8.01 Initialize Table

When reversing, it's best to try to focus on the bigger picture. It's safe to guess, based on the function names, that a "Generic Table" is probably some kind of data structure. It's also important to notice that this is a "generic" data structure.

With the information we know just from the name, it's probably a good idea to start by looking at the initialization function(s). This will hint at what a table contains, how many elements it has, and what initialized values there are.

Disassembly of `RtlInitializeGenericTable`:

```
LEA RAX, QWORD PTR DS:[RCX + 0x8]
XOR R10D, R10D
MOV QWORD PTR DS:[RCX], R10
MOV QWORD PTR DS:[RAX + 0x8], RAX
MOV QWORD PTR DS:[RAX], RAX
MOV QWORD PTR DS:[RCX + 0x18], RAX
MOV RAX, QWORD PTR SS:[RSP + 0x28]
MOV QWORD PTR DS:[RCX + 0x40], RAX
MOV QWORD PTR DS:[RCX + 0x20], R10
MOV QWORD PTR DS:[RCX + 0x28], RDX
MOV QWORD PTR DS:[RCX + 0x30], R8
MOV QWORD PTR DS:[RCX + 0x38], R9
RET
```

Notice that this function doesn't subtract from RSP like normal. That's because the function doesn't have any local variables.

RCX, RDX, R8, and R9 are all being used. This means this function takes at least 4 parameters. Right away we can see that RCX is some kind of data structure because of how it's being used. Remember, data structures are referenced from a base address (RCX in this case) plus an offset (such as +0x40). The first parameter of this function is likely a generic table, which we'll just call a table for simplicity. RCX+0x28 is also used, which is the fifth parameter in this case.

> "DS" is short for Data Segment, you can ignore it.

Parameter Types

Now we know the function takes 5 parameters, but what are their types? The first parameter, RCX, is probably a generic table. RDX, R8, and R9 are all very similar. They all use `MOV QWORD PTR DS:[RCX + OFFSET], RDX/R8/R9`. This tells us that the parameters are 8 byte/64-bit (QWORD) pointers. That's a fairly broad data type, but it's all we have for now (and it's better than nothing).

Data

Let's analyze what's being moved into the data structure. Based on our previous observations, we can safely assume that the data structure in RCX is a class or struct. However, keep an open mind just in case we're wrong. For example, it could also be a fancy array that is managed by a series of functions.

- The first line passes the address of the second member in the table (RCX+0x8) into RAX.
- R10 is zeroed out.
- R10 (0) is moved into the first member (RCX). RCX is the first member in the table (also the address used to reference the table).
- RAX (the second member) is then moved into the element at RAX+0x8, which is the 3rd member because RAX already contains RCX+0x8. This is effectively RCX+0x8+0x8 or RCX+0x10.
- The address that RAX holds (the second member) is then moved into the second member of the data structure (RAX, which holds RCX+0x8).
- RAX is then moved into the element at RCX+0x18, which is the fourth element.

Before continuing further, that was a little all over the place. To help us understand, let's correlate the assembly with C code.

```
LEA RAX, QWORD PTR DS:[RCX + 0x8]
XOR R10D, R10D
MOV QWORD PTR DS:[RCX], R10         ;Struct->Member1 = 0;
MOV QWORD PTR DS:[RAX + 0x8], RAX   ;Struct->Memeber3 = &Struct->Member2;
MOV QWORD PTR DS:[RAX], RAX         ;Struct->Memeber2 = &Struct->Member2;
MOV QWORD PTR DS:[RCX + 0x18], RAX  ;Struct->Memeber4 = &Struct->Member2;
```

Take a look at members 2-4, they're all pointers. They are initialized to the address of the second member. This could mean a few things. It could just be some way of initializing the pointers to a valid address, and there isn't anything special about them. It's also possible that they all have some relation. For example, what if `Member2` points to a structure, and the other members point to something in that structure. We can't be certain yet, but I'd guess that they're related. Keep an eye on how the members are used and maybe we can find an answer.

Let's keep going. We're picking up at `MOV RAX, QWORD PTR SS:[RSP + 0x28]`.

- The fifth function parameter (RSP + 0x28) is passed into RAX.
- RAX is moved into the 9th element.
- R10 (0) is moved into the 5th element.
- RDX is moved into the 6th element.
- R8 is moved into the 7th element.
- R9 is moved into the 8th element.

Comparing assembly to C again:

```
MOV RAX, QWORD PTR SS:[RSP + 0x28]
MOV QWORD PTR DS:[RCX + 0x40], RAX  ;Struct->Member9 = Param5
MOV QWORD PTR DS:[RCX + 0x20], R10  ;Struct->Member5 = 0
MOV QWORD PTR DS:[RCX + 0x28], RDX  ;Struct->Member6 = Param2
MOV QWORD PTR DS:[RCX + 0x30], R8   ;Struct->Member7 = Param3
MOV QWORD PTR DS:[RCX + 0x38], R9   ;Struct->Member8 = Param4
RET
```

Bringing the Information Together

The table has 9 members (that we know of). We know that the 2nd, 3rd, and 4th members are all pointers to the second member. Members 1 and 5 are initialized to zero. Based on the fact that each members is 8 bytes and there are 9 members, we can conclude that the table has a size of 72 bytes. We can also determine that because the final offset is 0x40 so there is a total of 0x48 bytes of data, and therefore is 72 bytes.

> 0x48 bytes of data and not 0x40 bytes because the final element (offset of 0x40), as far as we know right now, has a size of 8 bytes. So the total size would be 0x40 + 0x8.

With this information we can understand the table a bit more. We can also construct a foundational layout of the table:

```
struct Table{
    UNKNOWN Member1;
    UNKNOWN_PTR Member2;
    UNKNOWN_PTR Member3;
    UNKNOWN_PTR Member4;
    UNKNOWN Member5;
    UNKNOWN Member6;
    UNKNOWN Member7;
    UNKNOWN Member8;
    UNKNOWN Member9;
};
```

That's it for this function. It was pretty small, simple, fun, and important.

<- Previous Lesson
Next Lesson -> - WIP

Chapter Home