
AutelTest 逆向笔记

目录

AutelTest 逆向笔记	1
总结	1
一、预备知识	2
二、反汇编过程	3
1. 导出表	3
2. 类之间的关系	4
3. 分析 ICar	4
4. 分析 CBenz	8
5. 分析 CVehicle	11

总结

总结写在前面，因为分析太啰嗦了，也有一些错误的地方。

（1）题目分析

三个类 ICar, CBenz, CVehicle。ICar 是 CBenz 的父类，CVehicle 的一个成员变量是 ICar。

未分析正确的函数：析构函数→基类的析构调用，ICar 的成员变量。

未分析的函数：含有容器的函数，涉及算法的函数：seed2key,setVin,calckey。

无名函数：CBenz_delete(sub_4010FC0)。

ICar 有三个成员，seed（4 字节），key（16 字节），key_length(4 字节)。在 CBenz 中，通过加密的算法，将 seed 转化成 key。由于首先的分析的 ICar，所以产生了失误。

题目的知识点：C++的全部语法及其反编译，简单的加密算法，PE 导出表了解，C++ 函数名的修饰符。

（2）个人总结

题目是比较中等偏下，对于我来说比较难。

这是我第一次逆向一个完整的 C++工程项目，本题涉及到容器和算法，这是我的知识盲区，也是平时我回避的点。总的来说，这次的逆向情况非常差，开始逆向这个工程，到

逆向完第一遍（12月2日下午3:00-12月3日凌晨3:00），用了12个小时。前面3天全部是在补基础知识。虽然累，收获还是很多。

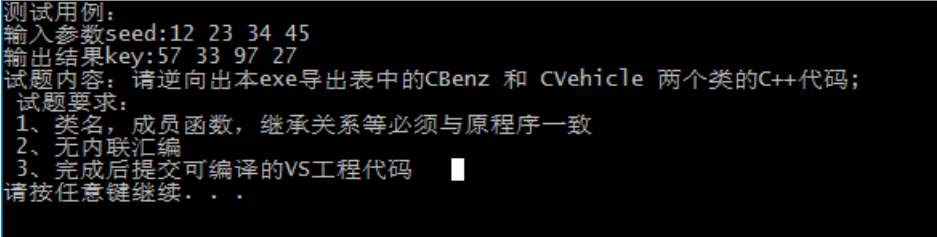
三天的学习，补充了很多知识，在逆向的过程中也在补。补充的过程中，发现自己的知识的不足，在下一次任务前去弥补，去强化。在分析的过程中，切忌焦虑，由于第一次逆向大的工程，没有头绪。同时不要怕错，一开始有点畏难便犹豫了几分钟，之后便直接大刀阔斧开始逆，一边逆向，一边修正之前分析的错误。

比较开心的一点，自己的代码能力比较强，第一次用C++编程，bug调试的时间居然不到30分钟。打字总共1个小时，bug调试20分钟，其余时间全部是分析。

下面的内容是我在分析的时候的记录，从接触到题，到第一次逆向完成。划线掉的是以前分析的，保留划线是为了便于以后的总结。有些错误有保留的价值。

一、预备知识

首先看看题目情况：



```
测试用例：
输入参数seed:12 23 34 45
输出结果key:57 33 97 27
试题内容：请逆向出本exe导出表中的CBenz 和 CVehicle 两个类的C++代码；
试题要求：
1、类名，成员函数，继承关系等必须与原程序一致
2、无内联汇编
3、完成后提交可编译的VS工程代码
请按任意键继续...
```

图 1 题目状况

导出表，C++类是重点，以及最后可以提交VS工程代码。导出表用LoadPE可以用直接查看，然后exe直接放到IDA里面去，可以进行接下来的分析了。

但是实际情况不是这样的。之后会详细的说明。

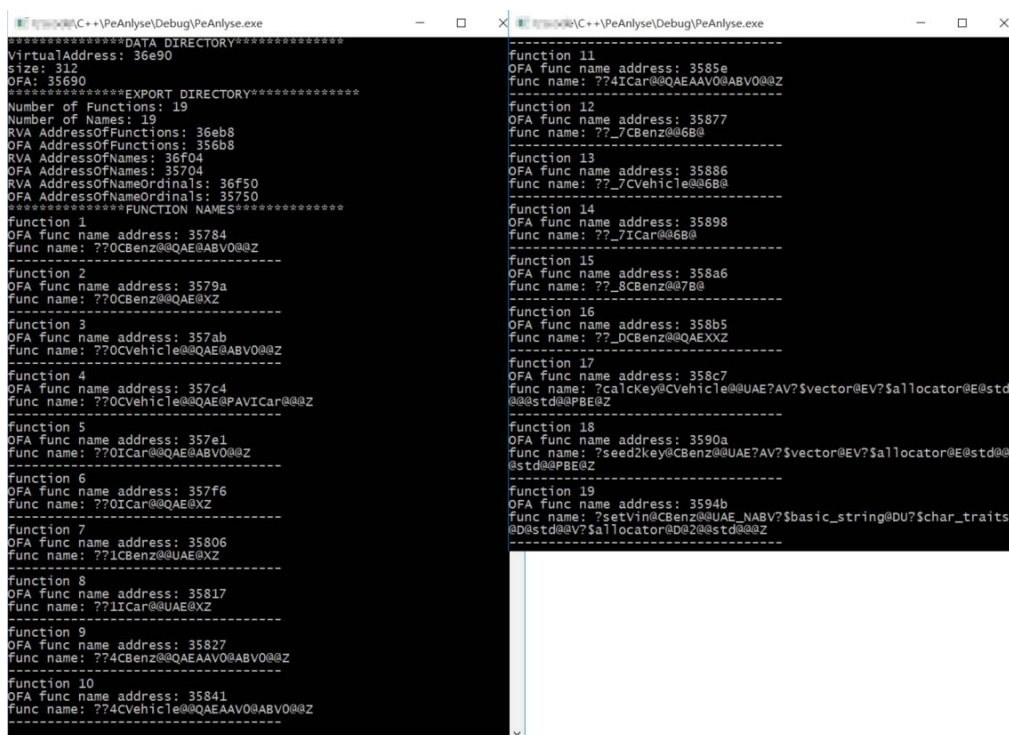
补习的知识点（星期四晚拿到题）：

PE 知识点：从可选PE头到导出表，用C语言完成解析器（1天）

C++知识点及其反汇编：结构体，类，类成员，多态，虚函数，友元函数，运算符的重载，Vector容器，C++反编译后函数修饰，C++头文件格式（2天）

二、反汇编过程

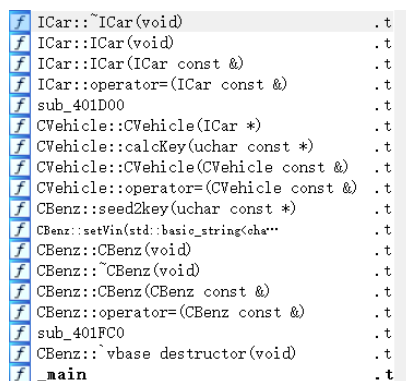
1. 导出表



```
*****DATA DIRECTORY*****
VirtualAddress: 36e90
Size: 312
OFA: 35690
*****EXPORT DIRECTORY*****
Number of Functions: 19
Number of Names: 19
RVA AddressOfFunctions: 36eb8
RVA AddressOfNames: 36f04
OFA AddressOfNames: 35704
RVA AddressOfNameOrdinals: 36f50
OFA AddressOfNameOrdinals: 35750
*****FUNCTION NAMES*****
Function 1
OFA func name address: 35784
Func name: ??70CBenz@@QAE@ABV0@@Z
-----
Function 2
OFA func name address: 3579a
Func name: ??70CBenz@@QAE@XZ
-----
Function 3
OFA func name address: 357ab
Func name: ??70CVehicle@@QAE@ABV0@@Z
-----
Function 4
OFA func name address: 357c4
Func name: ??70CVehicle@@QAE@PAVICar@@@Z
-----
Function 5
OFA func name address: 357e1
Func name: ??70ICar@@QAE@ABV0@@Z
-----
Function 6
OFA func name address: 357f6
Func name: ??70ICar@@QAE@XZ
-----
Function 7
OFA func name address: 35806
Func name: ??71CBenz@@UAE@XZ
-----
Function 8
OFA func name address: 35817
Func name: ??71ICar@@UAE@XZ
-----
Function 9
OFA func name address: 35827
Func name: ??74CBenz@@QAEAAV0@ABV0@@Z
-----
Function 10
OFA func name address: 35841
Func name: ??74CVehicle@@QAEAAV0@ABV0@@Z
-----
Function 11
OFA func name address: 3585e
Func name: ??74ICar@@QAEAAV0@ABV0@@Z
-----
Function 12
OFA func name address: 35877
Func name: ??7CBenz@@6B@
-----
Function 13
OFA func name address: 35886
Func name: ??7CVehicle@@6B@
-----
Function 14
OFA func name address: 35898
Func name: ??7ICar@@6B@
-----
Function 15
OFA func name address: 358a6
Func name: ??8CBenz@@7B@
-----
Function 16
OFA func name address: 358b5
Func name: ??DCBenz@@QAE@XZ
-----
Function 17
OFA func name address: 358c7
Func name: ?calcKey@CVehicle@@@UAE?AV?$vector@EV?$allocator@E@std@@@std@@PBE@Z
-----
Function 18
OFA func name address: 3590a
Func name: ?seed2key@CBenz@@@UAE?AV?$vector@EV?$allocator@E@std@@@std@@PBE@Z
-----
Function 19
OFA func name address: 3594b
Func name: ?setVin@CBenz@@@UAE_NABV?$basic_string@DU?$char_traits@D@std@@v?$allocator@D@Z@@@std@@@Z
```

图 2 导出表

可以看见整个导出表的全貌，但是里面有许多 C++ 反编译的修饰符。尝试使用 IDA 进行分析。



```
f ICar::~ICar(void) .t
f ICar::~ICar(void) .t
f ICar::~ICar(ICar const &) .t
f ICar::operator=(ICar const &) .t
f sub_401D00 .t
f CVehicle::CVehicle(ICar *) .t
f CVehicle::calcKey(uchar const *) .t
f CVehicle::CVehicle(CVehicle const &) .t
f CVehicle::operator=(CVehicle const &) .t
f CBenz::seed2key(uchar const *) .t
f CBenz::setVin(std::basic_string<char... .t
f CBenz::CBenz(void) .t
f CBenz::~CBenz(void) .t
f CBenz::CBenz(CBenz const &) .t
f CBenz::operator=(CBenz const &) .t
f sub_401FC0 .t
f CBenz::~vbase destructor(void) .t
f _main .t
```

图 3 IDA 分析

IDA 中的函数只有 15 个，在 IDA 中查看导出表。

```

.rdata:00436F84 a0cbenzQaeAbv0Z db '??0CBenz@@QAE@ABV0@@Z',0
.rdata:00436F84 a0cbenzQaeXz db '??0CBenz@@QAE@XZ',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00436F9A a0cvehicleQaeAb db '??0CVehicle@@QAE@ABV0@@Z',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00436FAB a0cvehicleQaePa db '??0CVehicle@@QAE@PAVICar@@@Z',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00436FC4 a0icarQaeAbv0Z db '??0ICar@@QAE@ABV0@@Z',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00436FE1 a0icarQaeXz db '??0ICar@@QAE@XZ',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00436FF6 a1cbenzUaeXz db '??1CBenz@@UAE@XZ',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00437006 a1icarUaeXz db '??1ICar@@UAE@XZ',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00437017 a4cbenzQaeav0A db '??4CBenz@@QAEAAV0@ABV0@@Z',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00437027 a4cvehicleQaeaa db '??4CVehicle@@QAEAAV0@ABV0@@Z',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00437041 a4icarQaeav0Ab db '??4ICar@@QAEAAV0@ABV0@@Z',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:0043705E a7cbenz6b db '??_7CBenz@@6B@',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00437086 a7cvehicle6b db '??_7CVehicle@@6B@',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00437098 a7icar6b db '??_7ICar@@6B@',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:004370A6 a8cbenz7b db '??_8CBenz@@7B@',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:004370B5 a0cbenzQaeXxz db '??_DCBenz@@QAE@XZ',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:004370B5 aCalkeyCvehicl db '?calKey@CVehicle@@UAE?AV?$vector@EV?$allocator@std@@@std@@PBE@'
.rdata:004370C7 db '?',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:0043710A aSeed2keyCbenzU db '?seed2key@CBenz@@UAE?AV?$vector@EV?$allocator@std@@@std@@PBE@Z',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:0043710A aSetvinCbenzUae db '?setVin@CBenz@@UAE_NABV?$basic_string@DU?$char_traits@D@std@@V?Sa'
.rdata:0043714B ; DATA XREF: .rdata:off_436F04f0

```

图 4 IDA 导出表-2

和解析的情况是一致的。

解析完成了导出表，将大致的分析，这几个类之间的关系。

2. 类之间的关系

根据题目的要求，只需要逆向 CBenz 和 CVehicle 两个类就行了。但是最后需要一个可以编译的 VS 工程，就是 C++ 文件吧。明显是个坑吧，CBenz 和 CVehicle 两个类的父类是 ICar，这个能直接看出来。所以 ICar 必须分析，否则后面两个类怎么编译通过。

明确一下分析的目标:ICar，CBenz，CVehicle 三个类。类的成员可以不用分析，这个看情况，最好能分析，不然之后的函数里面的内容怎么办。

之后明确一下分析的顺序，首先是 ICar，然后是之后的两个类。更加细节的关系，在之后进行分析。

3. 分析 ICar

(1) ICar 拥有的函数

```

.rdata:00436FC4 a0cvehicleQaePa db '??0CVehicle@@QAE@PAVICar@@@Z',0
.rdata:00436FC4 ; DATA XREF: .rdata:off_436F04f0
.rdata:00436FE1 a0icarQaeAbv0Z db '??0ICar@@QAE@ABV0@@Z',0
.rdata:00436FE1 ; DATA XREF: .rdata:off_436F04f0
.rdata:00436FF6 a0icarQaeXz db '??0ICar@@QAE@XZ',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00437006 a1cbenzUaeXz db '??1CBenz@@UAE@XZ',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00437017 a1icarUaeXz db '??1ICar@@UAE@XZ',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00437027 a4cbenzQaeav0A db '??4CBenz@@QAEAAV0@ABV0@@Z',0
.rdata:00437027 ; DATA XREF: .rdata:off_436F04f0
.rdata:00437041 a4cvehicleQaeaa db '??4CVehicle@@QAEAAV0@ABV0@@Z',0
.rdata:00437041 ; DATA XREF: .rdata:off_436F04f0
.rdata:0043705E a4icarQaeav0Ab db '??4ICar@@QAEAAV0@ABV0@@Z',0
.rdata:0043705E ; DATA XREF: .rdata:off_436F04f0
.rdata:00437077 a7cbenz6b db '??_7CBenz@@6B@',0 ; DATA XREF: .rdata:off_436F04f0
.rdata:00437086 a7cvehicle6b db '??_7CVehicle@@6B@',0
.rdata:00437086 ; DATA XREF: .rdata:off_436F04f0
.rdata:00437098 a7icar6b db '??_7ICar@@6B@',0 ; DATA XREF: .rdata:off_436F04f0

```

图 5 ICar 拥有的函数

一共有 5 个，但是 IDA 最后解析了 4 个，不用在意细节，开始分析每个函数的修饰符的含义。

(2) ICar 函数名分析

函数修饰符参考文献：

https://blog.csdn.net/qq_34992845/article/details/54564358

<https://www.jb51.net/article/36722.htm>

<https://blog.csdn.net/liweigao01/article/details/78351464>

https://en.wikiversity.org/wiki/Visual_C%2B%2B_name_mangling

这里记录几个比较重要的修饰符

- ① public: @@QAE – const 存在 @@QBE
- ② protect: @@IAE – const 存在 @@IBE
- ③ private: @@AAE – const 存在 @@ABE
- ④ 参数是类示例对象的引用: @@AAV1 – const 存在 @@ABV1
- ⑤ 有参数 @Z 无 Z
- ⑥ 返回值为空 X

其实可以丢给 IDA 直接分析的，还是自己分析一遍，稳妥一点。对于没有查到的，就只有靠 IDA 了，必尽时间是有限的，还有经验的问题。从上到下列出分析的函数名称：

```
public: ICar(ICar const &)
```

```
public: void ICar(ICar
```

这里还是看看 IDA 的分析吧 (TT__TT):

```
.text:00401CA0 ; _DWORD __thiscall ICar::ICar(ICar *this, const struct ICar *)
.text:00401CA0 public ??0ICar@@QAE@ABV0@@Z
.text:00401CA0 ??0ICar@@QAE@ABV0@@Z proc near ; DATA XREF: .rdata:off_436EB8+0
.text:00401CA0 arg_0 = dword ptr 8
.text:00401CA0
.text:00401CA0 push ebp
.text:00401CA1 mov ebp, esp
.text:00401CA3 mov eax, [ebp+arg_0]
.text:00401CA6 push esi
.text:00401CA7 xor edx, edx
.text:00401CA9 mov esi, ecx
.text:00401CAB push 0FFFFFFFFh
.text:00401CAD lea ecx, [esi+4]
.text:00401CB0 mov dword ptr [esi], offset ??_7ICar@@6B@ ; const ICar::'vftable'
.text:00401CB6 add eax, 4
.text:00401CB9 push edx
.text:00401CBA mov dword ptr [ecx+18h], 0Fh
.text:00401CC1 mov [ecx+14h], edx
.text:00401CC4 push eax
.text:00401CC5 mov [ecx+4], dl
.text:00401CC8 call sub_402700
.text:00401CCD mov eax, esi
.text:00401CCF pop esi
.text:00401CD0 pop ebp
.text:00401CD1 retn 4
.text:00401CD1 ??0ICar@@QAE@ABV0@@Z endp
```

图 6 IDA 分析 ICar 函数

这里省略了如何如找到这个函数位置的。当然是通过导出表，导出表中的函数地址。只是一个知识的问题，不做解释。

(3) 反编译函数

第一个：ICar 的构造函数：

先尝试 IDA 的 F5 一下，一般不要报希望，这里只关注逻辑，先大概的写一下，之后改就好了：

```
class ICar
{
public:
——char[4] carname; // 4byte
——int value2; // 4 bytes
——int128 value1; // 16 bytes
    char* seed;
    double key;
    int key_length;
}
ICar::ICar(const ICar* EleICar)
{
    this->key=0;
    this->key_length=15;
    this->seed = new CHAR[4];
    function_strcpy(this->seed, EleICar->seed, 0, -1);
}
```

Carname: 因为是一个 char 类型的，所以猜测是一个存储车名字的地方。（道通和车的联系非常紧密，十之八九）

Seed, key, key_length, 有之后的函数逆推出来的名字。

IDA 未编译的函数是否为类的私有函数：经过试验的验证（编写了验证的 C++ 的程序）发现，如果是 private 的成员，仍然是可以在 IDA 里面显示出来的，所以这个地方应该是一个其他的函数。并且通过题目要求可以看出，这个函数是可以不进行分析的，只需要体现其功能就可以了。

function_strcpy: 完全就是猜的，两个车的名字传到里面去，不是赋值是什么。两个类之间 seed 的 copy。

Call 调用的区别：通过资料的查询，thiscall, stdcall, fastcall 知道是在头文件里面配置，但是没有时间去研究了，有再说吧。

私有函数是否在导出表中：没有查到相应的资料，之后会接着查找，但是通过自己写函数进行测试，私有函数 IDA 是可以逆的。

第二个：ICar 无参构造函数

```
ICar::ICar()
{
    this->key=0;
    this->key_length=15;
    this->seed = new CHAR[4];
}
```

和前面那个够着函数是统一的，这里进一步确定每个变量的字节数是多少。

变量字节数的确定：

```
*(_DWORD *)this = &ICar::`vftable';
*((_DWORD *)this + 7) = 15;
*((_DWORD *)this + 6) = 0;
*((_BYTE *)this + 8) = 0;
```

图 7 变量自己数

虚表 4 个字节，所以第一个变量 carname 是 $8-4=4$ 个字节。 $6-8/4=4$ 双字，第二变量是 16 个字节。第三个变量是 1 个双字，4 个字节。

第三个：ICar 的析构函数

```
ICar::~~ICar()
{
    if(key >= 16)
        delete seed;
    this->seed = new CHAR[4];
    key=0;
    key_length=15;
}
```

这个析构函数有问题，有毛病！析构函数是清楚函数内容的，它居然还赋给值。

分析 delet 的对象：

```
01C50      push     esi
01C51      mov     esi, ecx
01C53      mov     dword ptr [esi], offset ??_7ICar@@6B@ ; const ICar
01C59      cmp     dword ptr [esi+1Ch], 10h
01C5D      jb      short loc_401C6B
01C5F      mov     eax, [esi+8]
01C62      push     eax ; void *
01C63      call    ???@YAXPAX@Z ; operator delete(void *)
```

图 8 delete 对象

Esi 里面是 this 指针，this+8 是 虚表和 carname。进一步猜测，名字应该是在堆中的。

第四个函数：操作数的重载=

```
ICar* ICar::operator=(const ICar* EleICar)
{
    function_strcpy(this->seed, EleICar->seed, 0, -1);
    return this;
}
```

第五个函数：// 这个就是一个析构函数，是编译器自动处理的一个东西，解释

由于 IDA 没有逆向出这个函数的名字，那么我们先放弃去弄其函数的名称。直接开始逆向了。从函数的整个架构来看，相当于一个析构函数的功能。还是命名一下:final_delet

```
void ICar::final_delet(char judge)
{
    if(judge & 2)
    {
        destructor_iterator(this, 32, carname, this->~ICar);
        if(judge & 1)
            delet[] carname;
    }
}
```

```

}
else
{
    if(value1>=16)
        delete carname;
    value2=15;
    value1=0;
    carname[3]=0;
    if( judge & 1)
        delete this;
}
}

```

destructor_iterator: 是一个循环销毁的迭代器，因为看不太懂，所以没有逆出来。里面有一个传入参数是方法，这个地方的代码我不会编写。同时对于这个类的析构函数，真的好奇怪，可能是我分析错了。

4. 分析 CBenz

(1)CBenz 拥有的函数

```

dd rva ??0CBenz@@QAE@ABV0@@Z, rva ??0CBenz@@QAE@XZ, rva ??0CVehicle@@QAE@ABV0@@Z
; DATA XREF: .rdata:00436EAC↑
dd rva ??0CVehicle@@QAE@PAVICar@@@Z, rva ??0ICar@@QAE@ABV0@@Z ; ICar::~ICar(void) ...
dd rva ??0ICar@@QAE@XZ, rva ??1CBenz@@QAE@XZ, rva ??1ICar@@QAE@XZ
dd rva ??4CBenz@@QAEAAV0@ABV0@@Z, rva ??4CVehicle@@QAEAAV0@ABV0@@Z
dd rva ??4ICar@@QAEAAV0@ABV0@@Z, rva ??_7CBenz@@@6B@, rva ??_7CVehicle@@@6B@
dd rva ??_7ICar@@@6B@, rva ??_8CBenz@@@7B@, rva ??_DCBenz@@QAE@XZ
dd rva ?calcKey@CVehicle@@QAE?AV?$vector@EV?$allocator@E@std@@@std@@PBE@Z
dd rva ?seed2key@CBenz@@QAE?AV?$vector@EV?$allocator@E@std@@@std@@PBE@Z
dd rva ?setVin@CBenz@@QAE_NABV?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@@std@@@Z

```

图 9 CBenz 拥有的函数

CBenz 拥有 9 个函数

(2) 反编译函数

第一个函数：CBenz 的构造函数

```

CBenz::CBenz()
{
    seed = new char[8];
    key = 0;
    key_length = 15;
}

```

CBenz 是 ICar 的子类，通过虚函数可以明显的分析到，CBenz 是 ICar 的子类。

ICar 的成员变量是 public 的，在 CBenz 的反汇编中，返现数据的结构是非常规整的，和 ICar 是一致的，只是多了 4 个字节，所以推测在 CBenz 中，只是对成员变量的一个复写，没有声明多的成员变量。

第二个函数：CBenz 含参构造函数

```

CBenz::CBenz(const CBenz* EleCar, int judge)

```



```

{
    char* temp_name;
    if (judge)
    {
        if(EleCar)
            temp_name=EleCar->seed;
        else
            temp_name=NULL;
        this->seed = new char[4];
        key = 0;
        key_length = 15;
        function_strcpy(this->seed, temp_name, 0, -1);
    }
}

```

第三个函数: CBenz 析构函数

这个略，就一个 retm

第四个函数: CBenz 运算符的重载 =

分析的步骤 IDA 框图看大致的流程，然后 table 切换成反编译的模式，看看代码的逻辑，然后自己进行编写。

函数的功能：将汽车的 carname 属性进行赋值

```

CBenz* CBenz::operator=(const CBenz & EleCar)
{
    function_strcpy(this->seed, EleCar.seed, 0, -1);
    return this;
}

```

在 IDA 中，其实是有两个分支的，我个人认为是没有什么影响（因为看不懂要表达的含义），所以按照这个函数大体的含义给逆向了一个代码。

第五个函数: CBenz_delete

未知的一个函数，同 ICar 中的 final_delete 类似，也是利用一个迭代器进行一个销毁的工作。这里直接按照大致的含义逆。

这个函数应该是一个用序号导出的函数，将函数名称给省略了。查了很多参考资料，才知道这个地方 compiler 做了很多事情。函数的功能就是释放一个数组的空间。

```

char* CBenz::CBenz_delete(char judge)
{
    char* result;
    if(judge & 2)
    {
        result = this->seed;
        delete[] this->seed;
    }
    else
    {
        CBenz* temp = new CBenz;
        delete this;
    }
}

```

```

}
return result;
}

```

这个函数，我放了。因为我无法理解这个函数的目的，里面的释放赋值有点混乱。

第六个函数: CBenz 的析构函数

参考资料:

<https://eli.thegreenplace.net/2015/c-deleting-destructors-and-virtual-operator-delete/>

<https://bbs.pediy.com/thread-172613.htm>

<http://www.it610.com/article/1807978.htm>

CBenz@@QAEXXZ: public 无参 无返回值

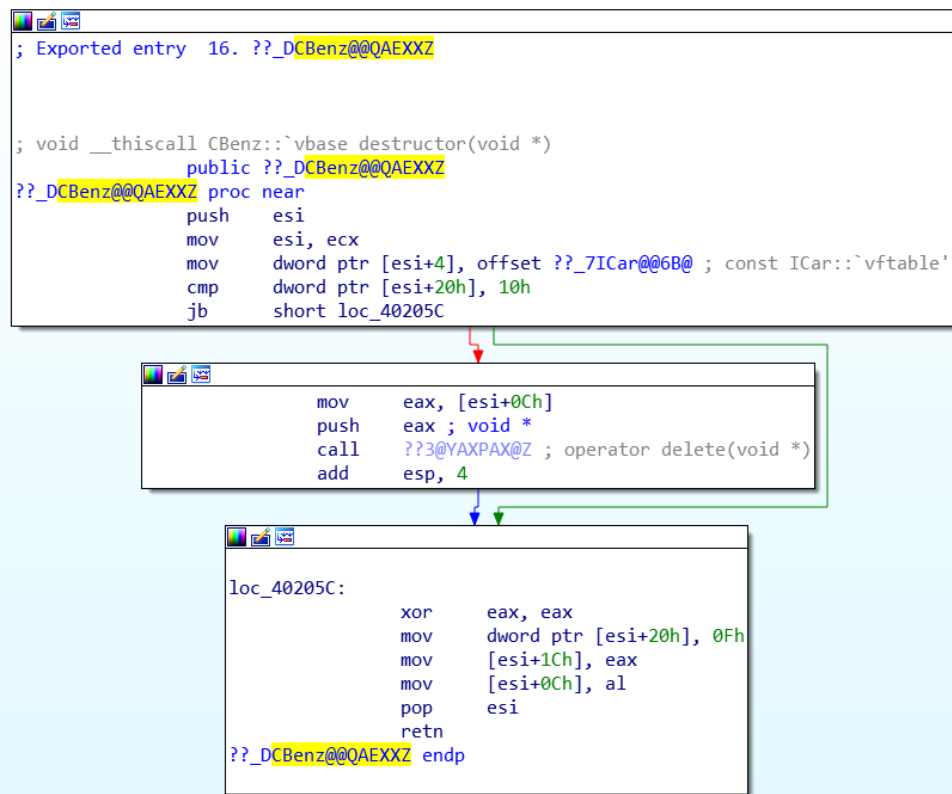


图 10 调用父类的析构函数

通过资料，知道这就是析构函数，是编译器的结果，这里是调用父类的迭代器。

这里查了一些外文的文献，知道是什么原理，但是代码无法完全逆出来。

第七个函数:

seed2key@CBenz@@UAE?AV?\$vector@EV?\$allocator@E@std@@@std@@@PBE@Z

public virtual <Vector<T>, allocator<E>>

这个函数是两个容器的函数，只学了 vector，还没有学习 allocator。知道这里是一个

泛型，但是没有办法将泛型驾驭好，这里直接逆成传入的参数 `const unsigned char *` 跳了，看接下来的接下来简单的函数。

第八个函数：

```
?setVin@CBenz@@@UAE_NABV?$basic_string@DU?$char_traits@D@std@@@V?$allocator@D@2@@@std@@@Z
```

这个函数是一个字符串的赋值。用最简单的方式实现一下。

```
virtual bool CBenz::setKin(char* Elestring)
{
    function_strcpy(this->seed, Elestring, 0, -1);
    return 1;
}
```

CBenz 的代码基本分析完成。

5. 分析 CVehicle

(1) CVehicle 拥有的函数

```
dd rva ??0CBenz@@QAE@ABV0@@Z, rva ??0CBenz@@QAE@XZ, rva ??0CVehicle@@QAE@ABV0@@Z
; DATA XREF: .rdata:00436EAC↑o
dd rva ??0CVehicle@@QAE@PAVICar@@@Z, rva ??0ICar@@QAE@ABV0@@Z ; ICar::~~ICar(void) ...
dd rva ??0ICar@@QAE@XZ, rva ??1CBenz@@UAE@XZ, rva ??1ICar@@UAE@XZ
dd rva ??4CBenz@@QAEAAV0@ABV0@@Z, rva ??4CVehicle@@QAEAAV0@ABV0@@Z
dd rva ??4ICar@@QAEAAV0@ABV0@@Z, rva ??_7CBenz@@6B@, rva ??_7CVehicle@@6B@
dd rva ??_7ICar@@6B@, rva ??_8CBenz@@7B@, rva ??_DCBenz@@QAE@XZ
dd rva ?calcKey@CVehicle@@UAE?AV?$vector@EV?$allocator@E@std@@@std@@PBE@Z
dd rva ?seed2key@CBenz@@UAE?AV?$vector@EV?$allocator@E@std@@@std@@PBE@Z
dd rva ?setVin@CBenz@@UAE_NABV?$basic_string@DU?$char_traits@D@std@@@V?$allocator@D@2@@@std@@@Z
```

图 11 CVehicle 拥有的函数

一共有 5 个函数，估计需要分析的函数比 5 个少。

(2) 反编译函数

第一个函数： CVehicle 含参构造函数

`CVehicle@@@QAE@ABV0@@@Z:`

Public const 有参

```
push    ebp
mov     ebp, esp
mov     eax, ecx
mov     ecx, [ebp+arg_0]
mov     dword ptr [eax], offset ??_7CVehicle@@6B@ ; const CVehicle::~`vftable'
mov     edx, [ecx+4]
mov     [eax+4], edx
pop     ebp
retn    4
```

图 12 汇编代码

该类有一个成员变量，可以直接赋值的类型。推测 CVehicle 的结构如下：

```
class CVehicle
{
```

```

public:
    int name; ICar* icar;
public:
    CVehicle();
    virtual ~CVehicle();
};

```

Name:自己随意取得名字，和车子相关的就想到了这个。

接下反编译构造函数的代码。

```

CVehicle::CVehicle(const CVehicle* EleCV)
{
    this->icar = EleCV->icar;
}

```

第二个函数：CVehicle 的含参构造函数

```

; _DWORD __thiscall CVehicle::CVehicle(CVehicle *this, struct ICar *)
        public ??0CVehicle@@QAE@PAVICar@@@Z
??0CVehicle@@QAE@PAVICar@@@Z proc near ; DATA XREF: .rdata:off_436EB84o

arg_0             = dword ptr 8

        push    ebp
        mov     ebp, esp
        mov     eax, ecx
        mov     ecx, [ebp+arg_0]
        mov     dword ptr [eax], offset ??_7CVehicle@@6B@ ; const CVehicle::~`vftable'
        mov     [eax+4], ecx
        pop     ebp
        retn    4

```

图 13 含参构造函数 2

CVehicle 的成员是一个 ICar 类的指针。此函数的反汇编如下

```

CVehicle::CVehicle(ICar* icar)
{
    this->icar = icar;
}

```

第三个函数：CVehicle 运算符的重载=

CVehicle@@QAEAAV0@ABV0@@@Z:

Public const & 有参

函数功能：将第一个成员变量进行赋值

```

CVehicle* CVehicle::operator=(const CVehicle & Cve)
{
    this->icar=Cve.icar;
    return this;
}

```

第四个函数：放置车钥匙

说明：之前命名的 carname 应该是错的，在函数的名称里提示了应该是车的 key，也就是密码，16 个字节用来存储加密后的密码，Value2 为 15，表示密文的长度。车子没有

了，钥匙可能还在，所以钥匙不会毁掉。这就是析构函数的原因。

最后一个函数涉及到了容器，我就不分析了。因为不会。

calcKey