

x64DBG

Program : WAP to perform Addition of two number a and b.

```
#include <stdio.h>

int main()
{
    int x = 5;
    int y = 11;

    int sum = 0;
    sum = x + y;

    printf("Addition of %d and %d is %d", x, y, sum);
    return 0;
}
```

Fig: C program to find sum of a and b, then print it

Debugging a simple C program for addition of two variables in x64dbg requires its basic setup, understanding of the interface navigation. Then, loading the executable, setting up breakpoints, stepping through the code, and analyzing issues. These steps are necessary for debugging any C program compiled into an exe.

```
>gcc -o add.exe add.c
```

Fig: Command to compile the C program into an exe

After compilation, open x64dbg and load the exe into it. Inside x64dbg there are Debugger Window, Disassembler Window, Memory Map, Registers, Stack, Breakpoints, etc, are some of the common area of interest for a tester.

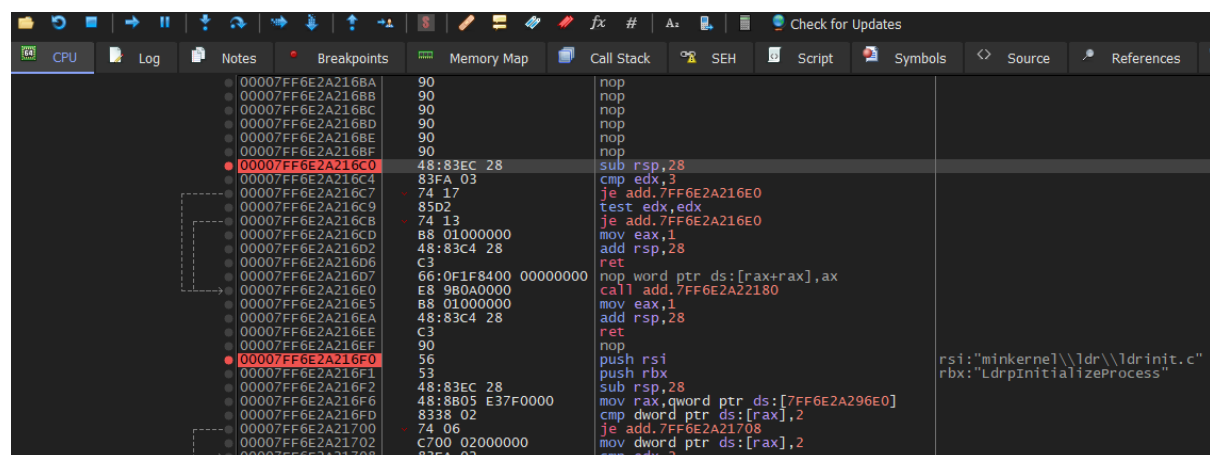


Fig: Disassembler Windows in view with Breakpoints set

Start the Program (using F9) and Stepping Through the Code, Step Over (F8) or Step Into (F7) to move to the next one or observe the behavior inside functions line by line simultaneously. In the CPU window, the assembly code is being executed, and the Registers tab will update in real-time to show the contents of the CPU registers. Next, Inspect the Call Stack (function calls that led to the current point in execution), Registers (current values stored in the CPU registers) and Variables (buffers) that loads like in this case a and b, then sum variables and print the sum.

Address	Disassembly	Comment
00007FF6E2A216D2	48:83C4 28	add rsp,28
00007FF6E2A216D6	C3	ret
00007FF6E2A216D7	66:0F1F8400 00000000	nop word ptr ds:[rax+rax],ax
00007FF6E2A216E0	E8 9B0A0000	call add.7FF6E2A22180
00007FF6E2A216E5	B8 01000000	mov eax,1
00007FF6E2A216EA	48:83C4 28	add rsp,28
00007FF6E2A216EE	C3	ret
00007FF6E2A216EF	90	nop
00007FF6E2A216F0	56	push rsi
00007FF6E2A216F1	53	push rbx
00007FF6E2A216F2	48:83EC 28	sub rsp,28
00007FF6E2A216F6	48:8B05 E37F0000	mov rax,qword ptr ds:[7FF6E2A296E0]
00007FF6E2A216FD	8338 02	cmp dword ptr ds:[rax],2
00007FF6E2A21700	74 06	je add.7FF6E2A21708
00007FF6E2A21702	C700 02000000	mov dword ptr ds:[rax],2
00007FF6E2A21708	83FA 02	cmp edx,2
00007FF6E2A2170B	74 13	je add.7FF6E2A21720
00007FF6E2A2170D	83FA 01	cmp edx,1
00007FF6E2A21710	74 4E	je add.7FF6E2A21760
00007FF6E2A21712	B8 01000000	mov eax,1
00007FF6E2A21717	48:83C4 28	add rsp,28
00007FF6E2A2171B	5B	pop rbx

Fig: Active Breakpoint

If the program is crashing or misbehaving, start the program and hit the breakpoint where the bug is occurring, Identify which function or part of the code is causing an issue, Check if Registers is empty or pointing to invalid memory. And the Stack window helps examine the contents of the stack which can reveal corrupt data or invalid function calls.

So, common issues that can occurs frequently especially in C/C++ programs are Memory Corruption, Buffer Overflows, Logic Errors. Whether it's correcting pointer issues, fixing logic flaws, or addressing a segmentation fault, the debugger's insights will guide helpful code changes.