Reversing C++ Assembly Code

Author: Jeffrey Farnan

INTRODUCTION

This report will examine the assembly code of a basic C++ program after decompilition. Identifiling object-oriented programming concepts such as classes and objects at the assembly level. Reconstruct classes to make them more understandable using the x64dbg Debugger.

The C++ Code

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

using std::endl;

class SimpleCat
{
public:
SimpleCat() {itsAge = 2; }
~SimpleCat() {}
int GetAge() const { return itsAge; }
void SetAge(int age) { itsAge = age; }
private:
int itsAge;
};

int main()
{
SimpleCat * Frisky = new SimpleCat; //create a new object
std::cout << "Frisky is " << (*Frisky).GetAge()
<< " years old" << endl;
(*Frisky).SetAge(5);
std::cout << "Frisky is " << (*Frisky).GetAge()
<< " years old" << endl;
delete Frisky; //delete object

system("pause");
return 0;
}
```

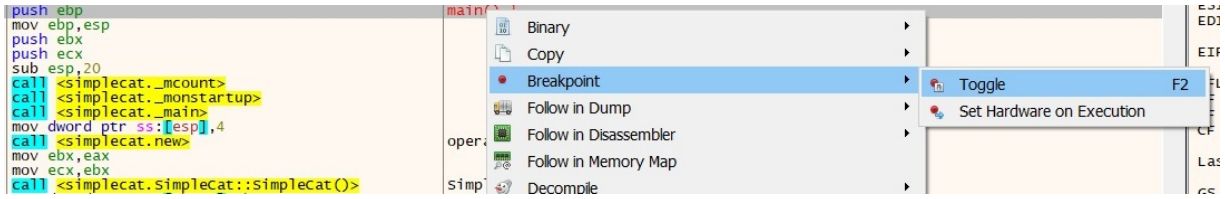
This is a simple C++ program which outputs two strings to the console. When the program is compiled the executable is loaded in the x64dbg Debugger where we locate the main method.

The main method

I go through the code labelling the functions allowing for better understanding of what is going on.

```
push dword ptr ds:[ecx-4]                    main() {
push ebp
mov ebp,esp
push ebx
push ecx
sub esp,20
call <simplecat._mcount>
call <simplecat._monstartup>
call <simplecat._main>
mov dword ptr ss:[esp],4
call <simplecat.new>                         operator new(4)
mov ebx,eax
mov ecx,ebx
call <simplecat.SimpleCat::SimpleCat()>      SimpleCat::SimpleCat
```

Setting a breakpoint on the main method and running the program, we can now proceed to go through the program step by step.



As we enter the main method we can see a new operator is used to allocate memory on the heap, in this case its four bytes. The simpleCat constructor is than called.

```
call simplecat.40D600
mov dword ptr ss:[esp],4
call <simplecat.new()>                       new(4);
mov ebx,eax
mov ecx,ebx
call <simplecat.simpleCat::simpleCat()>      constructor
mov dword ptr ss:[ebp-C],ebx
mov eax,dword ptr ss:[ebp-C]
mov ecx,eax
call <simplecat.simpleCat::getAge()>
mov ebx,eax                                  eax= (return age)
mov dword ptr ss:[esp+4],simplecat.489000    0x489000:"Frisky is "
mov dword ptr ss:[esp],simplecat.4881C0
call <simplecat.cout>
mov dword ptr ss:[esp],ebx
mov ecx,eax
call <simplecat.cerr>                        << operater(int);
sub esp,4
```

The simpleCat constructor

We can look at the constructor more closely here, when we enter it.

```
push ebp                                     simpleCat::simpleCat()
mov ebp,esp
sub esp,4
call <simplecat._mcount>
mov dword ptr ss:[ebp-4],ecx
mov eax,dword ptr ss:[ebp-4]
mov dword ptr ds:[eax],2                     return (2);
leave
ret                                          end_func
```

In the simpleCat constructor we can see the address of the object (in ecx) is placed into a local variable in this case is [ebp-4]. The value of [ebp-4] (address) placed into eax. The int value of 2 is placed into the address that eax holds. We exit that function and now enter the main method again.

```
mov ebx,eax
mov ecx,ebx
call <simplecat.SimpleCat::SimpleCat()>          SimpleCat::SimpleCat
mov dword ptr ss:[ebp-C],ebx                     ebx = object(2)
mov eax,dword ptr ss:[ebp-C]                     eax = object(2)
mov ecx,eax                                      object = getAge argument
call <simplecat.SimpleCat::getAge()>             SimpleCat_GetAge(object)
```

The object is returned in ebx, the object itself is a hexadecimal address but at that address hold's the value of 2. The object address is passed down to the next function which is simpleCat::getAge().

The simpleCat::gettAge() function

```
push ebp                                         simpleCat::getAge()
mov ebp,esp
sub esp,4
call <simplecat._mcount>
mov dword ptr ss:[ebp-4],ecx
mov eax,dword ptr ss:[ebp-4]
mov eax,dword ptr ds:[eax]
leave
ret
```

This function again places the object address into [ebp-4] on the stack. The address again is place into eax. Then the contents of [eax] which is 2 is placed into eax. We than exit this function back into the main method.

Display functions

Analysing the code I can see a lot of print statements and output to the console. I label these statements and run through the code.

```
call <simplecat.simpleCat::getAge()>             eax= (return age)
mov ebx,eax
mov dword ptr ss:[esp+4],simplecat.489000        0x489000:"Frisky is "
mov dword ptr ss:[esp],simplecat.4881C0
call <simplecat.cout>
mov dword ptr ss:[esp],ebx
mov ecx,eax
call <simplecat.cerr>                            << operater(int);
sub esp,4
mov dword ptr ss:[esp+4],simplecat.48900B        0x48900B:" years old"
mov dword ptr ss:[esp],eax
call <simplecat.cout>                            << " years old";
mov dword ptr ss:[esp],<simplecat.endl>
mov ecx,eax
call <simplecat.endl>                            cout << endl;
```

Looking at the code we have a load of print statements, ending with an end line statement. The next assembly instructions are setting up for the next function, the function is going to set the age now to five. Two arguments are being passéd to the function the object and the age, we now enter simpleCat::setAge()

```
mov eax,dword ptr ss:[ebp-C]          | int age =5;
mov dword ptr ss:[esp],5              |
mov ecx,eax                           | (obj, age)
call <simplecat.simpleCat::setAge()>  |
sub esp,4                             |
mov eax,dword ptr ss:[ebp-C]          |
mov ecx,eax                           |
```

The simpleCat::setAge() function

When we enter this function, the object address is in ecx, is moved into a variable [ebp-4]. The next instruction is to move this address into eax. The age of five is now moved from [ebp+8] into edx.

```
push ebp                              | simplecat::setAge(obj, 5)
mov ebp,esp                           |
sub esp,4                             |
call <simplecat._mount>               |
mov dword ptr ss:[ebp-4],ecx          | addres of object move into [ebp-4]
mov eax,dword ptr ss:[ebp-4]          | address move into eax
mov edx,dword ptr ss:[ebp+8]          |
mov dword ptr ds:[eax],edx            | eax = 5
leave                                 |
ret 4                                 | end_function
```

When we exit the function, eax is returned with the value of 5.

We return to the main code, where we encounter the getAge function again which will simply return the age of five into eax. We enter the print functions again and it will simply print out “Frisky is 5 years old”. We hit the endline function to end the print out statements.

```
call <simplecat.simpleCat::getAge()>  |
mov ebx,eax                           |
mov dword ptr ss:[esp+4],simplecat.489000  | 0x489000:"Frisky is "
mov dword ptr ss:[esp],simplecat.4881C0    |
call <simplecat.cout>                 | << "Frisky is";
mov dword ptr ss:[esp],ebx            |
mov ecx,eax                           |
call <simplecat.cerr>                 | << operater(int);
sub esp,4                             |
mov dword ptr ss:[esp+4],simplecat.48900B  | 0x48900B:" years old"
mov dword ptr ss:[esp],eax            |
call <simplecat.cout>                 |
mov dword ptr ss:[esp],<simplecat.endl>    |
mov ecx,eax                           |
call <simplecat.endl>                 | cout << endl;
```

The last set of assembly instructions we enter will destroy the object, free up memory space, pause the program and exit out of main.

```
call <simplecat.SimpleCat::~SimpleCat()>  | deconstructor
mov dword ptr ss:[esp],ebx                |
call <simplecat.delete()>                 |
mov dword ptr ss:[esp],simplecat.489016   | 0x489016:"pause"
call <simplecat.system>                   | system("pause");
mov eax,0                                 | return 0;
```

Looking at the output we see the following strings printed out.

```
Frisky is 2 years old
Frisky is 5 years old
Press any key to continue . . .
```