Code for Concinnity

Home
About
Contact
Hire Me
life
twitter
github
RSS Feed

Tutorial: Using WinDBG to call arbitrary functions — WinDBG kung-fu series

Most people developing for MS would praise how MSVS debugger is the best debugger out there (I doubt if many of them have actually used any other debuggers when they say that). That may be true for managed code debugging (.NET), but I just find that MSVS feels severely crippled when it comes to hard-core low level stuffs in C/C++.

This example shows you how to use the better tool for the job — WinDBG. While still severely crippled compared to GDB+Python scripting. Anyway…

The example program

```
#include <iostream>
#include <string>

class foo
{
public:
    foo(const char * name)
    {
        this->name = std::string(name);
    }

    void speak()
    {
        std::cout << "Hello, my name is " << name << std::endl;
    }

private:
    std::string name;
};

int main()
{
    foo * fred = new foo("fred");
    fred->speak();
    delete fred;
    return 0;
}
```

Our objectives

Have the output say "Hello, my name is chris" by creating a new `foo` object on the fly.

Now the action

Starting up…

```
> cl /EHsc /Zi /Fefoo.exe /Fdfoo.pdb foo.cpp
> windbg foo.exe
0:000> bm main; g
```

F10 a few times to stop at the line `fred->speak()`.

The first command we'll introduce is `.dvalloc`. We can allocate memory on the heap in the process' address space.

Since `foo::foo` needs a string, we'll first create a string.

```
0:000> * just put some arbitrary size
0:000> .dvalloc 100
Allocated 1000 bytes starting at 00150000
```

You'll notice that we got allocated 1000 bytes instead of 100 as we asked. That's OK. They just like to give us whole pages.

Now we'll put some string in that memory address

```
0:000> eza 0x150000 "chris"
```

Now we'll call `foo::foo`. All member methods have an implicit first parameter that should be `this`. Compiler usually inject that for us, but here we need to specify it.

Now we'll allocate some space for our `foo`. This time around we'll use `malloc`

```
0:000> .call malloc(100); g
Thread is set up for call, 'g' will execute.
WARNING: This can have serious side-effects,
including deadlocks and corruption of the debuggee.
.call returns:
void * 0x00c21310

foo!main+0x6b:
0126145b 8b4df0         mov     ecx,dword ptr [ebp-10h] ss:002b:0032fe2c=00c21258
```

The thing about `malloc` is that we can get the return value nicely stored in a pseudo-register `$callret`. I usually prefer that to `.dvalloc`. Having the return value in `$callret` aids a lot in scripting WinDBG. We'll not touch on that to not make things too difficult.

Now onto creating the `foo` object:

```
0:000> * recall these addresses we allocated before
0:000>.call foo::foo(0x21310, 0x150000); g
Thread is set up for call, 'g' will execute.
WARNING: This can have serious side-effects,
including deadlocks and corruption of the debuggee.
foo!main+0x6b:
0126145b 8b4df0         mov     ecx,dword ptr [ebp-10h] ss:002b:0032fe2c=00c21258
```

Great, the final touch is to swap out `fred` with our newly created object.

```
0:000>ep @@(&fred) 0x21310
0:000>g
```

Cool! You should get "Hello, my name is chris"

If you came here searching for "WinDBG call function" or something like that, you've probably came across this post on The Old New Thing. That one is more advanced than what we have here Probably this will serve as a gentler introduction.

Published by kizzx2, on October 26th, 2010 at 11:29 pm. Filled under: Useful tips Tags: windbg debug windows advanced • No Comments

No comments yet.

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Type the text

Privacy & Terms

Submit Comment