记录些kernel框架里C函数使用,学习这些C函数使用方法最好的办法就是看github上的文档.

//读出内存储存到一个buf里.
vm_read比较少用.所以用法不太清楚.一般用vm_read_overwrite,会覆盖用来存放读出数据的buf.这样好用.

vm_read_overwrite(vm_map_t 目标task,vm_address_t 开始读的地址,vm_size_t 读的大小(一次不能超过0x1000),vm_address_t 用来存放读出内容的buf,vm_size_t* 一般&size(即存放读出大小的变量的&))

//遍历内存区域
32位程序使用于vm_region_recurse,而64位程序使用vm_region_recurse_64,用法一样.一般会放在循环语句里面

vm_size_t size;//调用完函数后.值是地址所在的内存区域的大小.
unsigned int depth = 0;//不清楚
vm_region_submap_info_data_64_t info;//好像会返回更多有关信息,32位使用vm_region_submap_info_data_t
mach_msg_type_number_t info_count = VM_REGION_SUBMAP_INFO_COUNT_64;//32位为VM_REGION_SUBMAP_INFO_COUNT

vm_region_recurse_64(vm_map_t 目标task,vm_address_t 开始的内存地址,vm_size_t* 传入&size, natural_t* 传入&depth,vm_region_recurse_info_t 传入&info,mach_msg_type_number_t* 传入&info_count)

例子:

```
while(1){
        ret = vm_region_recurse_64(mach_task_self(),&addr,&size,&depth, (vm_region_info_t)&info,&info_count);
        if(ret!=KERN_SUCCESS)
            break;
        printf("addr:%lx size:%lx\n",addr,size);
        addr = addr + size;
    }

    printf("done\n");
```

//在一块内存中查找
memmem(void *用来查找的缓冲区buf,size_t 查找的范围大小,const void *查找的字符串,size_t 查找字符串的大小);
这里需要注意几点,我深有体会.
举个例子.
memmem(buf,strlen(buf),uuid字符串,uuid的大小).这里区分下sizeof和strlen.sizeof用与数组比较明确.对同一个字符串来说strlen会比sizeof多1.因为strlen算上了结尾的'\0'.就这样.要看好时候使用哪个.因为数组默认是不会自动加\0的.
用来查找的字符串(uuid字符串)参数那里,比如你需要找一个UUID的前几位为9498DADF-C1...这几位就够了.
那么用字符串的方法来查就是
char *uuid = "9498DADF"; //这是最简单的方法.但是如果你需要的到字节没有对应的字符就只能用hex值去查找了.
用char数组去查找.注意啊.不能直接弄成16进制格式,即0x9498DADF.这样是错的.太坑了.

```
char uuid[] = {0x39,0x34,0x39,0x38,0x44,0x41,0x44,0x46};
```
用转义字符也行.功能一样的.
```
char *uuid = “\x39\x34\x39\x38\x44\x41\x44\x46”;
```
嗯,要仔细看下.