

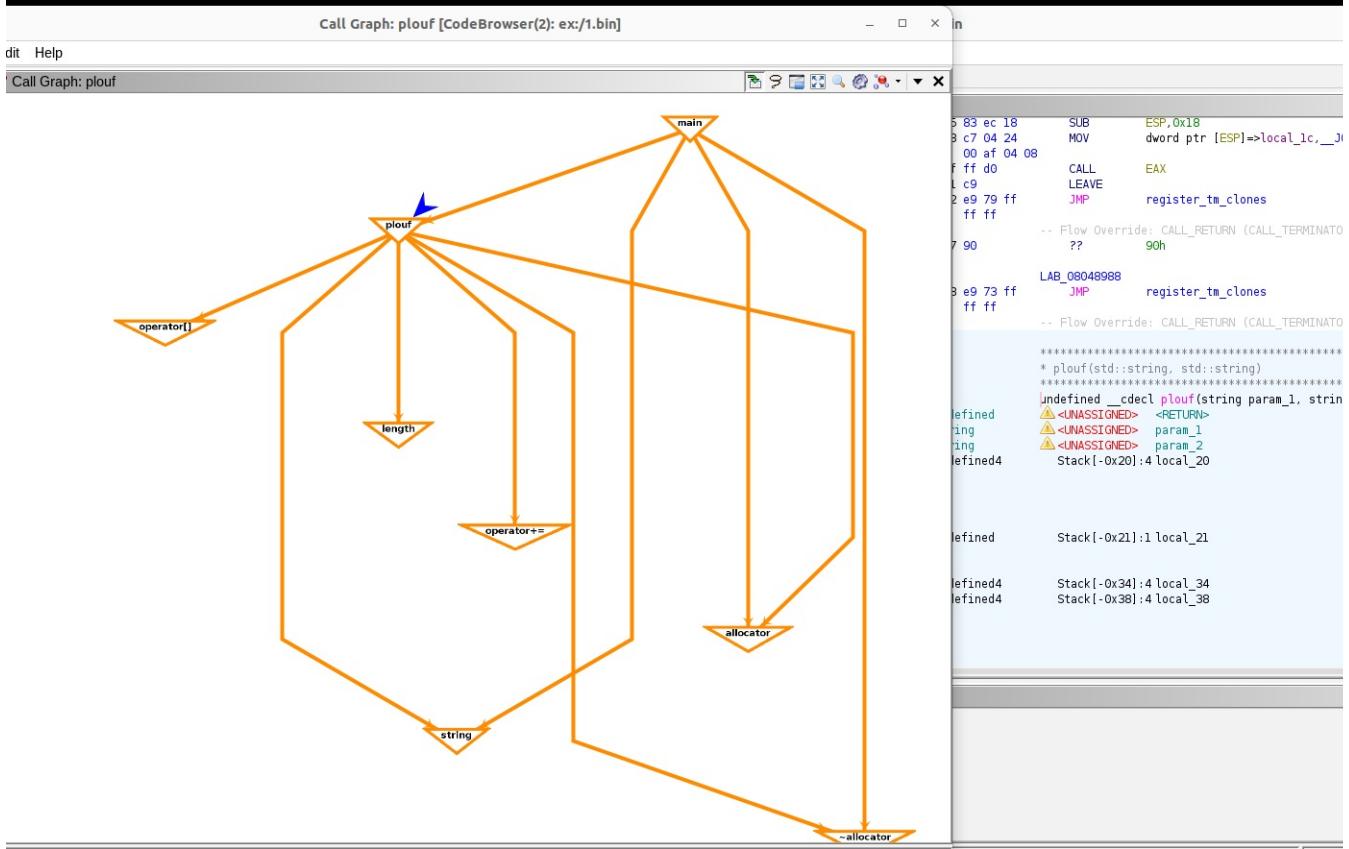
1.bin password

(gdb) break *0x08048b51

gdb) x/4wx \$eax 0xfffffcfd4: 0x08050ccc 0x08050bcd 0x08050bbc 0xfffffd000 (gdb) 0xfffffcfe4: 0xf7d3f000 0xf7ffd020 0xf7b36519 0xf7ef4a70 (gdb) set \$p = (unsigned int)\$eax (gdb) x/s \$p 0x8050ccc: "Here you have to understand a little C++_stuffs" (gdb)

I chose 0x08048b51 because it's the instruction immediately after the call to plouf in the main() function. Here's why that matters:

Why that address is key When you call a function in x86 (32-bit), the return value is placed in the EAX register. But you can only inspect EAX after the function has returned. So we need to break at the instruction right after the call plouf.



CodeBrowser(2): ex:/1.bin

File Edit Analysis BSim Graph Search Select Tools Window Help

Program Trees

- .dynstr
- .dynsym
- .gnu.hash
- .note.gnu.build-id
- .note.ABI-tag
- .interp
- segment_2.1
- .strtab
- .symtab
- .shstrtab

Program Tree x

Symbol Tree

- > f main
- > f plouf
 - param_1
 - param_2
 - > local_20
 - > local_21
 - > local_34
 - > local_38
 - > local_3c

Iter:

Data Type Manager

- Data Types
- BuiltinTypes
- 1.bin
- generic_clib
- generic_clib_64

Listing: 1.bin

```

undefined Stack[-0x21]:1 local_21
undefined4 Stack[-0x34]:4 local_34
undefined4 Stack[-0x38]:4 local_38

undefined4 Stack[-0x3c]:4 local_3c

_ZsploufSSs
plouf

0804898d 55 PUSH EBP
0804899e 89 e5 MOV EBP,ESP
08048990 57 PUSH EDI
08048991 56 PUSH ESI
08048992 53 PUSH EBX
08048993 83 ec 2c SUB ESP,0x2c
08048994 8d 45 e3 LEA EAX=>local_21,[EBP + -0x1d]
08048999 89 04 24 MOV dword ptr [ESP]=>local_3c,EAX
0804899c e8 bf fe CALL <EXTERNAL>::std::allocator<char>
ff ff
080489a1 8d 45 e3 LEA EAX=>local_21,[EBP + -0x1d]
080489a4 89 44 24 08 MOV dword ptr [ESP + local_34],EAX
080489a8 c7 44 24 MOV dword ptr [ESP + local_38],DATA

```

Console - Scripting

Post-call address for plouf from the provided image

From the XREF line in your image: _ZsploufSSs XREF[6]: Entry Point(*), main:08048b4c(c), 08048e94, 08048f30(*), 08049059(*), 0804905e(*) - Key detail: main:08048b4c(c) indicates a call to plouf at address 0x08048b4c inside main.

On 32-bit x86, a near call (call rel32) is 5 bytes long, so the instruction immediately after the call (the post-call "return address" you break on to read EAX) is: 0x08048b4c + 5 = 0x08048b51

That's why breaking at 0x08048b51 lets you inspect the return value from plouf in EAX.