反汇编这段汇编代码

.text:080484EB public vuln

.text:080484EB vuln proc near ; CODE XREF: main+16↓p

.text:080484EB

.text:080484EB var\_10 = dword ptr -10h

.text:080484EB var\_C = dword ptr -0Ch

.text:080484EB var\_8 = dword ptr -8

.text:080484EB var\_4 = dword ptr -4

.text:080484EB

.text:080484EB ; \_\_unwind {

.text:080484EB push ebp

.text:080484EC mov ebp, esp

.text:080484EE sub esp, 10h

.text:080484F1 mov [ebp+var\_10], 0Ah

.text:080484F8 mov [ebp+var\_4], 0

.text:080484FF mov [ebp+var\_C], 0

.text:08048506 jmp short loc\_804851D

.text:08048508 ; ---------------------------------------------------------------------------

.text:08048508

.text:08048508 loc\_8048508: ; CODE XREF: vuln+38↓j

.text:08048508 add [ebp+var\_4], 1

.text:0804850C cmp [ebp+var\_4], 5

.text:08048510 jle short loc\_8048519

.text:08048512 mov [ebp+var\_8], 2Ch

.text:08048519

.text:08048519 loc\_8048519: ; CODE XREF: vuln+25↑j

.text:08048519 add [ebp+var\_C], 1

.text:0804851D

.text:0804851D loc\_804851D: ; CODE XREF: vuln+1B↑j

.text:0804851D mov eax, [ebp+var\_C]

.text:08048520 cmp eax, [ebp+var\_10]

.text:08048523 jl short loc\_8048508

.text:08048525 mov eax, [ebp+var\_8]

.text:08048528 leave

.text:08048529 retn

.text:08048529 ; } // starts at 80484EB

.text:08048529 vuln endp

首先可以看到函数开始的内存地址为0x080484EB，定义一个名为vuln的函数，其中CODE XREF: main+16↓p代表这是一个交叉引用，在main函数中起始地址偏移16字节的位置有代码引用了该函数，

.text:080484EB public vuln

.text:080484EB vuln proc near ; CODE XREF: main+16↓p

.text:080484EB

因此该部分的代码应该反汇编为

void vuln()

然后该函数会定义四个双字（dword）类型的局部变量，其中var\_10相对于基指针向低地址偏移0x10个字节，var\_C相对于基指针向低地址偏移0xC个字节，var\_8和var\_4与其相同，观察后面的汇编指令可以知道给变量赋值的值为数值形式，并且变量大小为双字32位，所以可以将其设置为int类型,为了方便后面的步骤将四个变量分别定义为a,b,c,d

.text:080484EB var\_10 = dword ptr -10h

.text:080484EB var\_C = dword ptr -0Ch

.text:080484EB var\_8 = dword ptr -8

.text:080484EB var\_4 = dword ptr -4

.text:080484EB

因此该部分的代码应该反汇编为

int a;

int b;

int c;

int d;

接下来首先将ebp的内容入栈保存调用者的基址指针，将堆栈指针（esp）的值赋给基址指针（ebp）。这一步建立了当前函数的堆栈帧，将 ebp 设置为当前栈帧的基址，然后在栈上分配 16 （0x10）字节的空间用于局部变量。之后将0xA存储到var\_10的内存地址中，将0x0存储到var\_4和var\_c中，之后无条件跳转到标签 loc\_804851D

.text:080484EB ; \_\_unwind {

.text:080484EB push ebp

.text:080484EC mov ebp, esp

.text:080484EE sub esp, 10h

.text:080484F1 mov [ebp+var\_10], 0Ah

.text:080484F8 mov [ebp+var\_4], 0

.text:080484FF mov [ebp+var\_C], 0

.text:08048506 jmp short loc\_804851D

.text:08048508 ;

因此该部分的代码可以反汇编为

a=10;

d=0;

b=0;

首先定义一个代码位置的标签loc\_8048508，接下来将var\_4（d）变量加1，之后将该变量与立即数5进行比较，如果 var\_4 (d)的值小于或等于 5（Less than or Equal），则执行一个短距离跳转到 loc\_8048519 标签；如果var\_4(d)的值大于5则将0x2c保存到变量var\_8（c）中

.text:08048508

.text:08048508 loc\_8048508: ; CODE XREF: vuln+38↓j

.text:08048508 add [ebp+var\_4], 1

.text:0804850C cmp [ebp+var\_4], 5

.text:08048510 jle short loc\_8048519

.text:08048512 mov [ebp+var\_8], 2Ch

.text:08048519

因此该部分代码可以反汇编为

d++;

if(d>5)

{

c=44;

}

else{}

接下来这段指令只是将变量b加1，并且是在上面的if-else语句中,由此可以填入上一步中的else中

.text:08048519 loc\_8048519: ; CODE XREF: vuln+25↑j

.text:08048519 add [ebp+var\_C], 1

.text:0804851D

反汇编结果为

else

{

b++;

}

最后这一部分是程序开始后的跳转位置，首先将b的值加载到eax寄存器中，然后比较eax与a的值，即如果b<a则跳转到loc\_8048508位置，如果b>=a则将c的值加载到eax中设置函数的返回值，最后进行代码return

.text:0804851D loc\_804851D: ; CODE XREF: vuln+1B↑j

.text:0804851D mov eax, [ebp+var\_C]

.text:08048520 cmp eax, [ebp+var\_10]

.text:08048523 jl short loc\_8048508

.text:08048525 mov eax, [ebp+var\_8]

.text:08048528 leave

.text:08048529 retn

.text:08048529 ; } // starts at 80484EB

.text:08048529 vuln endp

这段程序可以反汇编成

if(b<a)

{

jump loc\_8048508;

}

else

{

return c;

}

因此最终我们可以得到的c语言程序为

void vuln(){

int a;

int b;

int c;

int d;

a=10;

d=0;

b=0;

while(b<a){

d++;

if(d>5){

c=44;

}

else{

b++;

}

}

return c;

}

将该程序进行格式优化可以得到最终的C程序

void vuln(){

int a=10;

int b=0;

int c;

int d=0;

while(b<a){

d++;

if(d>5){

c=44;

}

else{

b++;

}

}

return c;

}