easyCpp 攻防世界

原创

北风~ 于 2020-05-01 13:28:57 发布 876 收藏

分类专栏： 逆向与保护

本文链接：https://blog.csdn.net/weixin_45055269/article/details/105876587

逆向与保护 专栏收录该内容

65 篇文章 4 订阅

订阅专栏

工具

IDA+STL知识

思路展开

复杂语句，大量的C++ STL语句，vector容器等相关概念。

动态调试

整体梳理一遍逻辑：

16个整型输入

```
for ( i = 0; i <= 15; ++i )
{
  scanf("%d", &input_1[4 * i], v20);
  std::vector<int,std::allocator<int>>::push_back(&v24, &input_1[4 * i]);
}
```

生成16个斐波那契数（这里和上面的16 个输入对应起来）

```
for ( j = 0; j <= 15; ++j )
{
  LODWORD(input_begin) = fib(j);
  std::vector<int,std::allocator<int>>::push_back(&fibs, &input_begin);
}
```

输入经过transform和accumulate两个函数运算出结果

```
53  two_input_addr = __gnu_cxx::__normal_iterator<int *,std::vector<int,std::allocator<int>>>::operator+(
54                     &input_begin,
```

```
                    1LL);
  std::transform<__gnu_cxx::__normal_iterator<int *,std::vector<int,std::allocator<int>>>,std::back_insert_i
    two_input_addr,
    input_end,
    one_input_addr,
    (__int64)input_1);
  std::vector<int,std::allocator<int>>::vector(&v28, input_end, v11);
  v12 = std::vector<int,std::allocator<int>>::end(&one_input);
  v13 = std::vector<int,std::allocator<int>>::begin(&one_input);
  std::accumulate<__gnu_cxx::__normal_iterator<int *,std::vector<int,std::allocator<int>>>,std::vector<int,s
    (__int64)&input_begin,
    v13,
    v12,
    (__int64)&v28,
    v14,
    v15,
00000F6D main:68 (400F6D)
```

运算出的结果与斐波那契前16 个比较，不相等则错误。

```
if ( (unsigned __int8)std::operator!=<int,std::allocator<int>>((__int64)&input_encry, (__int64)&fibs) )
{
  puts("You failed!");
  exit(0);
}
```

斐波那契前16 个数是公开的，搞清楚transform和accumulate在干什么，逆推输入就可。

动调下断点：这几个函数和最终判断环环相扣，在这两个函数和最后的关键判断下断点，看输入就知道上一个操作在干什么。

输入

100 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45

transform：

```
0000000002487380  00 00 00 00 00 00 00 00  51 00 00 00 00 00 00 00  ........Q.......
0000000002487390  64 00 00 00 83 00 00 00  84 00 00 00 85 00 00 00  d...............
00000000024873A0  86 00 00 00 87 00 00 00  88 00 00 00 89 00 00 00  ................
00000000024873B0  8A 00 00 00 8B 00 00 00  8C 00 00 00 8D 00 00 00  ................
00000000024873C0  8E 00 00 00 8F 00 00 00  90 00 00 00 91 00 00 00  ................
00000000024873D0  00 00 00 00 00 00 00 00  31 EC 00 00 00 00 00 00  ........1.......
00000000024873E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

从第二个数开始每一个数都加上第一个数。

```
000017E3560  91 00 00 00 90 00 00 00  8F 00 00 00 8E 00 00 00  ..........
000017E3570  8D 00 00 00 8C 00 00 00  8B 00 00 00 8A 00 00 00  ..........
000017E3580  89 00 00 00 88 00 00 00  87 00 00 00 86 00 00 00  ..........
000017E3590  85 00 00 00 84 00 00 00  83 00 00 00 64 00 00 00  ..........
000017E35A0  00 00 00 00 00 00 00 00  61 EA 00 00 00 00 00 00  ........a.
000017E35B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ..........
```

accumulate

逆序

脚本

```
fib = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987][::-1]
for i in range(1,len(fib)):
 print (fib[i]-fib[0])
```

求出输入再输入就出flag
flag{987-377-843-953-979-985}

静态分析

c++逆向题代码很多，如何找到关键：
push_back() 在Vector最后添加一个元素（参数为要插入的值）
Iterator（迭代器），迭代器的作用：遍历字符
（back_inserter（尾部插）,inserter（插入指定位置）,front_inserter（头部插））
本题中大量迭代器的使用，真正的函数只有transform和accumulate，看到这里整体就明朗了。

0x01输入

```
for ( i = 0; i <= 15; ++i )
{
  scanf("%d", &input_1[4 * i], v20);
  std::vector<int,std::allocator<int>>::push_back(&v24, &input_1[4 * i]);
}
```

push_back() 在Vector最后添加一个元素（参数为要插入的值）
输入16个整型放到v24 Vector容器。

0x02生成16个斐波那契数

```
for ( j = 0; j <= 15; ++j )
{
  LODWORD(input_begin) = fib(j);
  std::vector<int,std::allocator<int>>::push_back(&fibs, &input_begin);
}
```

生成16个斐波那契数放到fibs Vector容器。

0x03.transform

点开可以看到一大堆iterator相关的代码，但关键是其中的lambda函数，也就是关键的一元操作。

```
  v8 = a3;
  v7 = a4;
  v12 = __readfsqword(0x28u);
  while ( (unsigned __int8)__gnu_cxx::operator!=<int *,std::vector<int,std::allocator<int>>>(&v10, &v9) )
  {
    v4 = (unsigned int *)__gnu_cxx::__normal_iterator<int *,std::vector<int,std::allocator<int>>>::operator*(
    v11 = main::{lambda(int)#1}::operator() const(&v7, *v4);
    v5 = std::back_insert_iterator<std::vector<int,std::allocator<int>>>::operator*(&v8);
    std::back_insert_iterator<std::vector<int,std::allocator<int>>>::operator=(v5, &v11);
    __gnu_cxx::__normal_iterator<int *,std::vector<int,std::allocator<int>>>::operator++(&v10);
    std::back_insert_iterator<std::vector<int,std::allocator<int>>>::operator++(&v8);
  }
  return v8;
}
```


lamda

```
__int64 __fastcall main::{lambda(int)#1}::operator() const(_DWORD **a1, int a2)
{
  return (unsigned int)(**a1 + a2);
}
```

简单的加法，再结合下面的

```
  __gnu_cxx::__normal_iterator<int *,std::vector<int,std::allocator<int>>>::operator++(&v10);
  std::back_insert_iterator<std::vector<int,std::allocator<int>>>::operator++(&v8);
```

operator的自加，就是后面的15个数，都加上第一个数，再返回。

0x04.accumulate

accumulate点进去，关键也在lambda

```
  v11 = a2;
  v10 = a3;
  v9 = a4;
  v15 = __readfsqword(0x28u);
  while ( (unsigned __int8)__gnu_cxx::operator!=<int *,std::vector<int,std::allocator<int>>>(&v11, &v10) )
  {
    v7 = *(_DWORD *)__gnu_cxx::__normal_iterator<int *,std::vector<int,std::allocator<int>>>::operator*(&v11);
    std::vector<int,std::allocator<int>>::vector(&v13, v9);
    main::{lambda(std::vector<int,std::allocator<int>>,int)#2}::operator() const(&v14, &a7, &v13, v7);
    std::vector<int,std::allocator<int>>::operator=(v9, &v14);
    std::vector<int,std::allocator<int>>::~vector(&v14);
    std::vector<int,std::allocator<int>>::~vector(&v13);
    __gnu_cxx::__normal_iterator<int *,std::vector<int,std::allocator<int>>>::operator++(&v11);
  }
  std::vector<int,std::allocator<int>>::vector(v12, v9);
  return v12;
}
0000133F _ZSt10accumulateIN9__gnu_cxx17__normal_iteratorIPiSt6vectorIiSaIiEEEES5_Z4mainEUlS5_iE0_ET0_T_S9_S8_T1_:21 (40133F)
```

点进去lambda

```
  v8 = a4;
  v12 = __readfsqword(0x28u);
  std::vector<int,std::allocator<int>>::vector(a1, a2, a3);
  std::vector<int,std::allocator<int>>::push_back(a1, &v8);
  v4 = std::back_inserter<std::vector<int,std::allocator<int>>>(v11);
  v5 = std::vector<int,std::allocator<int>>::end(v9);
  v6 = std::vector<int,std::allocator<int>>::begin(v9);
  std::copy<__gnu_cxx::__normal_iterator<int *,std::vector<int,std::allocator<int>>>,std::back_insert_iterator<std::vector<int,std::al
    v6,
    v5,
    v4);
  return v11;
}
```

操作的地方只有copy函数，函数一层套着一层的变量相互赋值，这里一定要做好注释，你会发现是尾部赋值给头部，就是逆序

0x05.脚本破解

```
fib = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987][::-1]
for i in range(1,len(fib)):
 print (fib[i]-fib[0])
```

求出输入再输入就出flag

flag{987-377-843-953-979-985}

总结

c++题目代码多，但逻辑总会出的简单