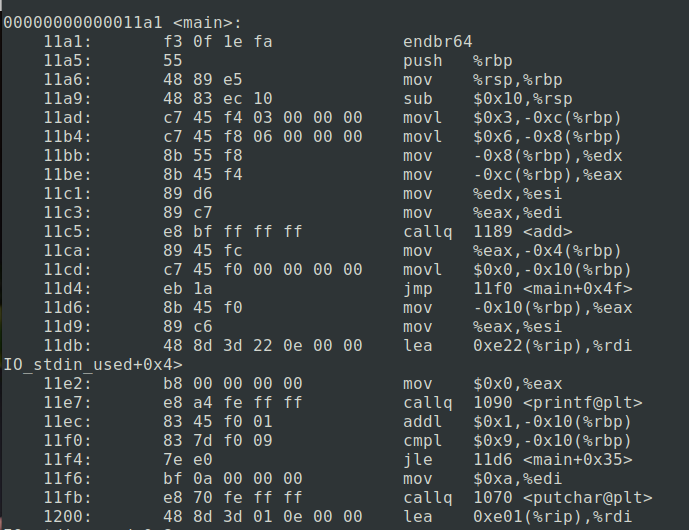
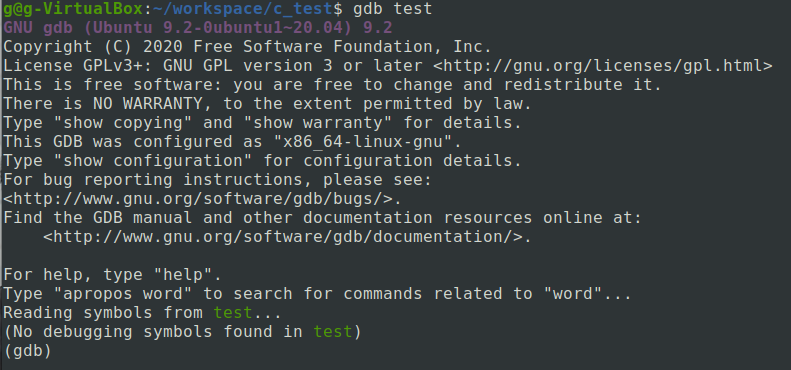
1. 使用objdump反汇编程序生成汇编代码，test是此例中用的测试程序。

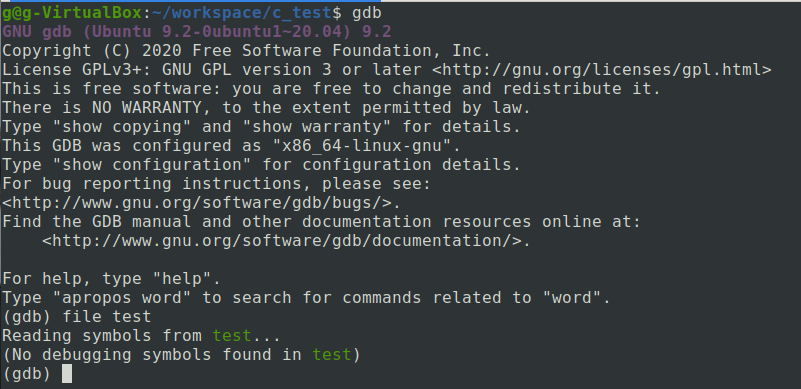
objdump -d test > test.asm



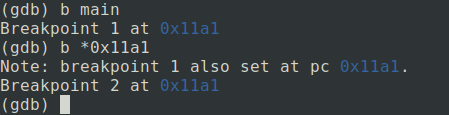
反汇编后的代码不会跟gdb有交互，主要是方便查看代码，调试的时候我们需要函数名称、地址、寄存器等等，这些都可以通过反汇编的代码得到。

1. 使用gdb <程序名>，或者运行gdb后使用file <程序名>加载程序。

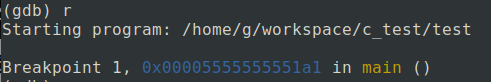




1. 使用“b/break 函数名/\*地址”创建断点。

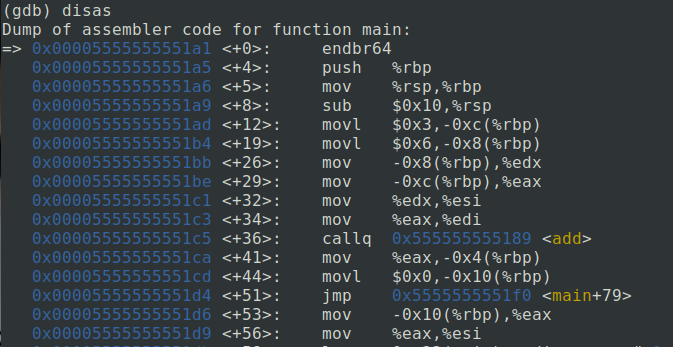


1. r/run开始运行程序，程序会在断点处停住。



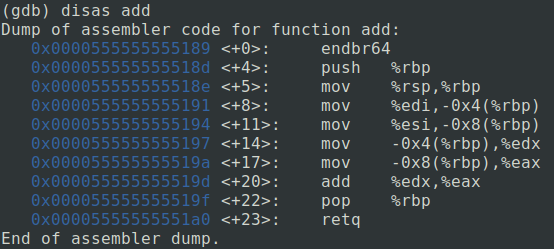
这里发现执行后的地址和创建断点显示的地址不同，跟反汇编的地址也不同，这是因为gcc默认编译生成pie（位置无关的可执行文件）程序，程序中的地址都是基于0的偏移值，实际执行的时候会随机分配一个基地址，所以程序运行的地址和反汇编的不一样。如果想要生成固定地址的代码，可以加入-no-pie参数，这样生成的代码地址在反汇编和运行时就是一样的。

1. 使用disas可以查看当前的汇编代码，并指示程序执行到哪里了。



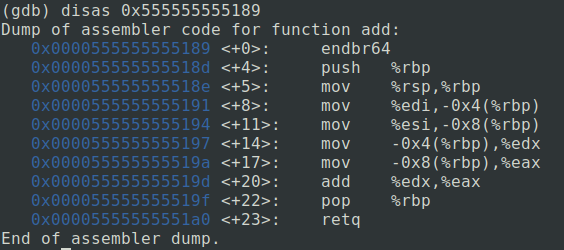
disas加函数名显示对应函数的汇编代码

disas <函数名>

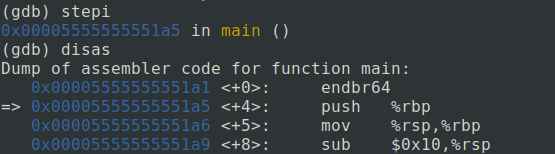


也可以指定地址查看汇编代码

disas <地址>

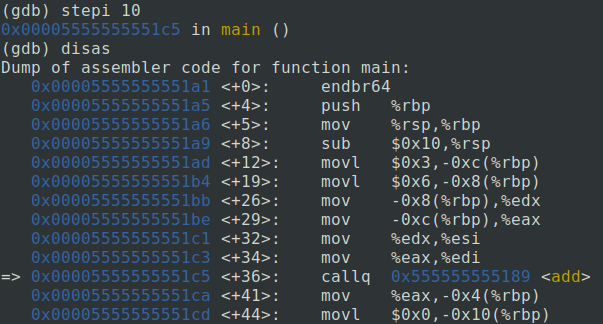


1. 使用stepi/nexti执行指令，stepi遇到函数调用会进入函数内部，nexti遇到函数会直接执行，不会进入函数内部。



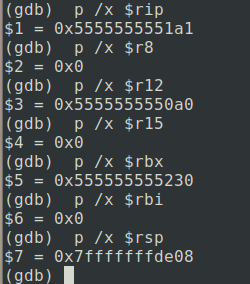
也可以在指令后加上数字执行多条指令

stepi/nexti <数字>

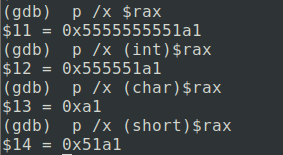


1. 查看寄存器的值

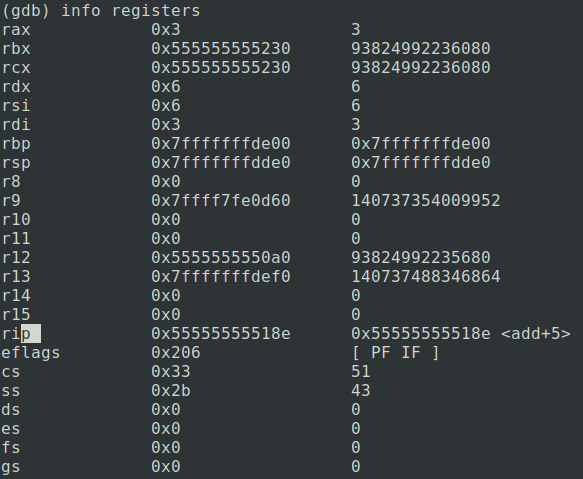
p 格式 $寄存器 ,格式可以是/x、/d、/t，分别表示16进制、十进制、二进制。



可以在寄存器前加类型显示指定位数的值

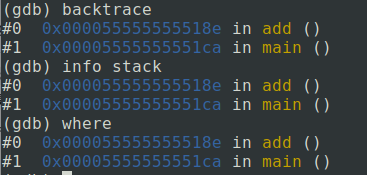


使用info r/registers查看所有寄存器的值

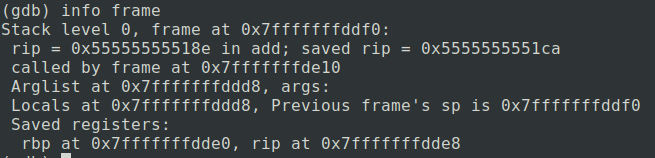


1. 查看栈调用情况

info stack、backtrace、where

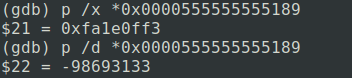


查看当前栈的详细信息



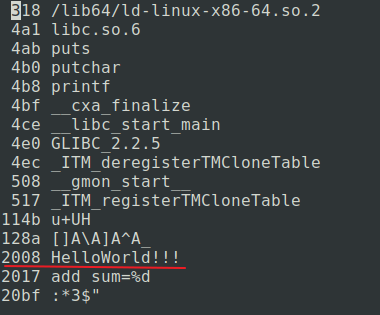
1. 打印某个地址处的值

p 格式 \*地址



如果某个地址存的是字符串，可以直接读取它

使用strings -d -t x test >string.txt可以提取可执行程序中的字符串，以及其对应的偏移地址。



偏移地址加上基址就是字符串的存储地址



如果地址值保存在寄存器中，可以通过寄存器访问地址处的值

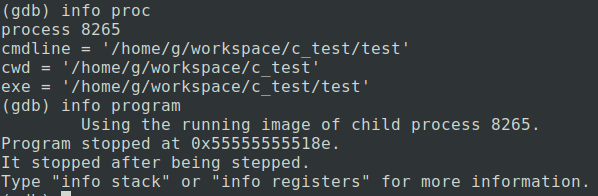
p 格式 \*(int \*)($寄存器)



1. 查看当前程序信息和执行状态

info proc

info program



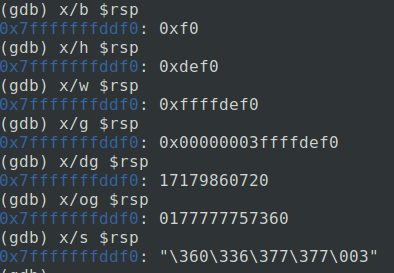
1. 使用x查看地址和寄存器的值

x/[个数][大小][格式] 地址/$寄存器

个数：显示几个对象

大小：表示对象的大小，(b=byte, h=half-word, w=word, g=giant (quad-word))

格式：如何显示，d=decimal, x=hex, o=octal等



想要熟练使用这些命令需要一定的练习，并且需要对汇编有一定的了解，虽然有点麻烦，不过掌握后可以更好的调试程序，快速定位问题，进而解决问题。