//memcpy(buf + 1, &p, 4);
*p++ = (offset & 0x000000ff);
*p++ = (offset & 0x0000ff00) >> 8;
*p++ = (offset & 0x00ff0000) >> 16;
*p++ = (offset & 0xff000000) >> 24;
*p++ = 0x90;

return 0;
}

static int restore_inline_hook(unsigned int old_func, unsigned char *old_opcode)
```

```
{
unsigned char *buf;

buf = (unsigned char *)old_func;
memcpy(buf, old_opcode, 5);

return 0;
}

static int opcode_test_init(void)
{
copy_opcode();
inline_hook_func(test1, new_test1, orig_opcode);
test1("hello,world", 3);

return 0;
}

static void opcode_test_exit(void)
{
printk("unload opcode test module.\n");
}

module_init(opcode_test_init);
module_exit(opcode_test_exit);
```

dmesg:

```
orig func addr: 0xe0b8d02a
0x560x830xfa0x030x530x890xc60x890xd30x7e0x0f0xba0x010x000x000x000xb80x9c0xd20xb80x
e00xe80xb60xff0xff0xff0x890xda0x890xf00x5b0x5e0xe90xab0xff0xff0xff0x57
s: 0xd22b92c0

found 0xe8: 0xe0b8d06f
orig offset: 0xffffffb6
orig func addr: 0xe0b8d02a
new_offset: 0xe8d3d50

found 0xe9: 0xe0b8d07a
orig offset: 0xffffffab
orig func addr: 0xe0b8d02a
new_offset: 0xe8d3d45

0x560x830xfa0x030x530x890xc60x890xd30x7e0x0f0xba0x010x000x000x000xb80x9c0xd20xb80x
e00xe80x500x3d0x8d0x0e0x000xda0x890xf00x5b0x5e0xe90x450x3d0x8d0x0e0x00
```

in new_test1.

hello,world

hello,world

hello,world

hello,world

hello,world

hello,world