五、程序的逆向识别（二）

案例分析：缓存区溢出

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

```
#include <stdio.h>
#include <string.h>

void foo()
{
    char dest[7]="12345";
    char* src = "aaaaaaa";
    strcat(dest, src);
    printf("%s\n", dest);
}

int main()
{
    foo();
    return 0;
}
```

案例分析：缓存区溢出

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

```
#include <stdio.h>
#include <stdlib.h>

void foo(void)
{
    int a, *p;
    p = (int*)((char *)&a + 12);
    *p += 12;
}
int main(void)
{
    foo();
    printf("First printf call\n");
    printf("Second printf call\n");
    return 0;
}
```

识别函数

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 程序都是由具有不同功能的函数组成的
- 在逆向分析中将重点放在函数的识别及参数的传递上，这样做可以将注意力集中在某一段代码上
- 函数是一个程序模块，用来实现一个特定的功能
- 一个函数包括函数名、入口参数、返回值、函数功能等部分

识别函数

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

❖ 函数的识别

- 程序通过调用程序来调用函数，在函数执行后又返回调用程序继续执行
- 调用函数的代码中保存了一个返回地址，该地址会与参数一起传递给被调用的函数
- 在绝大多数情况下，编译器都使用call和ret指令来调用函数及返回调用位置

识别函数

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

❖ 函数的识别

- call指令保存返回信息，即将其之后的指令地址压入栈的顶部
- 当遇到ret指令时返回这个地址
- call指令给出的地址就是被调用函数的起始地址
- ret指令则用于结束函数的执行

识别函数

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

❖ 函数的识别

```
int Add(int x,int y);
main( )
{
     int a=5,b=6;
     Add(a,b);
     return 0;

 }

Add(int x,int y)
{
     return(x+y);
}
```

❖ 函数的参数

- 函数传递参数有3种方式
  - 栈方式：需要定义参数在栈中的顺序，并约定函数被调用后由谁来平衡栈
  - 寄存器：确定参数存放在哪个寄存器中
  - 全局变量进行隐含参数传递
- 每种机制都有其优缺点，且与使用的编译语言有关

❖ 函数的参数（利用栈传递参数）

- 栈是一种“后进先出”(Last-In-First-Out, LIFO)的存储区
- 栈顶指针esp指向栈中第1个可用的数据项
- 在调用函数时，调用者依次把参数压入栈，然后调用函数
- 函数被调用以后，在栈中取得数据并进行计算
- 函数计算结束以后，由调用者或者函数本身修改栈，使栈恢复原样（即平衡栈数据）

❖ 函数的参数（利用栈传递参数）

- 当参数个数多于1个时，按照什么顺序把参数压入栈?
- 函数结束后，由谁来平衡栈?

约定类型	__cdecl（C 规范）	pascal	stdcall	Fastcall
参数传递顺序	从右到左	从左到右	从右到左	使用寄存器和栈
平衡栈者	调用者	子程序	子程序	子程序
允许使用 VARARG	是	否	是	

__cdecl 调用约定	pascal 调用约定	stdcall 调用约定
push par3 ;参数从右到左传递 push par2 push par1 call test1 add esp,0C ;平衡栈	push par1 ;参数从左到右传递 push par2 push par3 call test1 ;函数内平衡栈	push par3 ;参数从右到左传递 push par2 push par1 call test1 ;函数内平衡栈

函数的返回值

- 函数被调用执行后，将向调用者返回1个或多个执行结果，称为函数返回值
- 返回值最常见的形式是return操作符
- 还有通过参数按传引用方式返回值、通过全局变量返回值等

函数的返回值

- 用return操作符返回值
- 在一般情况下，函数的返回值放在eax寄存器中返回，如果处理结果的大小超过eax寄存器的容量，其高32位就会放到edx寄存器中

```
MyAdd(int x,int y)
{
    int temp;                    //局部变量
    temp = x+y;                  //计算
    return temp;                 //返回值
}
```

识别函数

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

主 程 序		MyAdd 函数	
push x	;参数 1	push ebp	;保存 ebp
push y	;参数 2	mov ebp, esp	;设置新的 ebp 指针
call MyAdd	;调用函数	sub esp, 4	;为局部变量分配空间
	;栈在 MyAdd 函数里平衡	mov ebx, [ebp+0C]	;取第 1 个参数
mov ..., eax	;返回值在 eax 中	mov ecx, [ebp+8]	;取第 2 个参数
		add ebx, ecx	;相加
		mov [ebp-4], ebx	;将结果放到局部变量中
		mov eax, [ebp-4]	;将局部变量返回 eax
		mov esp, ebp	;恢复现场
		add esp,4	;平衡栈
		ret	;返回

识别函数

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 给函数传递参数的方式有两种，分别是传值和传引用
- 进行传值调用时，会建立参数的一份复本，并把它传给调用函数，在调用函数中修改参数值的复本不会影响原始的变量值
- 传引用调用允许调用函数修改原始变量的值。调用某个函数，当把变量的地址传递给函数时，可以在函数中用间接引用运算符修改调用函数内存单元中该变量的值

识别函数

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

```
#include <stdio.h>
void max(int *a, int *b);

main( )
{
     int a=5,b=6;
     max(&a, &b);
     printf("a、b 中较大的数是%d",a);      //将较大的数显示出来
}

void max( int *a, int *b)
{
     if(*a < *b)
          *a=*b;                          //经比较，将较大的数放到变量 a 中
}
```

识别函数

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

```
                                                    ;设此时 esp=k
00401000    sub     esp, 00000008                   ;为局部变量分配内存
00401003    lea     eax, dword ptr[esp+04]          ;eax 指向变量，值为 k-4h
00401007    lea     ecx, dword ptr[esp]             ;ecx 指向变量，值为 k-8h
0040100B    push    eax                             ;指向参数 b 的字符指针入栈
0040100C    push    ecx                             ;指向参数 a 的字符指针入栈
0040100D    mov     [esp+08], 00000005              ;[esp+08]的值为 k-8h，将参数 a 的值放入
00401015    mov     [esp+0C], 00000006              ;[esp+0C]的值为 k-4h，将参数 b 的值放入
0040101D    call    00401040                        ;max(&a,&b)
00401022    mov     edx, [esp+08]                   ;利用变量[esp+08]返回函数值
00401026    push    edx
00401027    push    00407030
0040102C    call    00401060                        ;printf 函数
00401031    xor     eax, eax
00401033    add     esp, 18
00401036    retn

;max(&a, &b)函数的汇编代码
00401040    mov     eax, dword ptr [esp+08]         ;执行后，eax 就是指向参数 b 的指针
00401044    mov     ecx, dword ptr [esp+04]         ;执行后，ecx 就是指向参数 a 的指针
00401048    mov     eax, dword ptr [eax]            ;将参数 b 的值加载到寄存器 eax 中
0040104A    mov     edx, dword ptr [ecx]            ;将参数 a 的值加载到寄存器 edx 中
0040104C    cmp     edx, eax                        ;比较参数 a 和参数 b 的大小
0040104E    jge     00401052                        ;若 a<b，则不跳转
00401050    mov     dword ptr [ecx], eax            ;将较大的值放到参数 a 所指的数据区中
00401052    ret
```

识别数据结构

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 数据结构是计算机存储、组织数据的方式
- 在进行逆向分析时，确定数据结构以后，算法就很容易得到了
- 有些时候，事情也会反过来，即根据特定算法来判断数据结构

- 局部变量
  - 局部变量（Local Variables)是函数内部定义的一个变量
  - 其作用域和生命周期局限于所在函数内
  - 从汇编的角度来看，局部变量分配空间时通常会使用栈和寄存器

- 局部变量（**利用栈存放局部变量**）
  - 用 sub esp, 8 语句为局部变量分配空间
  - 用 [ebp-xxxx] 寻址调用这些变量
  - 用 [ebp+xxxx] 寻址调用参数（参数调用相对于ebp偏移量是正的）
  - 当函数退出时，用 add esp, 8 指令平衡栈

形 式 1	形 式 2	形 式 3

sub esp, n
……
add esp,n
	add esp,-n
	……
	sub esp,-n
		push reg
		……
		pop reg

- 局部变量（**利用栈存放局部变量**）

```
int add(int x,int y);

int main(void)
 {
      int a=5,b=6;      //声明局部变量 a 和 b，同时对变量进行初始化
      add(a,b);
      return 0;
 }

 int add(int x,int y)
 {
      int z;            //声明局部变量 z
      z=x+y;
      return(z);
 }
```

- 局部变量（**利用寄存器存放局部变量**）
  - 除了栈占用2个寄存器，编译器会利用剩下的6个通用寄存器尽可能有效地存放局部变量，这样可以少产生代码，提高程序的效率
  - 如果寄存器不够用，编译就会将变量放到栈中
  - 在进行逆向分析时要注意，局部变量的生存周期比较短，必须及时确定当前寄存器的变量是哪个变量

全局变量

- 全局变量作用于整个程序，它一直存在，放在全局变量的内存区中
- 在大多数程序中，常数一般放在全局变量中
- 全局变量通常位于数据区块（.data)的一个固定地址处，当程序需要访问全局变量时，一般会用一个固定的硬编码地址直接对内存进行寻址
- 全局变量可以被同一文件中的所有函数修改

```
mov eax, dword ptr [4084C0h]        ;直接调用全局变量，其中 4084C0h 是全局变量的地址
```

❖ 全局变量

```
int z;                          //全局变量 z
int add(int x,int y);

int main(void)
 {
    int a=5,b=6;
    z=7;
    add(a,b);
    return 0;
 }

 int add(int x,int y)
 {
    return(x+y+z);
 }
```

识别数据结构

数组

- 数组是相同数据类型的元素的集合
- 在内存中按顺序连续存放在一起
- 在汇编状态下访问，数组一般是通过基址加变址寻址实现的

```
int main(void)
{
 static int a[3]={0x11,0x22,0x33};
 int i,s=0,b[3];

for(i=0;i<3;i++)
{
     s=s+a[i];
     b[i]=s;
}

for(i=0;i<3;i++)
{
     printf("%d\n",b[i]);
}

     return 0;
}
```