MysteryFunc

I decided to introduce you to reversing with some DLL exports because we have the function names. Unfortunately, this isn't always the case. This function, even though it's quite small, will be a little preview of what to expect very soon.

Here is the disassembly of `MysteryFunc`:

```
00007FFF51C22000    48:890A     MOV QWORD PTR DS:[RDX], RCX                    MysteryFunc
00007FFF51C22003    48:8D41 04  LEA RAX, QWORD PTR DS:[RCX + 0x4]              rax:EntryPoint
00007FFF51C22007    48:894A 08  MOV QWORD PTR DS:[RDX + 0x8], RCX
00007FFF51C2200B    48:8942 10  MOV QWORD PTR DS:[RDX + 0x10], RAX             rax:EntryPoint
00007FFF51C2200F    48:8D41 08  LEA RAX, QWORD PTR DS:[RCX + 0x8]              rax:EntryPoint
00007FFF51C22013    48:8942 18  MOV QWORD PTR DS:[RDX + 0x18], RAX             rax:EntryPoint
00007FFF51C22017    48:8BC1     MOV RAX, RCX                                   rax:EntryPoint
00007FFF51C2201A    C3          RET
```

- `MOV QWORD PTR DS:[RDX], RCX` - This function seems to take two parameters. RDX looks like a pointer into some data structure because of how it's being used. This could be an array or a class (or something similar). This code will move RCX (first parameter) into [RDX] (address pointed to by the second parameter). RDX is a pointer of some kind.
- `LEA RAX, QWORD PTR DS:[RCX + 0x4]` - Loads the address of RCX + 0x4 into RAX. Because an offset from RCX is being used, this makes me think that RCX is some sort of array or structure.
- `MOV QWORD PTR DS:[RDX + 0x8], RCX` - Move RCX into RDX + 0x8.
- `MOV QWORD PTR DS:[RDX + 0x10], RAX` - Move RAX (address of RCX + 0x4) into RDX + 0x10. Okay, RDX is definitely some sort of structure or array.
- `LEA RAX, QWORD PTR DS:[RCX + 0x8]` - Load the address of RCX + 0x8 into RAX.
- `MOV QWORD PTR DS:[RDX + 0x18], RAX` - Move RAX into RDX + 0x18.
- `MOV RAX, RCX` - RCX is going to be our return value because it's moved into RAX. So this function is going to return a pointer or data structure.

Let's break it down. It appears that the function takes two parameters that are both pointers or data structures (struct, class, array, etc). RDX seems to be a pointer to a data structure. Maybe this function is some form of initialization/setup code for a data structure. It appears that the overall purpose for this function is to copy data from one data structure into another data structure. One thing that throws me off a bit is that the data structures are not aligned. What I mean by this is that the first element in one data structure is being moved into, for example, the second element of the other. So this isn't some data structure copy/duplication function. Let's reanalyze the code with our new found knowledge.

- `MOV QWORD PTR DS:[RDX], RCX` - The address of the RCX data structure is put into the first element of the RDX data structure.
- `LEA RAX, QWORD PTR DS:[RCX + 0x4]` - The address of the second parameter in the RCX data structure is moved into RAX. It seems like RAX is being used as a middle-man to move addresses into the structures.
- `MOV QWORD PTR DS:[RDX + 0x8], RCX` - RCX is moved into the second element in the RDX data structure. I know this is the second element not the third because RDX contains an address which was inside of RCX. This is x64, so the address is most likely 64 bits (8 bytes). Also, because ECX wasn't used, the full RCX register was likely being used. So the first element is 8 bytes, not 4 like we're used to.
- `MOV QWORD PTR DS:[RDX + 0x10], RAX` - Move RAX (the address of the second element in the RCX data structure) into the third element in the RDX data structure.

- `LEA RAX, QWORD PTR DS:[RCX + 0x8]` - Move the address of the third element in the RCX data structure into RAX.
- `MOV QWORD PTR DS:[RDX + 0x18], RAX` - Move RAX (the address of the third element in the RCX data structure) into the fourth element of the RDX data structure.
- `MOV RAX, RCX` - Return the first parameter, which is the base address of a data structure. This is also the data structure that is having its addresses moved into another data structure.

It appears that we are moving the addresses of one data structure into another data structure.

Hey, now it's starting to make sense! We still aren't sure what the data structures are. They could be classes, arrays, or any other data structure type. Let's take a look at some pseudo-code of what we can guess is going on.

If they are both classes:

```
class MyClass{
public:
        int x, y, z;
};
class AddrClass {
public:
        void* addrOfOldClass;
        int *x, *y, *z;
};

void* CopyClass(MyClass* oldClass, AddrClass* newClass) {
        newClass->addrOfOldClass = oldClass; //addr of oldClass
        newClass->x = (int*)&oldClass->x; //addr of oldClass->x
        newClass->y = (int*)&oldClass->y;
        newClass->z = (int*)&oldClass->z;
        return oldClass;
}

int main() {
        MyClass myClass1;
        myClass1.x = 10;
        myClass1.y = 20;
        myClass1.z = 30;

        AddrClass classAddrs;
        CopyClass(&myClass1, &classAddrs);
}
```

If they are both arrays:

```
void* CopyArray(int oldArray[], void* newArray[]) {
        newArray[0] = &oldArray; //addr of oldArray
        newArray[1] = &oldArray[0]; //addr of oldArray[0]
        newArray[2] = &oldArray[1];
        newArray[3] = &oldArray[2];
```

```
        return &oldArray;
}

int main() {
        int myArray[3] = { 1,2,3 };
        void* addrArray[4]; //array of pointers
        CopyArray(myArray, addrArray);
}
```

Hopefully now you understand what it is the `MysteryFunc` is doing. Here is the actual code of the function:

```
extern "C" __declspec(dllexport) void* MysteryFunc(Player* player, int* arr[]) {
        arr[0] = (int*)player;
        arr[1] = (int*)& player->score;
        arr[2] = (int*)& player->health;
        arr[3] = (int*)& player->name;
        return player;
}
```

> Remember, I wrote this code. In reality, you wouldn't have the source code.

As you can see, we nailed it. We couldn't have done any better using static analysis. There isn't any way we could have known if the parameters were classes or arrays with just static analysis. On a low-level, both structures and arrays are accessed the same way. We knew that the parameters were data structures, but it was impossible for us to know what kind. The only way we could have been more precise is by debugging a program that uses the DLL and analyzing how it uses `MysteryFunc`.

I really enjoyed this section. This sort of problem/puzzle solving is why I enjoy reversing so much. This was a simple example, I assure you that we will look at more complex examples soon.