Reversing and Calling `SayHello()`

The first thing I like to do is get a general idea of what the function is doing. The name of this function makes me think that it's going to "say" hello. This could be simply returning a string, or printing out "Hello". With this in mind, let's start.

This is the disassembly of `SayHello()`:

```
00007FFA21CD1E60    48 SUB RSP, 0x28                          SayHello
00007FFA21CD1E64    E8 CALL <dll.sub_7FFA21CD4800>
00007FFA21CD1E69    48 MOV RCX, RAX                           rax:EntryPoint
00007FFA21CD1E6C    48 ADD RSP, 0x28
00007FFA21CD1E70    E9 JMP <dll.sub_7FFA21CD4DB0>
```

This function doesn't seem to return anything. It calls one function and jumps to another. The first function is `sub_7FFA21CD4800`. This function is pretty big.

If we skim through this function there are two things that jump out at me. We can see that inside the function, the string `"Hello!"` is used. This is good to know because this clearly relates to the name of the function.

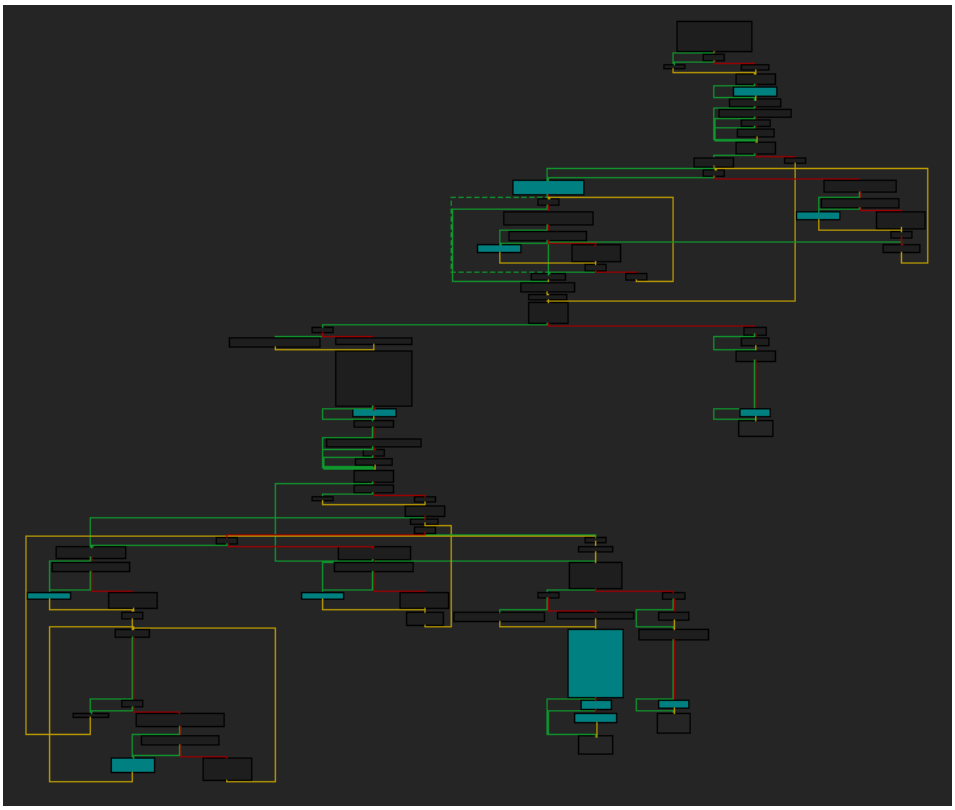
```
00007FFA21CD4920  0F JE dll.7FFA21CD49B0
00007FFA21CD4926  48 DEC RDI
00007FFA21CD4929  48 MOV RCX, QWORD PTR DS:[0x7FFA21D102D0]
00007FFA21CD4930  EB JMP dll.7FFA21CD48D2
00007FFA21CD4932  48 MOVSXD RAX, DWORD PTR DS:[RCX + 0x4]   rax:EntryPoint
00007FFA21CD4936  48 MOV RCX, QWORD PTR DS:[RAX + RSI + 0x48]  rax+rsi*1+48:sub_7FFA21CD8F14+9
00007FFA21CD493B  48 MOV RAX, QWORD PTR DS:[RCX]          rax:EntryPoint
00007FFA21CD493E  41 MOV R8D, 0x6
00007FFA21CD4944  48 LEA RDX, QWORD PTR DS:[0x7FFA21D09710]  00007FFA21D09710:"Hello!"
00007FFA21CD494B  FF CALL QWORD PTR DS:[RAX + 0x48]       rax+48:sub_7FFA21CD8F14+8
00007FFA21CD494E  48 CMP RAX, 0x6                         rax:EntryPoint
00007FFA21CD4952  75 JNE dll.7FFA21CD49B0
00007FFA21CD4954  48 TEST RDI, RDI
00007FFA21CD4957  7E JLE dll.7FFA21CD49B9
00007FFA21CD4959  48 MOV RAX, QWORD PTR DS:[0x7FFA21D102D0]  rax:EntryPoint
```

Let's continue skimming through and see if we can find something else helpful. Something that jumps out to me is towards the bottom there are references to `ios_base`.

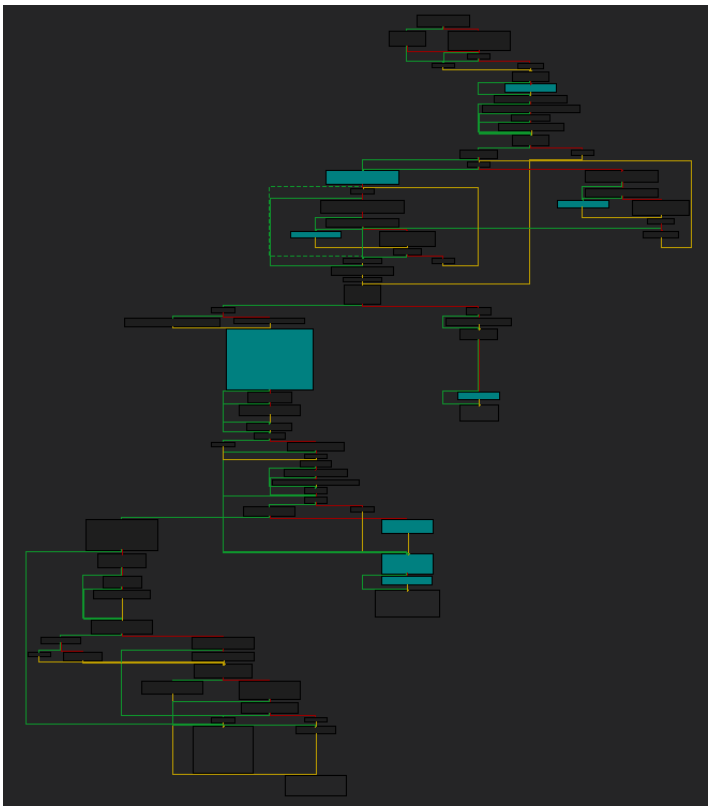
```
00007FFA21CD4A5D  A8 TEST AL, 0x4
00007FFA21CD4A5F  74 JE dll.7FFA21CD4A6A
00007FFA21CD4A61  48 LEA RBX, QWORD PTR DS:[0x7FFA21D096C8]  00007FFA21D096C8:"ios_base::badbit set"
00007FFA21CD4A68  EB JMP dll.7FFA21CD4A7E
00007FFA21CD4A6A  A8 TEST AL, 0x2
00007FFA21CD4A6C  48 LEA RBX, QWORD PTR DS:[0x7FFA21D096E0]  00007FFA21D096E0:"ios_base::failbit set"
00007FFA21CD4A73  48 LEA RAX, QWORD PTR DS:[0x7FFA21D096F8]  rax:EntryPoint, 00007FFA21D096F8:"ios_base::eofbit set"
00007FFA21CD4A7A  48 CMOVE RBX, RAX                       rax:EntryPoint
00007FFA21CD4A7E  BA MOV EDX, 0x1
00007FFA21CD4A83  48 LEA RCX, QWORD PTR SS:[RSP + 0x28]
00007FFA21CD4A88  E8 CALL <dll.sub_7FFA21CD1320>
00007FFA21CD4A8D  4C MOV R8, RAX                          rax:EntryPoint
00007FFA21CD4A90  48 MOV RDX, RBX
00007FFA21CD4A93  48 LEA RCX, QWORD PTR SS:[RSP + 0x50]
00007FFA21CD4A98  E8 CALL <dll.sub_7FFA21CD1C40>
00007FFA21CD4A9D  48 LEA RDX, QWORD PTR DS:[0x7FFA21D0D9E8]
00007FFA21CD4AA4  48 LEA RCX, QWORD PTR SS:[RSP + 0x50]
00007FFA21CD4AA9  E8 CALL <dll.sub_7FFA21CDA384>
```

Seeing the string `"Hello!"` and `ios_base` in the same function makes me suspicious. I'm starting to think this is a system function, possibly `std::cout`. The reason is that `ios_base` is the base class of the I/O stream. Let's put the theory to the test, I'll write a quick program that uses `std::cout` and try to determine if the two are similar. The quickest way to do this is to view the graph view of the two programs.

DLL Graph View:



Test Program Graph View:



They are somewhat similar, but not similar enough. We'll have to dig into the assembly. Let's just take a look at the start of the functions:

DLL:

```
00007FFA21CD4800 r$  48 MOV RAX, RSP                                          rax:EntryPoint
00007FFA21CD4803 .   41 PUSH R15
00007FFA21CD4805 .   48 SUB RSP, 0x80
00007FFA21CD480C .   48 MOV QWORD PTR DS:[RAX - 0x40], 0xFFFFFFFFFFFFFFFE    rax-40:sub_7FFA21CD8BB4+2E0
00007FFA21CD4814 .   48 MOV QWORD PTR DS:[RAX + 0x8], RBX
00007FFA21CD4818 .   48 MOV QWORD PTR DS:[RAX + 0x10], RSI
00007FFA21CD481C .   48 MOV QWORD PTR DS:[RAX + 0x18], RDI
00007FFA21CD4820 .   4C MOV QWORD PTR DS:[RAX + 0x20], R14
00007FFA21CD4824 .   48 LEA RSI, QWORD PTR DS:[0x7FFA21D102D0]
00007FFA21CD482B .   4C MOV R15, RSI
00007FFA21CD482E .   48 MOV QWORD PTR SS:[RSP + 0x28], RSI
00007FFA21CD4833 .   33 XOR EBX, EBX
00007FFA21CD4835 .   89 MOV DWORD PTR SS:[RSP + 0x20], EBX
00007FFA21CD4839 .   48 MOV RCX, QWORD PTR DS:[0x7FFA21D102D0]
00007FFA21CD4840 .   48 MOVSXD RDX, DWORD PTR DS:[RCX + 0x4]
00007FFA21CD4844 .   48 MOV RDI, QWORD PTR DS:[RDX + RSI + 0x28]
00007FFA21CD4849 .   48 TEST RDI, RDI
```

Test Program:

```
00007FF7951A2020 r$ MOV RAX, RSP                                          rax:KernelBaseDllInitialize
00007FF7951A2023 .  PUSH R15
00007FF7951A2025 .  SUB RSP, 0x80
00007FF7951A202C .  MOV QWORD PTR DS:[RAX - 0x40], 0xFFFFFFFFFFFFFFFE    rax-40:ConsoleCallServerWithBuffers+28
00007FF7951A2034 .  MOV QWORD PTR DS:[RAX + 0x8], RBX                     rax+8:KernelBaseDllInitialize+8
00007FF7951A2038 .  MOV QWORD PTR DS:[RAX + 0x10], RSI                    rax+10:KernelBaseDllInitialize+10
00007FF7951A203C .  MOV QWORD PTR DS:[RAX + 0x18], RDI                    rax+18:KernelBaseDllInitialize+18
00007FF7951A2040 .  MOV QWORD PTR DS:[RAX + 0x20], R14                    rax+20:KernelBaseDllInitialize+20
00007FF7951A2044 .  LEA RSI, QWORD PTR DS:[0x7FF7951D6240]
00007FF7951A204B .  MOV R15, RSI
00007FF7951A204E .  MOV QWORD PTR SS:[RSP + 0x28], RSI
00007FF7951A2053 .  XOR EDI, EDI
00007FF7951A2055 .  MOV DWORD PTR SS:[RSP + 0x20], EDI
00007FF7951A2059 .  MOV RCX, QWORD PTR DS:[0x7FF7951D6240]
00007FF7951A2060 .  MOVSXD RDX, DWORD PTR DS:[RCX + 0x4]
00007FF7951A2064 .  MOV RBX, QWORD PTR DS:[RDX + RSI + 0x28]
00007FF7951A2069 .  TEST RBX, RBX
```

The start of the functions look almost exactly the same. At this point, we can be pretty confident that the function `SayHello()` is calling is `std::cout`. Let's take a look at the jump instruction at the end of the `SayHello()` function and see what we can find there. Here is the code that is jumped to:

```
00007FFA21CD4DB0 r$ 40 PUSH RDI                                           sub_7FFA21CD4DB0
00007FFA21CD4DB2 .  48 SUB RSP, 0x40
00007FFA21CD4DB6 .  48 MOV QWORD PTR SS:[RSP + 0x20], 0xFFFFFFFFFFFFFFFE
00007FFA21CD4DBF .  48 MOV QWORD PTR SS:[RSP + 0x50], RBX
00007FFA21CD4DC4 .  48 MOV RBX, RCX
00007FFA21CD4DC7 .  48 MOV RAX, QWORD PTR DS:[RCX]                        rax:EntryPoint
00007FFA21CD4DCA .  48 MOVSXD RAX, DWORD PTR DS:[RAX + 0x4]               rax:EntryPoint
00007FFA21CD4DCE .  48 MOV RAX, QWORD PTR DS:[RAX + RCX + 0x40]           rax:EntryPoint
00007FFA21CD4DD3 .  48 MOV RCX, QWORD PTR DS:[RAX + 0x8]
00007FFA21CD4DD7 .  48 MOV QWORD PTR SS:[RSP + 0x30], RCX
00007FFA21CD4DDC .  48 MOV RAX, QWORD PTR DS:[RCX]                        rax:EntryPoint
00007FFA21CD4DDF .  FF CALL QWORD PTR DS:[RAX + 0x8]
00007FFA21CD4DE2 .  90 NOP
00007FFA21CD4DE3 .  48 LEA RCX, QWORD PTR SS:[RSP + 0x28]
00007FFA21CD4DE8 .  E8 CALL <dll.sub_7FFA21CD46A0>
00007FFA21CD4DED .  4C MOV R8, QWORD PTR DS:[RAX]                         rax:EntryPoint
00007FFA21CD4DF0 .  B2 MOV DL, 0xA                                        A:'\n'
00007FFA21CD4DF2 .  48 MOV RCX, RAX                                       rax:EntryPoint
00007FFA21CD4DF5 .  41 CALL QWORD PTR DS:[R8 + 0x40]
00007FFA21CD4DF9 .  0F MOVZX EDI, AL
00007FFA21CD4DFC .  48 MOV RCX, QWORD PTR SS:[RSP + 0x30]
00007FFA21CD4E01 .  48 TEST RCX, RCX
00007FFA21CD4E04 .v 74 JE dll.7FFA21CD4E1F
00007FFA21CD4E06 .  48 MOV RDX, QWORD PTR DS:[RCX]
00007FFA21CD4E09 .  FF CALL QWORD PTR DS:[RDX + 0x10]
00007FFA21CD4E0C .  48 TEST RAX, RAX                                      rax:EntryPoint
00007FFA21CD4E0F .v 74 JE dll.7FFA21CD4E1F
00007FFA21CD4E11 .  4C MOV R8, QWORD PTR DS:[RAX]                         rax:EntryPoint
00007FFA21CD4E14 .  BA MOV EDX, 0x1
00007FFA21CD4E19 .  48 MOV RCX, RAX                                       rax:EntryPoint
00007FFA21CD4E1C .  41 CALL QWORD PTR DS:[R8]
00007FFA21CD4E1F >  40 MOVZX EDX, DIL
00007FFA21CD4E23 .  48 MOV RCX, RBX
00007FFA21CD4E26 .  E8 CALL <dll.sub_7FFA21CD5920>
00007FFA21CD4E2B .  48 MOV RCX, RBX
00007FFA21CD4E2E .  E8 CALL <dll.sub_7FFA21CD4450>
00007FFA21CD4E33 .  48 MOV RAX, RBX                                       rax:EntryPoint
00007FFA21CD4E36 .  48 MOV RBX, QWORD PTR SS:[RSP + 0x50]
00007FFA21CD4E3B .  48 ADD RSP, 0x40
00007FFA21CD4E3F .  5F POP RDI
00007FFA21CD4E40 L. C3 RET
```

My first guess, based on the fact that "\n" is used and that we think `std::cout` is called, is that this function is `std::endl`. Also, it does look like a more condensed version of `std::cout`.

In this case, I'm pretty certain that I've figured it out. You could do further reversing if you want.

Implementing `SayHello()` In Our Own Program

So we've reversed the function, now let's use it. Before we use it we need to know what parameters the function takes, and it's data/return type. Looking at the function it doesn't seem to take any parameters. If it does take parameters it certainly doesn't use them. As for the return type, it doesn't seem to return anything either. So it seems like it takes no parameters and is of type void.

```
1  #include <iostream>
2  #include <Windows.h>
3
4  //void SayHello();                      //Function declaration (for reference)
5  typedef void(WINAPI* ISayHello)(void);  //Typedef the function for use.
6
7  int main()
8  {
9      HMODULE dll = LoadLibraryA("DLL.DLL"); //Load our DLL.
10
11     if (dll != NULL)
12     {
13         //Set SayHello to be the "SayHello()" function.
14         ISayHello SayHello = (ISayHello)GetProcAddress(dll, "SayHello");
15         if (SayHello != NULL) {
16             SayHello(); //Call the function.
17         }
18         else {
19             printf("Can't load the function.");
20         }
21     }
22 }
```

> There is a version of this code you can copy/paste at the bottom of the section.

Don't worry if you don't fully understand the code. If you've never done anything like this, then that code may be pretty intimidating. I'll walk you through it.

- The code is using the Windows library (Windows.h).
- typedef is used to define the function as a data type which will make other tasks easier. I called the imported function "ISayHello". I prepended an "I" to the start of the function to indicate that it was imported. This is not required but I do it to make the code easier to use. Also, when we import the function from the library we will be able to import it as "SayHello" without having naming conflicts.
- Line 9 is where I import the DLL.
- The if-statement on line 11 is checking that the DLL was loaded.
- Line 14 is importing the `SayHello()` function. The function is given the name of "SayHello" when it's imported. This is why I use the "I" naming convention.
- Line 15 checks to make sure that the function was loaded successfully.
- Line 16 calls the function.

Before you run this code, there are two important things you need to do.

1. Compile the code in "Release" mode.
2. Put the DLL in the same location as the executable that we've written to run the DLL.

```
D:\Dev\Langs\CPP\DLLTest\x64\Release>RunDLL.exe
Hello!

D:\Dev\Langs\CPP\DLLTest\x64\Release>
```

Awesome! As we can see, "Hello!" is printed out to the screen as we guessed it would be.

I don't know about you but I think it's really cool to see our code import and run a function from a DLL without us having any documentation, lib files, or header files.

Copy/Paste Code

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

//void SayHello(); //Function declaration (for reference)
typedef void(WINAPI* ISayHello)(void); //Typedef the function for use.

int main()
{
	HMODULE dll = LoadLibraryA("DLL.DLL"); //Load our DLL.

	if (dll != NULL)
	{
		//Set SayHello to be the "SayHello()" function.
		ISayHello SayHello = (ISayHello)GetProcAddress(dll, "SayHello");
		if (SayHello != NULL) {
			SayHello();	//Call the function.
		}
		else {
			printf("Can't load the function.");
		}
	}
}
```