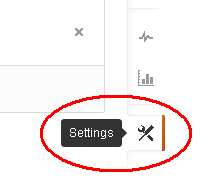
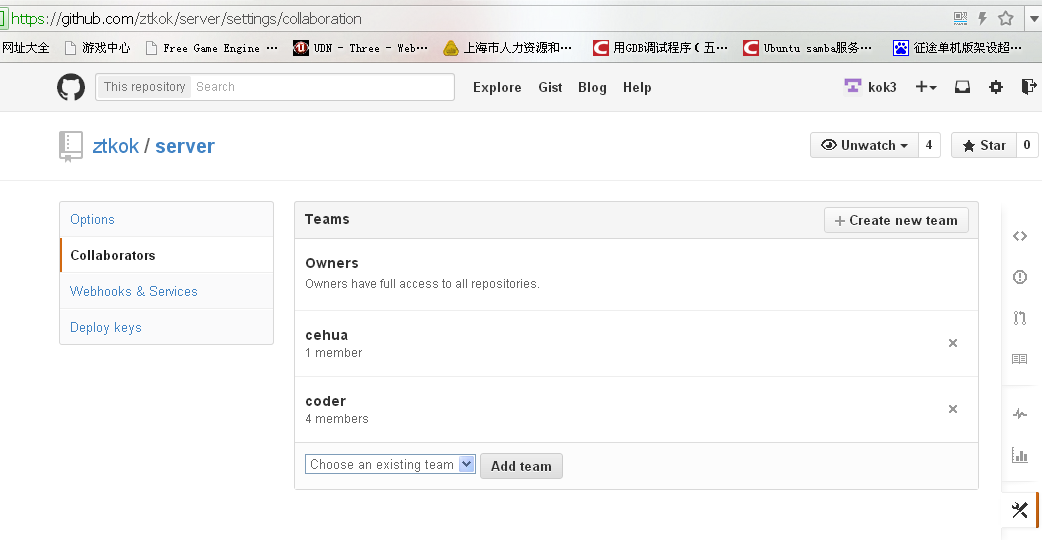
# 杂

## Github加权限





# 服务器宕机

Ch09 跨服

Ch50 亚伯

CH02 新区

## 常用操作

**宕机文件**

ll core.\*

**调试core文件**

gdb –c core.123 –se SessionServer.symbol

**查日志文件：**

grep 偶尔打打酱油 /log/sessionserver.log

grep 偶尔打打酱油 /log/sessionserver.log.141016-09 | grep 注册成功

**查看动态日志：**

tailf /log/sessionserver.log

**实时日志**

tailf /log/gatewayserver.log

**线程：**

Pstack

**pmap**

~/.gdbinit

pmap \_activeElment ‘数据类型‘

e.g.

std::map<DWORD， SkillStatusElement> \_activeElment

pmap \_avtiveElement std::pair<DWORD， SkillStatusElement>

**pset**

set<string> teststr;

要查看的时候操作如下：

pset testset

gdb显示

Set type =

std::set<std::basic\_string<char, std::char\_traits<char>, std::allocator<char> >,std::less<std::basic\_string<char, std::char\_traits<char>, std::allocator<char> > >,std::allocator<std::basic\_string<char, std::char\_traits<char>, std::allocator<char> > > >

Use pset <variable\_name> <element\_type> to see the elements in the set.

Set size = 1

2. 根据显示,输入

pset testset 'std::basic\_string<char, std::char\_traits<char>, std::allocator<char> >'

注意变量名称后面跟的模板T的类型最好用单引号'括上

<http://blog.csdn.net/hitzhang/article/details/5355129>

## 变长越界（堆栈破坏）

**缓冲区处理：（最后加多加2个验证字节）**

#define BUFFER\_CMD\_CHECK(cmd, name, len) CheckedBuf buffer\_##name(len, \_\_PRETTY\_FUNCTION\_\_)\

cmd \*name=(cmd\*)buffer\_##name.buffer(); constructInPlace(name);

然后定义下类 CheckedBuf :

class CheckedBuf  
{  
public:  
CheckedBuf(DWORD size, const char\* des)  
{  
\_size = size;  
\_des = des;  
\_buffer = new char[\_size + 2];  
bzero(\_buffer, \_size + 2);  
\_buffer[\_size] = 20;  
\_buffer[\_size + 1] = 12;  
}  
~CheckedBuf()  
{  
if(\_buffer[\_size] != 20 || \_buffer[\_size + 1] != 12 )  
{  
//log des 出界了  
}  
}  
  
char buffer(){ return \_buffer;}  
};

然后所有的变长定义 直接使用 BUFFER\_CMD\_CHECK(x,x,x);

**gcc打印堆栈：**

\_\_builtin\_return\_address(level)

level=0表示第一层地址

level=1表示第二层地址

level=2表示第三层地址

以此类推

使用时直接

printf("%p %P", \_\_builtin\_return\_address(0), \_\_builtin\_return\_address(1));

反编译

objdump -Cd yourExecutable > my.asm

查看printf的输出

在my.asm中找到对应的函数地址.

**VC6模拟\_\_builtin\_return\_address**

获取函数返回值：

gcc提供了获取当前函数的返回地址的一个关键字：

void \* \_\_builtin\_return\_address(int nLevel);

nlevel 参数指获取哪个函数的返回值，0表示当前函数，1表示当前函数的调用者的函数，为2依次类推；

下面是vc6的实现：

**[cpp]** [view plaincopy](http://blog.csdn.net/sstower/article/details/6906445)

1. **\_\_declspec** (**naked**) **void**\* \_\_builtin\_return\_address (**int** iLevel)
2. {
3. \_\_asm
4. {
5. push ebx;
7. mov eax, ebp;
8. mov ebx, **DWORD** PTR [esp + 8]; // iLevel
10. \_\_Next:
11. test ebx, ebx;
12. je  \_\_break;
13. dec ebx;
14. mov eax, **DWORD** PTR [eax];
15. jmp \_\_Next;
16. \_\_break:
18. mov eax, **DWORD** PTR [eax + 4];
19. pop ebx;
20. ret;
21. }
22. }

用一个测试例子来试一下：

**[cpp]** [view plaincopy](http://blog.csdn.net/sstower/article/details/6906445)

1. **int** sum(**int** a,**int** b)
2. {
3. **void** \*add=\_\_builtin\_return\_address(0);
4. **return** a+b;
5. }
6. **int** main(**int** argc, **char**\* argv[])
7. {

10. **int** c=sum(1,4);
12. **return** 0;
13. }

这里取得是当前函数的返回地址；调试过程中用汇编展开：

**[cpp]** [view plaincopy](http://blog.csdn.net/sstower/article/details/6906445)

1. 32:   **int** main(**int** argc, **char**\* argv[])
2. 33:   {
3. 004010A0   push        ebp
4. 004010A1   mov         ebp,esp
5. 004010A3   sub         esp,44h
6. 004010A6   push        ebx
7. 004010A7   push        esi
8. 004010A8   push        edi
9. 004010A9   lea         edi,[ebp-44h]
10. 004010AC   mov         ecx,11h
11. 004010B1   mov         eax,0CCCCCCCCh
12. 004010B6   rep stos    dword ptr [edi]
13. 34:
14. 35:
15. 36:       **int** c=sum(1,4);
16. 004010B8   push        4
17. 004010BA   push        1
18. 004010BC   call        @ILT+0(sum) (00401005)
19. 004010C1   add         esp,8
20. 004010C4   mov         dword ptr [ebp-4],eax
21. 37:
22. 38:       **return** 0;
23. 004010C7   xor         eax,eax
24. 39:   }

我们看到，sum函数的返回地址应该是004010C1， 然后跟进sum函数，查看执行后add的地址发现恰恰是004010C1，说明模仿成功。

## vi

**切分窗口**：cp chp

**切换窗口**CTRL+WW

**对齐：**v 方向键 =

## grep

## svn

清除svn保存的账号密码信息:

rm –rf ~/.subversion/auth

svn st

svn checkout <http://192.168.1.1/pro/domain>

## mysql

SERVERLIST

1.导出整个数据库  
　　mysqldump -u用户名 -p密码  数据库名 > 导出的文件名  
　　C:\Users\jack> mysqldump -uroot -pmysql sva\_rec  > e:\sva\_rec.sql  
   
　　2.导出一个表，包括表结构和数据  
　　mysqldump -u用户名 -p 密码  数据库名 表名> 导出的文件名  
　　C:\Users\jack> mysqldump -uroot -pmysql sva\_rec date\_rec\_drv> e:\date\_rec\_drv.sql  
   
　　3.导出一个数据库结构  
　　C:\Users\jack> mysqldump -uroot -pmysql -d sva\_rec > e:\sva\_rec.sql  
     
     4.导出一个表，只有表结构  
　　mysqldump -u用户名 -p 密码 -d数据库名  表名> 导出的文件名  
　　C:\Users\jack> mysqldump -uroot -pmysql -d sva\_rec date\_rec\_drv> e:\date\_rec\_drv.sql

**导入数据库：**

mysql>source d:wcnc\_db.sql

导出数据库：

**添加用户并设置权限：**

GRANT ALL ON \*.\* TO `username`@`%` IDENTIFIED BY ‘passwd’

**导出用户数据：**

Select \* from CHARBASE where charid=12345678 into outfile ‘/tmp/user12345678.sql’;

**导入用户数据：**

Load data infile ‘/tmp/user12345678.sql’ INTO TABLE CHARBASE;

导入后修改新增记录ACCID字段成现有的ACCID

## linux

查看用户：cat /etc/passwd

查看用户所属的组：id username

查看所有组：cat /etc/group

创建用户 # useradd usera

修改密码： passwd (自己修改)

passwd username（root）

**编码：**

   #vi /root/.bash\_profile  
   在文件尾加入  
   export LC\_ALL="zh\_CN.GB18030"  
   export LANG="zh\_CN.GB18030"

**打包解包：**

tar cvjf patch.tar.bz2(打包后的文件名) \*(需打包文件)

tar xjvf patch.tar.bz2

# 其它

## gdb

Server启动

SuperServer –d

#FLServer-d

RecordServer-d

BillServer-d

MiniServer-d

SessionServer-d

SceneServer-d

SceneServer-d

SceneServer-d

SceneServer-d

GatewayServer-d

#QueueServer-d

ArchiveServer -d

# 文件说明

WaterEntity 🡪 ModelEntity 负责创建更新？创建Model ModelNode

Water 🡪 Model 负责管理？渲染

WaterNode 🡪 ModeNode （可以包含很多NodeInstance）与Mesh挂钩

FrameShade FrameServer 对应datas/frame文件夹

Shader ShaderServer 对应datas/shades文件夹