C++逆向学习(一) string

ret2nullptr / 2019-04-24 08:58:00 / 浏览数 6595 安全技术 二进制安全 顶(5) 踩(0)

CTF比赛中C++的题越来越多，题目中经常出现`string`，`vector`等，而实际上手时发现常常迷失在"库函数"中，比如跟进了空间配置器相关函数

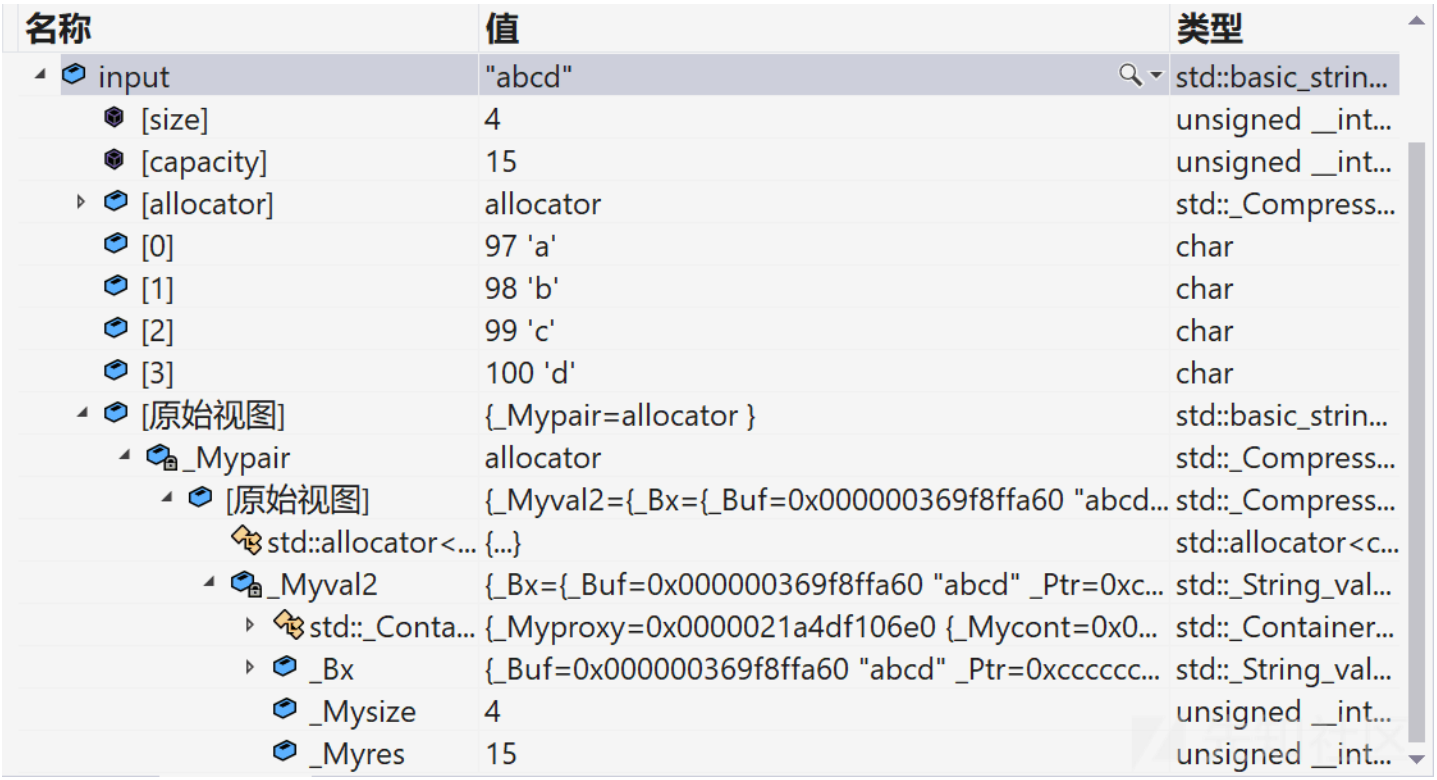
最近研究一下关于这些的底层机制与逆向，应该会写成一个系列

string

内存布局

`visual studio`的调试实在是太好用了，因此用它举例

定义一个`string`类，字符串为`abcd`，内存布局如下

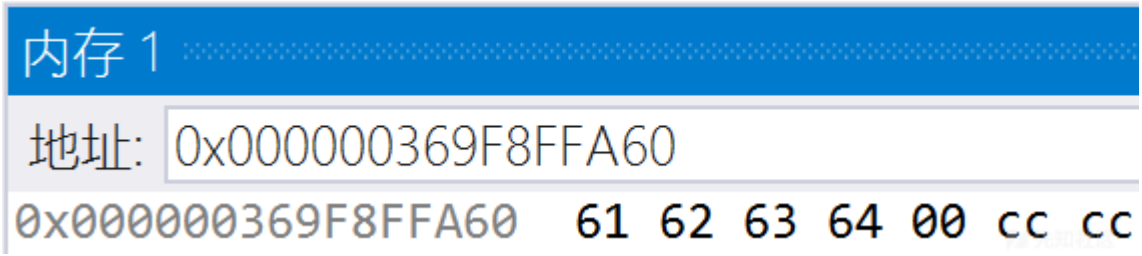


其中，`size`是当前字符串长度，`capacity`是最大的容量

可以发现，`capacity`比`size`大的多

而`allocator`是空间配置器，可以看到单独的字符显示

原始视图中可以得知，字符串的首地址



可以看到，`abcd`字符串在内存中也是以`\x00`结尾的

扩容机制

正是由于`capacity`开辟了更多需要的空间，来具体研究一下它的策略

```
#include<iostream>
#include<string>
#include<stdlib.h>
#include<windows.h>

using namespace std;

int main(int argc, char** argv) {
   string str;
   for (int i = 0; i < 100; i++) {
       str += 'a';
       std::cout << "size : " << str.size() << "   capacity : " << str.capacity() << std::endl;
   }
   system("pause");
   return 0;
}
```

从输出结果发现，capacity的变化为15 -> 31 -> 47 -> 70 -> 105

注意到15是二进制的1111，而31是二进制的11111，可能是设计成这样的？...

只有第一次变化不是1.5倍扩容，后面都是乘以1.5

当长度为15时，如下，两个0x0f表示长度，而第一行倒数第三个0f则表示的是当前的capacity

```
61 61 61 61 61 61 61 61 61 61 61 61 61 61 00 0f 00 00 00 00 00 00 0f 00 00  aaaaaaaaaaaaaaa...
00 00 00 00 00 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc 0f 00  .....???????????????
```

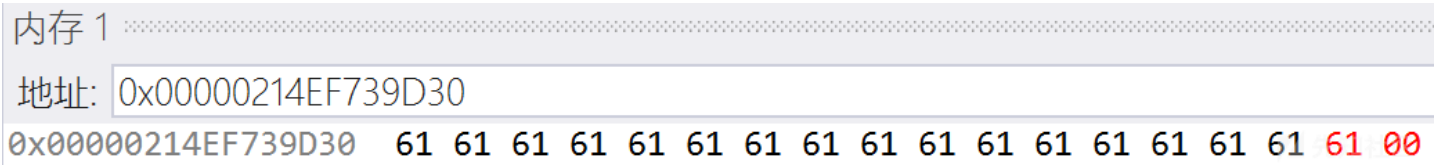
再次+='a'

```
30 9d 73 ef 14 02 00 00 61 61 61 61 61 61 61 00 10 00 00 00 00 00 00 00 1f 00 00  0?s?....aaaaaaa...
00 00 00 00 00 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc 10 00  .....???????????????
```

原先的capacity已经从0x0f变成了0x1f，长度也变成了16

而原先存储字符串的一部分内存也已经被杂乱的字符覆盖了

新的字符串被连续存储在另一块地址



vs的调试中，红色代表刚刚改变的值

不过原先使用的内存里还有一些aaaa...，可能是因为还没有被覆盖到

IDA视角

测试程序1

```
#include<iostream>
#include<string>

using namespace std;

int main(int argc, char** argv) {
   string input;
   cin >> input;
   for (int i = 0; i < 3; i++) {
       input += 'a';
   }
   for (int i = 0; i < 3; i++) {
       input.append("12345abcde");
   }
   cout << input << endl;
   return 0;
}
```

//visual studio 2019 x64 release

我用的IDA7.0，打开以后发现IDA似乎并没有对string的成员进行适合读代码的命名，只好自己改一下

```
size = 0i64;
capacity = 15i64;
LOBYTE(string) = 0;
std::operator___char_std::char_traits_char__std::allocator_char___(std::cin, &string, envp);
num_3 = 3i64;
num__3 = 3i64;
do
{
  _size = size;
  if ( size >= capacity )
  {                                             // Reallocate
    std::basic_string_char_std::char_traits_char__std::allocator_char___::_Reallocate_grow_by__lambda_319d5e083f45f90dcdce5dce53cbb275__char_(&string);
  }
  else
  {
    ++size;
    str_addr = (char *)&string;
    if ( capacity >= 0x10 )
      str_addr = (char *)string;
    *(_WORD *)&str_addr[_size] = 97;
  }
  --num__3;
}
while ( num__3 );
```

第一块逻辑，当size>capacity时，调用Rellocate_xxx函数

否则，就直接在str_addr后追加一个97，也就是a

```
do
{
  v7 = size;
  if ( capacity - size < 0xA )
  {                                             // Reallocate
    std::basic_string_char_std::char_traits_char__std::allocator_char___::_Reallocate_grow_by__lambda_65e615be2a453ca0576c979606f46740__char_const___unsigned
      &string,
      10ui64);
  }
  else
  {
    size += 10i64;
    v8 = (char *)&string;
    if ( capacity >= 0x10 )
      v8 = (char *)string;
    v9 = &v8[v7];
    memmove_0(v9, "12345abcde", 0xAui64);
    v9[10] = 0;
  }
  --num_3;
}
while ( num_3 );
```

第二块逻辑，这次因为用的是append()，每次追加10个字符，即使是一个QWORD也无法存放，所以看到的是memmove_0函数

最后是v9[10] = 0，也是我们在vs中看到的，追加后，仍然会以\x00结尾

一开始我没想明白，+='a'为什么没有设置\x00■■

后来才发现，*(_WORD*)&str_addr[_size] = 97;

这是一个WORD，2个byte，考虑小端序，\x00已经被写入了

至于其中的Reallocate_xxx函数，有点复杂...而且感觉也没必要深入了，刚刚已经在vs里了解扩容机制了

最后还有一个delete相关的

```
if ( capacity >= 0x10 )
{
  v12 = string;
  if ( capacity + 1 >= 0x1000 )
  {
    v12 = (_BYTE *)*((_QWORD *)string - 1);
    if ( (unsigned __int64)((_BYTE *)string - v12 - 8) > 0x1F )
      _invalid_parameter_noinfo_noreturn();
  }
  operator_delete(v12);
}
```


之前在做题时经常分不清作者写的代码、库函数代码，经常靠动态调试猜，多分析之后发现清晰了不少

测试程序2

```
#include<iostream>
#include<string>

using namespace std;

int main(int argc, char** argv) {
   string input1;
   string input2;
   string result;
   std::cin >> input1;
   std::cin >> input2;
   result = input1 + input2;
   std::cout << result;

   return 0;
}

//g++-4.7 main.cpp
```

这次用g++编译，发现逻辑很简明，甚至让我怀疑这是C++吗...

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4; // [rsp+10h] [rbp-50h]
  char v5; // [rsp+20h] [rbp-40h]
  char v6; // [rsp+30h] [rbp-30h]
  char v7; // [rsp+40h] [rbp-20h]

  std::string::string((std::string *)&v4);
  std::string::string((std::string *)&v5);
  std::string::string((std::string *)&v6);
  std::operator>><char,std::char_traits<char>,std::allocator<char>>(&std::cin, &v4);
  std::operator>><char,std::char_traits<char>,std::allocator<char>>(&std::cin, &v5);
  std::operator+<char,std::char_traits<char>,std::allocator<char>>(&v7, &v4, &v5);
  std::string::operator=(&v6, &v7);
  std::string::~string((std::string *)&v7);
  std::operator<<<char,std::char_traits<char>,std::allocator<char>>(&std::cout, &v6);
  std::string::~string((std::string *)&v6);
  std::string::~string((std::string *)&v5);
  std::string::~string((std::string *)&v4);
  return 0;
}
```

调用了一次operator+，然后operator=赋值，最后输出

但是用vs编译，IDA打开就很混乱...下次再仔细分析一下

测试程序3

```
#include<iostream>
#include<string>

using namespace std;

int main(int argc, char** argv) {
   string input1;
   string input2;
   std::cin >> input1;
   std::cin >> input2;

   //■■■
   for(auto c:input2){
       input1 += c;
   }

   std::cout << input1;
   return 0;
}

//g++-4.7 main.cpp -std=c++11
```

仍然是g++编译的，IDA打开后虽然没有友好的命名，需要自己改，但是逻辑很清晰

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char *iter_char; // rax
  char input1; // [rsp+20h] [rbp-50h]
  char input2; // [rsp+30h] [rbp-40h]
  __int64 input2_begin; // [rsp+40h] [rbp-30h]
  __int64 input2_end(); // [rsp+50h] [rbp-20h]
  std::string *input2_addr; // [rsp+58h] [rbp-18h]

  std::string::string((std::string *)&input1);
  std::string::string((std::string *)&input2);
  std::operator>><char,std::char_traits<char>,std::allocator<char>>(&std::cin, &input1);
  std::operator>><char,std::char_traits<char>,std::allocator<char>>(&std::cin, &input2);
  input2_addr = (std::string *)&input2;
  input2_begin = std::string::begin((std::string *)&input2);// 迭代器begin()
  input2_end() = std::string::end(input2_addr); // 迭代器end()
  while ( (unsigned __int8)__gnu_cxx::operator!=<char *,std::string>(&input2_begin, &input2_end()) )
  {
    iter_char = (char *)__gnu_cxx::__normal_iterator<char *,std::string>::operator*(&input2_begin);
    std::string::operator+=(&input1, (unsigned int)*iter_char);
    __gnu_cxx::__normal_iterator<char *,std::string>::operator++(&input2_begin);// ++  ->  ++*(&input2_begin)
  }
  std::operator<<<char,std::char_traits<char>,std::allocator<char>>(&std::cout, &input1);
  std::string::~string((std::string *)&input2);
  std::string::~string((std::string *)&input1);
  return 0;
}
```


for(auto c:input2)这句是一个"语法糖"，迭代地取出每一个字符，追加到input1上

IDA中可以看到，迭代器begin■end，通过循环中的operator!=判断是否已经结束，再通过operator+=追加，最后通过operator++来改变迭代器input2_begin的值

这里命名应该把input2_begin改成iterator更好一些，因为它只是一开始是begin

小总结

逆向水深...动态调试确实很容易发现程序逻辑，但是有反调试的存在

多练习纯静态分析也有助于解题，看得多了也就能分辨库函数代码和作者的代码了

上一篇：XSS相关一些个人Tips 下一篇：CVE-2019-6976：Lib...

1. 3 条回复


图灵的橘猫 2019-04-24 13:31:07

卧槽还能这么整？先点赞投币收藏一波，良心up主关注了

0 回复Ta


jzwxZZZ 2019-04-27 23:01:58

感觉配上string内存分配相关的源码分析一下会更好一些

0 回复Ta

ret2nullptr 2019-04-28 08:57:45

@jzw**** 因为是在windows下，用的visual studio，那个代码风格有点难读，遂放弃

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板