Machine-Advanced 回课

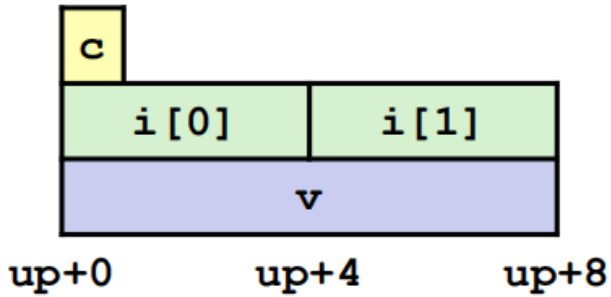
孔朝哲

2020 年 10 月 22 日

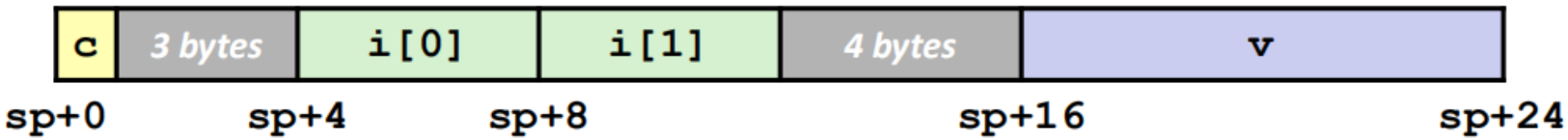
Union Allocation

- Allocate according to largest element
- Can only use one field at a time

```
union U1 {
  char c;
  int i[2];
  double v;
} *up;
```




```
struct S1 {
  char c;
  int i[2];
  double v;
} *sp;
```



Union Application

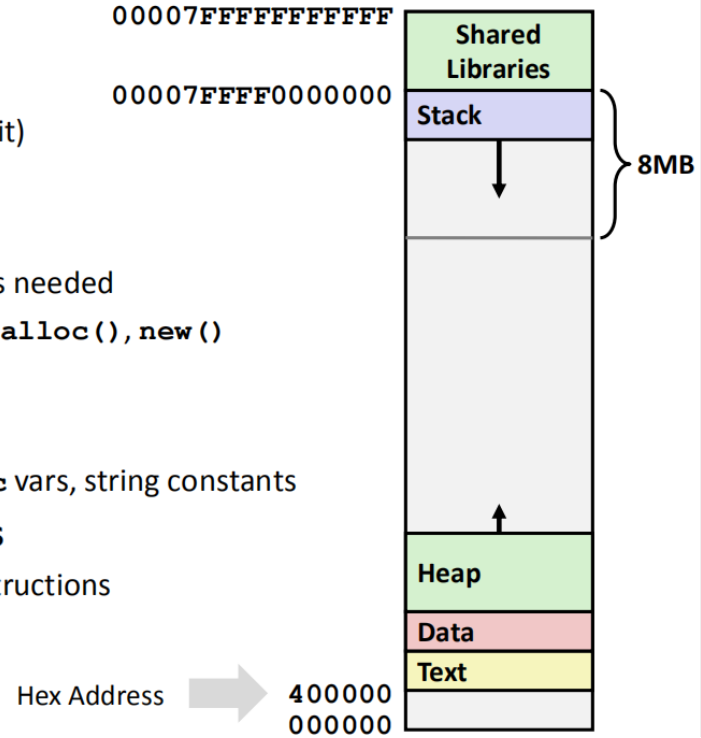
Unions are usually used with the company of a discriminator: a variable indicating which of the fields of the union is valid. For example, let's say you want to create your own Variant type:

```
struct my_variant_t {
    int type;
    union {
        char char_value;
        short short_value;
        int int_value;
        long long_value;
        float float_value;
        double double_value;
        void* ptr_value;
    };
};
```

x86-64 Linux Memory Layout

not drawn to scale

- **Stack**
  - Runtime stack (8MB limit)
  - E. g., local variables
- **Heap**
  - Dynamically allocated as needed
  - When call `malloc()`, `calloc()`, `new()`
- **Data**
  - Statically allocated data
  - E.g., global vars, `static` vars, string constants
- **Text / Shared Libraries**
  - Executable machine instructions
  - Read-only

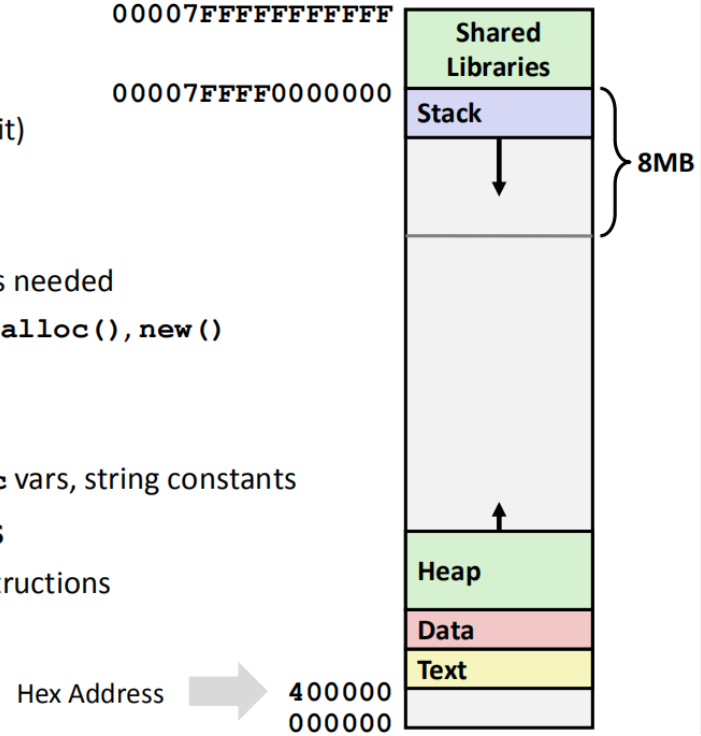


x86-64 Linux Memory Layout

x86-64 Linux Memory Layout

not drawn to scale

- **Stack**
  - Runtime stack (8MB limit)
  - E. g., local variables
- **Heap**
  - Dynamically allocated as needed
  - When call `malloc()`, `calloc()`, `new()`
- **Data**
  - Statically allocated data
  - E.g., global vars, `static` vars, string constants
- **Text / Shared Libraries**
  - Executable machine instructions
  - Read-only



ulimit -s unlimited
ulimit -s 16384

String Library Code

- **Implementation of Unix function `gets()`**

```
/* Get string from stdin */
char *gets(char *dest)
{
    int c = getchar();
    char *p = dest;
    while (c != EOF && c != '\n') {
        *p++ = c;
        c = getchar();
    }
    *p = '\0';
    return dest;
}
```

  - No way to specify limit on number of characters to read
- **Similar problems with other library functions**
  - **`strcpy`, `strcat`**: Copy strings of arbitrary length
  - **`scanf`, `fscanf`, `sscanf`,** when given **`%s`** conversion specification

Buffer Overflow

```
/* Echo Line */
void echo()
{
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```

Buffer Overflow

```
/* Echo Line */
void echo()
{
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```

如何修改?

Buffer Overflow

```
36  void echo_1()
37  {
38      char buf[4];
39      while(fgets(buf,4,stdin)!=nullptr)
40          printf("%s",buf);
41      printf("\n");
42  }
```

Buffer Overflow

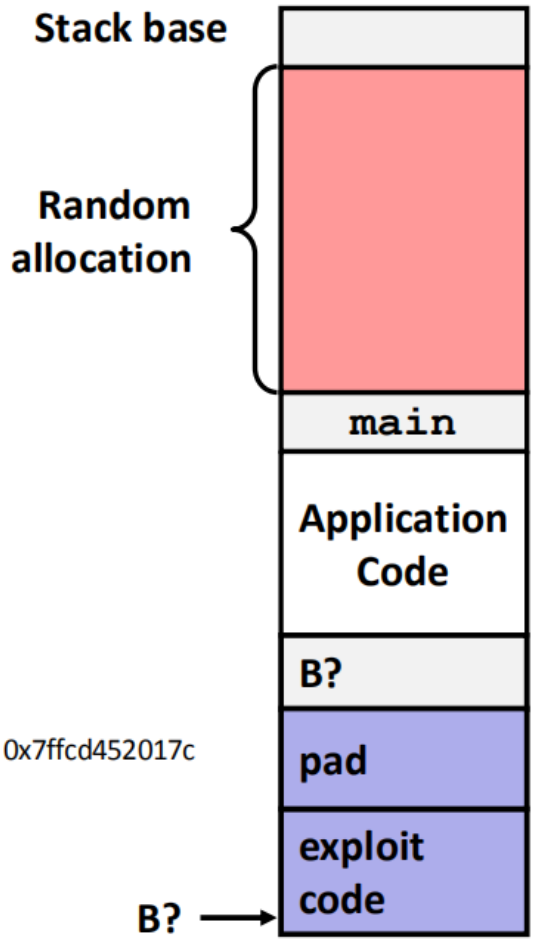
```
44  char* my_gets()
45  {
46      int len=4;
47      int n=0;
48      char *s=(char*)malloc(len);
49      if(s==nullptr)return nullptr;
50      char c=getchar();
51      while(c!=EOF && c!='\n')
52      {
53          if(n>=len-1)
54          {
55              len*=2;
56              char *s1=(char*)malloc(len);
57              if(s1==nullptr)
58              {
59                  free(s);
60                  return nullptr;
61              }
62              memcpy(s1,s,n);
63              free(s);
64              s=s1;
65          }
66          s[n++]=c;
67          c=getchar();
68      }
69      s[n]='\0';
70      return s;
71  }
72  void echo_2()
73  {
74      char *buf=my_gets();
```

Randomized Stack Offsets

- **Randomized stack offsets**
  - At start of program, allocate random amount of space on stack
  - Shifts stack addresses for entire program
  - Makes it difficult for hacker to predict beginning of inserted code
  - E.g.: 5 executions of memory allocation code

```
local    0x7ffe4d3be87c    0x7fff75a4f9fc    0x7ffeadb7c80c    0x7ffeaea2fdac    0x7ffcd452017c
```

    - Stack repositioned each time program executes

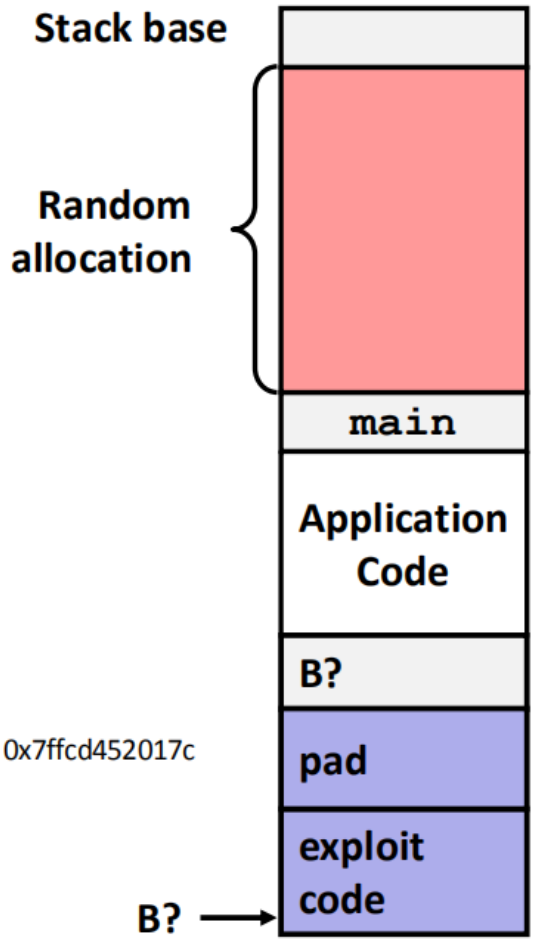


Randomized Stack Offsets

- **Randomized stack offsets**
  - At start of program, allocate random amount of space on stack
  - Shifts stack addresses for entire program
  - Makes it difficult for hacker to predict beginning of inserted code
  - E.g.: 5 executions of memory allocation code

```
local   0x7ffe4d3be87c   0x7fff75a4f9fc   0x7ffeadb7c80c   0x7ffeaea2fdac   0x7ffcd452017c
```

    - Stack repositioned each time program executes



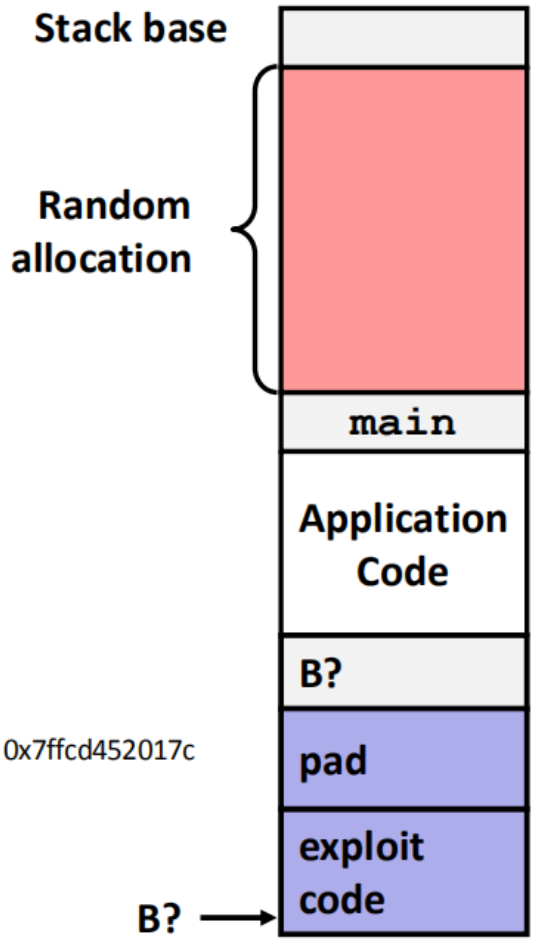
ASLR 的一部分。

Randomized Stack Offsets

- **Randomized stack offsets**
  - At start of program, allocate random amount of space on stack
  - Shifts stack addresses for entire program
  - Makes it difficult for hacker to predict beginning of inserted code
  - E.g.: 5 executions of memory allocation code

```
local    0x7ffe4d3be87c    0x7fff75a4f9fc    0x7ffeadb7c80c    0x7ffeaea2fdac    0x7ffcd452017c
```

    - Stack repositioned each time program executes



ASLR 的一部分。
可以被 $nop\ sled\ hack$ (?)

Randomized Stack Offsets

```
long local;
printf("%p\n",&local);
```

在 ubuntu 20.04 上运行 10000 次，最小值为 0x7ffc000b3b90，最大值为 0x7fffffea0280，值域为 2^{50}。

No-Execute

只有编译器产生的代码的那部分区域才是可执行的，其他部分只允许读和写。这样就消除了攻击者向系统中插入可执行代码的能力。

之前 x86 体系将读和执行访问控制合并成一个 1 位的标志，现在已经有了 $NX(No-Execute)$ 位。

No-Execute

只有编译器产生的代码的那部分区域才是可执行的，其他部分只允许读和写。这样就消除了攻击者向系统中插入可执行代码的能力。

之前 x86 体系将读和执行访问控制合并成一个 1 位的标志，现在已经有了 $NX(No-Execute)$ 位。

可以被 $ROP(Return\ Oriented\ Programming)\ hack$ 。

Stack Canary

开启 Canary 保护的 stack 结构大概如下:

Stack Canary

泄露栈中的 Canary

Canary 设计为以字节 `\x00` 结尾，本意是为了保证 Canary 可以截断字符串。 泄露栈中的 Canary 的思路是覆盖 Canary 的低字节，来打印出剩余的 Canary 部分。 这种利用方式需要存在合适的输出函数，并且可能需要第一溢出泄露 Canary，之后再次溢出控制执行流程。

```
void vuln() {
    char buf[100];
    for(int i=0;i<2;i++){
        read(0, buf, 0x200);
        printf(buf);
    }
}
int main(void) {
    init();
    puts("Hello Hacker!");
    vuln();
```

one-by-one 爆破 Canary

对于 Canary，虽然每次进程重启后的 Canary 不同 (相比 GS，GS 重启后是相同的)，但是同一个进程中的不同线程的 Canary 是相同的， 并且 通过 fork 函数创建的子进程的 Canary 也是相同的，因为 fork 函数会直接拷贝父进程的内存。我们可以利用这样的特点，彻底逐个字节将 Canary 爆破出来。 在著名的 offset2libc 绕过 linux64bit 的所有保护的文章中，作者就是利用这样的方式爆破得到的 Canary: 这是爆破的 Python 代码:

```
print "[+] Brute forcing stack canary "

start = len(p)
stop = len(p)+8

while len(p) < stop:
   for i in xrange(0,256):
      res = send2server(p + chr(i))

      if res != "":
         p = p + chr(i)
         #print "\t[+] Byte found 0x%02x" % i
         break

      if i == 255:
```

Stack Canary

如果 Canary 已经被非法修改，此时程序流程会走到 `__stack_chk_fail`。`__stack_chk_fail` 也是位于 glibc 中的函数，默认情况下经过 ELF 的延迟绑定，定义如下。

```
eglibc-2.19/debug/stack_chk_fail.c

void __attribute__ ((noreturn)) __stack_chk_fail (void)
{
  __fortify_fail ("stack smashing detected");
}

void __attribute__ ((noreturn)) internal_function __fortify_fail (const char *msg)
{
  /* The loop is added only to keep gcc happy.  */
  while (1)
    __libc_message (2, "*** %s ***: %s terminated\n",
                    msg, __libc_argv[0] ?: "<unknown>");
}
```

这意味可以通过劫持 `__stack_chk_fail` 的 got 值劫持流程或者利用 `__stack_chk_fail` 泄漏内容 (参见 stack smash)。

Stack Canary

进一步，对于 Linux 来说，fs 寄存器实际指向的是当前栈的 TLS 结构，fs:0x28 指向的正是 stack_guard。

```
typedef struct
{
  void *tcb;        /* Pointer to the TCB.  Not necessarily the
                       thread descriptor used by libpthread.  */
  dtv_t *dtv;
  void *self;       /* Pointer to the thread descriptor.  */
  int multiple_threads;
  uintptr_t sysinfo;
  uintptr_t stack_guard;
  ...
} tcbhead_t;
```

如果存在溢出可以覆盖位于 TLS 中保存的 Canary 值那么就可以实现绕过保护机制。

References

https://stackoverflow.com/questions/4788965/when-would-anyone-use-a-union-is-it-a-remnant-from-the-c-only-days

https://codeforces.com/blog/entry/63140

https://ctf-wiki.github.io/ctf-wiki/pwn/linux/mitigation/canary-zh/

https://eternalsakura13.com/2018/04/24/starctf_babystack/

Thanks for listening!