

网络安全 – 缓冲区溢出攻击

曹越

国家网络安全学院

武汉大学

yue.cao@whu.edu.cn

上周回顾 - 1

DoS攻击思想及方法

- 缓冲区满、连接复位

拒绝服务攻击的分类

- 物理、配置、资源消耗、服务中断

DoS攻击的实现方式

- 资源消耗、服务中止、物理破坏等

几个例子？

上周回顾 - 2

抵御拒绝服务

DDOS vs DOS

被DDOS攻击的现象

- 被攻击主机上有大量等待的TCP连接
- 端口随意
- 大量源地址为假的无用的数据包
- 高流量的无用数据造成网络拥塞
- 利用缺陷，反复发出服务请求，使受害主机无法及时处理正常请求
- 严重时会造成死机

反弹技术

缓冲区溢出概述

缓冲区溢出攻击的方法和步骤

缓冲区溢出攻击的防御技术

缓冲区溢出概述

缓冲区溢出攻击的方法和步骤

缓冲区溢出攻击的防御技术

缓冲溢出概述

缓冲区的定义

- 连续的一段存储空间，动态变量在程序运行时被分配缓冲区

缓冲区溢出的定义

- 指写入缓冲区的数据量超过该缓冲区能容纳的最大限度，造成溢出的数据改写了与该缓冲区相邻的原始数据的情形。

缓冲溢出的原因

造成缓冲区溢出的根本原因

- 代码在操作缓冲区时，没有对缓冲区边界进行检查
- 使得写入缓冲区的数据量超过缓冲区能够容纳的范围，从而导致溢出的数据改写了与该缓冲区相邻存储单元的内容。
- C语言中许多字符串处理函数如：Strcpy、Strcat、Gets、Sprintf等都没有对数组越界加以检测和限制。

缓冲溢出的危害

利用缓冲区溢出实现在本地或者远程系统上，实现任意执行代码的目的，从而进一步达到对被攻击系统的完全掌控；

利用缓冲区溢出进行DoS(Denial of Service)攻击；

利用缓冲区溢出破坏关键数据，使系统的稳定性和有效性受到不同程度的影响；

实现蠕虫程序

- 曾在2001年造成大约26亿美元损失的Code Red蠕虫就是利用了Microsoft IIS中的缓冲区溢出进行攻击
- 2002年的Sapphire蠕虫和2004年的Witty蠕虫也都利用了缓冲区溢出进行攻击。

缓冲溢出的危害（续）

缓冲区溢出可以成为攻击者实现攻击目标的手段，但是单纯地溢出缓冲区并不能达到攻击的目的

- 在绝大多数情况下，一旦程序中发生缓冲区溢出，系统会立即中止程序并报告“fault segment”。例如缓冲区溢出，将使返回地址改写为一个非法的、不存在的地址，从而出现core dump错误，不能达到攻击目的。

只有对缓冲区溢出“适当地”加以利用，攻击者才能通过其实现攻击目标。

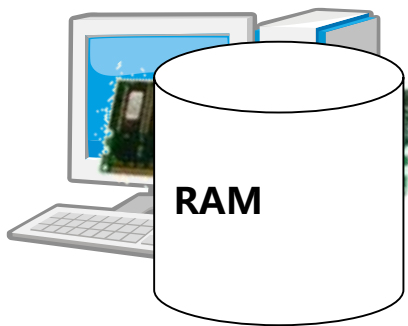
- 超出缓冲空间，覆盖正常程序或数据，使计算机转向预测程序

缓冲区溢出概述

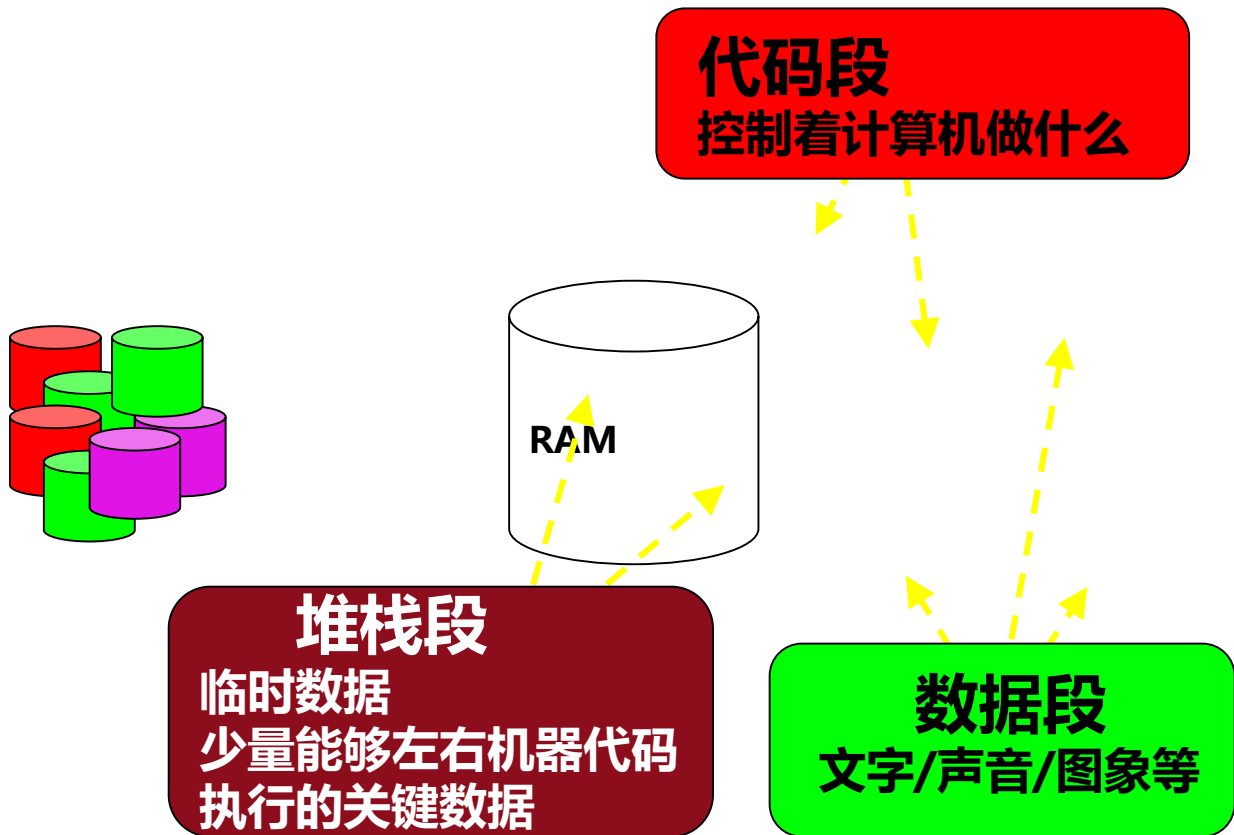
缓冲区溢出攻击的方法和步骤

缓冲区溢出攻击的防御技术

缓冲区溢出攻击原理



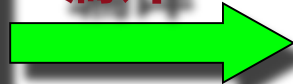
缓冲区溢出攻击原理



堆栈段工作示范

```
void proc(int i)
{
    int local;
    local=i;
}
void main()
{
    proc(1);
}
```

编译



```
main:  push    1
       call   proc

proc:  . . .
       push   ebp
       mov    ebp, esp
       sub    esp, 4
       mov    eax,
[ebp+08]
       mov    [ebp-4],
eax
       add    esp, 4
       pop    ebp
       ret    4
```

堆栈段工作示范

High



```
void proc(int i)
{
    int local;
    local=i;
}
void main()
{
    proc(1);
}
```

编译

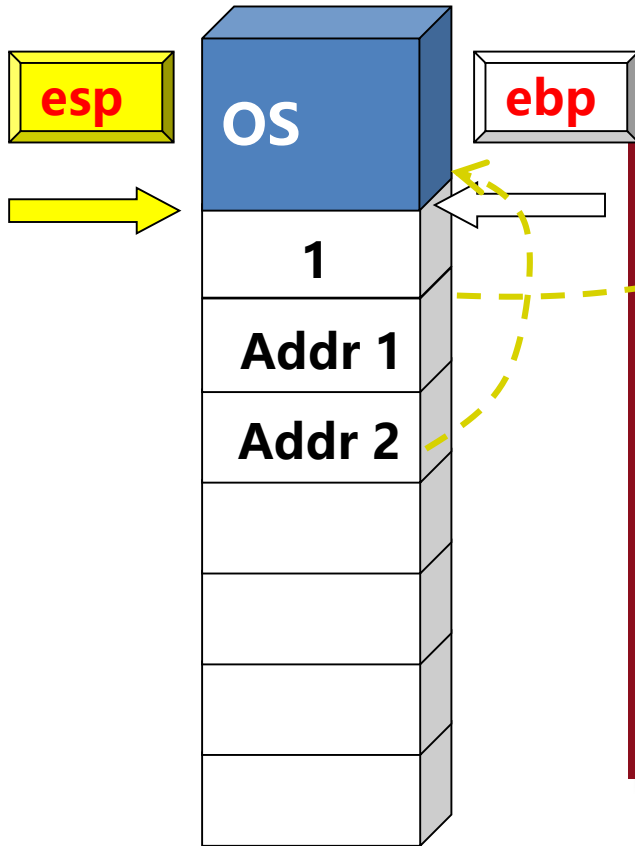


```
main:  push    1
       call   proc
       . . .
proc:  push    ebp
       mov     ebp, esp
       sub     esp, 4
       mov     eax,
[ebp+08]
       mov     [ebp-4],
eax
       add     esp, 4
       pop     ebp
       ret     4
```

Low

堆栈段工作示范

High



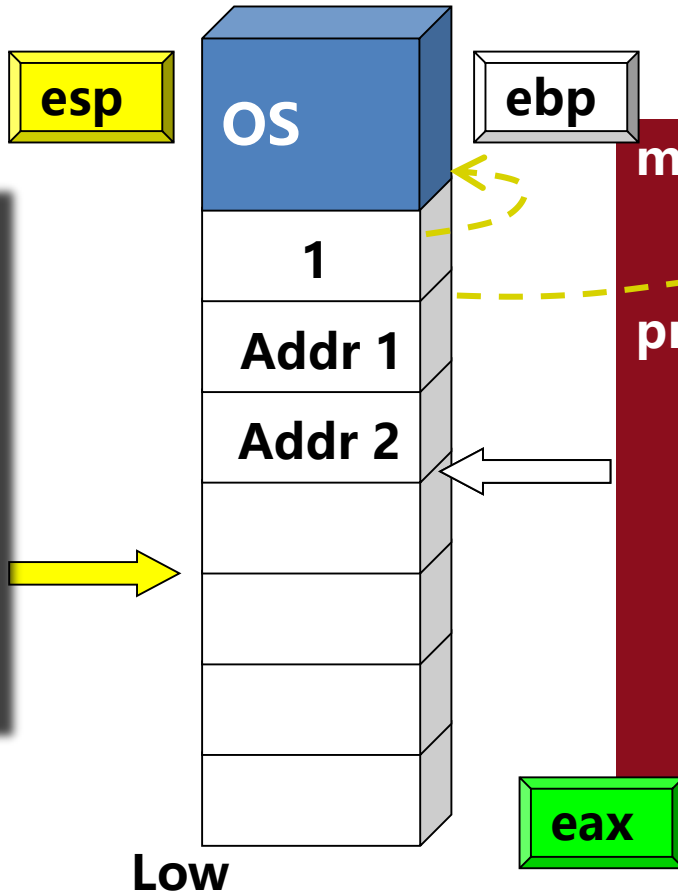
```
main:  push    1  
       call   proc
```

```
proc:  push    ebp  
       mov     ebp, esp  
       sub     esp, 4  
       mov     eax, [ebp+08]  
       mov     [ebp-4], eax  
       add     esp, 4  
       pop     ebp  
       ret     4
```

Low

堆栈段工作示范

High

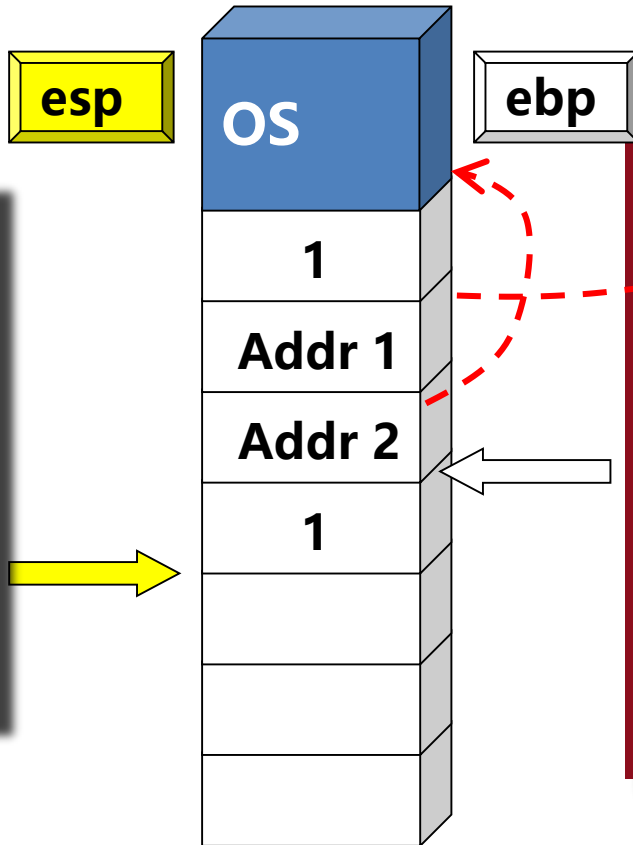


```
main:  push    1
       call    proc

proc:  push    ebp
       mov     ebp, esp
       sub     esp, 4
       mov     eax, [ebp+08]
       mov     [ebp-4], eax
       add     esp, 4
       pop     ebp
       ret     4
```


堆栈段工作示范

High



Low

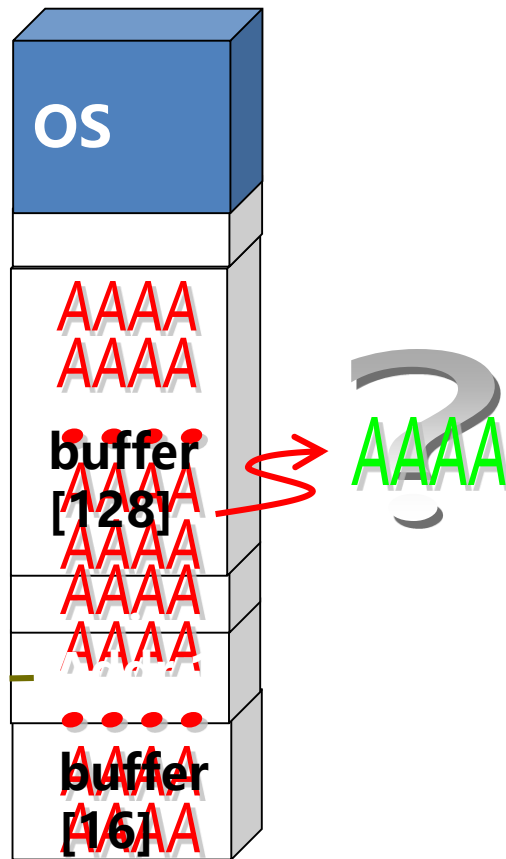
```
main:  push    1
       call    proc

proc:  push    ebp
       mov     ebp, esp
       sub     esp, 4
       mov     eax, [ebp+08]
       mov     [ebp-4], eax
       add     esp, 4
       pop     ebp
       ret     4
```

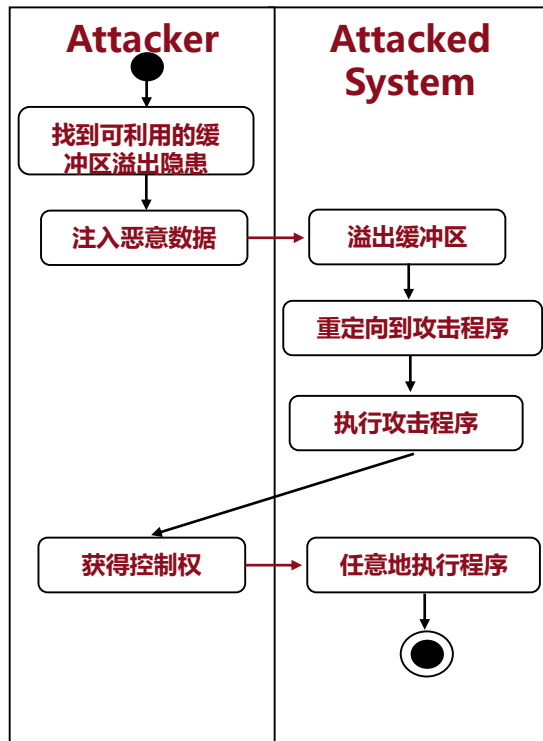
缓冲区溢出攻击原理

```
void function(char *str)
{
    char buffer[16];
    strcpy(buffer,str);
}

void main()
{
    int t;
    char buffer[128];
    for(i=0;i<127;i++)
        buffer[i]='A';
    buffer[127]=0;
    function(buffer);
    printf("This is a test\n");
}
```



缓冲溢出攻击的原理



恶意数据可以通过
命令行参数、
环境变量、
输入文件或者
网络数据注入

缓冲区溢出概述

缓冲区溢出攻击的方法和步骤

缓冲区溢出攻击的防御技术

黄金规则

在编写 C 和 C++ 代码时，对于如何管理来自用户的数据，您应该谨慎从事。

如果一个函数从某个不可信赖的来源接收到缓冲区，请遵守以下这些规则：

- **要求代码传递缓冲区长度**
- **检查内存**
- **采取防御措施**

采取防御措施

在操作或复制不可信赖的数据，时如果误用了某些函数，这些函数就有可能存在严重的安全问题。**联想前面的知识？**

当检查代码中的缓冲区溢出错误时，应该随着数据在代码内的流动而对其进行跟踪，并质疑该数据的有关假设。

如果编写 C++ 代码，请考虑使用自定义字符串操作类来对字符串进行操作，而不要直接对各字节进行操作。唯一潜在的缺点是可能会导致性能下降

缓冲区溢出攻击的防御技术

基于软件的防御技术

- 类型安全的编程语言
- 相对安全的函数库
- 修改的编译器
- 内核补丁
- 静态分析方法
- 动态检测方法

基于硬件的防御技术

- 处理器结构方面的改进

基于软件的防御技术 – 安全的编程语言

类型安全近似于所谓的内存安全（就是限制从内存的某处，将任意的字节复制到另一处的能力）

Java, C#, Visual Basic等属于类型安全的编程语言

缺点

- **性能代价**
- **类型安全的编程语言自身的实现可能存在缓冲区溢出问题**

基于软件的防御技术 – 相对安全的函数库

例如在使用C的标准库函数时，做如下替换

- strcpy -> strncpy
- strcat -> strncat

例如Microsoft Visual Studio
提供的C的标准函数库strsafe;

缺点

- 使用不当仍然会造成缓冲区溢出问题。

基于软件的防御技术 - 修改的编译器

增强边界检查能力的C/C++编译器

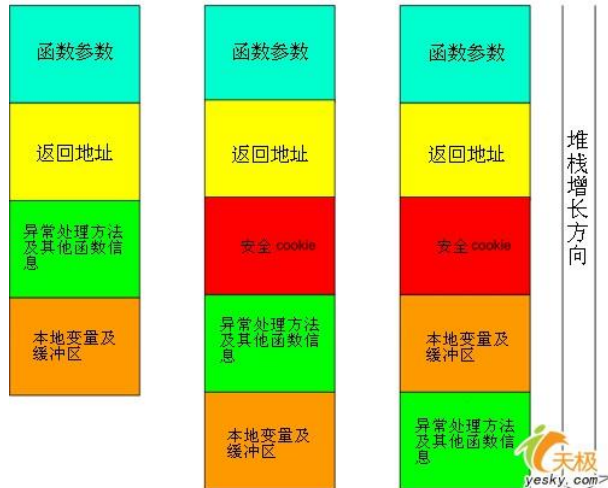
- 提供动态检测缓冲区溢出的能力

返回地址的完整性保护

- 将堆栈上的返回地址备份到另一个内存空间；
在函数执行返回指令前，将备份的返回地址重新写回堆栈。

缺点

- 性能代价
- 检查方法仍不完善



基于软件的防御技术 – 内核补丁

将堆栈标志为不可执行来阻止缓冲区溢出攻击

将堆或者数据段标志为不可执行

例如 Linux 的内核补丁 Openwall 、 RSX、 kNoX、 ExecShield和PaX等实现了不可执行堆栈，并且RSX、 kNoX、 ExecShield、 PaX还支持不可执行堆。

缺点：对于一些需要堆栈/堆/数据段为可执行状态的应用程序不合适；需要重新编译原来的程序，如果没有源代码，就不能获得这种保护。

基于软件的防御技术 - 静态分析方法 1

字典检查法

- 遍历源程序查找其中使用到的不安全的库函数和系统调用。
- 缺点：误报率很高，需要配合大量的人工检查工作。

程序注解法

- 包括缓冲区的大小，指针是否可以为空，输入的有效约定等等。
- 缺点：依赖注释的质量

基于软件的防御技术 - 静态分析方法 2

整数分析法

- 将字符串形式化为一对整数，表明字符串长度(以字节数为单位)以及目前已经使用缓冲区的字节数。通过这样的形式化处理，将缓冲区溢出的检测转化为整数计算
- 缺点：仅检查C中进行字符串操作的标准库函数。检查范围很有限

控制流程分析法

- 将源程序中的每个函数抽象成语法树，然后再把语法树转换为调用图/控制流程图来检查函数参数和缓冲区的范围。
- 缺点：对于运行时才会显露的问题无法进行分析、存在误报的可能

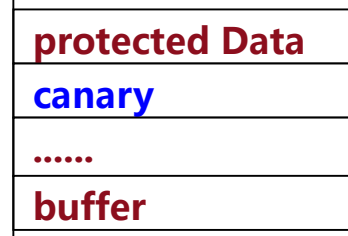
基于软件的防御技术 - 动态检测方法

Canary-based 检测方法

- 将canary(一个检测值)放在缓冲区和需要保护的数据之间，并且假设如果从缓冲区溢出的数据改写了被保护数据，检测值也必定被改写。
- 缺点：很多工具通过修改编译器实现检测功能，需要重新编译程序；这种方法无法检测能绕过检测值的缓冲区溢出攻击。

输入检测方法

- 向运行程序提供不同的输入，检查在这些输入条件下程序是否出现缓冲区溢出问题。不仅能检测缓冲区溢出问题，还可以检测其它内存越界问题。
- 缺点：系统性能明显降低。输入检测方法的检测效果取决于输入能否激发缓冲区溢出等安全问题的出现。



基于硬件的防御技术 - 处理器结构方面的改进

64位处理器的支持

- Intel/AMD 64位处理器引入称为NX(No Execute)或者AVP(Advanced Virus Protection)的新特性, 将以前的CPU合为一个状态存在的“数据页只读”和“数据页可执行”分成两个独立的状态。
- ELF64 SystemV ABI通过寄存器传递函数参数而不再放置在堆栈上, 使得64位处理器不仅可以抵制需要注入攻击代码的缓冲区溢出攻击还可以抵制return-to-libc这类的攻击。
- 缺点: 无法抵制 “borrowed code chunks” 这类的攻击。

思考题

1. 什么程序会发生缓冲区溢出?
2. 缓冲区溢出攻击的一般目标是什么?
3. 要让程序跳转到安排好的地址空间执行, 一般有哪些方法?

谢谢!