看雪pwn入门--基础篇

原创

泣当歌 于 2021-03-28 15:27:52 发布 330 收藏 6

文章标签： 安全 信息安全 pwn

本文链接：https://blog.csdn.net/qidangge/article/details/114695067

1.pwn理论基础

小菜：

什么是溢出

概念：在计算机中，当要表示的数据超出计算机使用的数据表示范围时，产生数据的溢出

原因：使用非类型安全的语言入C/C++等

以不可靠的方式存取或者复制内存缓冲区

编译器设置内存缓冲区太靠近关键数据结构

什么是pwn

“pwn"是一个黑客俚语词，指攻破设备或者系统。发音类似"砰”。就是向目标发送特定的数据，使其执行我们发送的错误代码。

（1）*汇编语言的基础（请点击）*

（2）*vi编辑器的基础操作（请点击））*

（3）*chmod命令的解析*

（请点击）用于提权

（4）gcc编译

请点击

用于对编写的程序进行编译得到可执行文件

（5）*gdb反汇编的基础命令*

***disass** main反汇编出来main函数的汇编代码*

```
root@YRJ:/home/xuenixiang/桌面/Pwn# gdb test
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
pwndbg: loaded 178 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
Reading symbols from test...(no debugging symbols found)...done.
pwndbg> disass main
Dump of assembler code for function main:
   0x0000000000000614 <+0>:     push   rbp
   0x0000000000000615 <+1>:     mov    rbp,rsp
   0x0000000000000618 <+4>:     sub    rsp,0x10
   0x000000000000061c <+8>:     mov    esi,0x2
   0x0000000000000621 <+13>:    mov    edi,0x1
   0x0000000000000626 <+18>:    call   0x5fa <test>
   0x000000000000062b <+23>:    mov    DWORD PTR [rbp-0x4],eax
   0x000000000000062e <+26>:    mov    eax,0x0
   0x0000000000000633 <+31>:    leave
   0x0000000000000634 <+32>:    ret
End of assembler dump.
pwndbg>
```


设置断点

设置断点可以通过b或者break设置断点，断点的设置可以通过函数名、行号、文件名+函数名、文件名+行号以及偏移量、地址等进行设置。

break 函数名

break 行号

break 文件名:函数名

break 文件名:行号

break +偏移量

break -偏移量

break *地址

```
pwndbg> b main
Breakpoint 2 at 0x555555554618
pwndbg> b *0x00000000000005fa
Breakpoint 3 at 0x5fa
```

运行

r是运行（run）

n是单步执行，next遇到函数不会进入函数内部

si是步入，step会执行到函数内部

c是继续运行，调试时，使用continue命令继续执行程序。程序遇到断点后再次暂停执行；如果没有断点，就会一直执行到结束。

删除断点

删除断点通过命令包括：

delete <断点id>：删除指定断点

delete：删除所有断点

clear

clear 函数名

clear 行号

clear 文件名：行号

clear 文件名：函数名

查看断点

info br

简写：i b

图片概述：

```
gcc -g  main.c                          //在目标文件加入源代码的信息
gdb a.out

(gdb) start                             //开始调试
(gdb) n                                 //一条一条执行
(gdb) step/s                            //执行下一条，如果函数进入函数
(gdb) backtrace/bt                      //查看函数调用栈帧
(gdb) info/i locals                     //查看当前栈帧局部变量
(gdb) frame/f                           //选择栈帧，再查看局部变量
(gdb) print/p                           //打印变量的值
(gdb) finish                            //运行到当前函数返回
(gdb) set var sum=0                     //修改变量值
(gdb) list/l 行号或函数名                 //列出源码
(gdb) display/undisplay sum             //每次停下显示变量的值/取消跟踪
(gdb) break/b  行号或函数名               //设置断点
(gdb) continue/c                        //连续运行
(gdb) info/i breakpoints                //查看已经设置的断点
(gdb) delete breakpoints 2              //删除某个断点
(gdb) disable/enable breakpoints 3      //禁用/启用某个断点
(gdb) break 9 if sum != 0               //满足条件才激活断点
(gdb) run/r                             //重新从程序开头连续执行
(gdb) watch input[4]                    //设置观察点
(gdb) info/i watchpoints                //查看设置的观察点
(gdb) x/7b input                        //打印存储器内容，b--每个字节一组，7--7组
(gdb) disassemble                       //反汇编当前函数或指定函数
(gdb) si                                // 一条指令一条指令调试 而 s 是一行一行代码
```

复制

（5）整体练习：栈帧的形成与释放

栈帧其实就是一个函数执行的环境，就是一个函数执行的时候，他的函数参数、函数的局部变量，函数执行完返回的地址。

2.保护和溢出

小菜

2.1***常见的保护***

2.1.1 Stack Protector（栈保护）

具体效果：当启用栈保护后，函数开始执行的时候会先往栈里插入cookie信息，当函数真正返回的时候会验证cookie信息是否合法，如果不合法就停止程序运行。攻击者在覆盖返回地址的时候往往也会将cookie信息给覆盖掉，导致栈保护检查失败而阻止shellcode的执行。在Linux中我们将cookie信息称为canary，所以这种保护方式也被称为**CANNAPY**。

2.1.2 NX（DEP）

NX即No-eXecute(不可执行）的意思。
基本原理：将数据所在内存页标识为不可执行，当程序溢出成功转入shellcode时，程序会尝试在数据页面上执行指令，此时CPU就会抛出异常，而不是去执行恶意指令。
注：NX等同于Windows下的DEP。

2.1.3 PIE（ASLR）

基本原理：标准的可执行程序需要固定的地址，并且只有被装载到这个地址才能正确执行，PIE能使程序像共享库一样在主存任何位置装载，这需要将程序编译成位置无关，并链接为ELF共享对象。
注：一般情况下NX和地址空间分布随机化ASLR会同时工作。内存地址随机化机制（address space layout randomization)，有以下三种情况
0 - 表示关闭进程地址空间随机化。
1 - 表示将mmap的基址，stack和vdso页面随机化。
2 - 表示在1的基础上增加栈（heap）的随机化。
可以防范基于Ret2libc方式的针对DEP的攻击。ASLR和DEP配合使用，能有效阻止攻击者在堆栈上运行恶意代码。

2.2 检查保护情况

checksec+文件名

注：要在root权限下操作

```
root@qidangge-virtual-machine:/home/qidangge/桌面/pwn# checksec hello
[*] '/home/qidangge/\xe6\xa1\x8c\xe9\x9d\xa2/pwn/hello'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```


2.3 保护的关闭与开启（gcc指令）

博客详细记录

2.3.1 栈保护的开启与关闭

注：默认不开启
gcc -fstack-protector -o hello hello.c //启用堆栈保护，不过只为局部变量中含有 char 数组的函数插入保护代码

gcc -fstack-protector-all -o test test.c //启用堆栈保护，为所有函数插入保护代码

gcc -fno-stack-protector -o test test.c //禁用栈保护

```
root@qidangge-virtual-machine:/home/qidangge/桌面/pwn# checksec hello
[*] '/home/qidangge/\xe6\xa1\x8c\xe9\x9d\xa2/pwn/hello'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

2.3.2 NX（DEP）的开启与关闭

gcc编译器默认开启了NX选项，如果需要关闭NX选项，可以给gcc编译器添加-z execstack参数。
例如：
gcc -z execstack -o hello hello.c

```
root@qidangge-virtual-machine:/home/qidangge/桌面/pwn# checksec hello
[*] '/home/qidangge/\xe6\xa1\x8c\xe9\x9d\xa2/pwn/hello'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      PIE enabled
    RWX:      Has RWX segments
```


如果关闭后要开启NX，输入指令：
gcc -z noexecstack -o test test.c // 开启NX保护

2.3.3 PIE(ASLR)的开启与关闭

注：默认不开启
gcc -fpie -pie -o test test.c // 开启PIE,此时强度为1
gcc -fPIE -pie -o test test.c // 开启PIE，此时为最高强度2

2.4 查看调用的函数

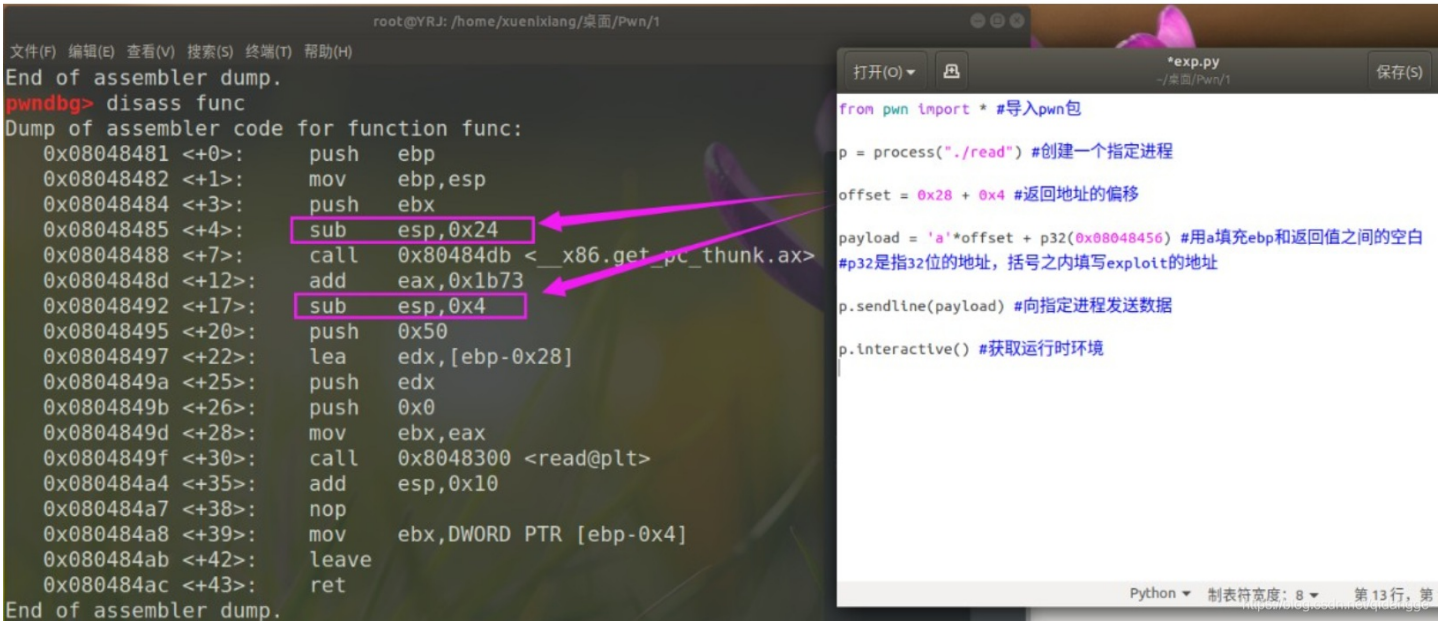
objdump命令是用查看目标文件或者可执行的目标文件的构成的gcc工具。（linux命令大全）
objdump -t -j .text hello//查看hello程序的.text段有哪些函数
各个表示：
-j name
–section=name 仅仅显示指定名称为name的section的信息

-t
–syms 显示文件的符号表入口。类似于nm -s提供的信息

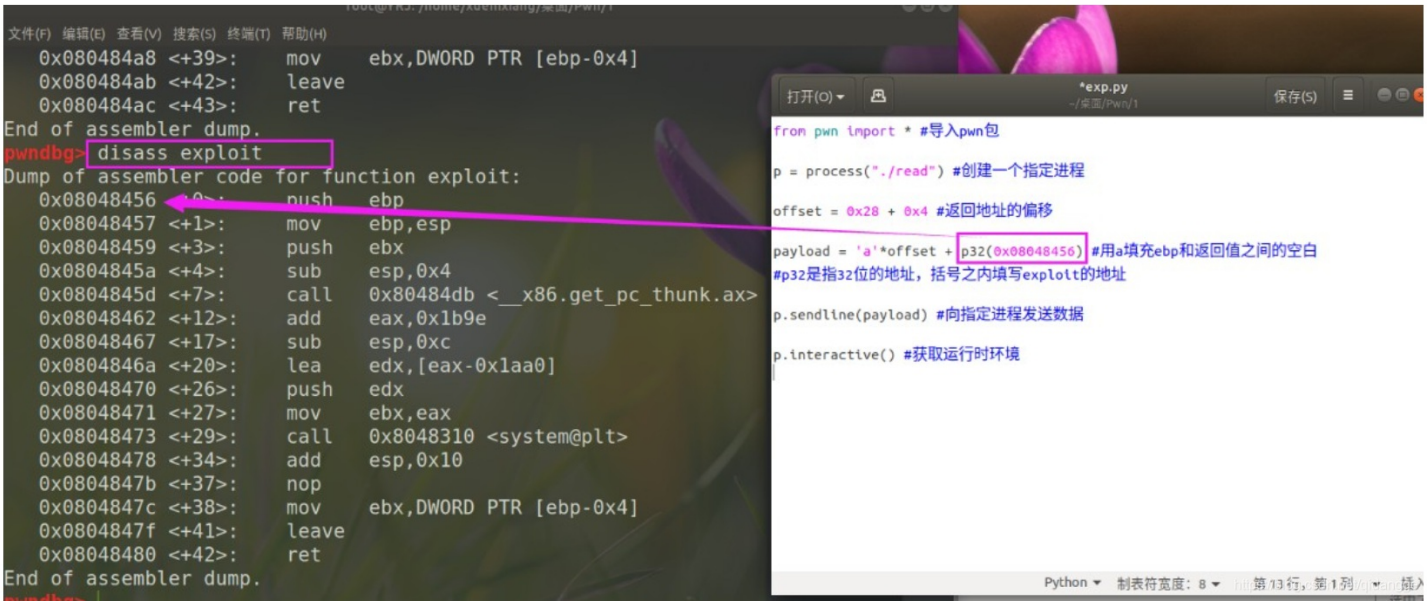
```
root@qidangge-virtual-machine:/home/qidangge/桌面/pwn# objdump -t -j .text hello

hello:     文件格式 elf64-x86-64

SYMBOL TABLE:
00000000000004f0 l    d  .text  0000000000000000              .text
0000000000000520 l     F .text  0000000000000000              deregister_tm_clones
0000000000000560 l     F .text  0000000000000000              register_tm_clones
00000000000005b0 l     F .text  0000000000000000              __do_global_dtors_aux
00000000000005f0 l     F .text  0000000000000000              frame_dummy
00000000000006b0 g     F .text  0000000000000002              __libc_csu_fini
0000000000000640 g     F .text  0000000000000065              __libc_csu_init
00000000000004f0 g     F .text  000000000000002b              _start
0000000000000614 g     F .text  0000000000000025              main
00000000000005fa g     F .text  000000000000001a              test
```


4.一个简单exp的编写

3.计算返回值偏移



4.查看exploit的地址



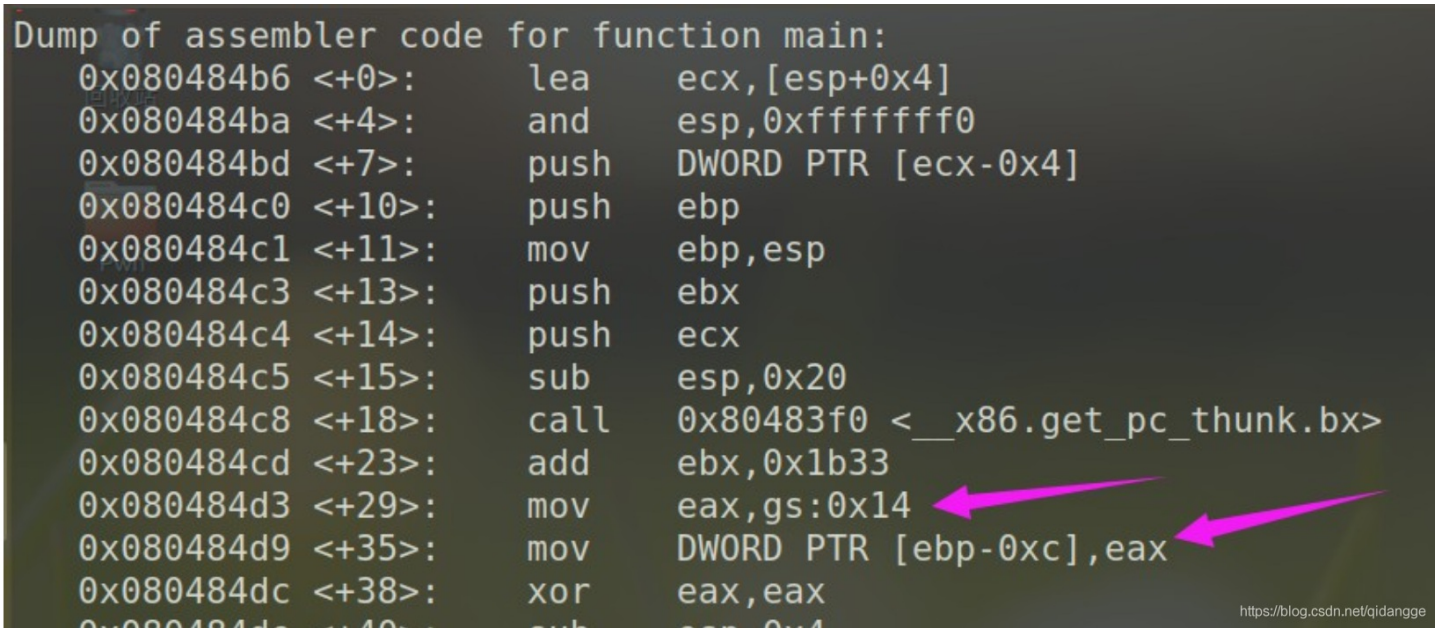
什么是exp?payload是什么？怎么使用python获取？

渗透中 PoC、Exp、Payload 与 Shellcode 的区别

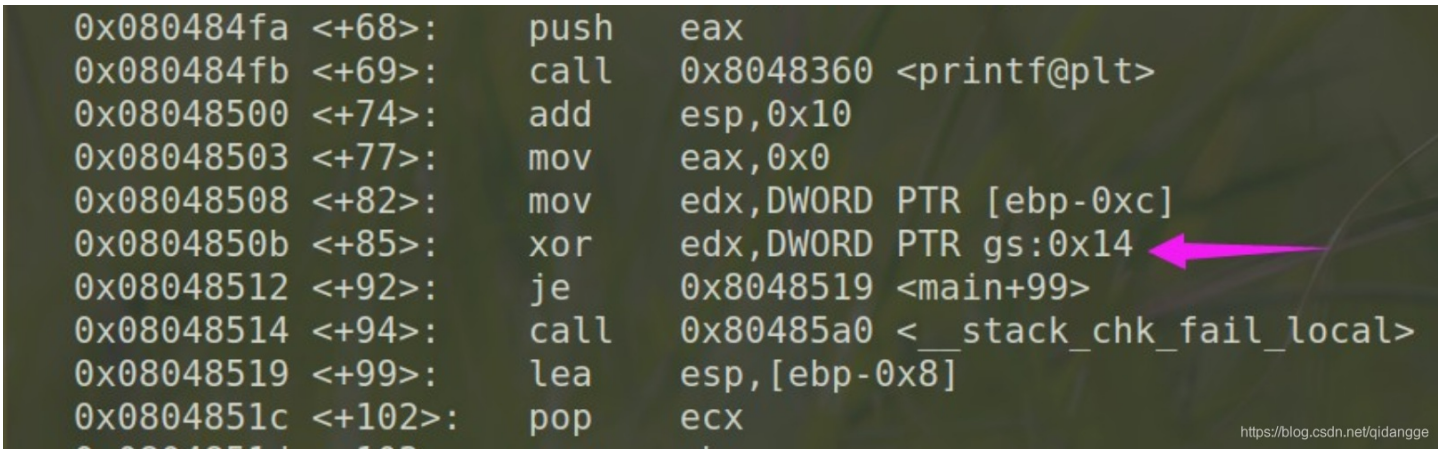
exp解析及pwntools相关使用

3.什么是canary保护

简单来说，canary保护就是在返回地址之前插入一个随机数据，在返回前先校验此数据是否被更改，如果被更改则je跳转到不可执行。
复习一下：栈保护的开启与关闭（上方）
实战演示：



可以发现：程序运行后，先将一个数据入栈到栈底（最后出去）。



在函数结束前，会使用xor指令进行校验，不一致则je跳转，无法返回。

4.print漏洞概述及其调试

https://bbs.ichunqiu.com/thread-42943-1-1.html?from=bkyl
+https://bbs.pediy.com/thread-250858.htm