LD_PRELOAD

atdog

Hook Function

可以用來攔截 gets, scanf, read …

鉤住之後

可以幫助你做各種事情

Logger, Filter, Reverse, Patch, …

最基本的用法

```
all:
	gcc hook.c -m32 -o hook.so -fPIC -shared -ldl \
	-D_GNU_SOURCE
	gcc -m32 -o main main.c
```

```
ubuntu [~/notes/hook/hook_gets] ➡  ./main
Input: my input
Process: my input
ubuntu [~/notes/hook/hook_gets] ➡  LD_PRELOAD=./hook.so ./main
Input: my input
Hook gets()
Process: my input
```

main.c

```c
#include<stdio.h>

main() {
    char str[50];
    printf("Input: ");
    fflush(stdout);
    gets(str);
    /* process */
    printf("Done and exit. %d \n", test());
}
```

hook.c

```c
#include <stdio.h>
#include <dlfcn.h>
#include <unistd.h>

char *gets(char *s) {
    char* (*old_gets)(char *message);
    old_gets = dlsym(RTLD_NEXT, "gets");

    old_gets(s);

    printf("Hook gets()\n");
}
```

最基本的用法

```
all:
	gcc hook.c -m32 -o hook.so -fPIC -shared -ldl \
	-D_GNU_SOURCE
	gcc -m32 -o main main.c
```

```
ubuntu [~/notes/hook/hook_gets] ➡  ./main
Input: my input
Process: my input
ubuntu [~/notes/hook/hook_gets] ➡  LD_PRELOAD=./hook.so ./main
Input: my input
Hook gets()
Process: my input
```

Logger

Attack & Defence 通常可以做封包分析

但拿到的封包是delay 3-5 rounds 以上的

Logger 可以讓你**第一時間**取得攻擊流量

hook.c

```c
/* Logger */
char *log;
asprintf(&log, "[2014-12-22 00:00:00] Message: %s\n", s);
FILE *fh = fopen("./input.log", "a+");
fwrite(log, strlen(log), 1, fh);
fclose(fh);
```

```
ubuntu [~/notes/hook/hook_gets] ➡ LD_PRELOAD=./hook.so ./main
Input: This is my first logger
Process: This is my first logger
ubuntu [~/notes/hook/hook_gets] ➡ cat input.log
[2014-12-22 00:00:00] Message: This is my first logger
```

Filter - IO

最有效的短解可以靠 filter 來完成

可以把不想要進到程式的 Input 濾掉 (**/bin/sh**, **flag**)

也可以把流出的 flag 濾掉

hook.c

```c
/* Filter */
char *filter_str = "flag";
char* ins = strstr(s, filter_str);
if(ins) {
    strncpy(ins, "****", 4);
}
```

```
ubuntu [~/notes/hook/hook_gets] ➡  LD_PRELOAD=./hook.so ./main
Input: cat flag;
Process: cat ****;
```

Reverse Engineering with LD_PRELOAD

逆向有時候需要 binary patch或是debug
透過 hook 可以幫助快速分析

main.c

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv) {
    char passwd[] = "foobar";

    if (argc < 2) {
        printf("usage: %s <given-password>\n", argv[0]);
        return 0;
    }

    if (!strcmp(passwd, argv[1])) {
        printf("Green light!\n");
        return 1;
    }

    printf("Red light!\n");
    return 0;
}
```

當密碼正確時 Green light
否則 Red light.

```
ubuntu [~/notes/hook/hook_strcmp] ➡  ./main foobar
Green light!
ubuntu [~/notes/hook/hook_strcmp] ➡  ./main test
Red light!
```

Hook 後不管什麼 Password 都可以過

```
ubuntu [~/notes/hook/hook_strcmp] ➡  LD_PRELOAD=./hook.so ./main test
S1 eq foobar
S2 eq test
Green light!
ubuntu [~/notes/hook/hook_strcmp] ➡  LD_PRELOAD=./hook.so ./main abc
S1 eq foobar
S2 eq abc
Green light!
ubuntu [~/notes/hook/hook_strcmp] ➡  LD_PRELOAD=./hook.so ./main foobar
S1 eq foobar
S2 eq foobar
Green light!
```

Hook strcmp() - always return equal

```c
#include <stdio.h>
#include <string.h>

int strcmp(const char *s1, const char *s2) {

    printf("S1 eq %s\n", s1);
    printf("S2 eq %s\n", s2);

    // ALWAYS RETURN EQUAL STRINGS!
    return 0;
}
```

Hook init function

main.c

```c
#include <stdio.h>

void doit() {
    char str[50];
    printf("Input: ");
    fflush(stdout);
    gets(str);
    /* process */
    printf("Process: %s\n", str);
}

main() {
    doit();
}
```

hook.c

```c
#include <stdio.h>

__attribute__ ((constructor)) void init(void) {
    printf("init hooked.\n");
}
```

可以 hook main 之前的 function init

```
ubuntu [~/notes/hook/hook_init] ➡  LD_PRELOAD=./hook.so ./main
init hooked.
Input: test
Process: test
```

當 hook init 後，可以直接 dynamic patch 程式

不會有 binary code長度限制問題
可以輕易控制原程式中的 function
還可以做很多雜七雜八的事情

再看一次 main.c

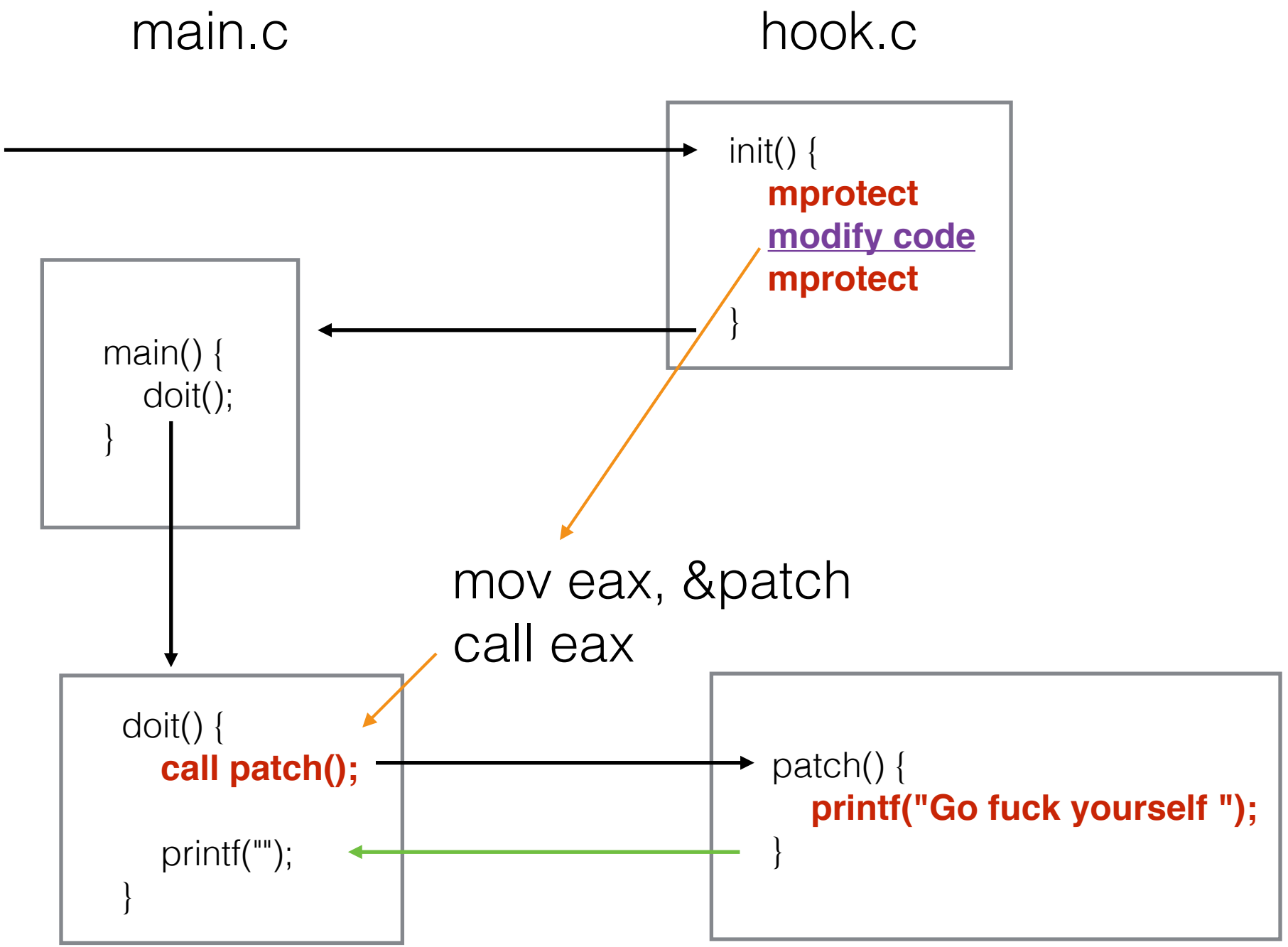
```c
#include <stdio.h>

void doit() {
    char str[50];
    printf("Input: ");
    fflush(stdout);
    gets(str);
    /* process */
    printf("Process: %s\n", str);
}

main() {
    doit();
}
```

hook.c

```c
void patch() {
    printf("go fuck yourself.\n");
}
```


main.c
hook.c
init() {
mprotect
modify code
mprotect
}
main() {
doit();
}
mov eax, &patch
call eax
doit() {
call patch();
printf("");
}
patch() {
printf("Go fuck yourself ");
}

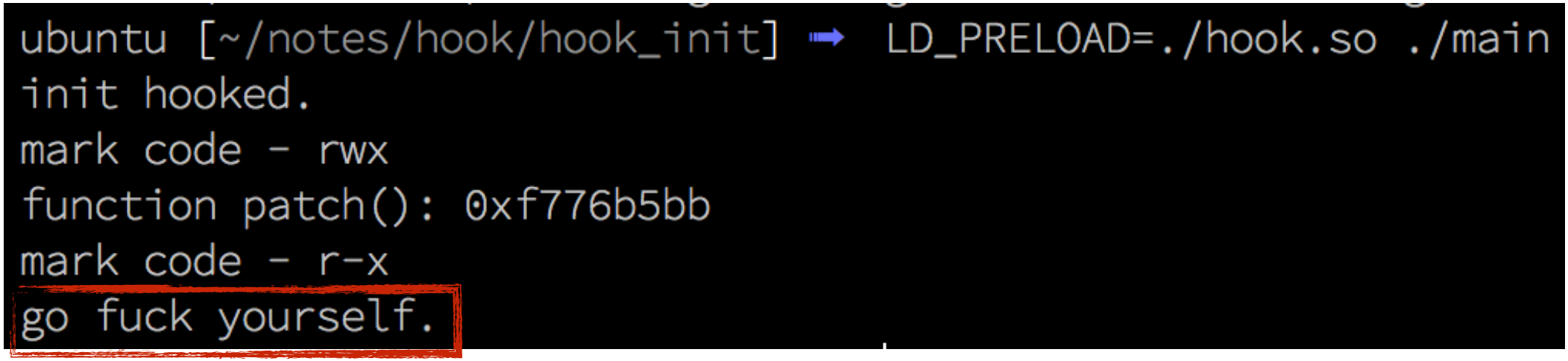
hook.c

```
if( mprotect(0x08048000, 0x1000, PROT_WRITE|PROT_READ|PROT_EXEC) == -1) {
    printf("mprotect fail\n");
}
else {
    printf("mark code - rwx\n");
}

printf("function patch(): %p\n", &patch);
char *ptr_op = 0x080484ed;
*ptr_op = 0xb8;
int *ptr_arg = 0x080484ee;
*ptr_arg = &patch;
ptr_op = 0x080484f2;
*ptr_op = 0xff;
*(ptr_op+1) = 0xe0;

if( mprotect(0x08048000, 0x1000, PROT_READ|PROT_EXEC) == -1) {
    printf("mprotect fail\n");
}
else {
    printf("mark code - r-x\n");
}
```

```
mov eax, &patch
call eax
```


ubuntu [~/notes/hook/hook_init] ➦ LD_PRELOAD=./hook.so ./main
init hooked.
mark code - rwx
function patch(): 0xf776b5bb
mark code - r-x
go fuck yourself.

Block IP

透過 xinetd 啟動的 service 可以從
env['REMOTE_HOST'] 取得**來源 IP**

搭配 Hook 可以直接阻擋其連線

GDB

(gdb) set environment LD_PRELOAD=./hook.so

以上行為都有可能導致比賽中 service check fail

比賽同時也要觀察 service check 可能 fail 的原因來修正為了 defence 設定的保護機制