[Pwn2Own 2016] Flash Player最新安全特性分析及绕过思路

文选@腾讯电脑管家网络攻防小组

About Us

4月7日

腾讯电脑管家发现 Adobe flash 漏洞 *14* 个

Tencent PC Manager working with Trend Micro's ZDI (CVE-2016-1018)

willj of Tencent PC Manager (CVE-2016-1020, CVE-2016-1021, CVE-2016-1022, CVE-2016-1023, CVE-2016-1024, CVE-2016-1025, CVE-2016-1026, CVE-2016-1027, CVE-2016-1028, CVE-2016-1029, CVE-2016-1031, CVE-2016-1032, CVE-2016-1033)

5月5日

腾讯电脑管家发现 Adobe Reader 漏洞 *6* 个

kelvinwang of Tencent PC Manager (CVE-2016-1081, CVE-2016-1082, CVE-2016-1083, CVE-2016-1084, CVE-2016-1085, CVE-2016-1086)

5月12日

腾讯电脑管家发现 Adobe Flash 漏洞 *7* 个

willJ of Tencent PC Manager (CVE-2016-4109, CVE-2016-4110, CVE-2016-4111, CVE-2016-4112, CVE-2016-4113, CVE-2016-4114, CVE-2016-4115)

6月16日

腾讯电脑管家发现 Adobe Flash 漏洞 *12* 个

willJ of Tencent PC Manager (CVE-2016-4122, CVE-2016-4123, CVE-2016-4124, CVE-2016-4125, CVE-2016-4127, CVE-2016-4128, CVE-2016-4129, CVE-2016-4130, CVE-2016-4131, CVE-2016-4134, CVE-2016-4166)

kelvinwang of Tencent PC Manager (CVE-2016-4133)

5月

腾讯电脑管家发现 微软 漏洞 *2* 个

CVE-2016-0174, Liang Yin of Tencent PC Manager working with Trend Micro's Zero Day Initiative (ZDI)

CVE-2016-0175, Liang Yin of Tencent PC Manager working with Trend Micro's Zero Day Initiative (ZDI)

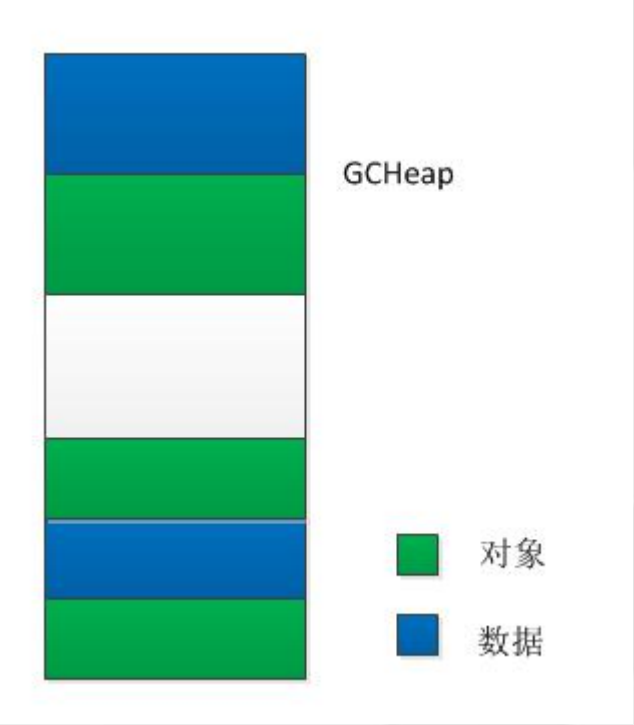
大纲

- **早期Flash特性及攻击思路**
  - 不安全的内存设计
  - 脆弱的object
  - 攻击流程
- **Flash新的安全特性**
  - 隔离堆
  - object加固
  - CFG
- **Flash内存管理**
  - GCHeap内存结构
  - 内存分配器结构
    FiexedMalloc/GCAlloc/GCLargeAlloc
- **新的利用思路**
  - 隔离堆绕过
  - 内存不连续问题
  - 绕过CFG

早期Flash特性 – 不安全的内存设计

GCHeap

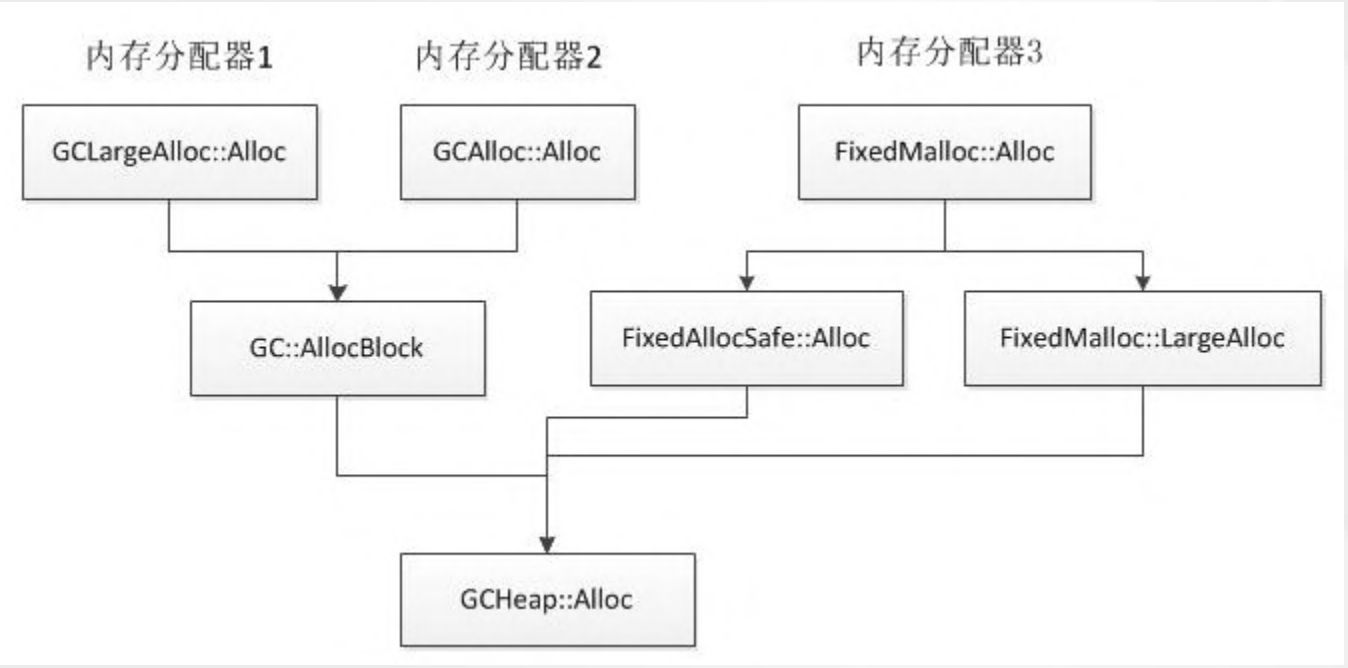
- GCHeap负责ActionScript中的内存分配
- 所有的对象和数据都在一个GCHeap空间
- 容易实现堆喷，内存布局

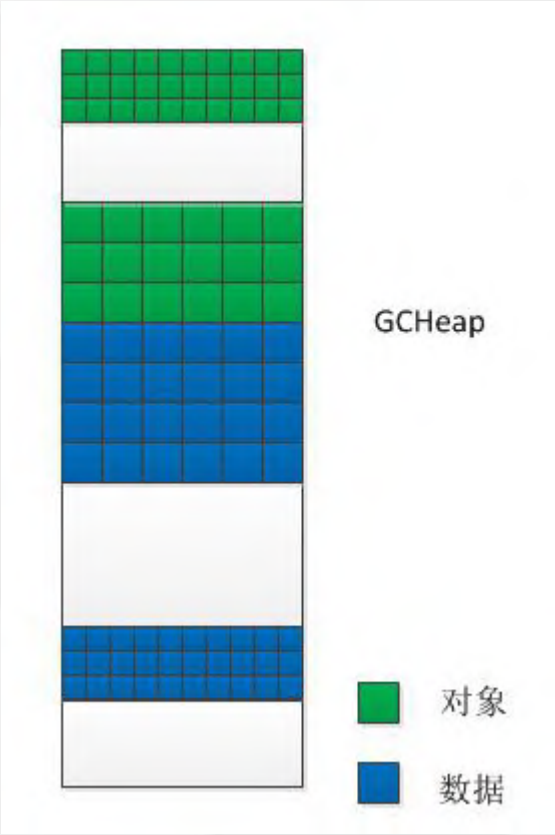


早期Flash特性 – 不安全的内存设计

3个内存分配器

–相近大小的对象可以连续，内存布局操作容易





早期Flash特性 – 脆弱的object

Vector/Array/ByteArray/...

关键数据（如长度、数据缓冲区指针）无校验、没有异地备份，无法防止被修改

```
0:011> dd 042bcfb0 l28/4;$$ Vector.<int> object
042bcfb0  038ce390 00000002 03e21da8 0423ce80
042bcfc0  042b2150 00000000 04336000 00000000
042bcfd0  00000000 00000000
0:011> dd 04336000 ; $$ data of Vector.<int> object
04336000  000004e1 03db2000 55667000 55667001
04336010  55667002 00000000 55667004 55667005
04336020  55667006 55667007 55667008 55667009
04336030  5566700a 5566700b 5566700c 5566700d
04336040  5566700e 5566700f 55667010 55667011
04336050  55667012 55667013 55667014 55667015
04336060  55667016 55667017 55667018 55667019
04336070  5566701a 5566701b 5566701c 5566701d
```

早期Flash特性 – 脆弱的object

Vector.<uint>

Length

n		Data[0]	Data[1]	...	Data[n]

漏洞触发前

```
0:007> dd 0228d000
0228d000  000003f0 014f3000 55667000 55667001
0228d010  55667002 55667003 55667004 55667005
0228d020  55667006 55667007 55667008 55667009
0228d030  5566700a 5566700b 5566700c 5566700d
0228d040  5566700e 5566700f 55667010 55667011
0228d050  55667012 55667013 55667014 55667015
0228d060  55667016 55667017 55667018 55667019
0228d070  5566701a 5566701b 5566701c 5566701d
```

漏洞触发后

```
0:007> dd 0228d000
0228d000  400003f0 014f3000 55667000 55667001
0228d010  55667002 55667003 55667004 55667005
0228d020  55667006 55667007 55667008 55667009
0228d030  5566700a 5566700b 5566700c 5566700d
0228d040  5566700e 5566700f 55667010 55667011
0228d050  55667012 55667013 55667014 55667015
0228d060  55667016 55667017 55667018 55667019
0228d070  5566701a 5566701b 5566701c 5566701d
```

早期Flash 攻击流程

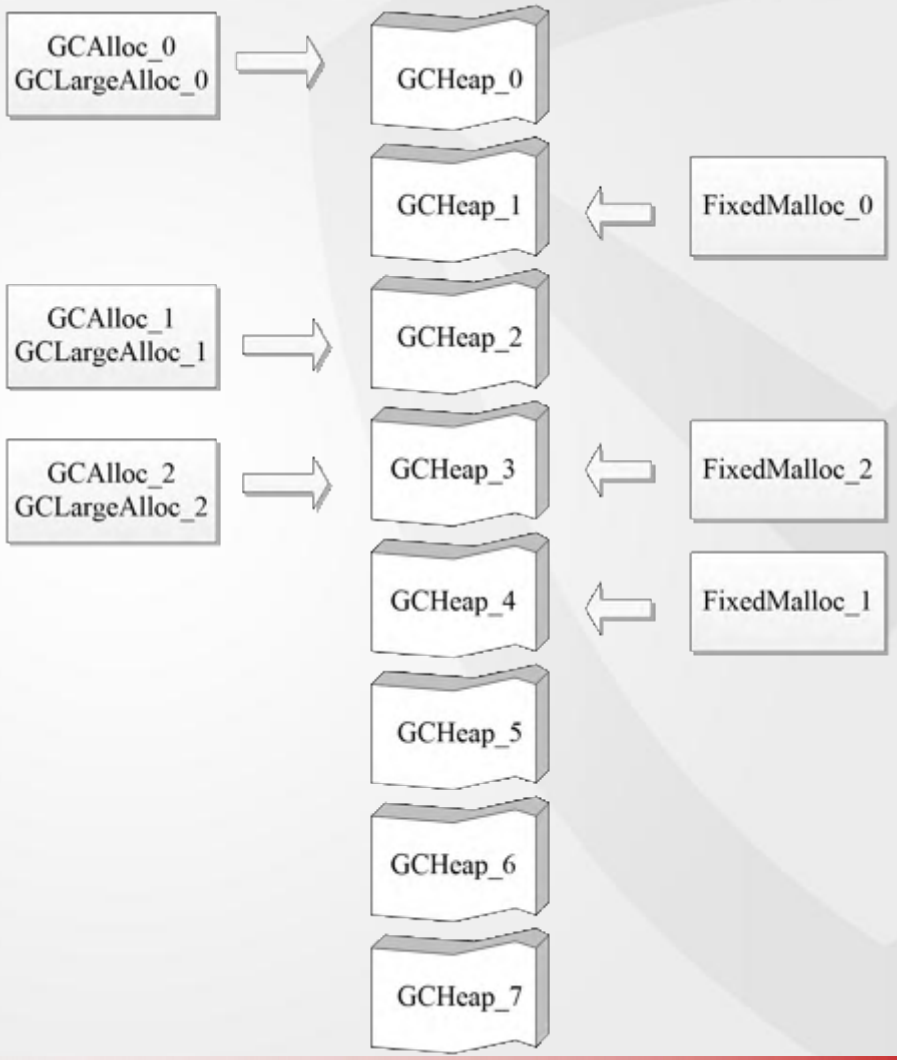

FLASH
None for you!
你丫弱爆了

转折点

新的安全特性 - 隔离堆

对象隔离

- 堆的数目1---> 8
- 分配器数目3--->9



新的安全特性 - 隔离堆

堆地址随机化

–不同堆的地址随机且相差很大

–同一堆中内存块也可能不连续

```
0:040> $$>< "C:\wdb_scripts\print_GCHeaps.txt"
GCHeap_0:
    000002e3e50b0000----000002e3e80b0000
GCHeap_1:
    000003e50de80000----000003e50fe80000
GCHeap_2:
    000003cd48ba0000----000003cd49ba0000
GCHeap_3:
    0000052bc7690000----0000052bc8690000
GCHeap_4:
    0000026727cb0000----000002672ccb0000
    0000021cf3800000----0000021cf4800000
    0000033eaf320000----0000033eb0320000
    000005ebd1310000----000005ebd7310000
GCHeap_5:
    000003f9d10b0000----000003f9d20b0000
GCHeap_6:
    00000560e2db0000----00000560e3db0000
GCHeap_7:
    0000023921030000----0000023922030000
0:040> ?000003e50de80000-000002e3e50b0000
Evaluate expression: 1104492167168 = 00000101`28dd0000
0:040> ?0000026727cb0000-0000021cf3800000
Evaluate expression: 318704910336 = 0000004a`344b0000
```

新的安全特性 - object加固

Security Cookie

- 用于异或加密object中的关键数据
- 针对漏洞利用中的有长度的对象
- 在GCHeap初始化时生成
- 存放在flash.ocx的.data区中

```
DWORD CreateSecurityCookie(){
    HCRYPTPROV Rnd;
    DWORD SecurityCookie = 0;
    if(CryptAcquireContext(&Rnd, NULL, NULL, PROV_RSA_FULL,
    CRYPT_VERIFYCONTEXT)) {
        if(CryptGenRandom(Rnd, 4, (BYTE*)(&SecurityCookie)))
            CryptReleaseContext(Rnd,0);
    }
    if(!SecurityCookie)
        DebugBreak();
    return SecurityCookie;
}
```

新的安全特性 - object加固

ByteArray

```
class Buffer : public FixedHeapRCObject
{
public:
    virtual void destroy();
    virtual ~Buffer();
    uint8_t* array;
    uint32_t capacity;
    uint32_t length;
};
```

```
class Buffer : public FixedHeapRCObject
{
public:
    virtual void destroy();
    virtual ~Buffer();
    uint8_t* array;
    uint32_t capacity;
    uint32_t length;
    uint32_t array_xored;
    uint32_t capacity_xored;
    uint32_t length_xored;
};
```

```
0:014> dd 03336040 160/4; $$ ByteArray
03336040  10d197d0 00000002 0328dba0 0331b388
03336050  03336058 00000044 10d19768 10d19770
03336060  10d19764 10d5e934 048984c0 033ab000
03336070  03707b50 00000000 00000000 00000000
03336080  10d2bc8c 033a9200 00000000 00000000
03336090  10d1975c 00000003 00000000 00000000
0:014> dd 033a9200 128/4; $$ ByteArray.m_buffer
033a9200  10d19750 00000001 037c5000 00001000
033a9210  00000004 00000000 a48ddad4 a7f19ad4
033a9220  a7f18ad0 a7f18ad4
0:014> dd Flash32_20_0_0_306+102acdc 11; $$ the secret cookie(lenXor)
1102acdc  a7f18ad4
0:014> ?037c5000 ^ a7f18ad4; $$ 计算缓存区地址异或后的值
Evaluate expression: -1534207276 = a48ddad4
0:014> db 037c5000 14; $$ 缓存区的内容
037c5000  11 22 33 44                                      ."3D
```

新的安全特性 - object加固

Vector

- vector object中新增了一个length字段
- vector data中的length存放的是异或后的值

```
0:014> dd 032b3130 130/4; $$ Vector.<int>
032b3130  10d29dc0 00000002 048e6430 032ad430
032b3140  03262150 00000000 037bd000 033ab000
032b3150  000004e1 00000000 00000000 00000000
0:014> dd 037bd000 ; $$ data of Vector.<int>
037bd000  a7f18e35 55667000 55667001 55667002
037bd010  00000000 55667004 55667005 55667006
037bd020  55667007 55667008 55667009 5566700a
037bd030  5566700b 5566700c 5566700d 5566700e
037bd040  5566700f 55667010 55667011 55667012
037bd050  55667013 55667014 55667015 55667016
037bd060  55667017 55667018 55667019 5566701a
037bd070  5566701b 5566701c 5566701d 5566701e
0:014> ?000004e1  ^ a7f18ad4; $$ 计算length异或后的值
Evaluate expression: -1477341643 = a7f18e35
```

新的安全特性 - CFG(Control Flow Guard)

间接调用前检查地址是否合法

```
mov     [rsp+arg_10], rbx
mov     [rsp+arg_18], rsi
push    rdi
sub     rsp, 30h
mov     rax, cs:__security_cookie
xor     rax, rsp
mov     [rsp+38h+var_10], rax
mov     rax, [rcx]
mov     rdi, rcx
mov     rsi, rdx
mov     rbx, [rax+8]
mov     rcx, rbx        ; Parameter
test    rdx, rdx
jnz     short loc_100335B8A
mov     [rsp+38h+var_18], 1
call    cs:__guard_check_icall_fptr
lea     r8d, [rsi+1]
mov     rcx, rdi
lea     rdx, [rsp+38h+var_18]
call    rbx
jmp     short loc_100335BB0
```

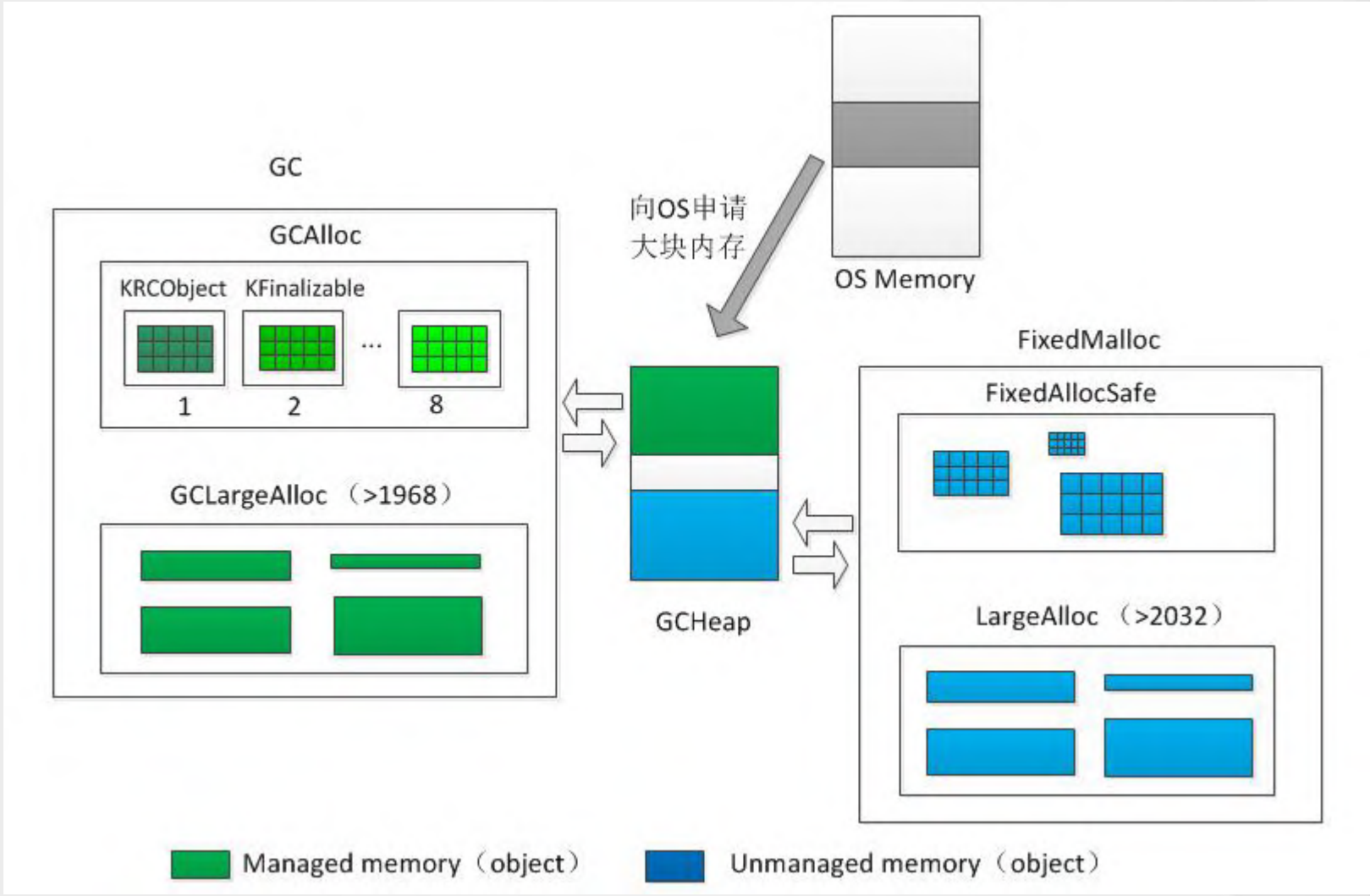
我们的几套方案没了！！！

Flash内存管理 -GCHeap 内存分类

三类内存

Memory type	特点	存储的内容
Managed memory	GC（Garbage Collect）管理，不需显示释放	Managed object（主要是ScriptObjects）
Unmanaged memory	Flash管理、需要显示释放	Unmanaged object（如Bitmap Data、Video Data、Sound Data...）
System memory	GCHeap负责分配释放	GCHeap memory的信息（如块大小、块使用情况等）

Flash内存管理 - GCHeap内存结构



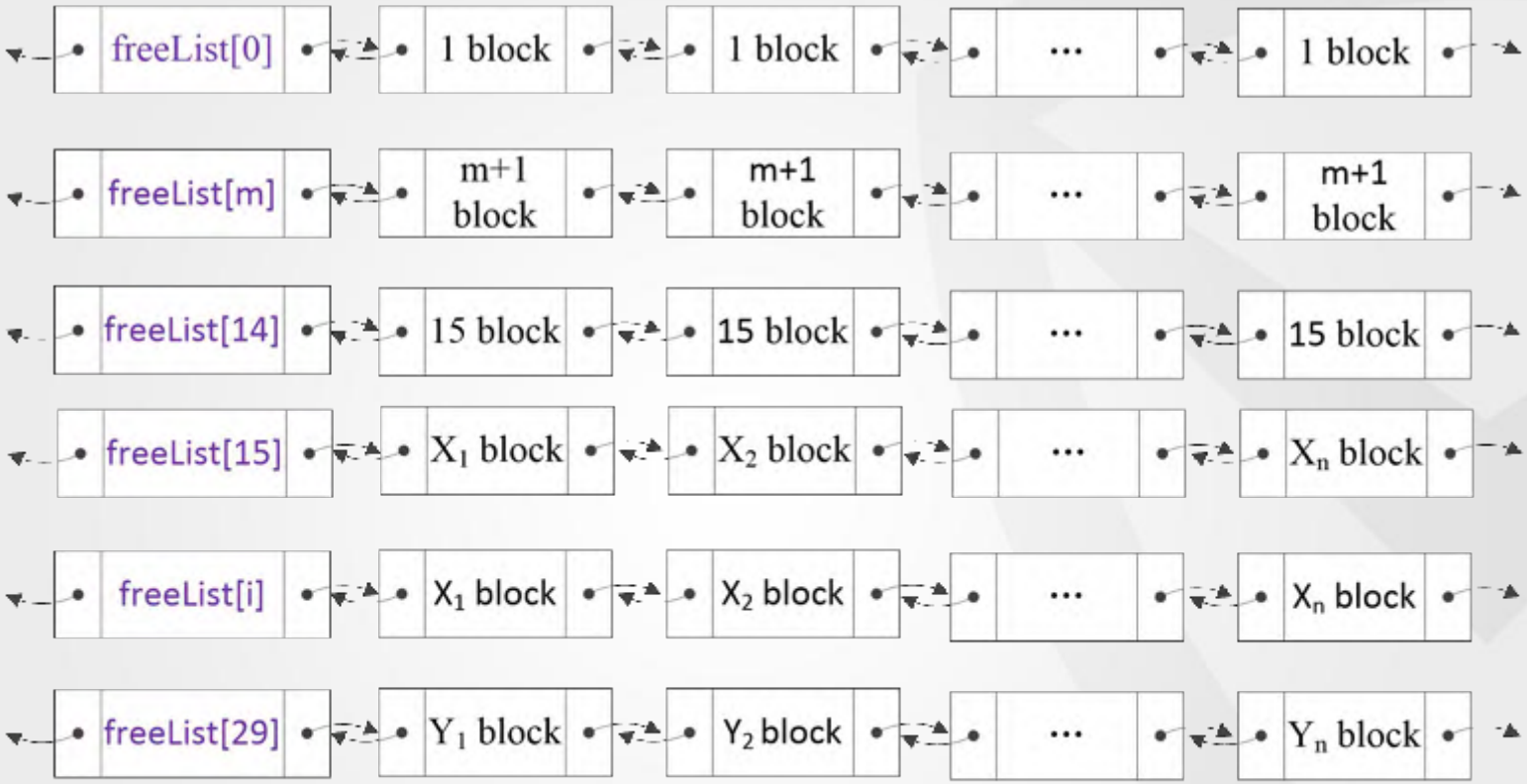
Flash内存管理 - GCHeap关键数据结构

```
class GCHeap
{
    Region * freeRegion;
    Region * nextRegion;
    Region * lastRegion;
    HeapBlock * blocks;
    HeapBlock freeLists[30];
}
```

```
class Region
{
    Region *prev;
    char *baseAddr;
    char *reserveTop;
    char *commitTop;
    size_t blockId;
};
```

```
class HeapBlock
{
    char *baseAddr;   // base address of block's memory
    size_t size;          // size of this block
    size_t sizePrevious; // size of previous block
    HeapBlock *prev;       // prev entry on free list
    HeapBlock *next;       // next entry on free list
 }
```

Flash内存管理 -FreeList



freeList[i] , 15 <= i <= 28:

$X_1 <= X_2 <= X_n$, $16 + (i - 15) * 8 <= X_p < 16 + (i - 15 + 1) * 8$, $1 <= p <= n$

freeList[29]:

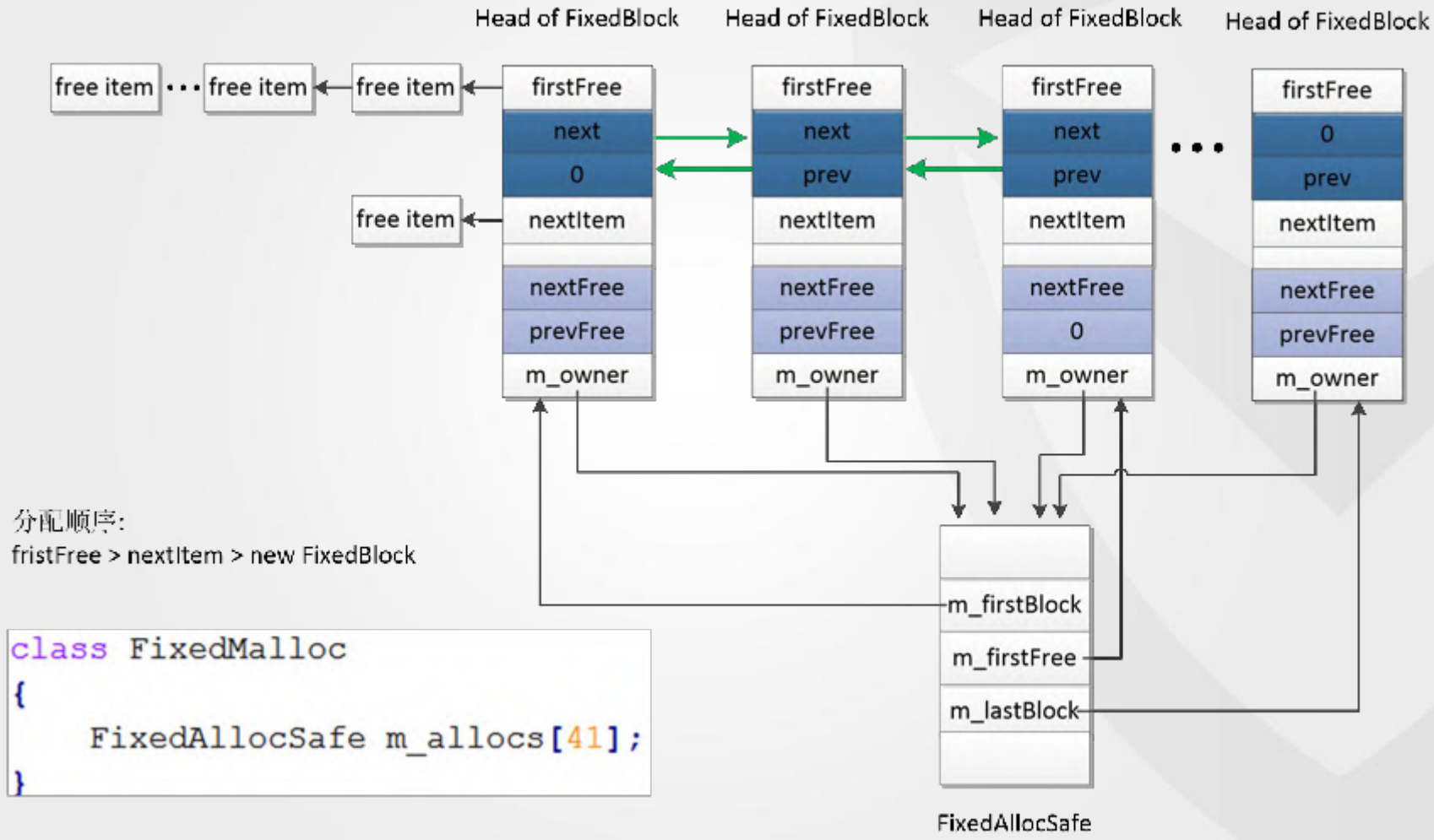
$Y_1 <= Y_2 <= Y_n$, $Y_p >= 128$, $1 <= p <= n$

Flash内存管理 - GCHeap分配顺序

GCHeap分配HeapBlock内存的顺序

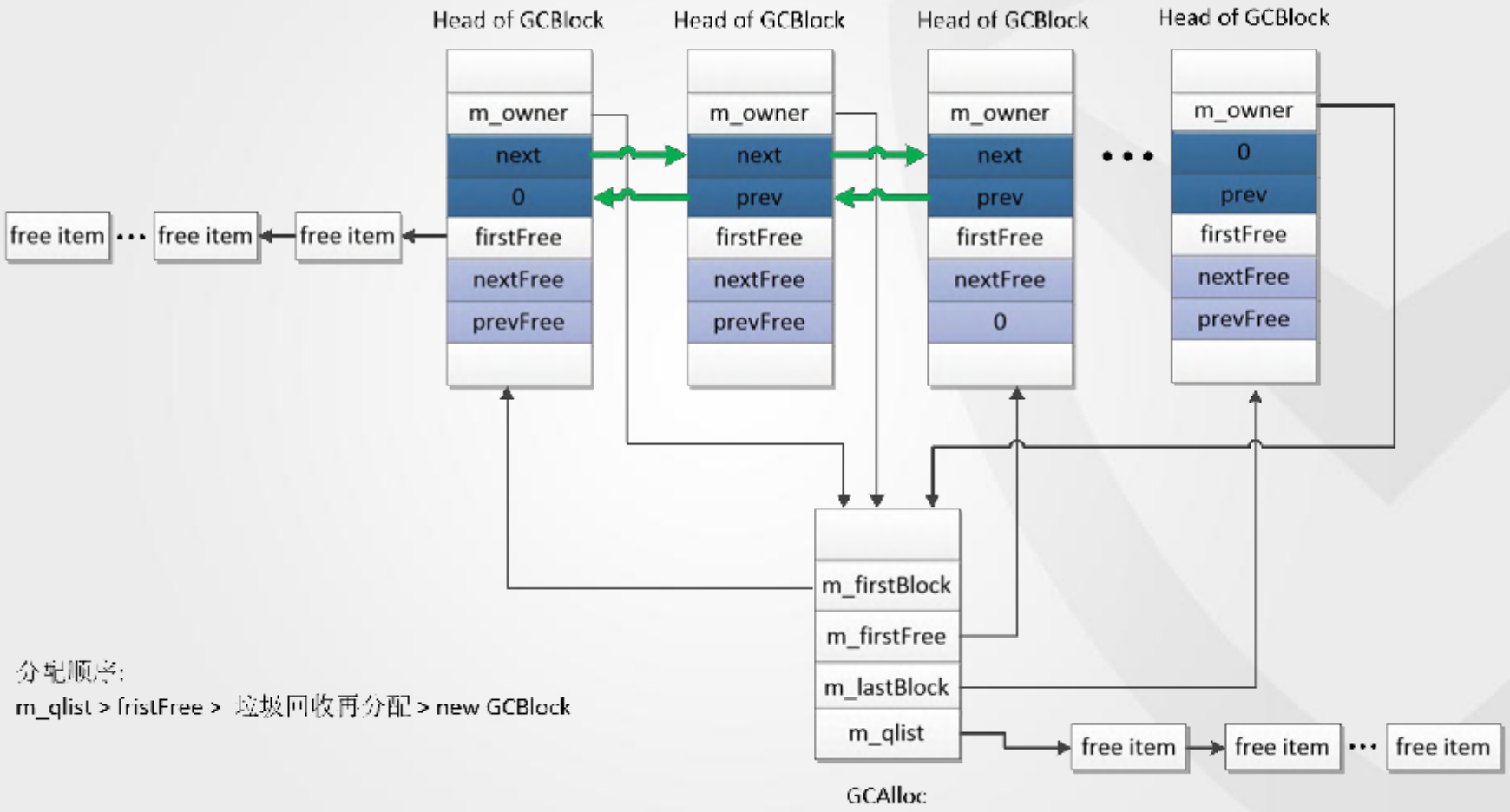
1. freeList[30]
2. Region内存中uncommitted内存
3. 申请一块新的Region内存

Flash内存管理 - 内存分配器之FixedMalloc

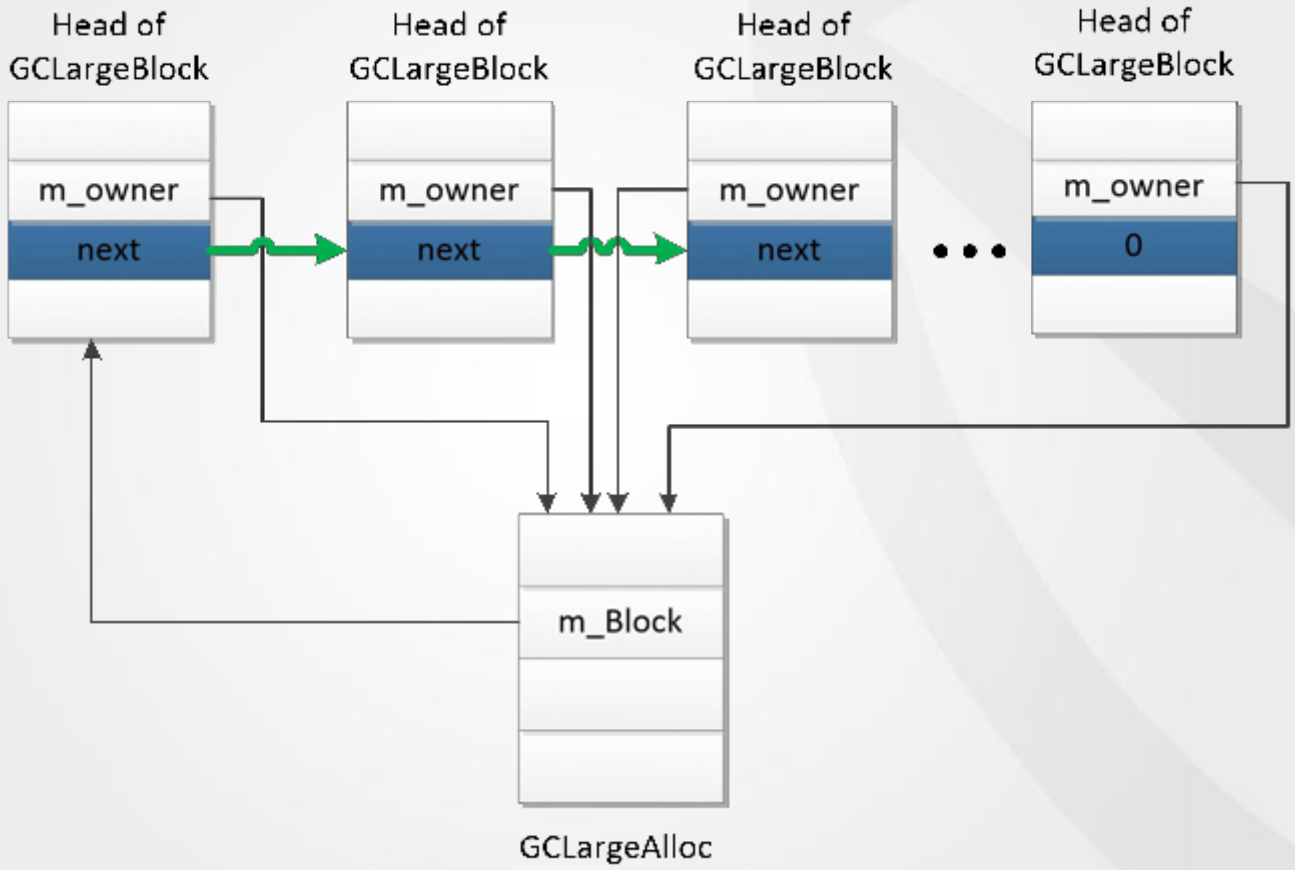



```
class FixedMalloc
{
    FixedAllocSafe m_allocs[41];
}
```

Flash内存管理 - 内存分配器之GCAlloc



Flash内存管理 -内存分配器之 GCLargeAlloc



新的利用思路 - 隔离堆绕过

修改FixedBlock，控制firstFree指针。

```
0:040> dq 000003e50ef17000 ; $$ FixedBlock in GCHeap_1
000003e5`0ef17000  000003e5`0ef17468 000003e5`0ef174a0
000003e5`0ef17010  00000000`00000000 000003e5`0ee83000
000003e5`0ef17020  00000000`00380013 00000000`00000000
000003e5`0ef17030  000003e5`0ee08000 00007ffd`80c07f60
000003e5`0ef17040  00007ffd`80607fa0 00000000`00000008
000003e5`0ef17050  000003e5`0ef17078 000003e5`0ee83fc8
000003e5`0ef17060  000003e5`0deaafe8 00000000`00000011
000003e5`0ef17070  00000000`0000fdf8 00007ffd`80607fa0
```

修改FixedBlock头部的firstFree为GCHeap_3中的某一地址

```
0:040> dq 000003e50ef17000
000003e5`0ef17000  0000052b`c76a0000 000003e5`0ef174a0
000003e5`0ef17010  00000000`00000000 000003e5`0ee83000
000003e5`0ef17020  00000000`00380013 00000000`00000000
000003e5`0ef17030  000003e5`0ee08000 00007ffd`80c07f60
000003e5`0ef17040  00007ffd`80607fa0 00000000`00000008
000003e5`0ef17050  000003e5`0ef17078 000003e5`0ee83fc8
000003e5`0ef17060  000003e5`0deaafe8 00000000`00000011
000003e5`0ef17070  00000000`0000fdf8 00007ffd`80607fa0
```

新的利用思路 - 隔离堆绕过

使用同一类型进行内存布局

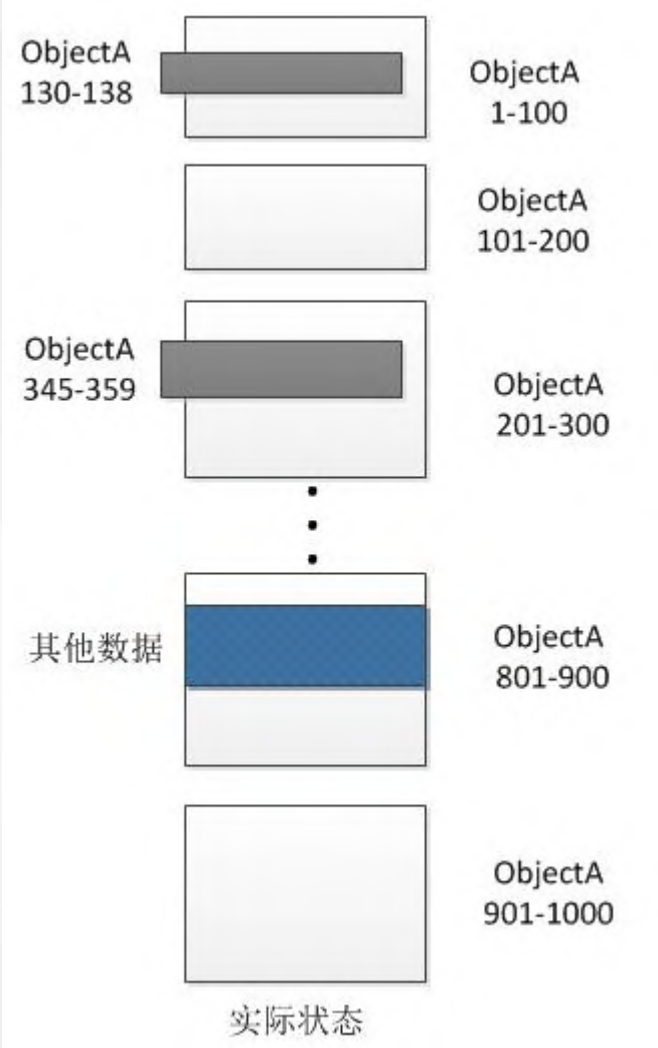
所在位置	数据 类型
GCHeap_0	ScriptObject（String，Array，自定义对象，etc）
GCHeap_1	buffer object of ByteArray
GCHeap_2	data of Array，data of Vector.<Object>
GCHeap_3	data of Vector.<int>，data of ByteArray
GCHeap_4	data of String，data of loader，data of BitmapData

新的利用思路 - 内存不连续问题

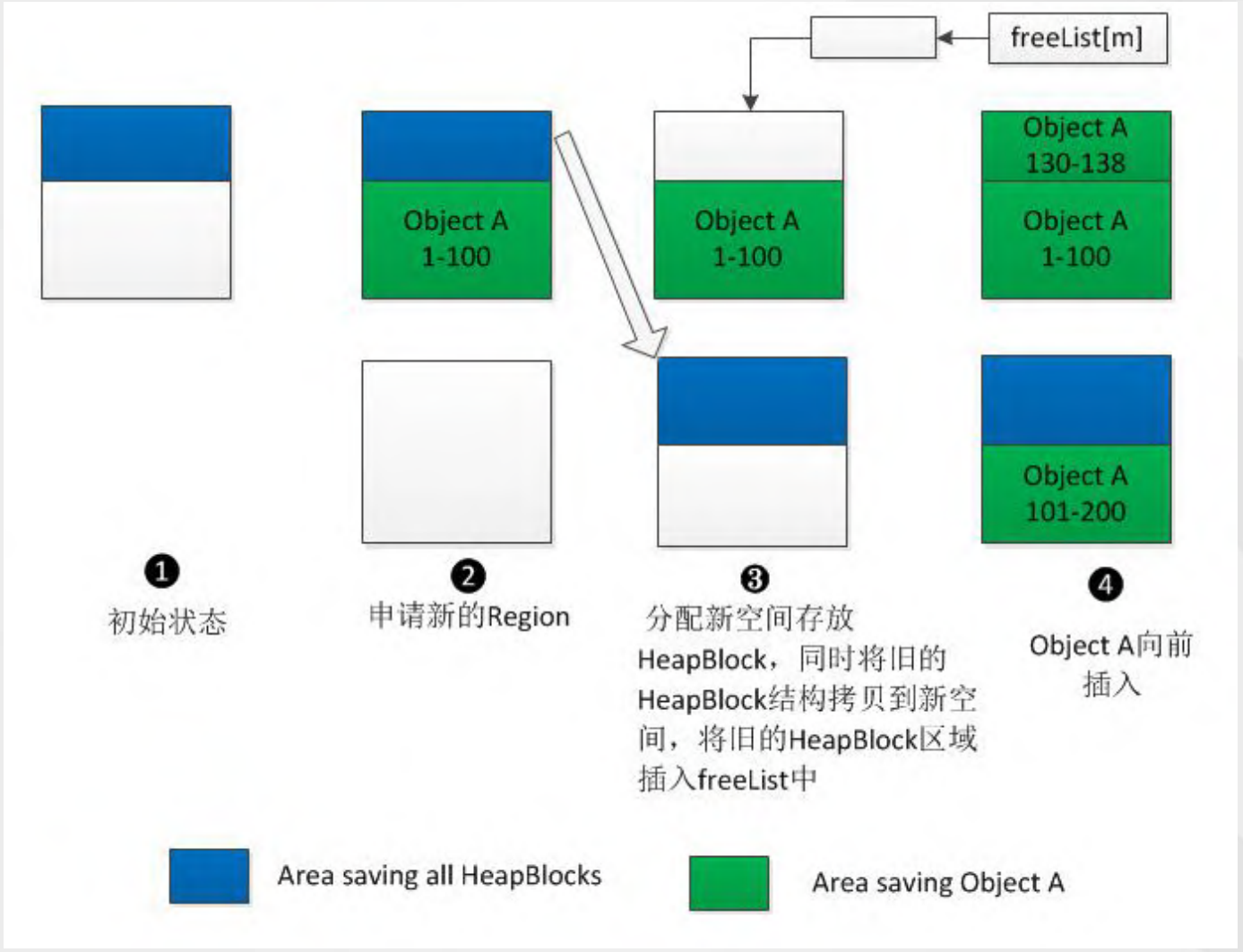
```
for(i = 0; i < 1000; i++)
{
    ObjectAArray[i] = new ObjectA();
}
```

问题：

- 有部分后分配的ObjectA会插入到前面的ObjectA的内存中，？？
- 某个ObjectA的内存区域中会有一块其他数据，使得该内存中的ObjectA不连续，？？



新的利用思路 - 内存不连续问题



新的利用思路 - 内存不连续问题

解决方法

1. 分配2000个Object A
2. 释放，HeapBlock Area足够大
3. 再分配1000个Object A。

新的利用思路 - 绕过CFG

寻找没有CFG保护的间接调用

如部分JIT动态生成的代码

替换CFG中的Guard Check Function

参考 Zhang Yunhai，ByPass Control Flow Guard Comprehensively

覆盖栈上的返回地址

由于有GS，SAFESEH，SEHOP机制，需精准控制

3秒攻破Adobe Flash Player

Pwn2Own：腾讯安全联队3秒攻破Flash

2016-03-17 09:15　it168网站原创　作者: 网络　编辑: 姜惠田　0条评

【IT168 资讯】北京时间17日凌晨，中国队Pwn2Own世界黑客大赛再传喜讯，腾讯安全联队的Sniper战队以3秒的神速完全“碾轧”Adobe Flash。短短3秒，不仅成功攻破该插件，而且实现系统级访问，拿到该项目全额积分13分。

Q&A

谢 谢