Tool Types

Debugger

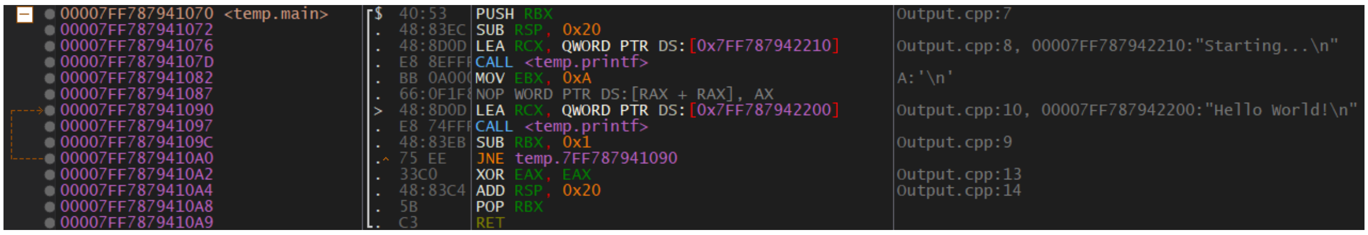
You've probably heard of a debugger before. It allows a program to step through their code and analyze what it's doing line-by-line. We will be using these ourselves. Unfortunately for us, we won't have the source code, instead we will be looking at Assembly.

A crash course on debugging can be found in 0x302-Debugging.

Both Visual Studio and x64dbg have debuggers built into them.

Disassembler

A disassembler will take a binary and present it's Assembly code. Disassemblers are the backbone of reverse engineering. Here is the disassembly of a small function:



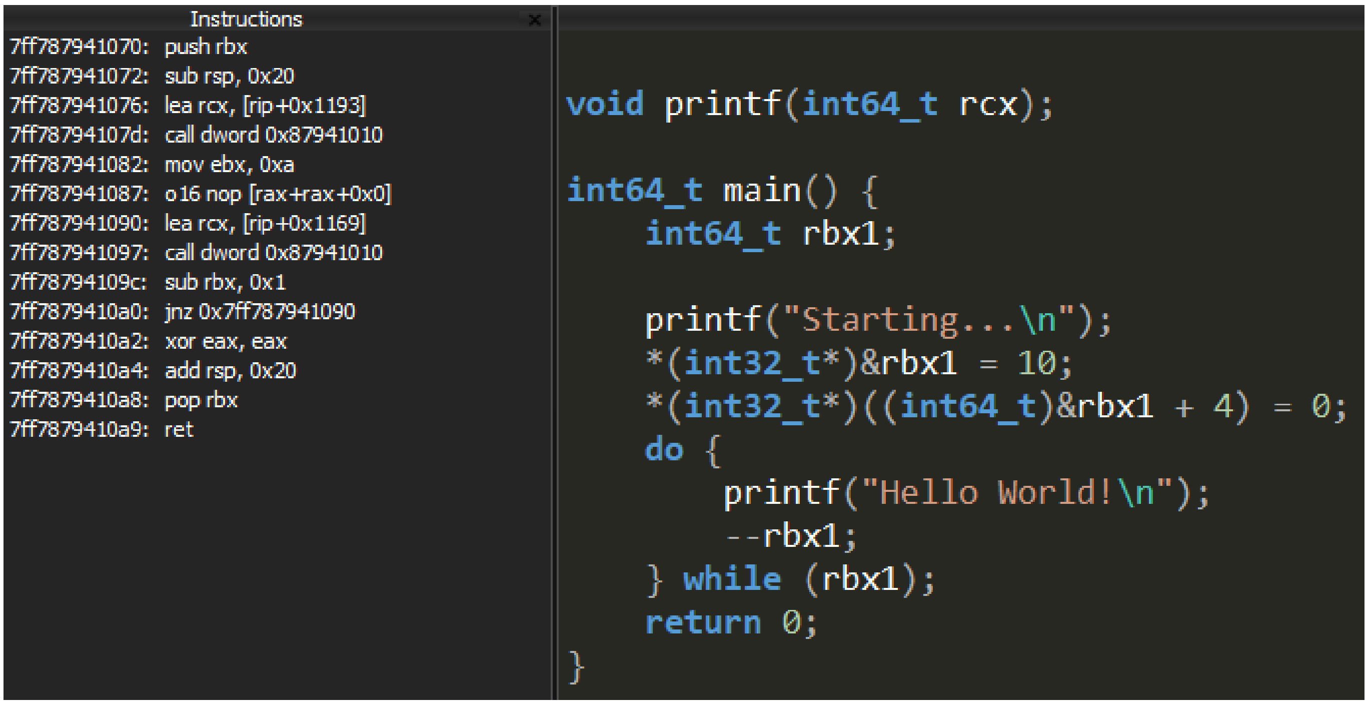
```
00007FF787941070 <temp.main>  $ 40:53     PUSH RBX                                Output.cpp:7
00007FF787941072              . 48:83EC   SUB RSP, 0x20
00007FF787941076              . 48:8D0D   LEA RCX, QWORD PTR DS:[0x7FF787942210]  Output.cpp:8, 00007FF787942210:"Starting...\n"
00007FF78794107D              . E8 8EFFF  CALL <temp.printf>
00007FF787941082              . BB 0A000  MOV EBX, 0xA                            A:'\n'
00007FF787941087              . 66:0F1F8  NOP WORD PTR DS:[RAX + RAX], AX
00007FF787941090              > 48:8D0D   LEA RCX, QWORD PTR DS:[0x7FF787942200]  Output.cpp:10, 00007FF787942200:"Hello World!\n"
00007FF787941097              . E8 74FFF  CALL <temp.printf>
00007FF78794109C              . 48:83EB   SUB RBX, 0x1                            Output.cpp:9
00007FF7879410A0              .^ 75 EE    JNE temp.7FF787941090
00007FF7879410A2              . 33C0      XOR EAX, EAX                            Output.cpp:13
00007FF7879410A4              . 48:83C4   ADD RSP, 0x20                           Output.cpp:14
00007FF7879410A8              . 5B        POP RBX
00007FF7879410A9              . C3        RET
```

Ghidra and x64dbg can disassemble a program. Visual Studio also allows you to see the disassembled version of your code. This can be extremely helpful for learning or testing. If you want to enable this you can do so by setting a breakpoint in the program then going to Debug > Windows > Disassembly.
You can view the memory by going to Debug > Windows > Memory > Memory #

Decompiler

A decompiler attempts to take a compiled program and turn it back into the original source code. These are often inaccurate but they are useful for identifying parameters, loops, conditionals, and sometimes structures. They can also be a good source to *help* identify data types (never trust decompiler data types, only use them as additional help).

Here is the decompiled version of the function shown in the disassembler portion:



```
7ff787941070:  push rbx
7ff787941072:  sub rsp, 0x20
7ff787941076:  lea rcx, [rip+0x1193]
7ff78794107d:  call dword 0x87941010
7ff787941082:  mov ebx, 0xa
7ff787941087:  o16 nop [rax+rax+0x0]
7ff787941090:  lea rcx, [rip+0x1169]
7ff787941097:  call dword 0x87941010
7ff78794109c:  sub rbx, 0x1
7ff7879410a0:  jnz 0x7ff787941090
7ff7879410a2:  xor eax, eax
7ff7879410a4:  add rsp, 0x20
7ff7879410a8:  pop rbx
7ff7879410a9:  ret
```

```c
void printf(int64_t rcx);

int64_t main() {
    int64_t rbx1;

    printf("Starting...\n");
    *(int32_t*)&rbx1 = 10;
    *(int32_t*)((int64_t)&rbx1 + 4) = 0;
    do {
        printf("Hello World!\n");
        --rbx1;
    } while (rbx1);
    return 0;
}
```

Ghidra and x64dbg both have decompiler. Ghidra has one built-in and x64dbg has a plugin called "Snowman" that does it.

Viewers

There are different ways to represent a program or parts of a program. Hex viewers and editors will show you the program represented in hexadecimal. There are even programs that will try to view the program as ASCII.

Utilities

There are various other tools that we can use to gather more information about a binary. For example, DUMPBIN can be used to look at imports, exports, headers, symbols, and more.

The best way to figure out all of these tools is to use them.

Static vs Dynamic Analysis

- Static Analysis - Analyzing at a binary when it's not running. In other words, viewing the binary as it is on disk. I also consider reversing a DLL in x64dbg using DLLLoader to be static analysis. This is because we can't step through the exported functions (they are never called).
- Dynamic Analysis - Analyzing the binary as it's running in memory.