第三届XMan夏令营

栈溢出102

讲师：Anciety 丁湛钊

目录
Contents


X-Man
Part 01
基本介绍
个人介绍、战队介绍、课程介绍
X-Man

基本介绍

- Anciety 丁湛钊
- Team Eur3kA(r3kapig)
- Github: Escapingbug
- Pwn(rev学习者 @Atum)
- I’m never yr teacher, source is.
- QQ: 641880047
- 微信: ding641880047
- Email: dzz@anciety.me(preferred)

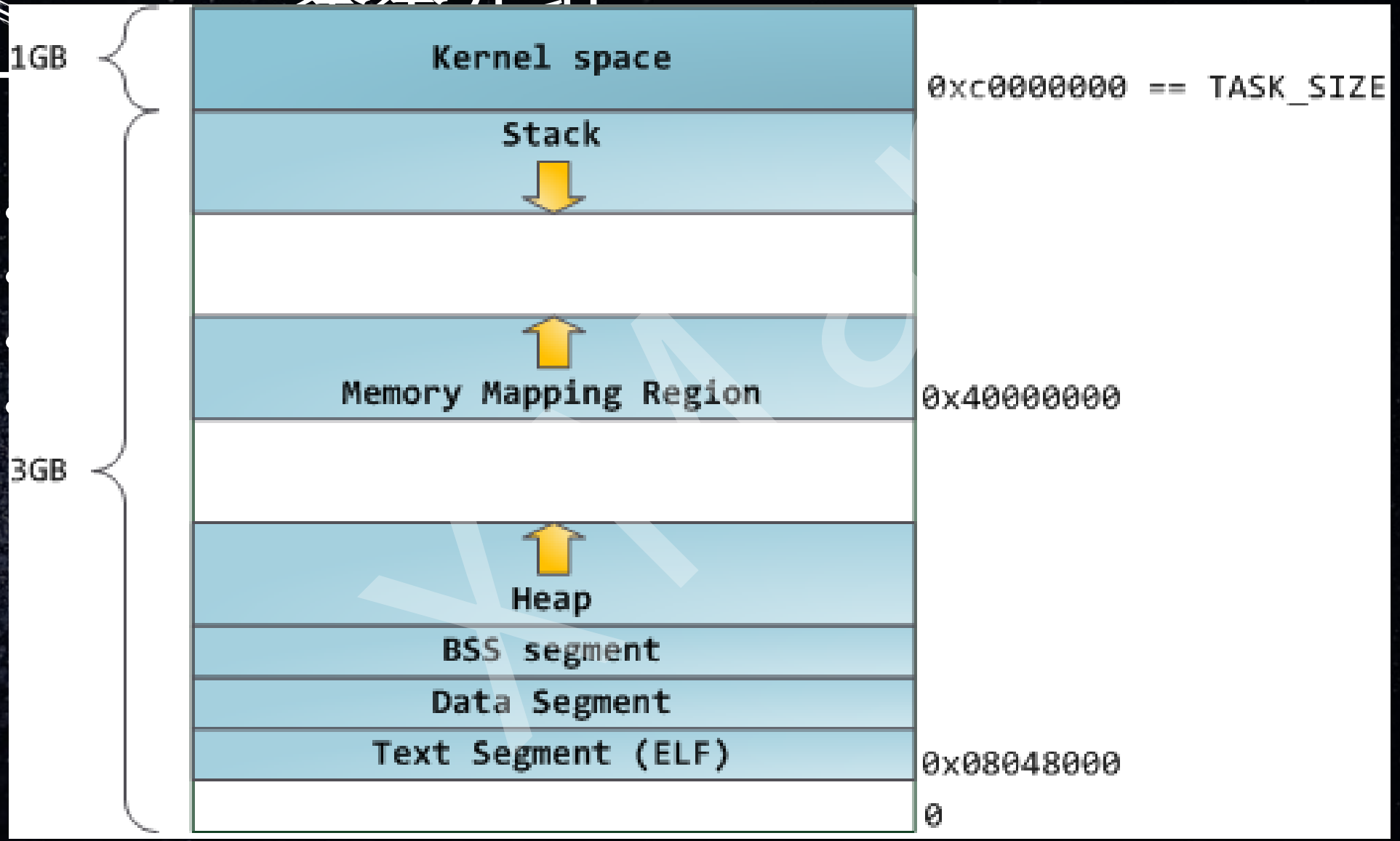
基本介绍

- Team Eur3kA
- x FlappigPig: r3kapig
- N1CTF 1st
- 0CTF/TCTF online 2nd /offline 5th
- WCTF 7th
- Defcon 26 qualified
- Mail: lgcpku@gmail.com 欢迎上船

基本介绍

- First-class source code！ 源码为一等公民
- 官方文档优先。
- 按需听讲，困了睡觉。
- 随时提问，不保证会。

基本介绍


1GB
3GB
Kernel space
Stack
Memory Mapping Region
Heap
BSS segment
Data Segment
Text Segment (ELF)
0xc0000000 == TASK_SIZE
0x40000000
0x08048000
0

基本介绍

- Register 寄存器： eax, ebx, ecx, edx, edi, esi, esp, eip, eflags…
- 调用约定 – 64: rdi, rsi, rdx, rcx, r8, r9, stack
- 调用约定 -- 32: stack
- 调一下看看函数调用过程！

基本介绍

- 系统调用：与操作系统内核交互的api
- 有独特的调用约定
- 32: eax – syscall 号, ebx, ecx, edx 参数
- 64: rax – syscall 号, rdi, rsi, rdx, rcx, r8, r9参数

基本介绍

调一下：函数调用约定，系统调用过程

Part 02

工具使用

Gdb/ida/pwntools简介

工具使用——IDA

- Interactive DisAssembler (IDA)
- 买不起
- 反汇编器
- Hexrays 反编译器
- IDA Python
- 类型支持、静态分析支持
- 动态调试支持

工具使用——IDA

IDA使用技巧

工具使用——IDA

- F5
- F5
- F5

工具使用——IDA

- 基本功能
  - 略（见逆向）

工具使用——IDA

- 反编译源码调试（没用过）
- F5好用
- 流程图好用
- 流程图带F5带地址，更好用
- General -> IDA options -> line prefix(graph) 开启流程图里的地址显示
- 伪代码 -> 右键 -> copy to assembly 显示对应关系（此时你的光标会停留在翻译成相应伪代码的asm位置）
- （一会再演示，两个系统太蛋疼了）

工具使用——IDA

- Shift+f1 -> 右键 insert struct 可以插入自定义结构体（甚至可以插入enum）（C++语法）
- Y -> 切换类型（切换正确的类型可以让hexray翻译出更加合理的伪代码）（函数类型也可以更换）
- N -> 切换命名
- 让你的伪代码看起来和源代码一样

工具使用——GDB

GDB使用技巧

工具使用——GDB

- 装个插件(gef/pwndbg/peda)
- 装个带符号的libc
- 嫌麻烦？Github：escapingbug/ancypwn
  用docker搭环境

工具使用——GDB

- 复杂情况请学会用python (gdb python)
- 虽然大多数情况下不需要，毕竟做的是pwn题
- 常见命令：搜索gdb cheatsheet
- GDB可以有符号，如果有debug信息请利用好
- 没有符号有源码？编译一个带debug信息的，加上去！
- Libc source可以帮助你事半功倍

工具使用——pwntools

pwntools使用技巧

工具使用——pwntools

看官方文档

工具使用——pwntools

- IO问题很关键，read, readline, gets, scanf会影响你的结果
- 遇事不决先sleep(或raw_input)
- Debug log_level！
- 注意context！

工具使用——pwntools

- 小功能： cyclic pattern (cyclic(), cyclic_find())
- 小功能: fmtstr_payload（依然可能手搓）
- 其他小功能。。（没用过）


Part 03
格式化字符串漏洞
格式化字符串漏洞基本介绍
xman

格式化字符串漏洞

正题之前...

格式化字符串漏洞

很好..你将接收来自glibc的馈赠

格式化字符串漏洞

简要介绍一下成因

```
printf(“%s”, s); // intended
printf(s); // wtf?
```

格式化字符串漏洞

可变参数101：

- C语言可变参数用...表示
- 可以接受任意长度参数
- va_list: va_start, va_end, va_arg
- 实现：从栈上往后取

格式化字符串漏洞

一个桑坡：

```
va_list ap; //定义一个va_list类型变量
va_start(ap,fmt);  //获取第二个参数的地址
m = va_arg(ap,int);  //第二个参数是int类型，获取值
d = va_arg(ap,double); //第三个参数是double类型，获取值
ptr = va_arg(ap,char*); //第四个参数是char*类型，获取值
va_end(ap);
```

格式化字符串漏洞

回去想想成因：

我们控制了本不应该被控制的
格式化字符串

格式化字符串漏洞

来以printf为例：

```
/* Write formatted output to stdout from the format string FOR
/* VARARGS1 */
int
__printf (const char *format, ...)
{
  va_list arg;
  int done;

  va_start (arg, format);
  done = vfprintf (stdout, format, arg);
  va_end (arg);

  return done;
}

#undef _IO_printf
```

```
199   /* The function itself.  */
200   int
201   vfprintf (FILE *s, const CHAR_T *format, va_list ap)
```

```
2047  all_done:
2048    free (args_malloced);
2049    free (workstart);
2050    /* Unlock the stream.  */
2051    _IO_funlockfile (s);
2052    _IO_cleanup_region_end (0);
2053
2054    return done;
2055  }
2056
```

格式化字符串漏洞

慢慢读？看看文档：

https://linux.die.net/man/3/printf

- 控制了格式化字符串，而其实需要用到多少个参数是由格式化字符串定的。
- 没有？C语言并不知道，反正把栈上的内容当参数
- %s 后面的指针对应地址，可以读出

n

The number of characters written so far is stored into the integer indicated by the *int* * (or variant) pointer argument. No argument is converted.

格式化字符串漏洞

- 现在就简单了，不过还需要控制可变参数部分
- %m$X, m十进制整数, 第几个参数

格式化字符串漏洞

思路：

- 想办法找到控制的参数（通过%m$X）
- 利用%mX控制输出的字符数量（写入内容）
- 利用%n写入（参数已控制，写入位置控制）
- Arbitrary-address-write 任意地址写入
- 读？%s

格式化字符串漏洞

简单示例

格式化字符串漏洞

注意事项：

- Printf_chk可以防御：加强了%m$X和%n(禁止)，但是无法防御读取（可以读出后面地址）
- Sprintf, sscanf等一切带格式化字符串的都可以采取相似思路


Part 04
栈溢出
栈溢出各种手法原理及攻击方法介绍
xman

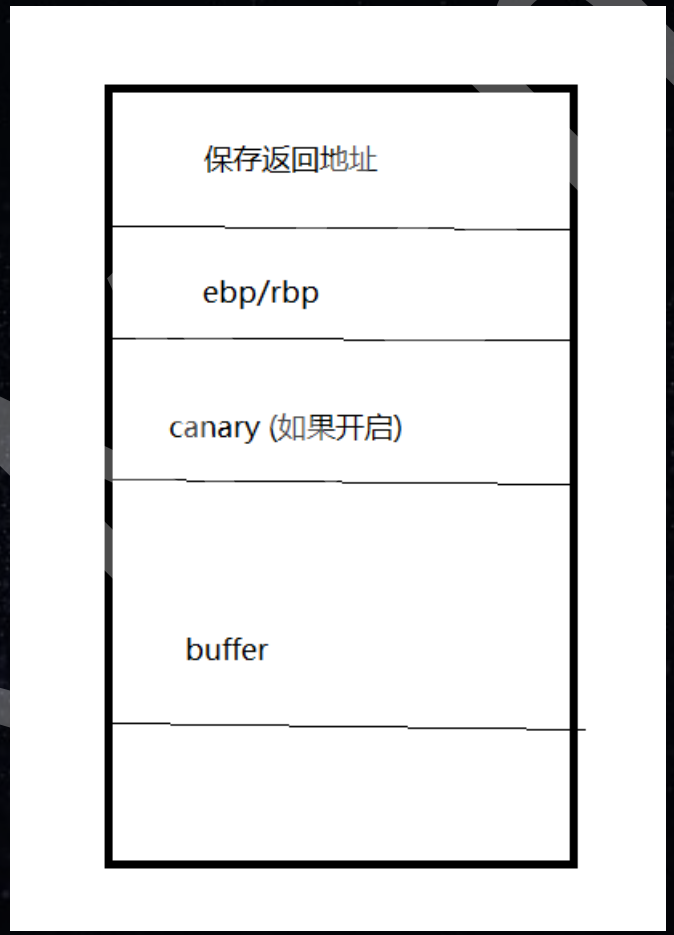
栈溢出漏洞 ret2shellcode

栈上分配数据未验证长度

```
1  void vuln(void) {
2    char buf[10]; // 10字节
3    scanf("%s", buf); // 输入任意字节长度
4    read(0, buf, 200); // 输入超过10字节长度
5  }
```

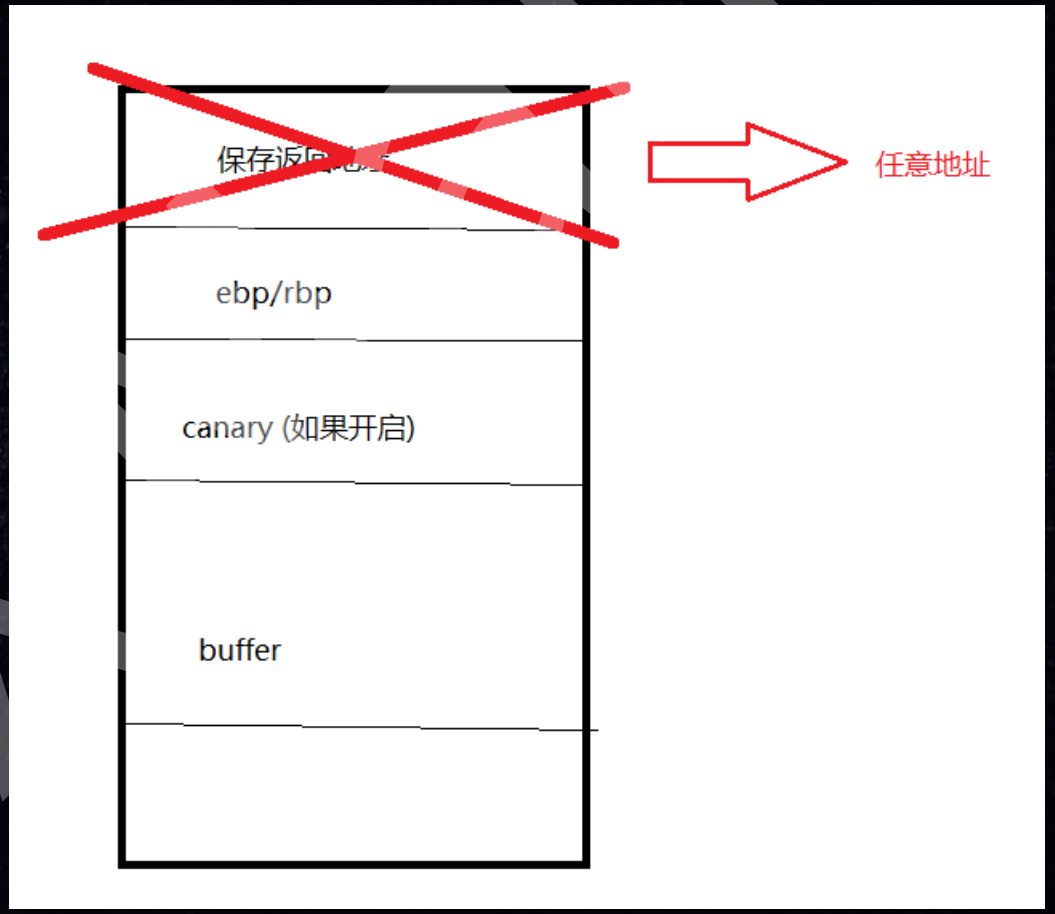
栈溢出漏洞 ret2shellcode

栈上的内容



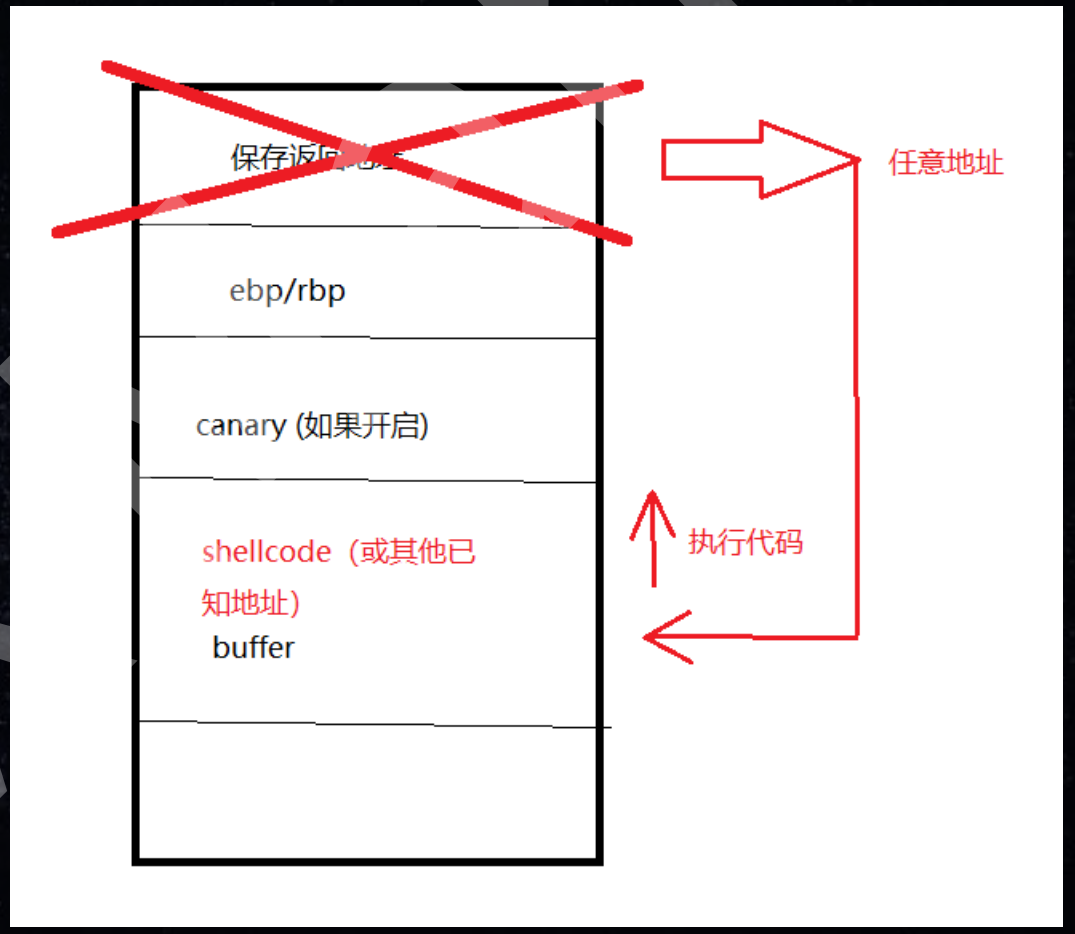
栈溢出漏洞 ret2shellcode

通过buffer超长使得后面内容被覆盖，执行任意地址代码



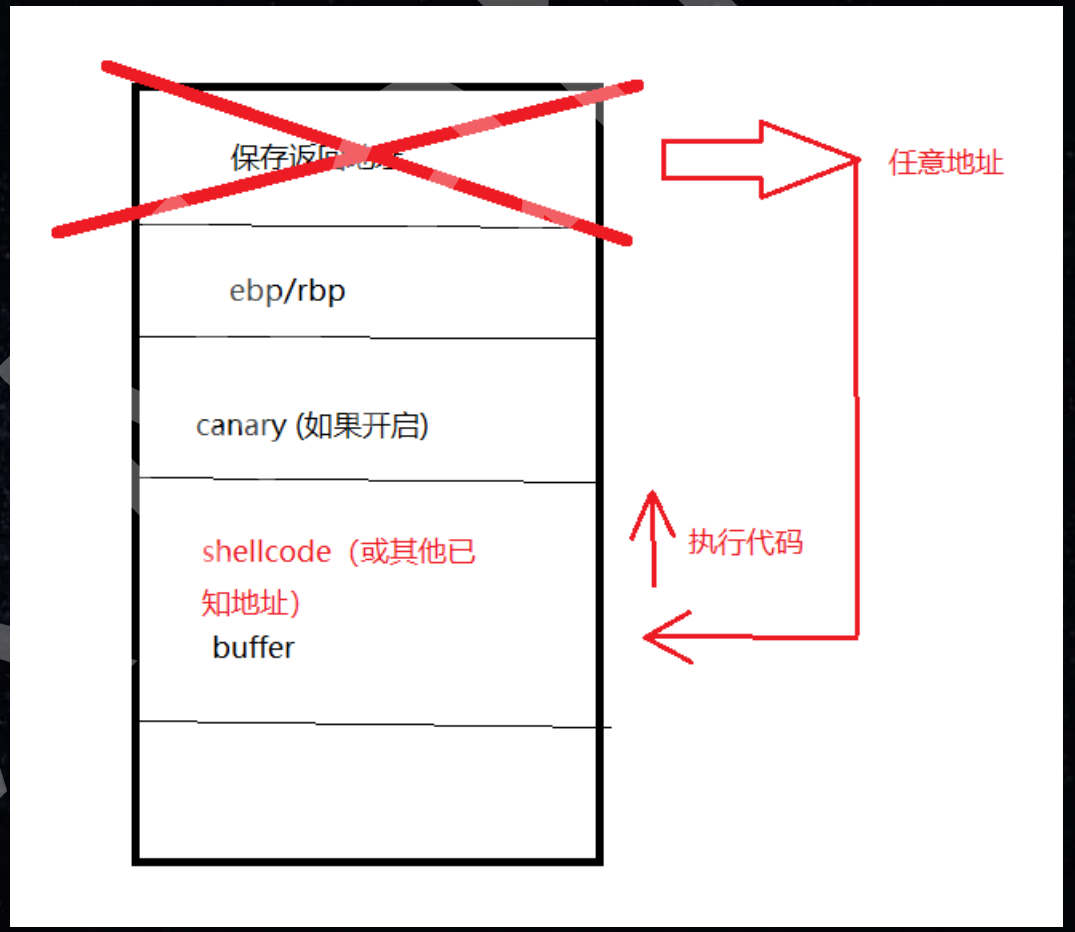
栈溢出漏洞 ret2shellcode

Shellcode:
输入数据时输入的代码



栈溢出漏洞 ret2shellcode

要求：
数据可以被执行
（具有执行权限）



栈溢出漏洞 ROP

如果无法被执行呢？

栈溢出漏洞 ROP

ROP(Return Oriented Programming)

栈溢出漏洞 ROP

现在我们无法执行“数据”（shellcode），
有什么可以执行的？

代码段内代码！

栈溢出漏洞 ROP

不过还是有个问题，一旦开始执行，如何停下来？

控制流变化（ret， jmp, call）
(ROP, JOP, COP)

栈溢出漏洞 ROP

思路：
使用代码段已有代码，但是有用指令之后，需要有ret来回到我们的控制。
XX指令; ret; → gadget (xx指令为我们需要的)
栈已控制，ret目标可控制，ret到下一个gadget
Done.

栈溢出漏洞 ROP

工具:
ROPgadget(或者其他可以搜索gadget的工具)

ret2syscall, ret2libc, ret2csu, ret2reg → 不同gadget
Ret2vdso, ret2vsyscall → 特殊gadget（关注mapping！）
Ret2csu ☹ → 不会用工具…(--depth 20)

栈溢出漏洞 ROP

-- ROP讲完了？

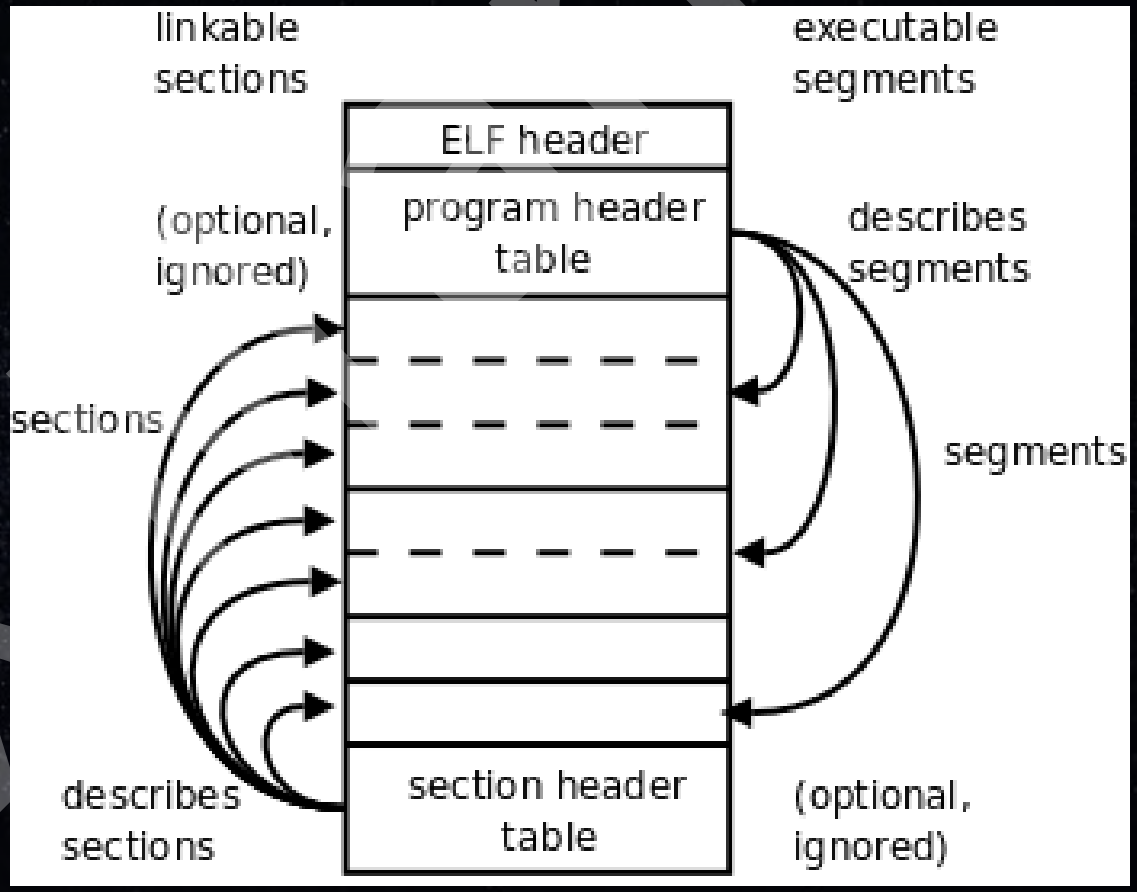
-- 讲完了。理解汇编，剩下的就是找gadget拼程序了。

栈溢出漏洞 dl-resolve

Ret2dlresolve:
没有libc地址
需要调用外部函数
控制了已知地址内容

栈溢出漏洞 dl-resolve

接下来说说ELF..



栈溢出漏洞 dl-resolve

PLT & GOT:

- PLT: push index; call GOT; jmp PLT[0]
- GOT: 保存已解析地址，未解析则为下一条指令地址 (jmp PLT[0])

栈溢出漏洞 dl-resolve

PLT[0] olve

```
27          .text
28          .globl _dl_runtime_resolve
29          .type _dl_runtime_resolve, @function
30          .align 16
31          cfi_startproc
32  _dl_runtime_resolve:
33          cfi_adjust_cfa_offset(16) # Incorporate PLT
34          subq $56,%rsp
35          cfi_adjust_cfa_offset(56)
36          movq %rax,(%rsp)        # Preserve registers otherwise clobbered.
37          movq %rcx, 8(%rsp)
38          movq %rdx, 16(%rsp)
39          movq %rsi, 24(%rsp)
40          movq %rdi, 32(%rsp)
41          movq %r8, 40(%rsp)
42          movq %r9, 48(%rsp)
43          movq 64(%rsp), %rsi     # Copy args pushed by PLT in register.
44          movq 56(%rsp), %rdi     # %rdi: link_map, %rsi: reloc_index
45          call _dl_fixup          # Call resolver.
46          movq %rax, %r11         # Save return value
47          movq 48(%rsp), %r9      # Get register content back.
```

栈溢出漏洞 dl-resolve

_dl_runtime_resolve -> _dl_fixup -> _dl_lookup_symbol_x

```
66  _dl_fixup (
67  # ifdef ELF_MACHINE_RUNTIME_FIXUP_ARGS
68            ELF_MACHINE_RUNTIME_FIXUP_ARGS,
69  # endif
70            /* GKM FIXME: Fix trampoline to pass bounds so we can do
71               without the `__unbounded' qualifier.  */
72            struct link_map *__unbounded l, ElfW(Word) reloc_arg)
73  {
74    const ElfW(Sym) *const symtab
75      = (const void *) D_PTR (l, l_info[DT_SYMTAB]);
76    const char *strtab = (const void *) D_PTR (l, l_info[DT_STRTAB]);
77
78    const PLTREL *const reloc
79      = (const void *) (D_PTR (l, l_info[DT_JMPREL]) + reloc_offset);
80    const ElfW(Sym) *sym = &symtab[ELFW(R_SYM) (reloc->r_info)];
81    void *const rel_addr = (void *)(l->l_addr + reloc->r_offset);
82    lookup_t result;
83    DL_FIXUP_VALUE_TYPE value;
```

```
117
118         result = _dl_lookup_symbol_x (strtab + sym->st_name, l, &sym, l->l_scope,
119                                       version, ELF_RTYPE_CLASS_PLT, flags, NULL);
120
```

栈溢出漏洞 dl-resolve

现在要从index获取到symbol的字符串，过程？

1. Reloc

```
typedef struct elf32_rel {
  Elf32_Addr        r_offset;
  Elf32_Word        r_info;
} Elf32_Rel;

typedef struct elf64_rel {
  Elf64_Addr r_offset;          /* Location
at which to apply the action */
  Elf64_Xword r_info;           /* index
and type of relocation */
} Elf64_Rel;
```

```
66  _dl_fixup (
67  # ifdef ELF_MACHINE_RUNTIME_FIXUP_ARGS
68            ELF_MACHINE_RUNTIME_FIXUP_ARGS,
69  # endif
70            /* GKM FIXME: Fix trampoline to pass bounds so we can do
71               without the `__unbounded' qualifier.  */
72            struct link_map *__unbounded l, ElfW(Word) reloc_arg)
73  {
74    const ElfW(Sym) *const symtab
75      = (const void *) D_PTR (l, l_info[DT_SYMTAB]);
76    const char *strtab = (const void *) D_PTR (l, l_info[DT_STRTAB]);
77
78    const PLTREL *const reloc
79      = (const void *) (D_PTR (l, l_info[DT_JMPREL]) + reloc_offset);
80    const ElfW(Sym) *sym = &symtab[ELFW(R_SYM) (reloc->r_info)];
81    void *const rel_addr = (void *)(l->l_addr + reloc->r_offset);
82    lookup_t result;
83    DL_FIXUP_VALUE_TYPE value;
```

栈溢出漏洞 dl-resolve

现在要从index获取到symbol的字符串，过程？

1. Reloc

```
/* The following are used with relocations */
#define ELF32_R_SYM(x) ((x) >> 8)
#define ELF32_R_TYPE(x) ((x) & 0xff)

#define ELF64_R_SYM(i)
            ((i) >> 32)
#define ELF64_R_TYPE(i)
            ((i) & 0xffffffff)
```

```
66  _dl_fixup (
67  # ifdef ELF_MACHINE_RUNTIME_FIXUP_ARGS
68            ELF_MACHINE_RUNTIME_FIXUP_ARGS,
69  # endif
70            /* GKM FIXME: Fix trampoline to pass bounds so we can do
71               without the `__unbounded' qualifier.  */
72            struct link_map *__unbounded l, ElfW(Word) reloc_arg)
73  {
74    const ElfW(Sym) *const symtab
75      = (const void *) D_PTR (l, l_info[DT_SYMTAB]);
76    const char *strtab = (const void *) D_PTR (l, l_info[DT_STRTAB]);
77
78    const PLTREL *const reloc
79      = (const void *) (D_PTR (l, l_info[DT_JMPREL]) + reloc_offset);
80    const ElfW(Sym) *sym = &symtab[ELFW(R_SYM) (reloc->r_info)];
81    void *const rel_addr = (void *)(l->l_addr + reloc->r_offset);
82    lookup_t result;
83    DL_FIXUP_VALUE_TYPE value;
```

栈溢出漏洞 dl-resolve

现在要从index获取到symbol的字符串，过程？

2. Sym “JMPREL” + R_SYM

```
typedef struct elf32_sym{
  Elf32_Word    st_name;
  Elf32_Addr    st_value;
  Elf32_Word    st_size;
  unsigned char         st_info;
  unsigned char         st_other;
  Elf32_Half    st_shndx;
} Elf32_Sym;
```

```
66  _dl_fixup (
67  # ifdef ELF_MACHINE_RUNTIME_FIXUP_ARGS
68            ELF_MACHINE_RUNTIME_FIXUP_ARGS,
69  # endif
70            /* GKM FIXME: Fix trampoline to pass bounds so we can do
71               without the `__unbounded' qualifier.  */
72            struct link_map *__unbounded l, ElfW(Word) reloc_arg)
73  {
74    const ElfW(Sym) *const symtab
75      = (const void *) D_PTR (l, l_info[DT_SYMTAB]);
76    const char *strtab = (const void *) D_PTR (l, l_info[DT_STRTAB]);
77
78    const PLTREL *const reloc
79      = (const void *) (D_PTR (l, l_info[DT_JMPREL]) + reloc_offset);
80    const ElfW(Sym) *sym = &symtab[ELFW(R_SYM) (reloc->r_info)];
81    void *const rel_addr = (void *)(l->l_addr + reloc->r_offset);
82    lookup_t result;
83    DL_FIXUP_VALUE_TYPE value;
```

栈溢出漏洞 dl-resolve

现在要从index获取到symbol的字符串，过程？

2. Sym “JMPREL” + R_SYM

```
typedef struct elf64_sym {
  Elf64_Word st_name; /* Symbol name, index in string tbl */
  unsigned char st_info;  /* Type and binding attributes */
  unsigned char st_other;  /* No defined meaning, 0 */
  Elf64_Half st_shndx;  /* Associated section index */
  Elf64_Addr st_value;  /* Value of the symbol */
  Elf64_Xword st_size;  /* Associated symbol size */
} Elf64_Sym;
```

```
66  _dl_fixup (
67  # ifdef ELF_MACHINE_RUNTIME_FIXUP_ARGS
68            ELF_MACHINE_RUNTIME_FIXUP_ARGS,
69  # endif
70            /* GKM FIXME: Fix trampoline to pass bounds so we can do
71               without the `__unbounded' qualifier.  */
72            struct link_map *__unbounded l, ElfW(Word) reloc_arg)
73  {
74    const ElfW(Sym) *const symtab
75      = (const void *) D_PTR (l, l_info[DT_SYMTAB]);
76    const char *strtab = (const void *) D_PTR (l, l_info[DT_STRTAB]);
77
78    const PLTREL *const reloc
79      = (const void *) (D_PTR (l, l_info[DT_JMPREL]) + reloc_offset);
80    const ElfW(Sym) *sym = &symtab[ELFW(R_SYM) (reloc->r_info)];
81    void *const rel_addr = (void *)(l->l_addr + reloc->r_offset);
82    lookup_t result;
83    DL_FIXUP_VALUE_TYPE value;
```

栈溢出漏洞 dl-resolve

现在要从index获取到symbol的字符串，过程？

2. Sym “JMPREL” + R_SYM

```
typedef struct elf64_sym {
  Elf64_Word st_name; /* Symbol name, index in string tbl */
  unsigned char st_info;  /* Type and binding attributes */
  unsigned char st_other;  /* No defined meaning, 0 */
  Elf64_Half st_shndx;  /* Associated section index */
  Elf64_Addr st_value;  /* Value of the symbol */
  Elf64_Xword st_size;  /* Associated symbol size */
} Elf64_Sym;
```

```
66  _dl_fixup (
67  # ifdef ELF_MACHINE_RUNTIME_FIXUP_ARGS
68            ELF_MACHINE_RUNTIME_FIXUP_ARGS,
69  # endif
70            /* GKM FIXME: Fix trampoline to pass bounds so we can do
71               without the `__unbounded' qualifier.  */
72            struct link_map *__unbounded l, ElfW(Word) reloc_arg)
73  {
74    const ElfW(Sym) *const symtab
75      = (const void *) D_PTR (l, l_info[DT_SYMTAB]);
76    const char *strtab = (const void *) D_PTR (l, l_info[DT_STRTAB]);
77
78    const PLTREL *const reloc
79      = (const void *) (D_PTR (l, l_info[DT_JMPREL]) + reloc_offset);
80    const ElfW(Sym) *sym = &symtab[ELFW(R_SYM) (reloc->r_info)];
81    void *const rel_addr = (void *)(l->l_addr + reloc->r_offset);
82    lookup_t result;
83    DL_FIXUP_VALUE_TYPE value;
```

栈溢出漏洞 dl-resolve

现在要从index获取到symbol的字符串，过程？

3. Str “STRTAB” + st_name

```
66  _dl_fixup (
67  # ifdef ELF_MACHINE_RUNTIME_FIXUP_ARGS
68              ELF_MACHINE_RUNTIME_FIXUP_ARGS,
69  # endif
70              /* GKM FIXME: Fix trampoline to pass bounds so we can do
71                 without the `__unbounded' qualifier.  */
72              struct link_map *__unbounded l, ElfW(Word) reloc_arg)
73  {
74    const ElfW(Sym) *const symtab
75      = (const void *) D_PTR (l, l_info[DT_SYMTAB]);
76    const char *strtab = (const void *) D_PTR (l, l_info[DT_STRTAB]);
77
78    const PLTREL *const reloc
79      = (const void *) (D_PTR (l, l_info[DT_JMPREL]) + reloc_offset);
80    const ElfW(Sym) *sym = &symtab[ELFW(R_SYM) (reloc->r_info)];
81    void *const rel_addr = (void *)(l->l_addr + reloc->r_offset);
82    lookup_t result;
83    DL_FIXUP_VALUE_TYPE value;
```

栈溢出漏洞 dl-resolve

然而比较恶心的地方。。
vesion != NULL的话..

```
if (l->l_info[VERSYMIDX (DT_VERSYM)] != NULL)
  {
    const ElfW(Half) *vernum =
      (const void *) D_PTR (l, l_info[VERSYMIDX (DT_VERSYM)]);
    ElfW(Half) ndx = vernum[ELFW(R_SYM) (reloc->r_info)] & 0x7fff;
    version = &l->l_versions[ndx];
    if (version->hash == 0)
      version = NULL;
  }
```

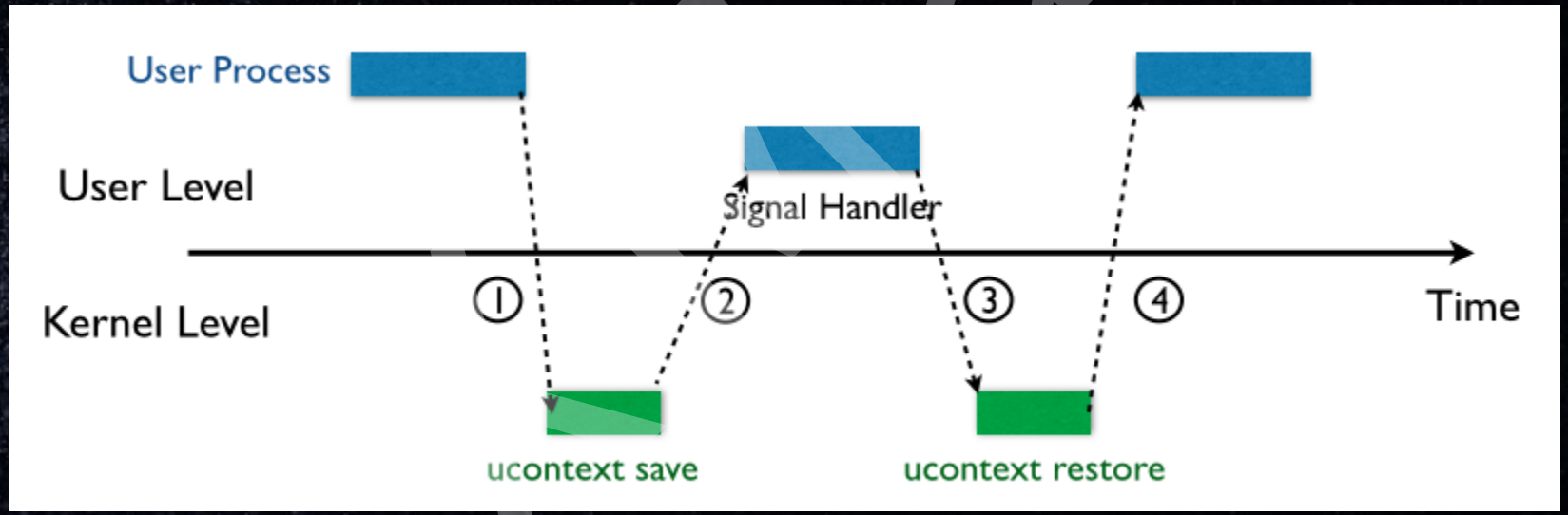
```
117
118         result = _dl_lookup_symbol_x (strtab + sym->st_name, l, &sym, l->l_scope,
119                                       version, ELF_RTYPE_CLASS_PLT, flags, NULL);
120
```

栈溢出漏洞 dl-resolve

调一下

栈溢出漏洞 SROP

说说signal...



栈溢出漏洞 SROP

sigreturn frame 32

```
199  /*
200   * The 32-bit signal frame:
201   */
202  struct sigcontext_32 {
203          __u16                   gs, __gsh;
204          __u16                   fs, __fsh;
205          __u16                   es, __esh;
206          __u16                   ds, __dsh;
207          __u32                   di;
208          __u32                   si;
209          __u32                   bp;
210          __u32                   sp;
211          __u32                   bx;
212          __u32                   dx;
213          __u32                   cx;
214          __u32                   ax;
215          __u32                   trapno;
216          __u32                   err;
217          __u32                   ip;
218          __u16                   cs, __csh;
219          __u32                   flags;
220          __u32                   sp_at_signal;
221          __u16                   ss, __ssh;
222
223          /*
224           * fpstate is really (struct _fpstate *) or (struct _xstate *)
225           * depending on the FP_XSTATE_MAGIC1 encoded in the SW reserved
226           * bytes of (struct _fpstate) and FP_XSTATE_MAGIC2 present at the end
227           * of extended memory layout. See comments at the definition of
228           * (struct _fpx_sw_bytes)
229           */
230          __u32                   fpstate; /* Zero when no FPU/extended context */
231          __u32                   oldmask;
232          __u32                   cr2;
233  };
234
```

栈溢出漏洞 SROP

sigreturn frame 64

```
238  struct sigcontext_64 {
239          __u64                   r8;
240          __u64                   r9;
241          __u64                   r10;
242          __u64                   r11;
243          __u64                   r12;
244          __u64                   r13;
245          __u64                   r14;
246          __u64                   r15;
247          __u64                   di;
248          __u64                   si;
249          __u64                   bp;
250          __u64                   bx;
251          __u64                   dx;
252          __u64                   ax;
253          __u64                   cx;
254          __u64                   sp;
255          __u64                   ip;
256          __u64                   flags;
257          __u16                   cs;
258          __u16                   gs;
259          __u16                   fs;
260          __u16                   ss;
261          __u64                   err;
262          __u64                   trapno;
263          __u64                   oldmask;
264          __u64                   cr2;
265
266          /*
267           * fpstate is really (struct _fpstate *) or (struct _xstate *)
268           * depending on the FP_XSTATE_MAGIC1 encoded in the SW reserved
269           * bytes of (struct _fpstate) and FP_XSTATE_MAGIC2 present at the end
270           * of extended memory layout. See comments at the definition of
271           * (struct _fpx_sw_bytes)
272           */
273          __u64                   fpstate; /* Zero when no FPU/extended context */
274          __u64                   reserved1[8];
275  };
```

栈溢出漏洞 SROP

调用sigreturn就会把当前栈帧上的内容当一个sigframe然后恢复各寄存器 → 控制寄存器！

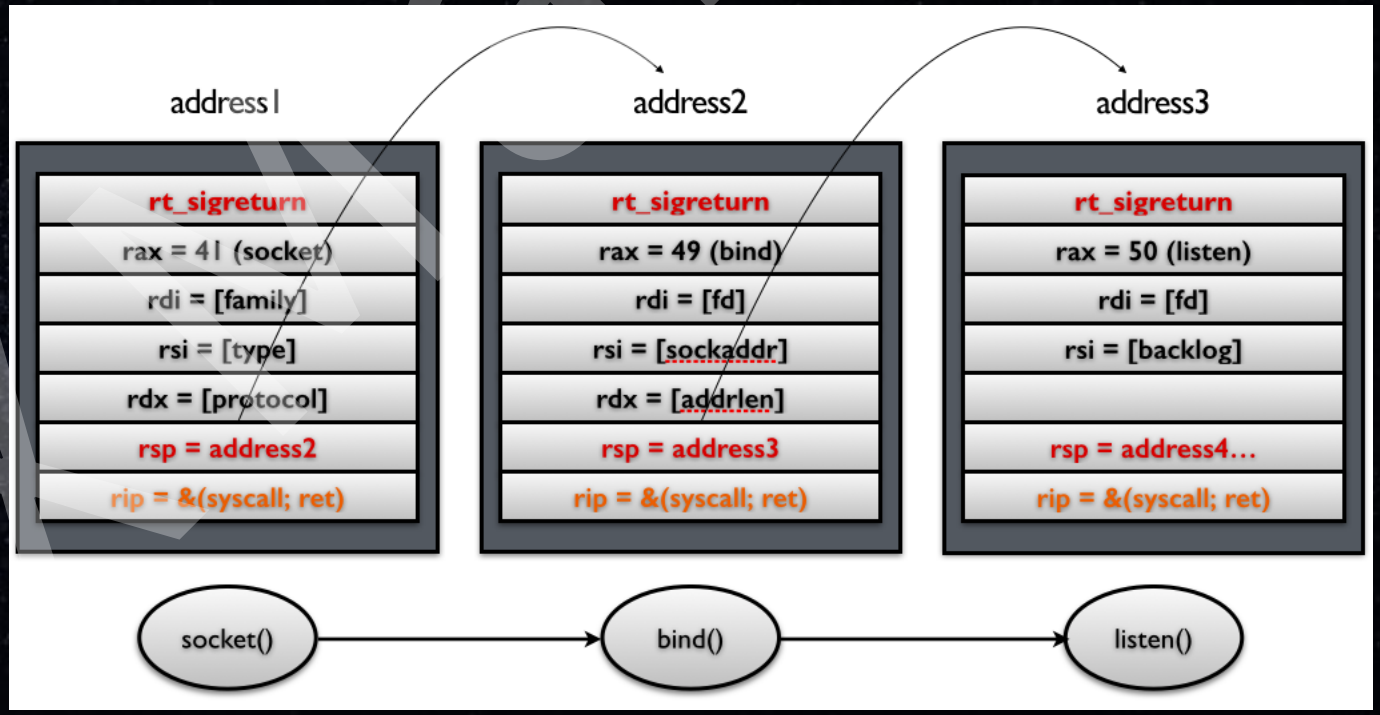
0x00	rt_sigreturn	uc_flags
0x11	&uc	uc_stack.ss_sp
0x20	uc_stack.ss_flags	uc_stack.ss_size
0x30	r8	r9
0x40	r10	r11
0x50	r12	r13
0x60	r14	r15
0x70	rdi = &"/bin/sh"	rsi
0x80	rbp	rbx
0x90	rdx	rax = 59 (execve)
0xA0	rcx	rsp
0xB0	rip = &syscall	eflags
0xC0	cs / gs / fs	err
0xD0	trapno	oldmask (unused)
0xE0	cr2 (segfault addr)	&fpstate
0xF0	__reserved	sigmask

栈溢出漏洞 SROP

控制pc → 继续sigreturn → rop!

SROP: Sigreturn ROP

只需要控制ax和sp



栈溢出漏洞 SROP

控制pc → 继续sigreturn → rop!

SROP: Sigreturn ROP

只需要控制ax和sp

栈溢出漏洞 SROP

一些天然gadget

On some systems SROP gadgets are randomised, on others, they are not

non-ASLR :-(

android

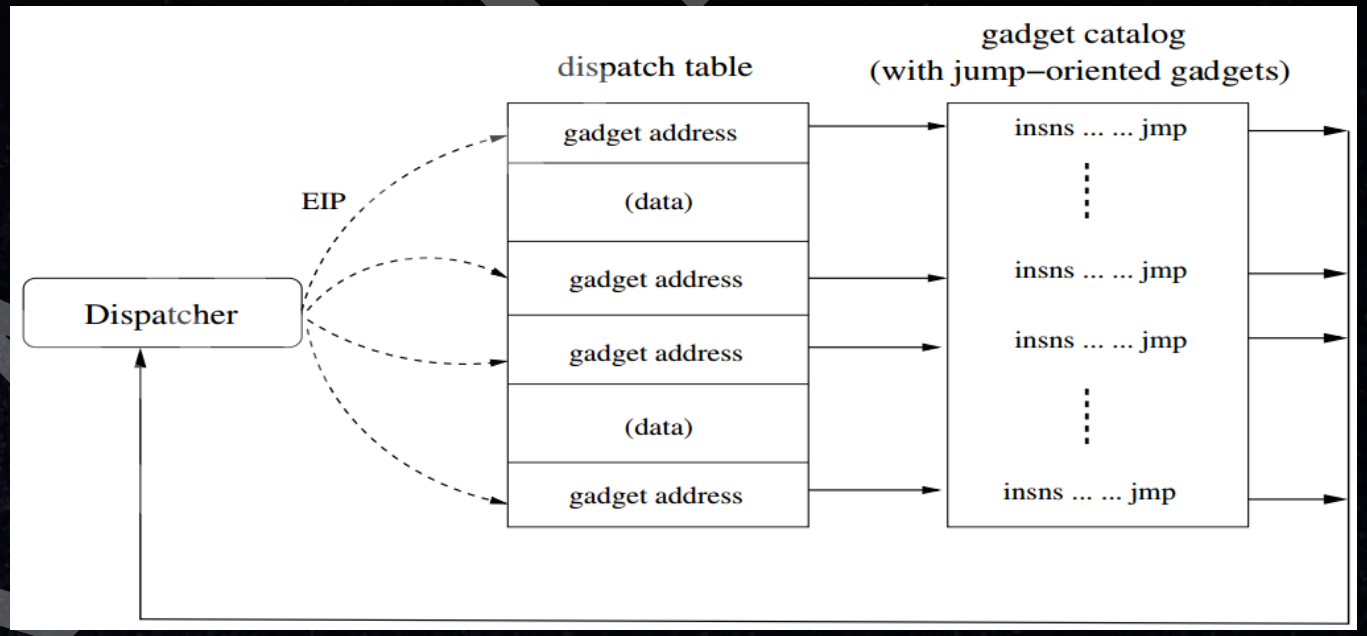
Operating system	Gadget	Memory map
Linux i386	sigreturn	[vdso]
Linux < 3.11 ARM	sigreturn	[vectors] 0xffff0000
Linux < 3.3 x86-64	syscall & return	[vsyscall] 0xffffffffff600000
Linux ≥ 3.3 x86-64	syscall & return	Libc
Linux x86-64	sigreturn	Libc
FreeBSD 9.2 x86-64	sigreturn	0x7fffffff000
Mac OSX x86-64	sigreturn	Libc
iOS ARM	sigreturn	Libsystem
iOS ARM	syscall & return	Libsystem

栈溢出漏洞 jop, cop

最后随便聊聊jop和cop。

利用jmp和call影响控制流。

见得比较少（我没见过）



栈溢出漏洞 tricks

- Stack pivot
- tls


Any Questions?
THANKS
x Man