栈溢出：这些都是套路

讲师：Atum

内容简介

INTRODUCTION

01 基础知识

02 栈溢出的保护机制

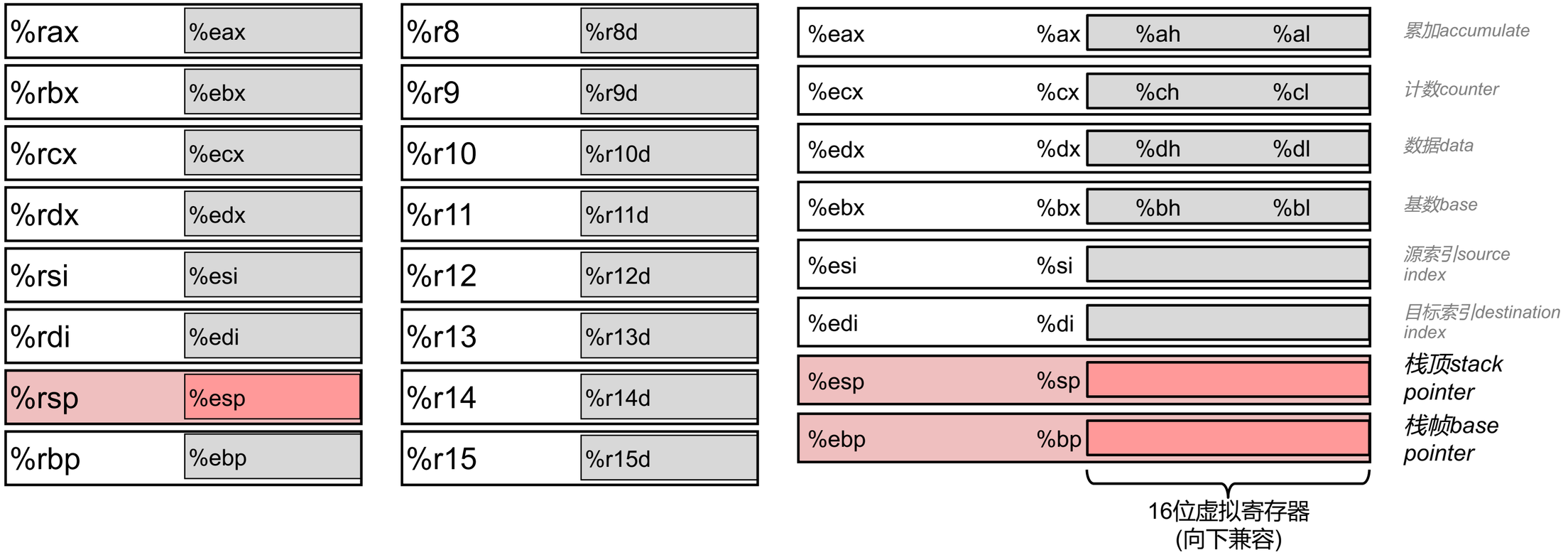
03 栈溢出的利用方法

基础知识

寄存器

重要的寄存器：rsp/esp, pc, rbp/ebp, rax/eax, rdi, rsi, rdx, rcx

作业：了解寄存器各个寄存器的作用，尤其是以上几个重要寄存器

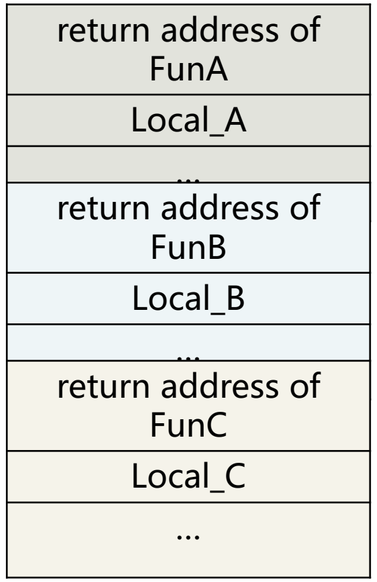


基础知识

栈：一种先进先出的数据结构

程序中的栈：

- 内存中的一块区域，用栈的结构来管理，从高地址向低地址增长
- 寄存器esp代表栈顶（即最低栈地址）
- 栈操作
  - 压栈（入栈）push sth-> [esp]=sth,esp=esp-4
  - 弹栈（出栈）pop sth-> sth=[esp],esp=esp+4
- 栈用于保存函数调用信息和局部变量




```
FunA() {
        local_A;
        FunB();
}
FunB() {
        local_B;
        FunC();
}
Main() {
        local_C;
        FunA();
}
```

作业：详细了解栈溢出漏洞的形式

- Google/Baidu
- 如http://blog.csdn.net/aemperor/article/details/47310593

基础知识

函数调用：call ，ret

调用约定：

- __stdcall，__cdecl，__fastcall，__thiscall，__nakedcall，__pascal

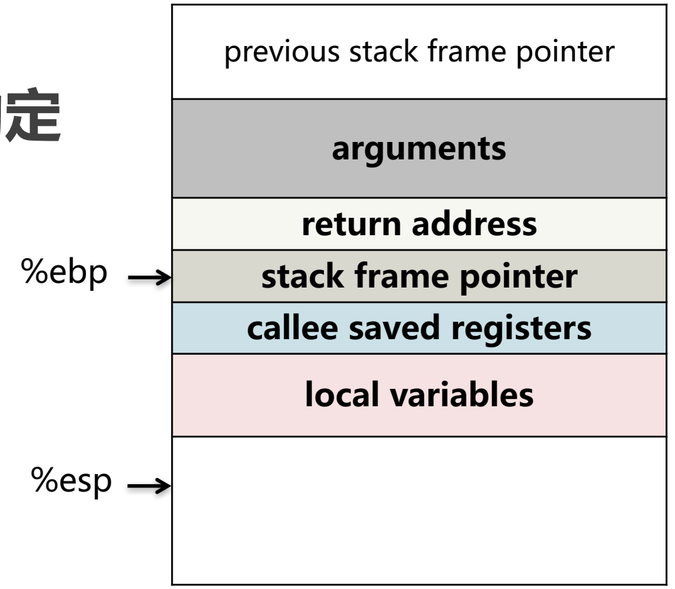
参数传递：取决于调用约定，默认如下

- X86 从右向左入栈，X64 优先寄存器，参数过多时才入栈

作业：了解默认调用约定时函数调用的过程、了解调用约定

函数调用相关指令解读

- Call func -> push pc, jmp func.
- Leave ->mov esp,ebp, pop ebp
- Ret -> pop pc



栈溢出的保护机制

栈上的数据无法被当成指令来执行

- 数据执行保护（NX/DEP）
- 绕过方法：ROP

让攻击者难以找到shellcode地址

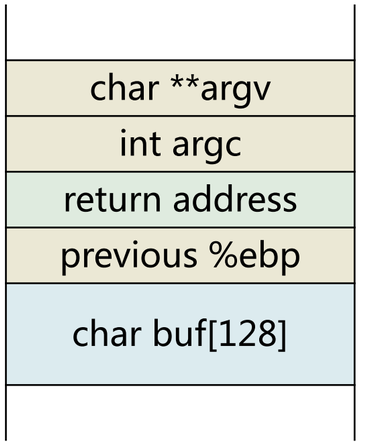
- 地址空间布局随机化（ASLR）
- 绕过方法：infoleak、ret2dlresolve、ROP

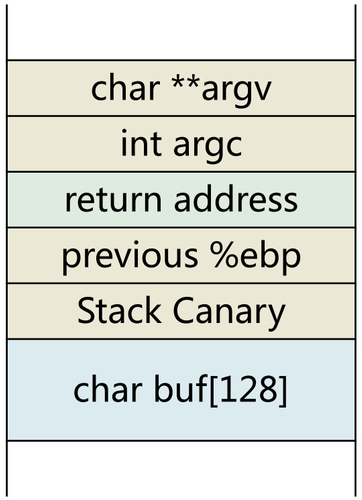
检测Stack Overflow

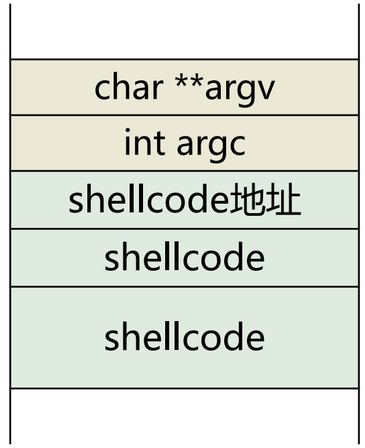
- Stack Canary/Cookie
- 绕过方法：infoleak

现在NX+Stack Cananry+ASLR基本是标配

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(int argc, char **argv) {
5
6      char buf[128];
7
8      if (argc < 2) return 1;
9
10     strcpy(buf, argv[1]);
11
12     printf("argv[1]: %s\n", buf);
13
14     return 0;
15 }
16
```







栈溢出的利用方法

现代栈溢出利用技术基础：ROP

利用signal机制的ROP技术：SROP

没有binary怎么办：BROP

劫持栈指针：stack pivot

利用动态链接绕过ASLR：ret2dl resolve、fake linkmap

利用地址低12bit绕过ASLR：Partial Overwrite

绕过stack canary：改写指针与局部变量、leak canary、overwrite canary

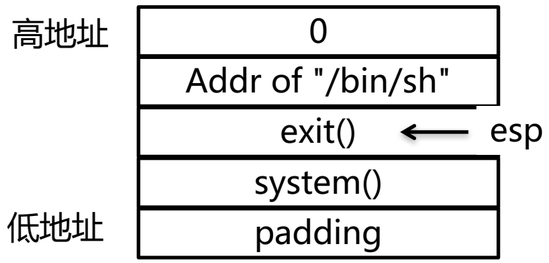
溢出位数不够怎么办：覆盖ebp，Partial Overwrite

现代栈溢出利用技术基础：ROP

一种代码复用技术，通过控制栈调用来劫持控制流。

Google 关键字：Ret2libc，ROP

```
p += pack("<I", 0x08052318) # pop %edx ; ret
p += pack("<I", 0x080c5f9f) # @ .data + 7
p += pack("<I", 0x0809951f) # xor %eax,%eax ; ret
p += pack("<I", 0x080788c1) # mov %eax,(%edx) ; ret
p += pack("<I", 0x08048254) # pop %ebx ; ret
p += pack("<I", 0x080c5f98) # @ .data
p += pack("<I", 0x08052341) # pop %edx ; pop %ecx ; pop %ebx ; ret
p += "AAAA" # padding
p += pack("<I", 0x080c5f9f) # @ .data + 7
p += "AAAA" # padding
p += pack("<I", 0x08052318) # pop %edx ; ret
p += pack("<I", 0x080c5f9f) # @ .data + 7
p += pack("<I", 0x0809951f) # xor %eax,%eax ; ret
p += pack("<I", 0x0804825e) # inc %eax ; ret
p += pack("<I", 0x0804825e) # inc %eax ; ret
p += pack("<I", 0x0804825e) # inc %eax ; ret
p += pack("<I", 0x0804825e) # inc %eax ; ret
p += pack("<I", 0x0804825e) # inc %eax ; ret
p += pack("<I", 0x0804825e) # inc %eax ; ret
p += pack("<I", 0x0804825e) # inc %eax ; ret
p += pack("<I", 0x0804825e) # inc %eax ; ret
p += pack("<I", 0x0804825e) # inc %eax ; ret
p += pack("<I", 0x0804825e) # inc %eax ; ret
p += pack("<I", 0x0804825e) # inc %eax ; ret
p += pack("<I", 0x08048260) # int $0x80
```



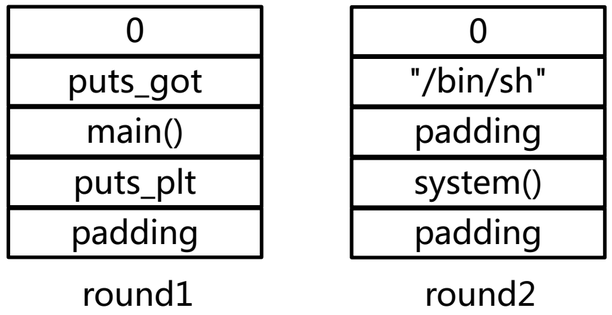
现代栈溢出利用技术基础：ROP

CTF中ROP常规套路：

- 第一次触发漏洞，通过ROP泄漏libc的address(如puts_got)，计算system地址，然后返回到一个可以重现触发漏洞的位置(如main)，再次触发漏洞，通过ROP调用system(“/bin/sh”)
- 直接execve(“/bin/sh”, [“/bin/sh”], NULL)，通常在静态链接时比较常用

习题：

- Defcon 2015 Qualifier：R0pbaby
- AliCTF 2016：vss
- PlaidCTF 2013: ropasaurusrex



作业1：根据r0pbaby的writeup重写exploit

作业2：尝试做一下vss和ropasaurusrex

利用signal机制的ROP技术-SROP

SROP: Sigreturn Oriented Programming

系统Signal Dispatch之前会将所有寄存器压入栈，然后调用signal handler，signal handler返回时会将栈的内容还原到寄存器。

如果事先填充栈，然后直接调用signal return，那在返回的时候就可以控制寄存器的值。

用的不是特别多，但是有时候很好用，推荐资料：

- http://angelboy.logdown.com/posts/283221-srop

  http://www.2cto.com/article/201512/452080.html

例题

Defcon 2015 Qualifier fuckup（这题比较难）

建议自己写一个demo自己测试

没有binary怎么办-BROP

BROP：Blind Return Oriented Programming

目标：在拿不到目标binary的条件下进行ROP

条件：必须先存在一个已知的stack overflow的漏洞，而且攻击者知道如何触发这个漏洞；服务器进程在crash之后会重新复活，并且复活的进程不会被re-rand

用的不是特别多，但是在CTF中出现过

推荐资料：

- http://ytliu.info/blog/2014/05/31/blind-return-oriented-programming-brop-attack-yi/
- http://ytliu.info/blog/2014/06/01/blind-return-oriented-programming-brop-attack-er/

例题：

HCTF 2016 出题人跑路了(pwn50)

作业（选做）：重现一下推荐资料2中实现

劫持栈指针：stack pivot

将栈劫持到其他攻击者控制的缓冲区

- 向目标缓冲区填入栈数据（如ROP Chains），然后劫持esp到目标缓冲区。劫持esp的方法有很多，最常用的就是ROP时利用可以直接改写esp的gadget，如pop esp，ret;
- 是一种相对常用的利用技术，不仅用于栈溢出，也可以用在其他可以劫持控制流的漏洞。

Stack Piovt的动机

- 溢出字节数有限，无法完成ROP
- 栈地址未知且无法泄漏，但是某些利用技术却要求知道栈地址(ret2 dlresolve)
- 劫持esp到攻击者控制的区域，也就变相的控制了栈中的数据，从而可以使非栈溢出的控制流劫持攻击也可以做ROP

劫持栈指针：stack pivot

Stack Pivot的利用条件：

- 存在地址已知且内容可控的buffer
  - bss段，由于bss段尾端通常具有很大的空余空间(pagesize-usedsize)，所以bss段段尾端也往往是stack pivot的目标，
  - 堆块，如果堆地址已泄且堆上的数据可被控制，那堆也可以作为stack pivot的目标
- 控制流可劫持
- 存在劫持栈指针的gadgets
  - 如pop esp, ret，除此之外还有很多，要具体binary具体分析

作业：

- EKOPARTY CTF 2016 fuckzing-exploit-200(基于栈溢出的stack pivot，必做作业)
- HACKIM CTF 2015 - Exploitation 5(基于堆溢出的stackpivot,选做作业)

利用动态链接绕过ASLR：ret2dl resolve、fake linkmap

动态链接的过程就是从函数名到函数地址转换的过程，所以我们可以通过动态链接器来解析任何函数，且无需任何leak

前置技能：了解动态链接的过程

- http://blog.chinaunix.net/uid-24774106-id-3053007.html
- 《程序员的自我修养》

伪造动态链接的相关数据结构如linkmap、relplt，详见以下内容

- http://rk700.github.io/2015/08/09/return-to-dl-resolve/
- http://angelboy.logdown.com/posts/283218-return-to-dl-resolve
- http://www.inforsec.org/wp/?p=389

利用动态链接绕过ASLR：ret2dl resolve、fake linkmap

习题：

- Codegate CTF Finals 2015 yocto(fake relplt) http://o0xmuhe.me/2016/10/25/yocto-writeup/
- HITCON QUALS CTF 2015 readable (fake linkmap)
- Hack.lu‘s 2015 OREO (http://wapiflapi.github.io/2014/11/17/hacklu-oreo-with-ret2dl-resolve/)

理论上任何可以stack pivot且FULLRELRO未开的题目都可以利用这种技术，所以可以试试用这种技术去做一些之前的习题

作业：选择一道题目去完成ret2dlresolve的利用

利用地址低12bit绕过ASLR：Partial Overwrite

在PIE开启的情况下，一个32地址的高20bit会被随机化，但是低12bit是不变的。

所以可以通过只改写低12bit来绕过PIE，不仅在栈溢出使用，在各种利用都经常使用。

Example:

Return address=0x?????abc

System(“/bin/sh”)=0x????def

Overwrite abc by def, we can prompt a shell

作业：了解Partial Overwrite

http://ly0n.me/2015/07/30/bypass-aslr-with-partial-eip-overwrite/

习题：

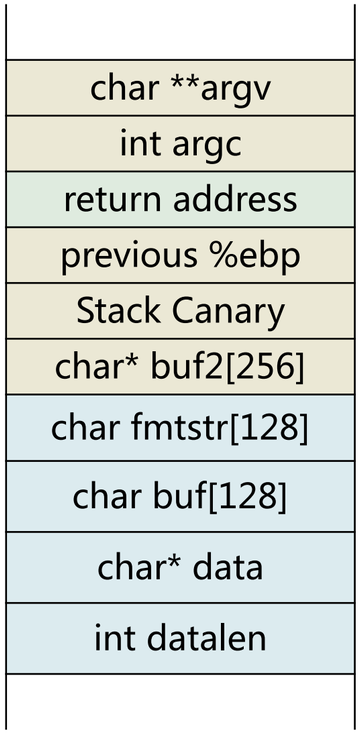
HCTF 2016 fheap(基于堆溢出的parital overwrite)

绕过stack canary

至此所讲的所有套路，一旦遇到Stack Canary均无法使用！！

绕过思路：

- 不覆盖Stack Canary, 只覆盖Stack Canary前的局部变量、指针。
  - 已经几乎不可行，因为编译器会根据占用内存大小从小到大排列变量
  - 但是在某些情况下依然可用，如右图可以覆盖fmtstr buf2.
- Leak Canary
  - 可以通过printf泄漏，Canary一般从00开始
- Overwrite Canary
  - Canary在TLS，TLS地址被随机化



溢出位数不够怎么办：覆盖ebp，Partial Overwrite

Func1:

Call func2

leave (mov esp ebp, pop ebp)

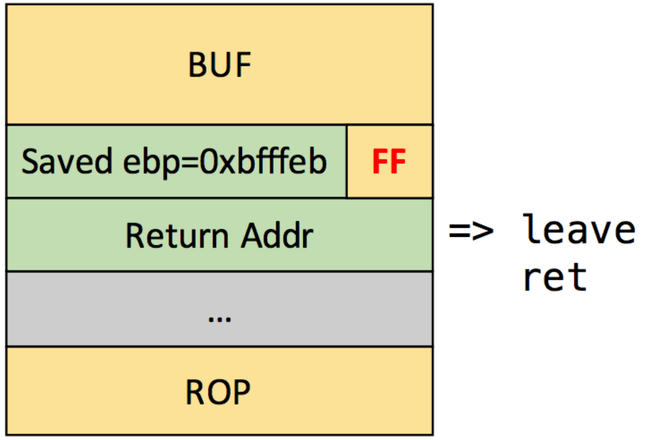
ret (pop ip)

Func 2:

Stack overflow

leave (mov esp ebp, pop ebp)

ret(pop ip)



可以覆盖Func2的ebp, 会影响到Func1的esp，进而影响func1的ip

可以部分覆盖返回地址

习题：

XMAN 2016 广外女生-pwn

Codegate CTF Finals 2015, chess

The End