

gdb? !

pwndbg是gdb的一个插件，先来安装它吧

首先对于调试，

设置断点

```
Bash |  
1 break + 【函数名】  
2 b +
```

启动程序

```
Bash |  
1 run
```

步过（汇编语句）

```
Bash |  
1 ni
```

步过（与ni的区别是碰到call某个函数会进入）

```
Bash |  
1 si
```

快速完成一个函数

```
Bash |  
1 finish
```

第一关

尝试从对应的地方找到目标的值吧

```

RAX 0x55555555575f (main) ← endbr64
RBX 0xfffffffffd0e8 → 0xfffffffffd3d4 ← '/mnt/hgfs/UbShare/ctf/gonghubeitctf/study_gdb/pwn'
RCX 0x5555555557d40 (_do_global_dtors_aux_flni_array_entry) → 0x5555555552c0 (_do_global_dtors_aux) ← endbr64
RDX 0xfffffffffd0f8 → 0xfffffffffd406 ← COLORTERM=truecolor
RDI 1
RSI 0x7fffffff0e8 → 0x7fffffff3d4 ← '/mnt/hgfs/UbShare/ctf/gonghubeitctf/study_gdb/pwn'
R8 0
R9 0x7ffff7fc380 (_dl_fini) ← endbr64
R10 0x4fbfbff
R11 0x20
R12 1
R13 0
R14 0x5555555557d40 (_do_global_dtors_aux_flni_array_entry) → 0x5555555552c0 (_do_global_dtors_aux) ← endbr64
R15 0x7ffff7ffd0e00 (_rtld_global) → 0x7ffff7ffe2e0 → 0x5555555554000 ← 0x10102464c457f
RBP 0x7ffff7fcfc8 → 0x7ffff7fd0e00 → 0
RSP 0x7ffff7fcfc8 → 0x7ffff7fd0e00 → 0x7ffff7fd0c0 ← 0
RIP 0x555555555767 (main+8) ← mov edi, 0

0x555555555767 <main+8>    mov edi, 0      EDI => 0
0x55555555576c <main+13>   call ttime@plt           <ttime@plt>
0x555555555771 <main+18>   mov edi, eax      EAX => 0
0x555555555773 <main+20>   call srand@plt        <srand@plt>
0x555555555778 <main+25>   mov eax, 0       EAX => 0
0x55555555577d <main+30>   call compare_with_random <compare_with_random>
0x555555555782 <main+35>   test eax, eax      EAX => 0
0x555555555784 <main+37>   jne main+64          <main+64>
0x555555555786 <main+39>   lea rax, [rip + 0xb13] RAX => 0x5555555562a0 ← 'You failed at Round 1.'
0x55555555578d <main+46>   mov rdi, rax
0x555555555790 <main+49>   call puts@plt        <puts@plt>

0:0000 | rbp rsp 0xfffffffffcfc8 → 0xfffffffffd0e00 → 0xfffffffffd0c0 ← 0
1:0008 | +008 0x7ffff7fcfc8 → 0x7ffff7621ca (_libc_start_call_main+122) ← mov edi, eax
2:0010 | +010 0x7ffff7fcfc0d → 0x7ffff7ffd010 → 0x5555555557d40 (_do_global_dtors_aux_flni_array_entry) → 0x5555555552c0 (_do_global_dtors_aux) ← endbr64
3:0018 | +018 0x7ffff7fcfc08 → 0x7ffff7fd0e8 → 0x7fffffff3d4 ← '/mnt/hgfs/UbShare/ctf/gonghubeitctf/study_gdb/pwn'
4:0020 | +020 0x7ffff7fcfc0 → 0x100000001
5:0028 | +028 0x7ffff7fcfc08 → 0x55555555575f (main) ← endbr64
6:0030 | +030 0x7ffff7fcfcf8 → 0x7fffffff0e8 → 0x7fffffff3d4 ← '/mnt/hgfs/UbShare/ctf/gonghubeitctf/study_gdb/pwn'
7:0038 | +038 0x7ffff7fcfcf8 ← 0x97adf11b3f3622ae

0 0x555555555767 main+8
1 0x7ffff7621ca __libc_start_call_main+122
2 0x7ffff762a28b __libc_start_main+139
3 0x555555555245 _start+37

```

不同寄存器的值

程序对应的汇编语句及其本次运行的地址

部分栈地址和栈上内容

函数调用和返回的序列

第二关

如何查看指定地址的值?

```

1 ↵ x/[n][f][u] addr

```

- **n** : 可选, 表示要显示的单元数量, 默认为1。
- **f** : 可选, 表示显示的格式, 常见格式包括:
 - d** : 十进制整数
 - x** : 十六进制
 - o** : 八进制
 - f** : 浮点数
- **u** : 可选, 表示数据单位, 常见单位包括:
 - b** : 字节
 - h** : 半字 (2字节)
 - w** : 字 (4字节)
 - g** : 巨字 (8字节)

Bash |

```
1 x 按十六进制格式显示变量。  
2 d 按十进制格式显示变量。  
3 u 按十六进制格式显示无符号整型。  
4 o 按八进制格式显示变量。  
5 t 按二进制格式显示变量。  
6 a 按十六进制格式显示变量。  
7 c 按字符格式显示变量。  
8 f 按浮点数格式显示变量。  
9 s 按字符串显示。  
10 b 按字符显示。  
11 i 显示汇编指令。
```

第三关

如何在调试的时候改变指定地址内存储的值？

Bash |

```
1 set? ? ? ?
```

第四关

如何查看堆相关的指令，请试试他们分别是做什么的吧：

Bash |

```
1 heap -v
```

Bash |

```
1 arena
```

Bash |

```
1 bins
```

Bash |

```
1 vis
```

