



软件安全漏洞挖掘 技术探讨

文伟平 博士

北京大学软件与微电子学院



北京¹大学

漏洞挖掘技术难点

◆ 漏洞挖掘本身的难度

- 函数模型 (Windows 7之前系统)
- 逻辑模型
- 安全开发平台
- 云计算
- 虚拟化

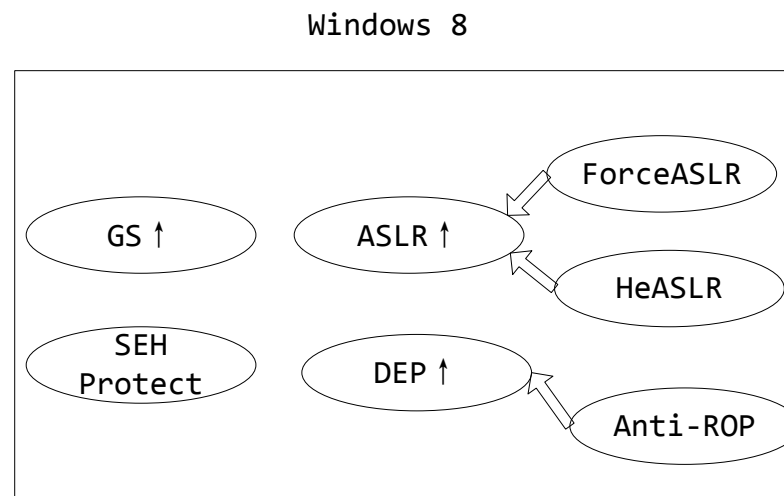
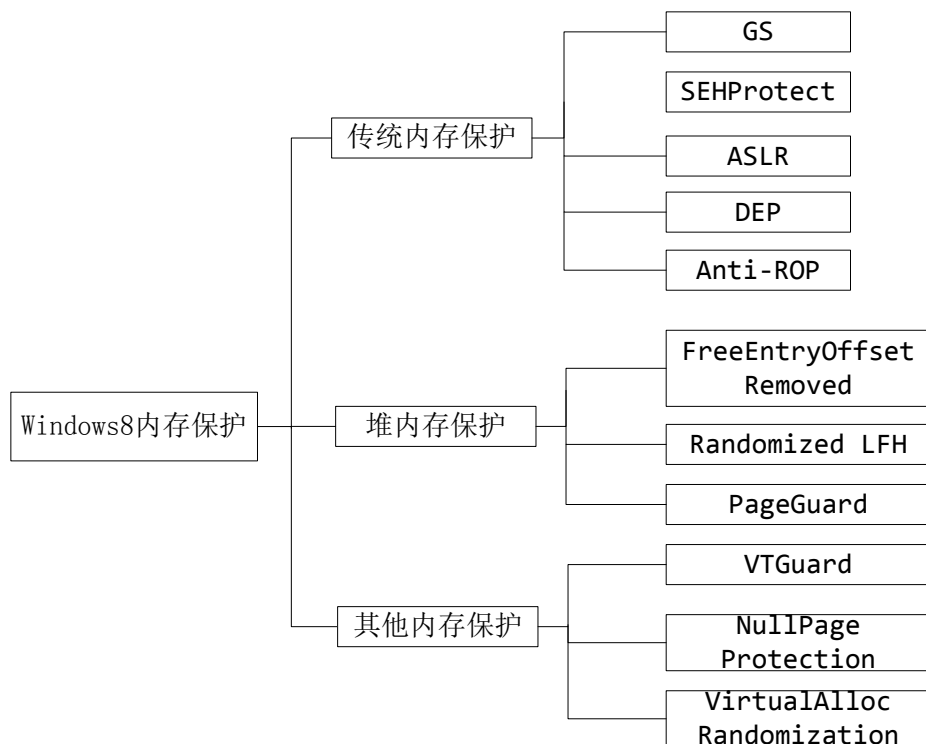
◆ 系统安全对抗

- Windows 7, 8系统安全防护机制
- IE 的Use after Free
- GS, SafeSEH, DEP和ASLR
- ROP, Heap Spray, JIT Spray
- SharedUserData





漏洞挖掘和利用的技术难点





软件漏洞挖掘相关技术

- ◆ 基于Crash信息的漏洞挖掘技术
- ◆ 参考安全补丁的漏洞挖掘技术
- ◆ 系统内核函数无序调用挖掘技术
- ◆ 基于协议握手的测试漏洞挖掘技术
- ◆ 基于浏览器对象UAF漏洞挖掘技术
- ◆ 基于FLASH对象UAF漏洞挖掘技术



基于Crash信息的漏洞挖掘技术

■ 系统转储文件

- 完全内存转储
- 核心内存转储
- 小内存转储

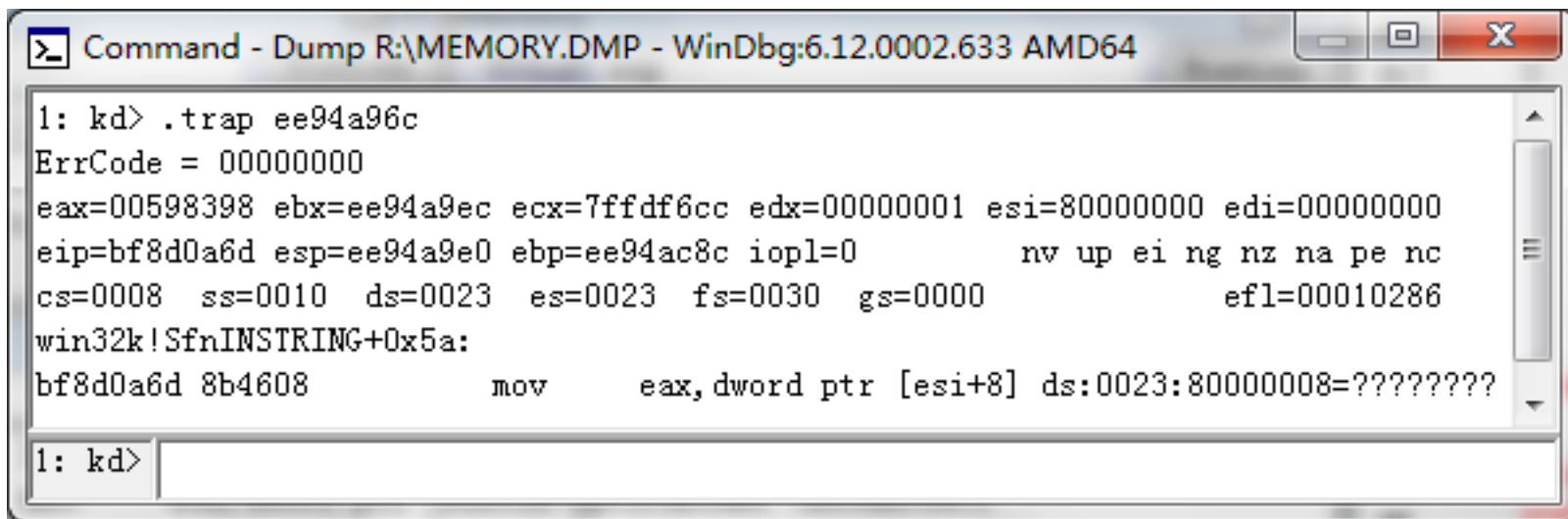
```
1 Microsoft (R) Windows Debugger Version 6.12.0002.633 AMD64
2 Copyright (c) Microsoft Corporation. All rights reserved.
3
4
5 Loading Dump File [R:\MEMORY.DMP]
6 Kernel Summary Dump File: Only kernel address space is available
7 (省略系统概括和符号信息)
8 *****
9 *
10 *                      Bugcheck Analysis
11 *
12 *****
13
14 Use !analyze -v to get detailed debugging information.
15
16 BugCheck 50, {80000008, 0, bf8d0a6d, 0}
17
18 Probably caused by : win32k.sys ( win32k!SfnINSTRING+5a )
19
20 Followup: MachineOwner
21 -----
22
```



基于Crash信息的漏洞挖掘技术

KTRAP_FRAME陷阱帧

```
1 1: kd> kv
2 ChildEBP RetAddr  Args to Child
3 .....
4 ee94a954 bf8d0a6d ..... nt!KiTrap0E+0xd0 (FP0: [0,0] TrapFrame @ ee94a96c)
5 .....
```



```
> Command - Dump R:\MEMORY.DMP - WinDbg:6.12.0002.633 AMD64

1: kd> .trap ee94a96c
ErrCode = 00000000
eax=00598398 ebx=ee94a9ec ecx=7ffdf6cc edx=00000001 esi=80000000 edi=00000000
eip=bf8d0a6d esp=ee94a9e0 ebp=ee94ac8c iopl=0         nv up ei ng nz na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010286
win32k!SfnINSTRING+0x5a:
bf8d0a6d 8b4608          mov     eax,dword ptr [esi+8] ds:0023:80000008=????????

1: kd>
```


基于Crash信息的漏洞挖掘技术

■ 定位函数

```
Command - Dump R:\MEMORY.DMP - WinDbg:6.12.0002.633 AMD64

1: kd> k
*** Stack trace for last set context - .thread/.cxr resets it
ChildEBP RetAddr
ee94ac8c bf80d382 win32k!SfnINSTRING+0x5a
ee94acc0 bf8585a6 win32k!xxxDefWindowProc+0xef
ee94acdc bf803690 win32k!xxxEventWndProc+0x67
ee94ad0c bf80ecbe win32k!xxxDispatchMessage+0x19e
ee94ad58 8054261c win32k!NtUserDispatchMessage+0x39
ee94ad58 7c92e4f4 nt!KiFastCallEntry+0xfc
WARNING: Frame IP not in any known module. Following frames may be wrong.
0007feb4 00000000 0x7c92e4f4

1: kd> |
```

- 最后执行的函数为win32k!SfnINSTRING，接下来应该利用IDA工具分析该函数的反汇编代码，判断是否存在逻辑缺陷，判断是否可以进一步利用。

基于Crash信息的漏洞挖掘技术

■ 静态分析相关代码，确定原因

■ win32k!SfnINSTRIN
G函数执行过程中，对用户
态空间传递过来的wParam、
lParam没有做任何判断，
复制[esi+8]处的数据时，
也没有限制[esi+8]是内核
地址还是用户态地址。如
果esi、[esi+8]是一个不可
访问的内存地址，那么将
会导致无法访问内存而蓝
屏崩溃。

MS11-054发现过程

① .text:BF8D0A37	mov esi, [ebp+lParam]
② .text:BF8D0AAD	push 1 ; int
.text:BF8D0AAF	lea eax, [ebp+var_21C]
.text:BF8D0AB5	push eax ; int
.text:BF8D0AB6	push [ebp+AllocationSize] ; AllocationSize
.text:BF8D0ABC	push [ebp+var_220] ; int
.text:BF8D0AC2	push 30h ; int
.text:BF8D0AC4	call _AllocCallbackMessage@20
.text:BF8D0AC9	mov ebx, eax
③ .text:BF8D0CB7	push eax
.text:BF8D0CB8	push dword ptr [esi+8]
.text:BF8D0CBB	push ebx
.text:BF8D0CBC	call _CaptureCallbackData@16
④ .text:BF8D0BC8	lea eax, [ebp+var_24C]
.text:BF8D0BCE	push eax
.text:BF8D0BCF	lea eax, [ebp+var_240]
.text:BF8D0BD5	push eax
.text:BF8D0BD6	push dword ptr [ebx]
.text:BF8D0BD8	push ebx
.text:BF8D0BD9	push 1Ah
.text:BF8D0BDB	call ds:__imp__KeUserModeCallback@20



参考安全补丁的漏洞挖掘技术

由补丁引发的思考

◆ 通常安全补丁对漏洞代码的修改及代码运行流程基本不会有太大的变化。而这种漏洞修补方式可能存在如下安全隐患：

◆ 1) 软件厂商修补漏洞缺乏全局考虑，通常只注重对漏洞点的修补；

◆ 2) 往往只考虑当前漏洞的上下文环境，而未必考虑到整个系统或者第三方代码对全局变量或逻辑条件带来的影响。



参考安全补丁的漏洞挖掘技术

参考安全补丁比对的漏洞挖掘思路

软件安全补丁面临安全隐患

分析补丁，找出补丁所
修补的代码位置以及实
际出现问题的代码位置

路径查找、条件执行、符
号执行，判断代码内部是
否存在新的安全漏洞

- 1、路径查找会找到所有可能的执行路径
- 2、条件执行会尝试执行这些路径，以判断当前路径是否是实际可执行的
- 3、符号执行通过代码变量的逻辑抽象与控制流相结合得到条件约束，最后通过约束求解的方法，来判断代码内部是否存在安全漏洞。

参考安全补丁的漏洞挖掘技术

参考安全补丁技术应用

以某漏洞为例验证参考安全补丁漏洞挖掘技术的有效性：

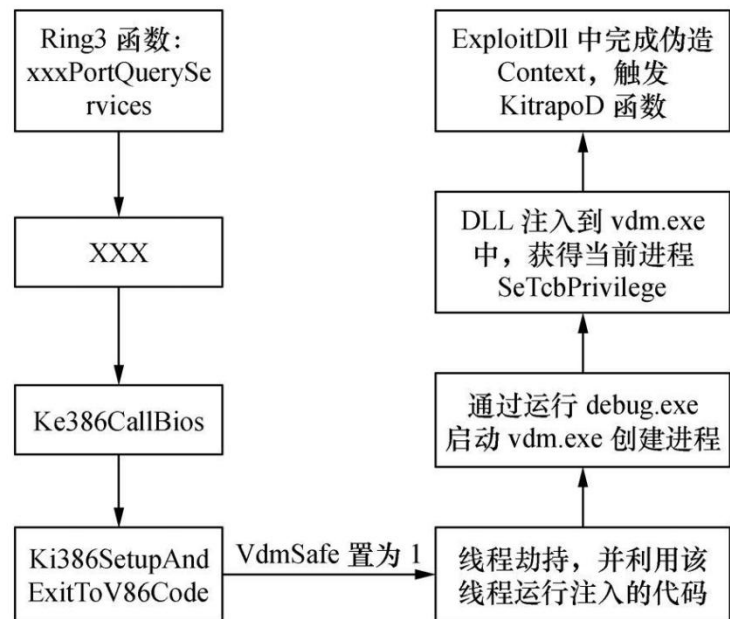
■ 专门为针对这个补丁而增设的标志位VdmSafe。漏洞点B及补丁点P已经定位。

```
mov     edi, [ebx+124h] ;
mov     byte ptr [edi+51h], 1
```

■ 通过分析可知，该补丁可能存在新的安全隐患，补丁后攻击代码实现攻击流程如右图所示：

表 1 _Ethread 结构补丁前后比较

补丁前	补丁后
kd> dt nt! _kthread	kd> dt nt! _kthread
.....
+ 0x051 Spare0: [3]	+ 0x051 VdmSafe: [1]
	+ 0x052 Spare0: [2]



参考安全补丁的漏洞挖掘技术

■ 参考安全补丁技术应用

- 北京大学软件安全研究小组发现的MS11-010进一步证明了参考安全补丁技术的有效性。

- 该漏洞是在分析MS10-011安全补丁的基础上，采用本方法发现的一个安全漏洞。

- MS10-011补丁为以下情况：B点和P点位于不同的函数中，补丁增加了运行至B点的逻辑条件，而这个逻辑条件是普通用户构造某些函数进行特定序列调用可以进行修改的，从而逻辑条件被恶意利用导致新的安全隐患。

- P点：补丁所修补的代码位置

- B点：实际出现问题的代码位置

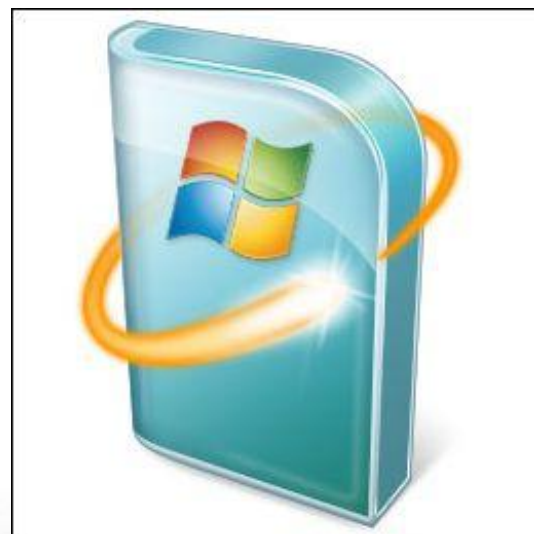
- B点和P点位于不同的基本块中，且B点和P点分布在不同函数中。这种安全漏洞修补方式，最有可能存在安全隐患。

系统内核函数无序调用挖掘技术

■ 系统内核漏洞

■ 随着Windows 7 / 8 操作系统发布，多种对抗内存攻击的安全防护机制使得传统的基于内存的攻击越来越困难，在这种情况下，内核漏洞往往可以作为突破安全防线的切入点。

- ✓ Windows内核模块，如win32k.sys等
- ✓ 第三方驱动程序
- ✓ 安全软件对SSDT、ShadowSSDT的处理



系统内核函数无序调用挖掘技术

■ API序列的乱序组合

- ◆ 一个正常功能的实现依赖一组API调用序列
- ◆ 任意改变这组序列，会有意想不到的结果
 - 序列中的函数多次调用
 - 调用顺序改变
 - 特定的条件或参数
- ◆ 序列中的API一般情况下是Undocumented API



系统内核函数无序调用挖掘技术

■ API乱序组合绕过MS10-011

- ◆ 关闭ApiPort时，系统将会向Csrss发送一个类型为LPC_PORT_CLOSED (0x5) 的消息。在CsrApiRequestThread函数中处理该类型消息有代码如下

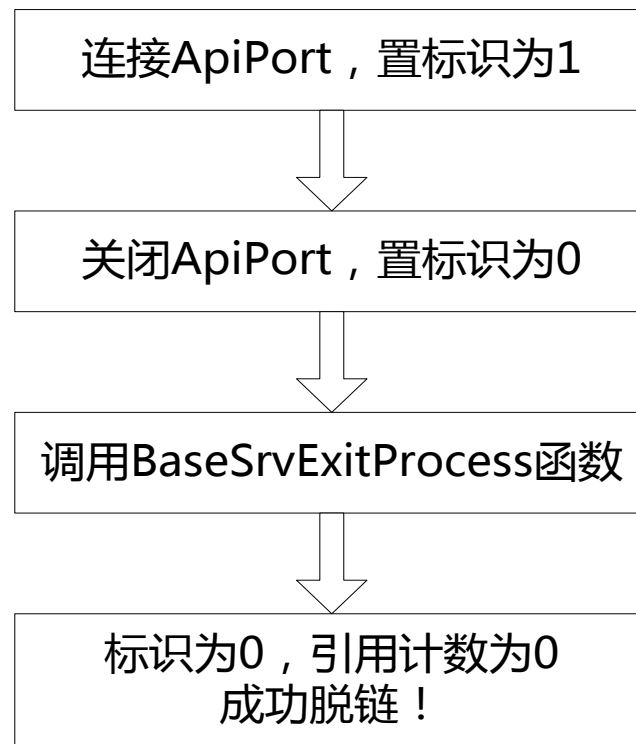
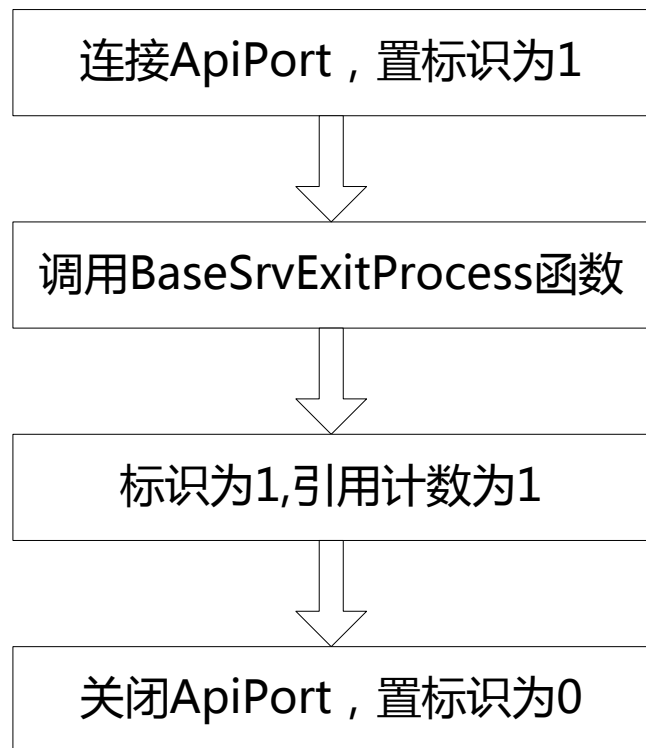
```
.text:75AA4780      and     byte ptr [eax+39h], 0DFh
.text:75AA4784      mov     eax, [ebp+var_1C]
.text:75AA4787      cmp     [eax+58h], esi
.text:75AA478A      jnz     short loc_75AA4798
.text:75AA478A
.text:75AA478C      or      byte ptr [eax+39h], 2
.text:75AA4790      push    [ebp+var_1C]
.text:75AA4793      call    CsrLockedDereferenceProcess(x)
```

■ 若走过这段流程，足以让我们绕过补丁了！



系统内核函数无序调用挖掘技术

API乱序组合绕过MS10-011补丁



基于协议握手的漏洞挖掘技术

■ 非开源网络软件漏洞挖掘

■ 非开源网络软件漏洞挖掘的“公理”——协议Fuzzing测试



■ 测试空间极大

小组主页: www.pku-exploit.com

联系 QQ: 513357938



北京大学

基于协议握手的漏洞挖掘技术

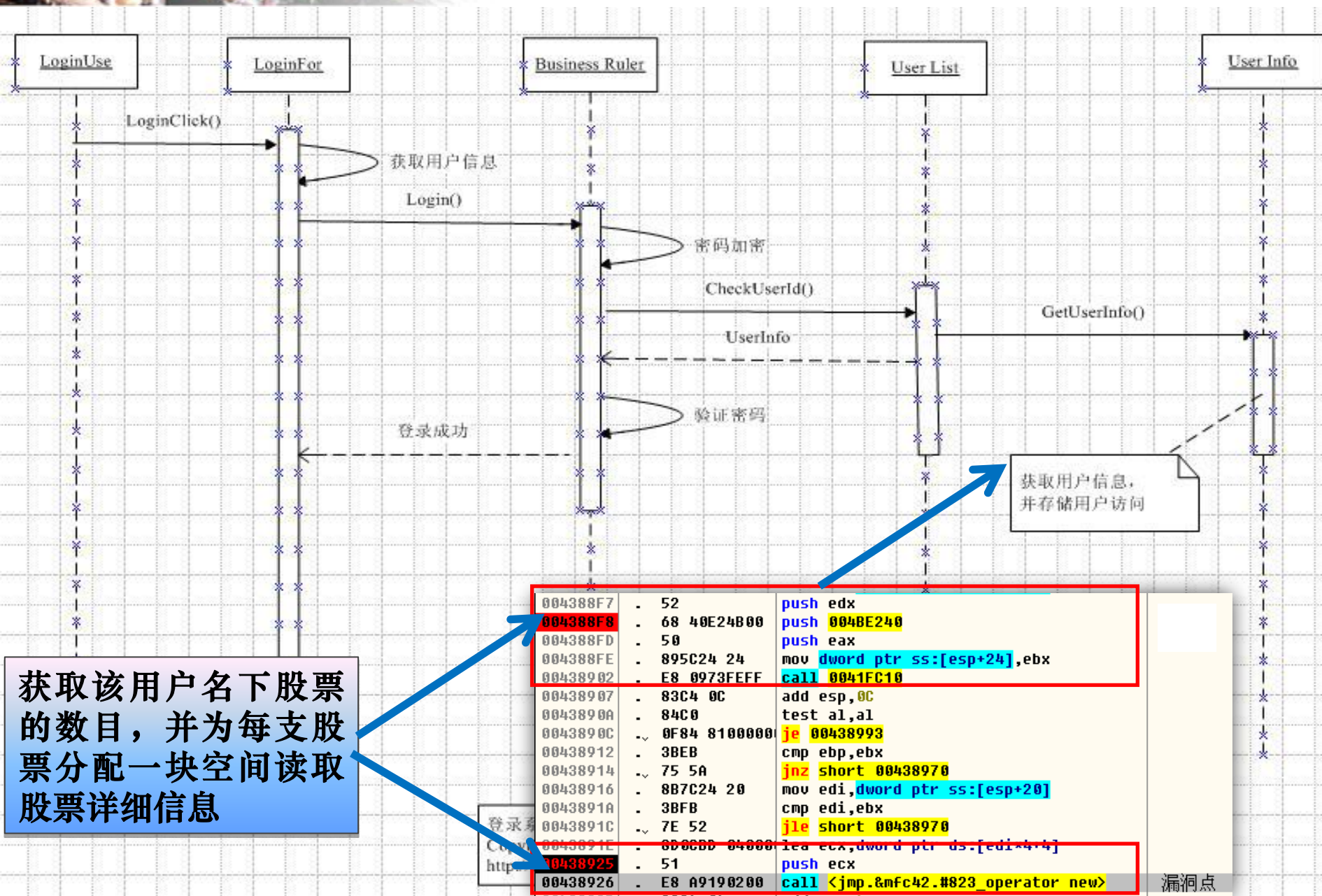
■ 共识：对测试用例进行裁剪

- 网络应用程序的潜在漏洞往往存在于对发送 / 接收的网络数据包进行处理的过程，进行Fuzzing测试时，如果不考虑到这个特性就盲目构造针对操作和函数调用的测试用例，是南辕北辙的行为，不可能获得良好效果。
- 容易发现的一点是，网络软件在进行数据处理时的网络状态不大可能是发送的第一个包，通常都是通信双方进行多次请求响应后的状态。
- 在这个思路的指引下，我们就要**模拟其网络通信流程**，构造合适的网络数据包作为测试用例。

■ 基于二进制的软件逆向分析

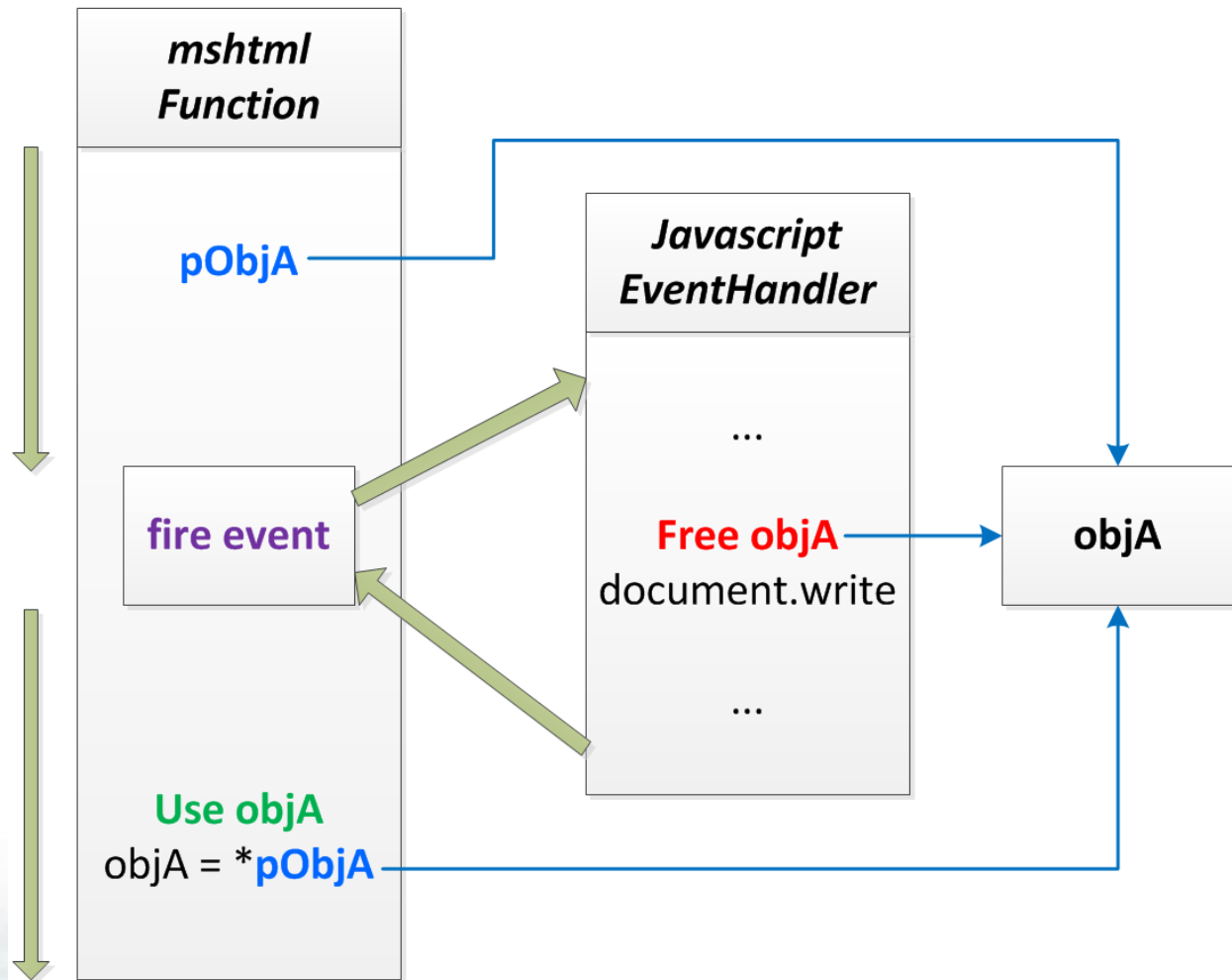


基于协议握手的漏洞挖掘技术



基于浏览器对象UAF漏洞挖掘技术

原因



基于浏览器对象UAF漏洞挖掘技术

■ 目前比较有效的Fuzz工具

■ Fuzzing Framework

■ Grinder

■ Fuzzer:

■ Cross Fuzz

■ ndujaFuzz

■ NodeFuzz

■ jsFunFuzz

■ 以及安全研究人员自己实现的各种Fuzzer

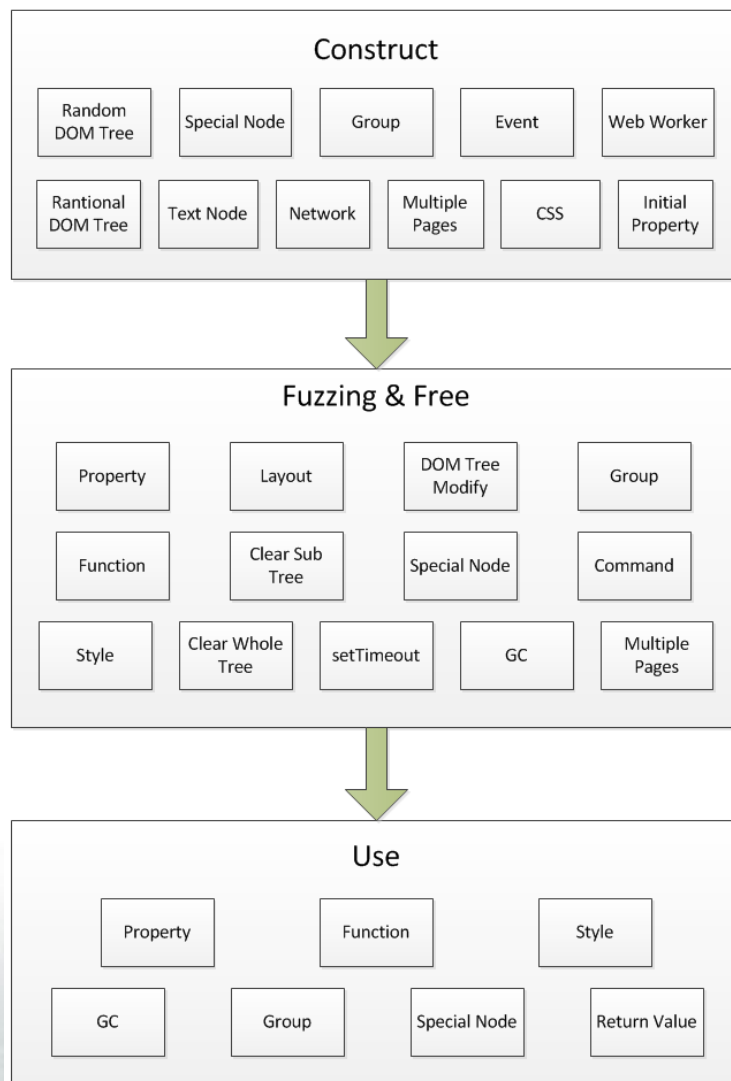
.....



基于浏览器对象UAF漏洞挖掘技术

UAF漏洞到底该怎么挖？

三步曲



基于浏览器对象UAF漏洞挖掘技术

■ 实现一个Browser Fuzzer?

阅读w3c等官方文档，构建字典。

建立最基础的dom树结构

```
struct DomEleInfo a_stru = { "a", { "accessKey", "animation", "animationDela
struct DomEleInfo abbr_stru = { "abbr", { "accessKey", "animation", "animati
struct DomEleInfo acronym_stru = { "acronym", { "accessKey", "animation", "a
struct DomEleInfo address_stru = { "address", { "accessKey", "animation", "a
struct DomEleInfo applet_stru = { "applet", { "accessKey", "align", "alt", "
```

```
char* interesting_str[] =
{
"undefined", "null", "true", "false", "'green'", "'tuesday'", "[]", "{}", "\"
};
```

```
char* commands_a[] =
{
"2D-Position", "backColor", "bold", "clearAuthenticationCache", "contentRe
"AbsolutePosition", "BackColor", "BackgroundImageCache", "ms-beginUndoUnit",
"DefaultParagraphSeparator", "Delete", "DirLTR", "DirRTL", "EditMode", "ms-endU
"InsertButton", "InsertFieldset", "InsertHorizontalRule", "InsertIFrame", "Ins
"Italic", "JustifyCenter", "JustifyFull", "JustifyLeft", "JustifyNone", "Justif
"OverWrite", "Paste", "ms-pasteContentOnly", "ms-pasteTextOnly", "PlayImage",
"SaveAs", "SelectAll", "SizeToControl", "SizeToControlHeight", "SizeToControlW
"Superscript", "UnBookmark", "Underline", "Undo", "Unlink", "Unselect"
};
```

```
char* standardAttributes[] = {
"nodeName", "nodeValue", "nodeType", "childNodes", "location", "name", "
};
```

```
for (int i = 0; i < max_element; i++)
{
/*Generate Dom element and link them to others*/
sprintf(outbuffer, "try { newNode_%X = document.createElementNS(HTML_NS, \"%s\"); } catch(e){}\n", i, domElemsList[rand() % 253], i);
fwrite(outbuffer, 1, strlen(outbuffer), tmp);

memset(outbuffer, 0, 0x1000);
if (i==0)
    sprintf(outbuffer, "try{ document.body.appendChild(newNode_%X);}catch(e){}\n", i);
else if (rand()%0x10 >=12)
    sprintf(outbuffer, "try{ document.body.appendChild(newNode_%X);}catch(e){}\n", i);
else
    sprintf(outbuffer, "try{ newNode_%X.appendChild(newNode_%X);}catch(e){}\n", rand()%i, i);
fwrite(outbuffer, 1, strlen(outbuffer), tmp);
memset(outbuffer, 0, 0x1000);

/*tweak properties of new generated element*/
sprintf(outbuffer, "newNode_%X", i);
tweak_attributes(tmp, outbuffer);
/*addlistener to element*/
memset(outbuffer, 0, 0x1000);
```

小组主页: www.pku-exploit.c

联系 Q Q: 513357938

基于浏览器对象UAF漏洞挖掘技术

■ 实现一个Browser Fuzzer?

从已有的测试样本构
建dom树

- Base Dom Tree Building
- randomNodes
- randomTree
- Generation algorithm
- document.createElement
- node.appendChild

2dcontext	2014/10/13 15:37	文件夹
ambient-light	2014/10/13 15:37	文件夹
animation-timing	2014/10/13 15:37	文件夹
app-uri	2014/10/13 15:37	文件夹
battery-status	2014/10/13 15:37	文件夹
common	2014/10/13 15:37	文件夹
conformance-checkers	2014/10/13 15:37	文件夹
content-security-policy	2014/10/13 15:37	文件夹
cors	2014/10/13 15:37	文件夹
custom-elements	2014/10/13 15:37	文件夹
dom	2014/10/13 15:37	文件夹
DOMEvents	2014/10/13 15:37	文件夹
domparsing	2014/10/13 15:37	文件夹
domxpath	2014/10/13 15:37	文件夹
eventsources	2014/10/13 15:37	文件夹
ext-xhtml-pubid	2014/10/13 15:37	文件夹
FileAPI	2014/10/13 15:37	文件夹
fonts	2014/10/13 15:37	文件夹
geolocation-API	2014/10/13 15:37	文件夹
hr-time	2014/10/13 15:37	文件夹
html	2014/10/13 15:38	文件夹
html-imports	2014/10/13 15:38	文件夹
html-longdesc	2014/10/13 15:38	文件夹
html-media-capture	2014/10/13 15:38	文件夹
http	2014/10/13 15:38	文件夹
images	2014/10/13 15:38	文件夹



基于浏览器对象UAF漏洞挖掘技术

■ 实现一个Browser Fuzzer?

- 构建跨引擎dom树
- IE里处理js有两个引擎，jscript.dll和jscript9.dll,通过上述代码，使IE在处理特定的js脚本使使用较老的jscript.dll来解析，并将所有生成的dom节点返回到新的jscript9.dll引擎继续后续的fuzz。

```
f = document.getElementById("a_frame");
var m = f.contentDocument.createElement("meta");
m.setAttribute("http-equiv", "X-UA-Compatible");
m.setAttribute("content", "IE=8");
f.contentDocument.head.appendChild(m);
var s = f.contentDocument.createElement("script");
s.setAttribute("language", "JScript.Encode");
s.type = 'text/javascript';
s.async = true;
s.src = '%s';
s.onload = function(){frame_ret_nodes = f.contentDocument.all; fuzz_ret_nodes(frame_ret_nodes); };
f.contentWindow.document.getElementsByTagName('head')[0].appendChild(s);
```

小组主页: www.pku-exploit.com

联系 QQ: 513357938



北京大学

基于浏览器对象UAF漏洞挖掘技术

■ 实现一个Browser Fuzzer?

- 为Dom树节点添加事件处理回调。

- DomEventListener 字典:

"DOMContentLoaded", "msvisibilitychange", "abort", "activate", "afterprint" “...

```
switch (rand() % 3)
{
case 0: break;
case 1:
    sprintf(outbuffer, "try{ var observerObject = new MutationObserver(moEH);\nobserverObject.observe(newNode_%X, { child
fwrite(outbuffer, 1, strlen(outbuffer), tmp);
    break;
case 2:
    listenedEvent.push_front(domEventsList[rand() % 146]);
    sprintf(outbuffer, "try{ newNode_%X.addEventListener(\"%s\", evEH, %X);} catch(e){}\n", i, listenedEvent.front(), rand()
fwrite(outbuffer, 1, strlen(outbuffer), tmp);
    break;
}
```



基于浏览器对象UAF漏洞挖掘技术

■ 实现一个Browser Fuzzer?

- 将dom树随机一些节点生成elementRange,供后续fuzz使用
- 生成elementRange的作用在于它会保存一份节点的引用,若Range中的节点在回调中被销毁,则存在触发UAF的可能。
- 回调函数的构造: 回调函数是整个fuzz工具中最重要的一部分,是否能有效的打乱dom关系树,是否能有效的使元素释放重用以及fuzz工具的效率都和回调函数的构造相关,后面fuzz部分的功能都是在回调函数中实现的。

```
function callback() //must be stable!!
{
    ....

    case 0:
        sprintf(outbuf, "newNode_XX", rand() % max_element);
        fuzz(tmp, outbuf);
        break;

    case 1:
        tweak_execCommand_cty_explicit(tmp);
        sprintf(outbuf, "tweak attributes.cty_explicit(newNode_XX,0);\n", rand() % max_element);
        fwrite(outbuf, 1, strlen(outbuf), tmp);

        sprintf(outbuf, "try { document.execCommand('%s\\', true, %s);} catch(e){}\n", commands_a[rand() % cmdCount], intresting_str[rand() % 49]);
        fwrite(outbuf, 1, strlen(outbuf), tmp);
        memset(outbuf, 0, 0x1000);
        break;

    case 2:
        fuzz_ele_range(tmp);
        tweak_execCommand_cty_explicit(tmp);
        sprintf(outbuf, "newNode_XX", rand() % max_element);
        ....
}
```

```
try
{
    var r = document.createRange();
    startNode = newNode_XX;
    r.setStart(startNode, 0);
    endNode = newNode_XX;
    r.setEnd(endNode, 0);
    allRanges.push(r);
}
catch(e){}
```



基于FLASH对象UAF漏洞挖掘技术

■ FLASH对象中UAF

- FLASH 对象的 UAF 大多出在 domainMemory 和 共享内存上。
- ActionScript 3 为了提高swf的处理效率，新增了一个在主线程和工作线程共享对象的特性。
- 当主线程(main thread)和工作线程(worker thread)之间的共享对象引用出现不同步的情况，就可能出现UAF漏洞。





■ 基于FLASH对象UAF漏洞挖掘技术

■ 最近的典型案例（domainMemory 的 UAF）

■ CVE-2015-0311

- ByteArray 对象的 Uncompress 操作中未实现 domainMemory 中相应对象引用的修改。导致对象在解压过程中因意外导致执行失败释放空间后，domainMemory 中仍存在对 ByteArray 数据段的引用。

■ CVE-2015-0313

- 子线程 worker 中对于 ByteArray 的 Clear 操作并未通知 domainMemory，导致 domainMemory 中仍存在对已释放内存块的引用。



■ 基于FLASH对象UAF漏洞挖掘技术

■ 原因

■ ByteArray对象结构

- m_buffer指向实际数据对象
- m_subscribers指向引用该对象的数据链表

```
private:
    Toplevel* const      m_toplevel;
    MMgc::GC* const      m_gc;
    WeakSubscriberList   m_subscribers;
    MMgc::GCObject*      m_copyOnWriteOwner;
    uint32_t             m_position;
    FixedHeapRef<Buffer> m_buffer;
    bool                 m_isShareable;

public:
    bool                 m_isLinkWrapper;
```

- 当ByteArray进行拷贝构造时，若ByteArray为可共享的，则拷贝构造函数直接引用原来的数据对象m_buffer,但忽略了对m_subscribers成员的拷贝。从而导致某些函数在对拷贝对象的m_buffer数据进行free/alloc操作时，忽略了某些引用指针,造成UAF。



■ 基于FLASH对象UAF漏洞挖掘技术

■ Flash 线程同步机制

Communication technique	Dispatches event when receiving data	Shares memory between workers
Worker shared properties	No	No, objects are copies not references
MessageChannel	Yes	No, objects are copies not references
Shareable ByteArray	No	Yes, memory is shared

reference:http://help.adobe.com/en_US/as3/dev/WS2f73111e7a180bd0-5856a8af1390d64d08c-7ffe.html

■ Flash自身支持三种线程间共享方式

- Worker Shared properties
- MessageChanel
- Shareable ByteArray

■ 其中只有Shareable ByteArray 是以共享内存的形式在线程间同步的。



■ 基于FLASH对象UAF漏洞挖掘技术

■ FLASH UAF 挖掘

■ 在多线程ActionScript3代码里设置共享对象

Passing data with a shared property

The most basic way to share data between workers is to use a shared property. Each worker maintains an internal dictionary of shared property values. the Worker object's `setSharedProperty()` method with two arguments, the key name and the value to store:

```
// code running in the parent worker  
bgWorker.setSharedProperty("sharedPropertyName", someObject);
```

Once the shared property has been set, the value can be read by calling the Worker object's `getSharedProperty()` method, passing in the key name:

```
// code running in the background worker  
receivedProperty = Worker.current.getSharedProperty("sharedPropertyName");
```

◆ http://help.adobe.com/en_US/as3/dev/WS2f73111e7a180bd0-5856a8af1390d64d08c-7ffe.html

小组主页: www.pku-exploit.com

联系 QQ: 513357938



北京大學

■ 基于FLASH对象UAF漏洞挖掘技术

■ 操作共享对象

```
function demo_fuzz(obj:Object):void
{
    //here you can operate on the object any way you want.
    obj.clear();
    obj = null;
    obj.someproperty = "what ever you want"
    .....
}
shareobj =
Worker.current.getSharedProperty("sharedPropertyNa
me");
demo_fuzz(shareobj);
```



添加事件回调

■ 添加事件回调

◆ 在页面对象上设置事件回调

回调类型

NETWORK_CHANGE, PASTE, REMOVED, RENDER,
LOCATION_CHANGE, HTML_DOM_INITIALIZE,
HTML_BOUNDS_CHANGE

example:

```
SharedObj.addEventListener(Event.ADDED, fuzzfunc);  
SharedObj.addEventListener(Event.ADDED_TO_STAGE, fuzzfunc);  
SharedObj.addEventListener(Event.REMOVED, fuzzfunc);  
SharedObj.addEventListener(Event.REMOVED_FROM_STAGE, fuzzfunc);  
.....
```



北京大学软件安全研究小组简介

■ 小组概况

■ 北京大学软件安全研究小组隶属于北京大学网络与软件安全保障教育部重点实验室、北京大学软件与微电子学院和北京大学软件研究所信息安全实验室，成员均为北京大学软件与微电子学院信息安全系和信息安全实验室的研究生，小组指导老师由北京大学软件与微电子学院、中国科学院软件研究所及北京邮电大学的专家和老教师组成，主要研究方向包括逆向工程、软件安全漏洞挖掘、软件安全漏洞分析、恶意代码及软件安全评估，目前已发表了软件安全研究方向论文四十余篇，并开发了多个用于软件安全研究实践的辅助工具。

■ 小组主页：<http://www.pku-exploit.com/>

小组主页：www.pku-exploit.com

联系 QQ: 513357938



北京大學

漏洞发现成果

■ 操作系统漏洞

◆ 发现Windows操作系统漏洞5个

➤ 已公开漏洞3个；

- MS11-010 (CVE编号: CVE2011-0030)
- MS11-054 (CVE编号: CVE2011-1886)

➤ 未公开漏洞3个，其中

- 本地拒绝服务2个
- 本地权限提升1个



漏洞发现情况

■ 其它漏洞发现

◆ 应用软件安全漏洞30个

➤ 已公开漏洞

- 某安全产品K驱动中1个任意内核地址写入漏洞，9个拒绝服务攻击漏洞；
- 某安全产品Q驱动中6个任意内核地址写入漏洞；
- 某安全产品T安全沙箱驱动6个任意内核地址写入漏洞。

➤ 未公开漏洞

- 网络电话软件漏洞1个；
- 其他应用软件漏洞7个。





谢谢!

小组主页: www.pku-exploit.com

联系 QQ: 513357938



北京大学