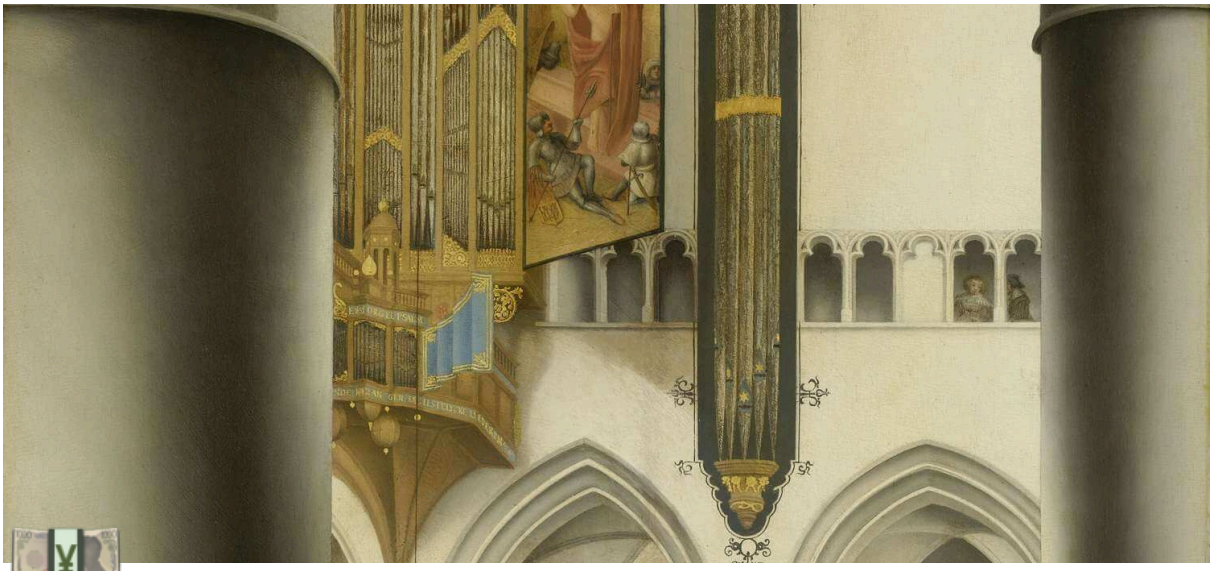

GDB调试

file <文件名>	加载被调试的可执行程序文件
run	重新开始运行文件
start	单步执行，运行程序，停在第一执行语句
list	查看原代码，简写l
set	设置变量的值
next	单步调试（逐过程，函数直接执行）,简写n
step	单步调试（逐语句：跳入自定义函数内部执行）,简写s
backtrace	查看函数的调用的栈帧和层级关系,简写bt
frame	切换函数的栈帧,简写f
info	查看函数内部局部变量的数值,简写i
finish	结束当前函数，返回到函数调用点
continue	继续运行,简写c
print	打印值及地址,简写p
quit	退出gdb,简写q

跟踪execve

```
catch exec
```

```
set follow-exec-mode new
```

程序运行参数

set args 指定运行时的参数（set args “xxx”）

show args 查看设置好的运行参数

设置断点

break 设置断点， 简写为b（b *0x401000）

info break 查看设置好的断点， 简写为i b

delete 删除断点，简写为d（d 1 删除第一个断点）

调试程序

run 运行程序 简写为 r

next 单步跟踪 一行一行的执行 简写为 n

step 步入 进入被调用函数 简写为s

finish 退出函数 简写为fin

until 在循环内单步跟踪时， 可跳出循环 简写为u

continue 继续运行程序到下一个断点 简写为c

查看运行时数据

print 打印有符号的变量 、 字符串、 表达式等值， 可简写为p

stack 查看栈数据后跟数字输出指定行数

x 以格式化的形式打印内存数据， 格式为x/FMT address , 格式字符串有 o （八进制） ， x （十六进制），d （十进制） ， u （无符号十进制） ， t （二进制） ， f （浮点数） ， a （地址） ， c （字符） ， s（字符串） ， z （十六进制对齐）。同时在FMT后面还可以加上每个单元

pwntool

```
s = process(file)    //本地启动程序
s = remote("host" , port)     // 向远程发起连接
s.sendlie()      // 输出一行 发送一行有换行符
s.sendlineafter()     //接受到某个数据发送一行
s.recvline()     //接受一行
s.recvuntil()    //直到接受到某行为止
s.send()         //直接发送没有换行符
s.sendafter()    //接受到某个字符串开始发送
s.recv()         //直接接受
```

基本指令

- help //帮助
- i //info，查看一些信息，只输入info可以看可以接什么参数，下面几个比较常用
  - i b //**常用**，info break 查看所有断点信息（编号、断点位置）
  - i r //**常用**，info registers 查看各个寄存器当前的值
  - i f //info function 查看所有函数名，**需保留符号**
- show //和info类似，但是查看调试器的基本信息，如：
  - show args //查看参数
- r d i / / * * 常 用 * * ， rdi //**常用**，rdi//**常用**，+寄存器名代表一个寄存器内的值，**用在地址上直接相当与一个十六进制变量**
- backtrace //查看调用栈
- q //quit 退出，**常用**

执行指令

- s //单步步入，遇到调用跟进函数中，相当于step into，**源码层面的一步**
  - si //**常用**，同s，**汇编层面的一步**
- n //单步补过，遇到电泳不跟进，相当于step over，**源码层面的一步**
  - ni //**常用**，同n，**汇编层面的一步**

- c //continue，**常用**，继续执行到断点，没断点就一直执行下去
- r //run，**常用**，重新开始执行

断点指令

下普通断点指令b(break)：

- b *(0x123456) //**常用**，给0x123456地址处的指令下断点
    - b *$ rebase(0x123456) //$rebase 在**调试开PIE的程序的时候**可以直接加上程序的随机地址
- b fun_name //**常用**，给函数fun_name下断点，**目标文件要保留符号才行**
    - b file_name:fun_name
- b file_name:15 //给file_name的15行下断点，**要有源码才行**
    - b 15
- b +0x10 //在程序当前停住的位置下0x10的位置下断点，同样可以-0x10，就是前0x10
- break fun if $rdi==5 //条件断点，rdi值为5的时候才断

删除、禁用断点：

- 使用info break(简写: i b)来查看断点编号
- delete 5 //**常用**，删除5号断点，直接delete不接数字删除所有
- disable 5 //**常用**，禁用5号断点
    - enable 5 //启用5号断点
- clear //清除下面的所有断点

内存断点指令watch：

- watch 0x123456 //0x123456地址的数据改变的时候会断
- watch a //变量a改变的时候会断
- info watchpoints //查看watch断点信息

捕获断点catch：

- catch syscall //syscall系统调用的时候断住
- tcatch syscall //syscall系统调用的时候断住，只断一次

- info break //catch的断点可以通过i b查看

除syscall外还可以使用的有：
1）throw: 抛出异常
2）catch: 捕获异常
3）exec: exec被调用
4）fork: fork被调用
5）vfork: vfork被调用
6）load: 加载动态库
7）load libname: 加载名为libname的动态库
8）unload: 卸载动态库
9）unload libname: 卸载名为libname的动态库
10）syscall [args]: 调用系统调用，args可以指定系统调用号，或者系统名称
1234567891011

打印指令

查看内存指令x：

- x /nuf 0x123456 //**常用**，x指令的格式是：x空格/nfu，nfu代表三个参数
  - n代表**显示几个单元（而不是显示几个字节，后面的u表示一个单元多少个字节)**，放在'/'后面
  - u代表一个单元几个字节，b(一个字节)，h(俩字节)，w(四字节)，g(八字节)
  - f代表显示数据的格式，**f和u的顺序可以互换，也可以只有一个或者不带n，用的时候很灵活**

x 按十六进制格式显示变量。
d 按十进制格式显示变量。
u 按十六进制格式显示无符号整型。
o 按八进制格式显示变量。
t 按二进制格式显示变量。
a 按十六进制格式显示变量。
c 按字符格式显示变量。
f 按浮点数格式显示变量。
s 按字符串显示。
b 按字符显示。

```
i 显示汇编指令。
1234567891011
```

- x /10gx 0x123456 //**常用**，从0x123456开始每个单元八个字节，十六进制显示是个单元的数据
- x /10xd $rdi //从**rdi指向的地址向后**打印10个单元，每个单元4字节的十进制数
- x /10i 0x123456 //**常用**，从0x123456处向后显示十条汇编指令

打印指令p(print)：

- p fun_name //打印fun_name的地址，**需要保留符号**
- p 0x10-0x08 //计算0x10-0x08的结果
- p &a //查看变量a的地址
- p *(0x123456) //查看0x123456地址的值，**注意和x指令的区别，x指令查看地址的值不用星号**
- p $rdi //显示rdi寄存器的值，**注意和x的区别，这只是显示rdi的值，而不是rdi指向的值**
    - p *($rdi) //显示rdi指向的值

打印汇编指令disass(disassemble)：

- disass 0x123456 //显示0x123456前后的汇编指令
- x /10i //**我一般喜欢用x显示指令**

打印源代码指令list：

- list //查看当前附近10行代码，**要有源码，list指令pwn题中几乎不用**，但为了完整性还是简单举几个例子
    - list 38 //查看38行附近10行代码
    - list 1,10 //查看1-10行
    - list main //查看main函数开始10行

修改和查找指令

修改数据指令set：

- set $rdi=0x10 //把rdi寄存器的值变为0x10

- set *(0x123456)=0x10 //0x123456地址的值变为0x10，**注意带星号**
- set args “abc” “def” “gh”//给参数123赋值
    - set args “python -c ‘print “1234\x7f\xde”’” //使用python给参数赋值不可见字符

查找数据：

- search rdi //从当前位置向后查包含rdi的指令，返回若干
    - search -h //查看search帮助，我也不太长用这个指令
- find “hello” //查找hello字符串，pwndbg独有
- ropgadget //查找ropgadget，pwndbg独有，**没啥用，可以用其他工具**

堆操作指令（pwndbg插件独有）

- arena //显示arena的详细信息
    - arenas //显示所有arena的基本信息
    - arenainfo //**好看的显示**所有arena的信息
- bins //**常用**，查看所有种类的堆块的链表情况
    - fastbins //单独查看fastbins的链表情况
    - largebins //同上，单独查看largebins的链表情况
    - smallbins //同上，单独查看smallbins的链表情况
    - unsortedbin //同上，单独查看unsortedbin链表情况
    - tcachebins //同上，单独查看tcachebins的链表情况
        - tcache //查看tcache详细信息
- heap //**数据结构的形式显示所有堆块**，会显示一大堆
    - heapbase //**查看堆起始地址**
    - heapinfo、heapinfoall //显示堆得信息，和bins的挺像的，**没bins好用**
    - parseheap //显示堆结构，**很好用**
- tracemalloc //**好用**，会跟提示所有操作堆的地方

其他pwndbg插件独有指令

- cyclc 50 //生成50个用来溢出的字符，如：aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaama
- $reabse //**开启PIE的情况的地址偏移**
  - b *$reabse(0x123456) //断住PIE状态下的二进制文件中0x123456的地方
  - codebase //打印PIE偏移，**与rebase不同，这是打印，rebase是使用**
- stack //**查看栈**
  - retaddr //打印包含返回地址的栈地址
- canary //直接看canary的值
- plt //查看plt表
  - got //查看got表
- hexdump //想IDA那样显示数据，**带字符串**

Cascadia Code