Exports

Let's start reversing this DLL. Because we want to implement this DLL into our own program, we don't care too much about what's going on internally. What we care about is what exported functions are available to us, what do they do, and what do we need to do to call them. To call a function we need to be sure that we get the function type and parameters correct.

Finding Exports

Let's take a look at the DLL's exports. I will show you two tools we can use to do this. The first tool is DUMPBIN. This comes with Visual Studio. DUMPBIN is available in the developer command prompt. Open the dev prompt then navigate to the directory of the DLL. To show the exports for a DLL use the command `dumpbin DLL.dll /EXPORTS`.

This is what you will see:

```
Dump of file DLL.dll

File Type: DLL

  Section contains the following exports for DLL.dll

    00000000 characteristics
    FFFFFFFF time date stamp
        0.00 version
           1 ordinal base
           6 number of functions
           6 number of names

    ordinal hint RVA      name

          1    0 00001F20 ?InitializePlayer@@YAXPEAVPlayer@@@Z
          2    1 00001ED0 ?PrintArray@@YAXQEADH@Z
          3    2 00001E80 ?PrintArray@@YAXQEAHH@Z
          4    3 00002000 MysteryFunc
          5    4 00001F50 PrintPlayerStats
          6    5 00001E60 SayHello

  Summary

        3000 .data
        3000 .pdata
       17000 .rdata
        1000 .reloc
        1000 .rsrc
       27000 .text
        1000 _RDATA
```

We can also use x64dbg. First, load the DLL into x64dbg. Once loaded, go to the "Symbols' tab. Keep pressing the run button until you see DLL.dll show up in the "Modules" column.

Here are the exports in x64dbg:

Address	Type	Ordinal	Symbol	Symbol (undecorated)
00007FFF51C28EF4	Export	0	OptionalHeader.AddressOfEntryPoint	
00007FFF51C21F20	Export	1	?InitializePlayer@@YAXPEAVPlayer@@@Z	void __cdecl InitializePlayer(class Player * __ptr64)
00007FFF51C21ED0	Export	2	?PrintArray@@YAXQEADH@Z	void __cdecl PrintArray(char * __ptr64 const,int)
00007FFF51C21E80	Export	3	?PrintArray@@YAXQEAHH@Z	void __cdecl PrintArray(int * __ptr64 const,int)
00007FFF51C22000	Export	4	MysteryFunc	
00007FFF51C21F50	Export	5	PrintPlayerStats	
00007FFF51C21E60	Export	6	SayHello	

Not all of the functions shown above are exports. Be sure to pay attention to whether a function is exported or imported. You can see if a function is exported or imported in the second column.

There are many other tools to find function exports. Another popular way is with a tool called "Dependency Walker".

Function Overriding, Mangling, and Decoration

What's with the random characters in some of the function names, such as ?PrintArray@@YAXQEAD@Z? This name mangling is done for function overriding. Function overriding allows for multiple functions of the same name that take different parameters. For example, you might want to have a function for addition. You'll probably want the ability to add integers or floats. To make this happen, you can use function overriding. Here is another example:

```
float Add(float x, float y){
  return x+y;
}

int Add(int x, int y){
  return x+y;
}

int main(){
  Add(2,3); //Calls Add(int x, int y);
  Add(4.5, 1.2); //Calls Add(float x, float y);
}
```

> When a function's name is mangled, it's called a decorated function.

In the DLL we'll be reversing we can see two functions called "PrintArray" that are both decorated. This means that they are overrides and one takes different parameters than the other. They most likely do the same thing, although this isn't a guarantee.

Decorated functions are *not* a feature in C. If a function isn't decorated that means it's going to be a C function. A function can manually be defined as a C function by prefixing the function declaration with `extern "C"`. Because of this, we can assume that some of the other functions such as `SayHello` are prefixed with `extern "C"`.

A function that doesn't have any overrides does *not* need to be defined as a C function. As you can see, `InitializeClass` is decorated but there are no other functions with the same name, so it's never overwritten. One nice thing about decorated functions is that we will know their return type and parameters. We can see that `InitializeClass` takes a pointer to a class called "`Player`" and it returns void.

You may also notice "__cdecl". This is short for C Declaration which is a calling convention. They are prefixed with this because they are C functions, although they don't have to follow the cdecl calling convention.