Inside Flash: Flash Exploit Detection Uncovered

Ga1ois & Bo Qu

About us

- Security Researchers in PANW
- Work
  - IPS
  - APT Detection
- After work
  - Vulnerability discovery
  - Exploit technique researching

Agenda

- Identify (Possible) Exploit
- Stop Exploit
- Detect Exploit

Part 1 : Identify Possible Exploit

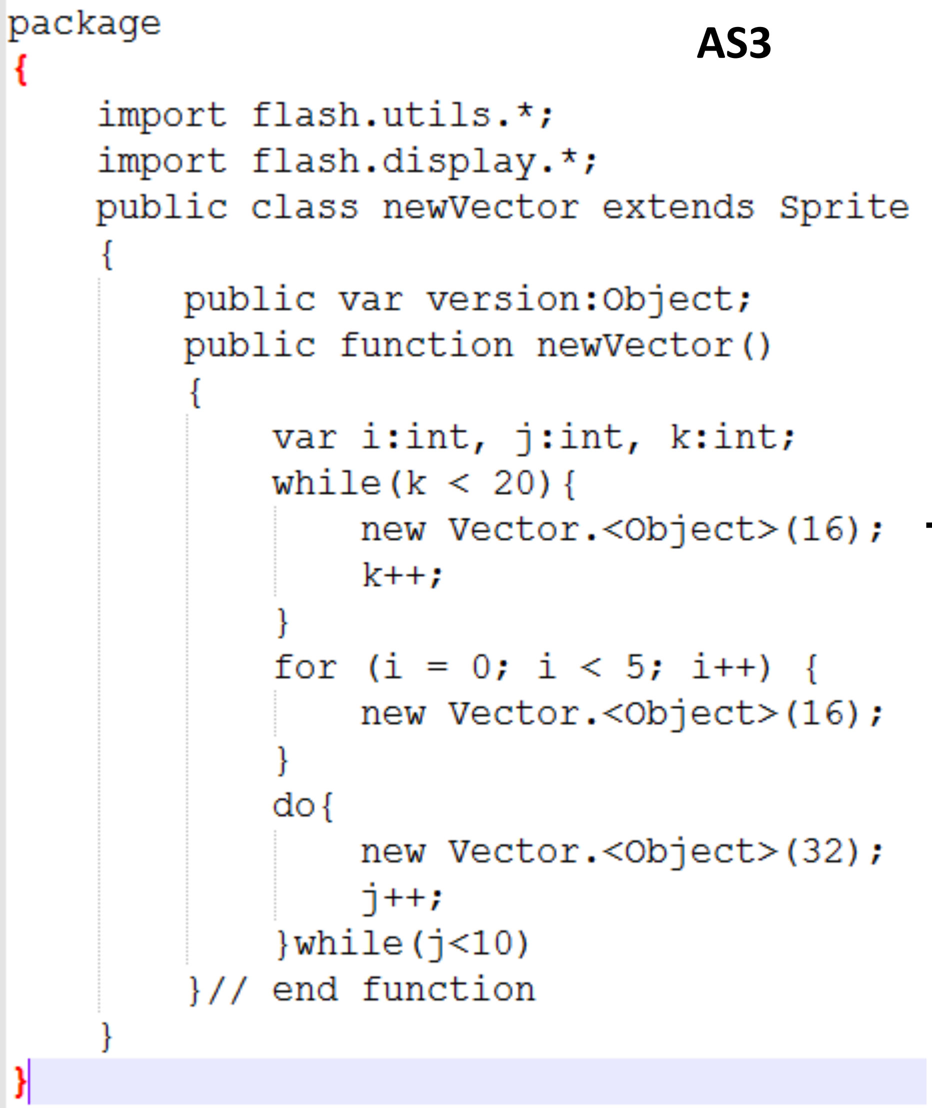
Find vector in loop using static detection

Find opcode pattern of loop

- Compile tools
  - As3compile.exe
  - Asc.jar– two embeded abc file needed
  - Mxmlc in flexsdk
  - Flash Builder
  - Flash CS* professional
- Command line decompile tools
  - Swfdump.exe in swftools
  - Swfdump.py in mecheye-fusion
  - Swfdump.jar/swfdump.exe in flex – we use it
- 3 types of loop
  - For
  - While
  - Do/while

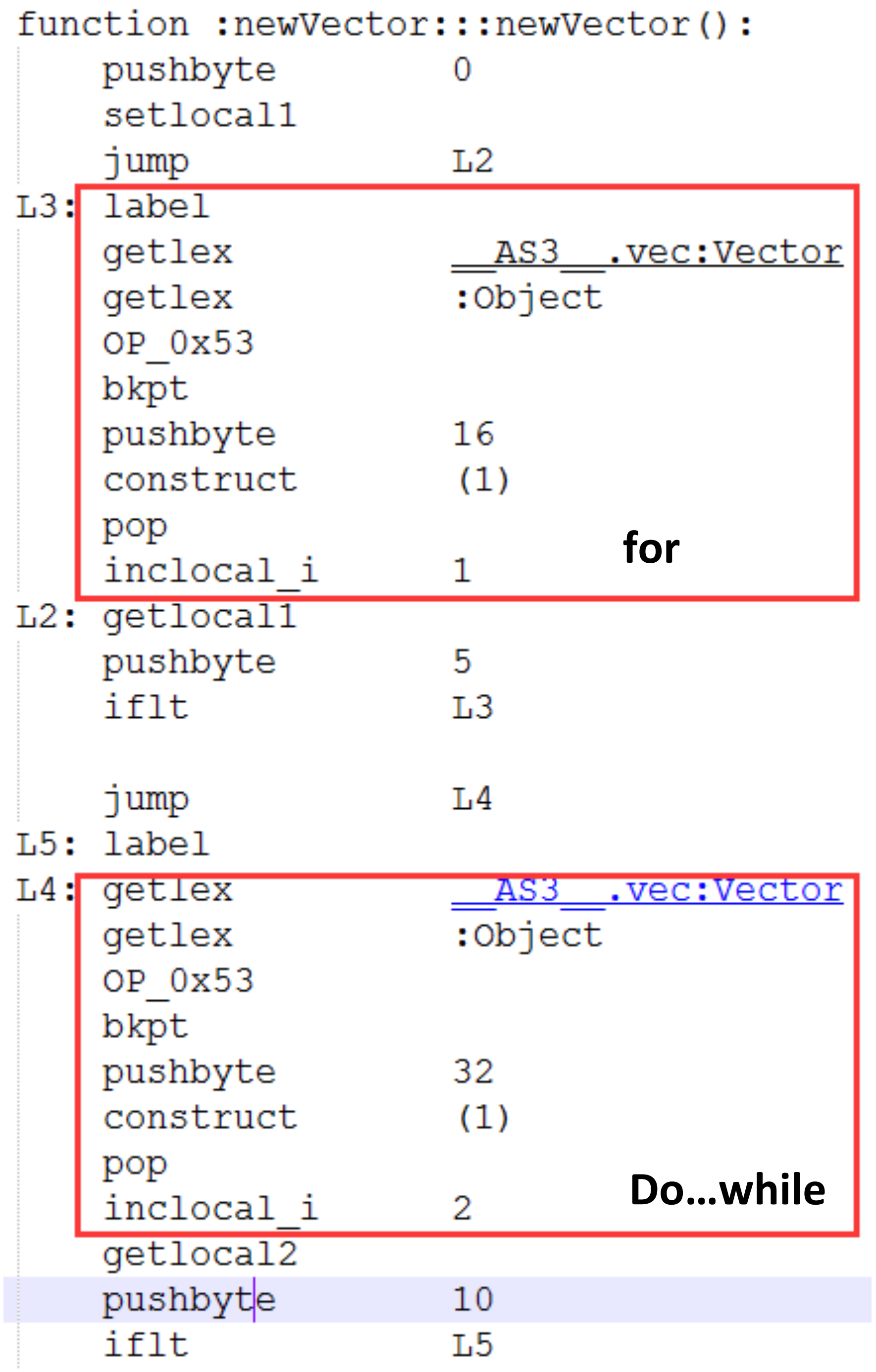
Simplest situation

AS3



```
package
{
    import flash.utils.*;
    import flash.display.*;
    public class newVector extends Sprite
    {
        public var version:Object;
        public function newVector()
        {
            var i:int, j:int, k:int;
            while(k < 20){
                new Vector.<Object>(16);
                k++;
            }
            for (i = 0; i < 5; i++) {
                new Vector.<Object>(16);
            }
            do{
                new Vector.<Object>(32);
                j++;
            }while(j<10)
        }// end function
    }
}
```

MXMLC → **SWF** → **Swfdump**

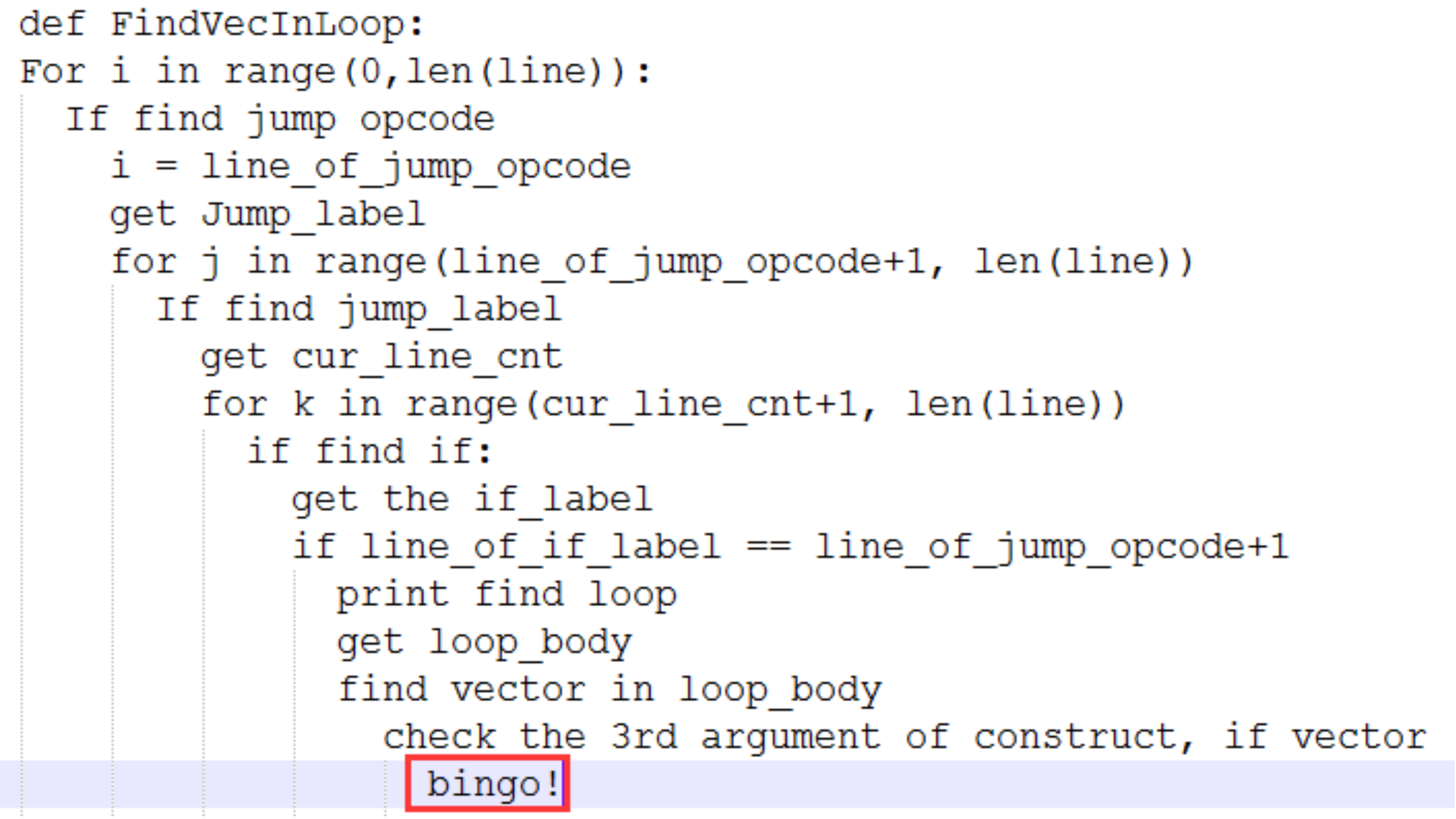


```
function :newVector::newVector():
    pushbyte        0
    setlocal1
    jump            L2
L3: label
    getlex          __AS3__.vec:Vector
    getlex          :Object
    OP_0x53
    bkpt
    pushbyte        16
    construct       (1)
    pop
    inclocal_i      1
L2: getlocal1
    pushbyte        5
    iflt            L3

    jump            L4
L5: label
L4: getlex          __AS3__.vec:Vector
    getlex          :Object
    OP_0x53
    bkpt
    pushbyte        32
    construct       (1)
    pop
    inclocal_i      2
    getlocal2
    pushbyte        10
    iflt            L5
```

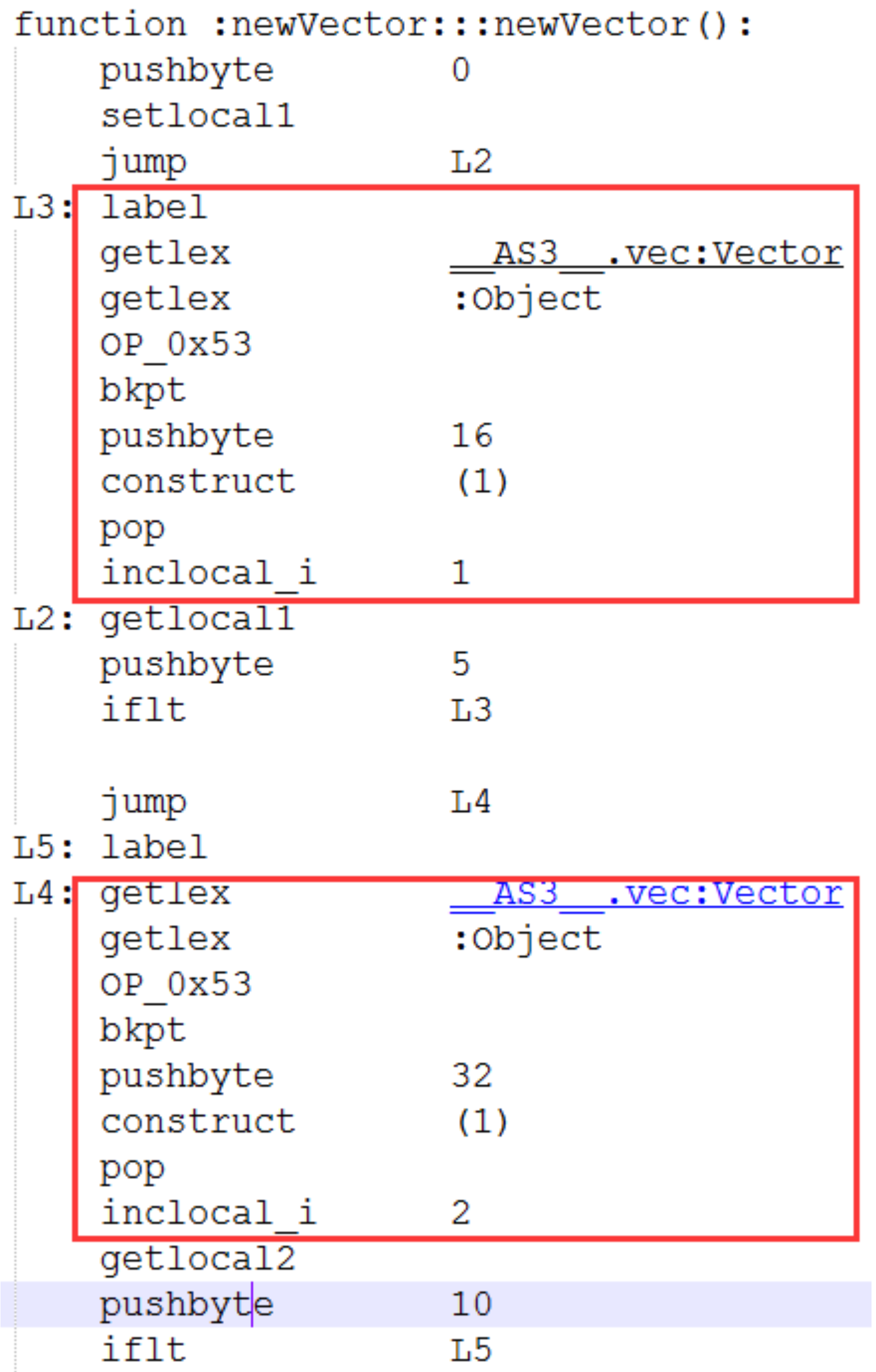
for

Do…while

Algorithm



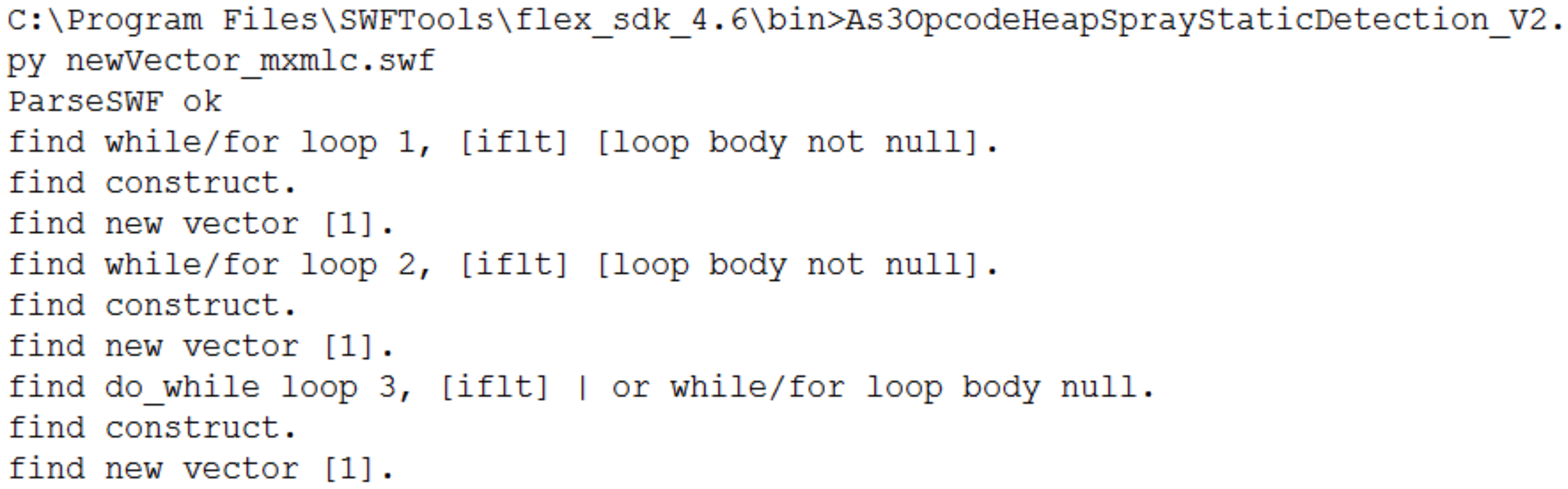
```
def FindVecInLoop:
For i in range(0,len(line)):
  If find jump opcode
    i = line_of_jump_opcode
    get Jump_label
    for j in range(line_of_jump_opcode+1, len(line))
      If find jump_label
        get cur_line_cnt
        for k in range(cur_line_cnt+1, len(line))
          if find if:
            get the if_label
            if line_of_if_label == line_of_jump_opcode+1
              print find loop
              get loop_body
              find vector in loop_body
                check the 3rd argument of construct, if vector
                  bingo!
```



```
function :newVector::newVector():
    pushbyte        0
    setlocal1
    jump            L2
L3: label
    getlex          __AS3__.vec:Vector
    getlex          :Object
    OP_0x53
    bkpt
    pushbyte        16
    construct       (1)
    pop
    inclocal_i      1
L2: getlocal1
    pushbyte        5
    iflt            L3

    jump            L4
L5: label
L4: getlex          __AS3__.vec:Vector
    getlex          :Object
    OP_0x53
    bkpt
    pushbyte        32
    construct       (1)
    pop
    inclocal_i      2
    getlocal2
    pushbyte        10
    iflt            L5
```

Find opcode pattern of loop

- demo



```
C:\Program Files\SWFTools\flex_sdk_4.6\bin>As3OpcodeHeapSprayStaticDetection_V2.
py newVector_mxmlc.swf
ParseSWF ok
find while/for loop 1, [iflt] [loop body not null].
find construct.
find new vector [1].
find while/for loop 2, [iflt] [loop body not null].
find construct.
find new vector [1].
find do_while loop 3, [iflt] | or while/for loop body null.
find construct.
find new vector [1].
```

Limitation and solution

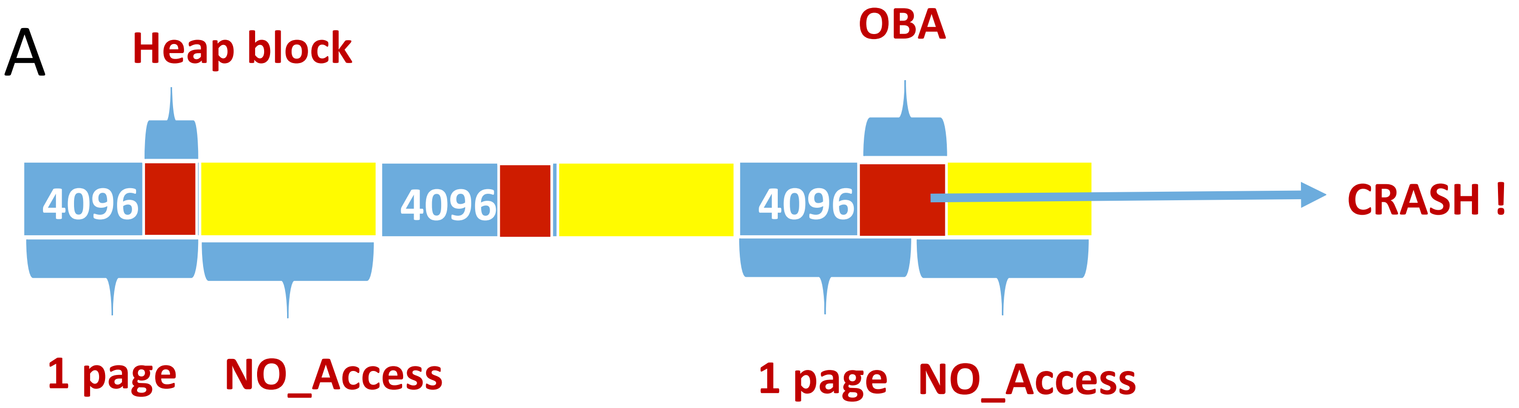
- Bad news
  - Loadbytes for obfuscation
  - Not use loop[jmp or goto or repeat one statement for many times]
  - Function calls in loop body
- Good news
  - Hook and generate inner real exploit SWF
  - The pattern itself can be detected
  - Check deeper

Part 2 : Stop Exploit

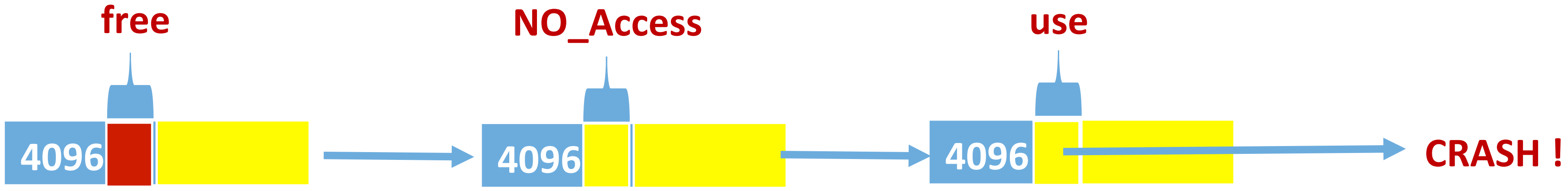
A Lightweight PageHeap for FixedMalloc in Flash

Page heap on windows process heap

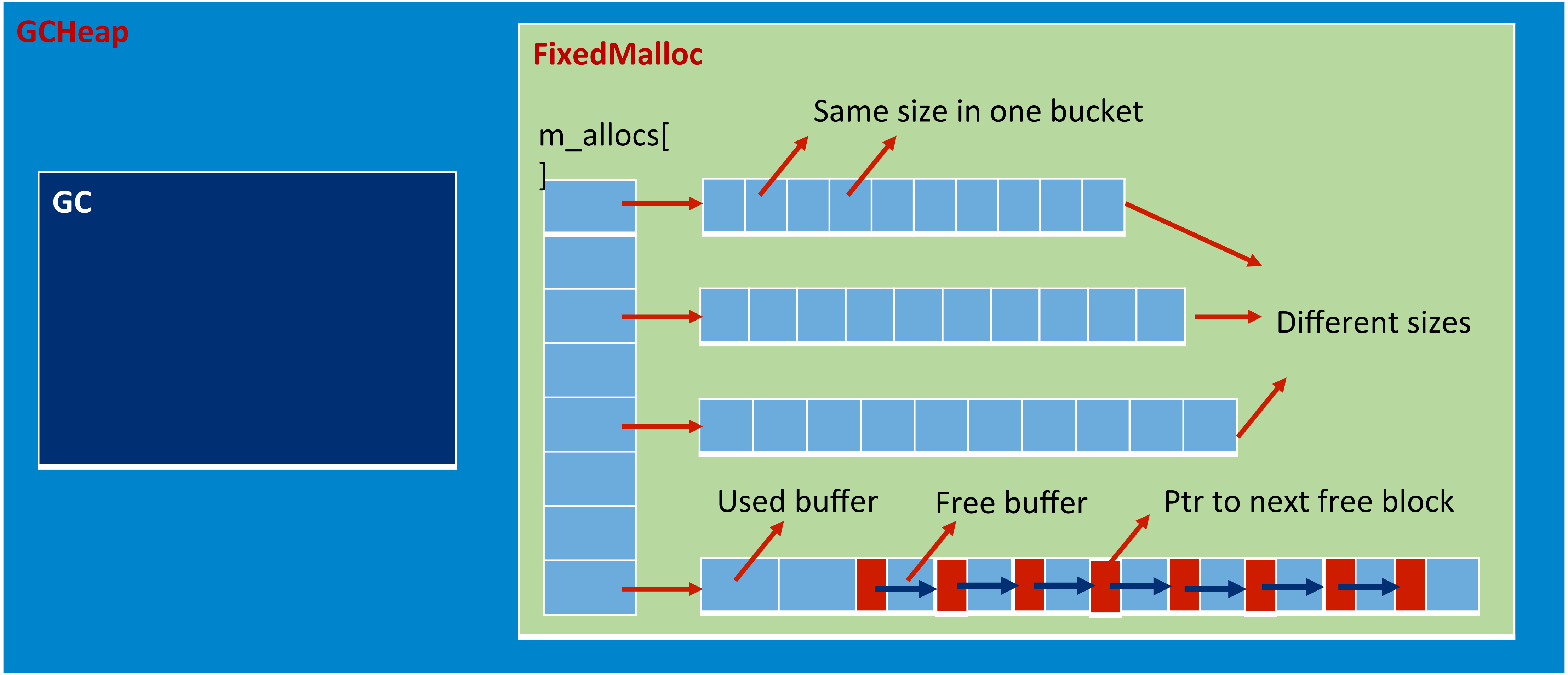
- A diagnostic option that can detect OBA(Out of Bounds Access) and UAF(Use After Free) bugs
- OBA



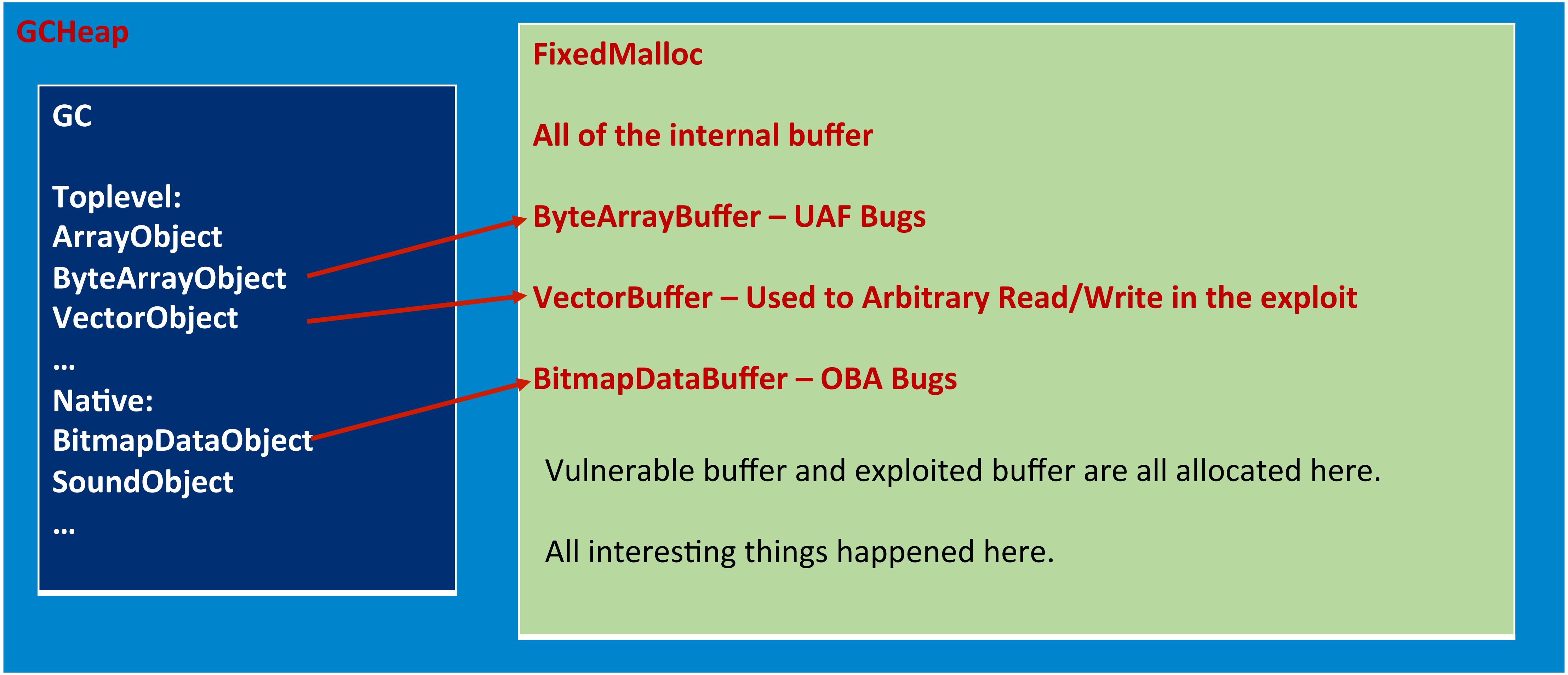
- UAF



Custom Heap in Flash MMgc



Custom Heap in Flash MMgc



GCHeap

GC

Toplevel:
ArrayObject
ByteArrayObject
VectorObject
…
Native:
BitmapDataObject
SoundObject
…

FixedMalloc

All of the internal buffer

ByteArrayBuffer – UAF Bugs

VectorBuffer – Used to Arbitrary Read/Write in the exploit

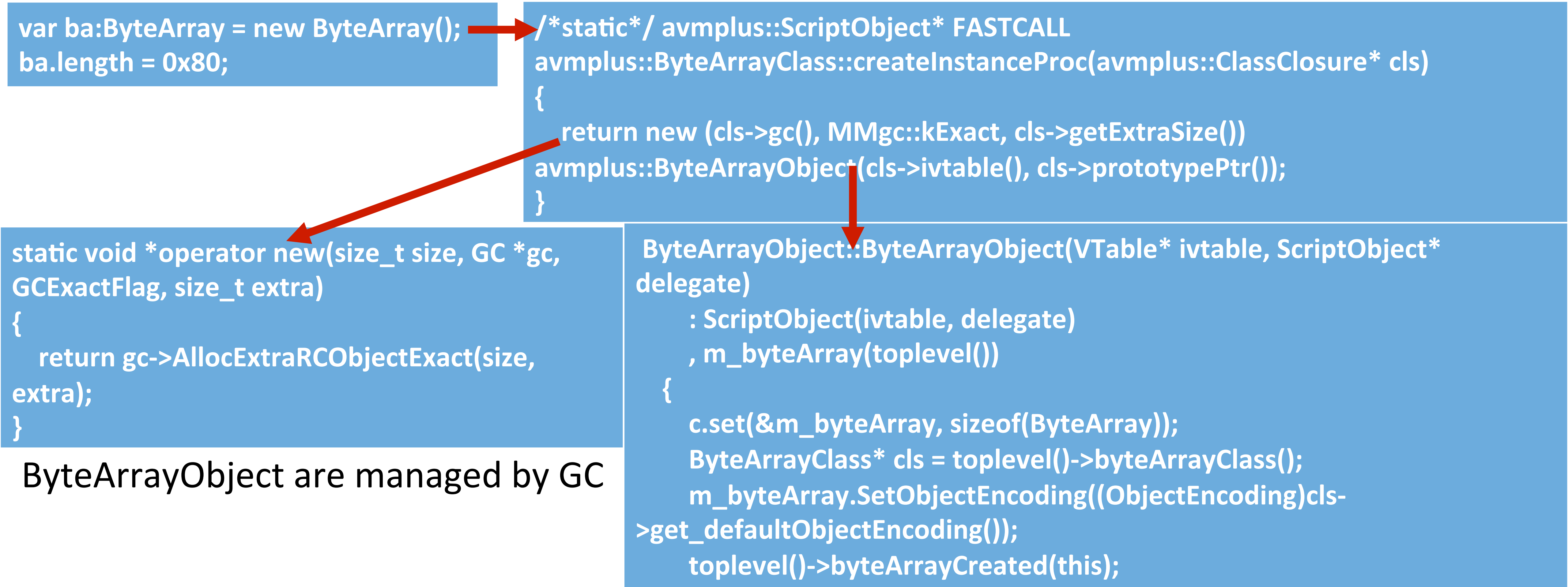
BitmapDataBuffer – OBA Bugs

Vulnerable buffer and exploited buffer are all allocated here.

All interesting things happened here.

From AS3 To Memory

Take ByteArray As An Example



```
var ba:ByteArray = new ByteArray();
ba.length = 0x80;
```

```
/*static*/ avmplus::ScriptObject* FASTCALL
avmplus::ByteArrayClass::createInstanceProc(avmplus::ClassClosure* cls)
{
   return new (cls->gc(), MMgc::kExact, cls->getExtraSize())
avmplus::ByteArrayObject(cls->ivtable(), cls->prototypePtr());
}
```

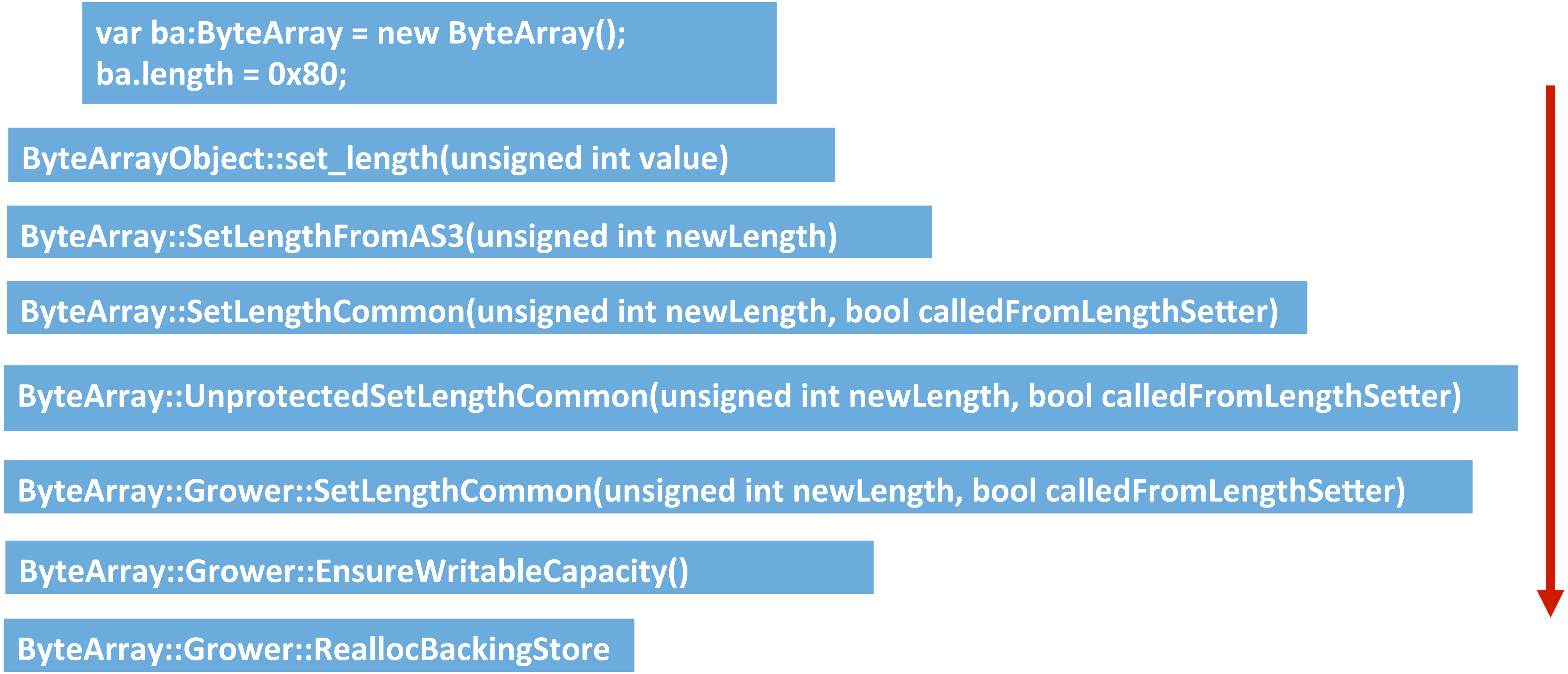
```
static void *operator new(size_t size, GC *gc,
GCExactFlag, size_t extra)
{
   return gc->AllocExtraRCObjectExact(size,
extra);
}
```

ByteArrayObject are managed by GC

```
 ByteArrayObject::ByteArrayObject(VTable* ivtable, ScriptObject*
delegate)
        : ScriptObject(ivtable, delegate)
        , m_byteArray(toplevel())
    {
        c.set(&m_byteArray, sizeof(ByteArray));
        ByteArrayClass* cls = toplevel()->byteArrayClass();
        m_byteArray.SetObjectEncoding((ObjectEncoding)cls-
>get_defaultObjectEncoding());
        toplevel()->byteArrayCreated(this);
```

From AS3 To Memory

Take ByteArray As An Example



```
var ba:ByteArray = new ByteArray();
ba.length = 0x80;
```

```
ByteArrayObject::set_length(unsigned int value)
ByteArray::SetLengthFromAS3(unsigned int newLength)
ByteArray::SetLengthCommon(unsigned int newLength, bool calledFromLengthSetter)
ByteArray::UnprotectedSetLengthCommon(unsigned int newLength, bool calledFromLengthSetter)
ByteArray::Grower::SetLengthCommon(unsigned int newLength, bool calledFromLengthSetter)
ByteArray::Grower::EnsureWritableCapacity()
ByteArray::Grower::ReallocBackingStore
```

From AS3 To Memory

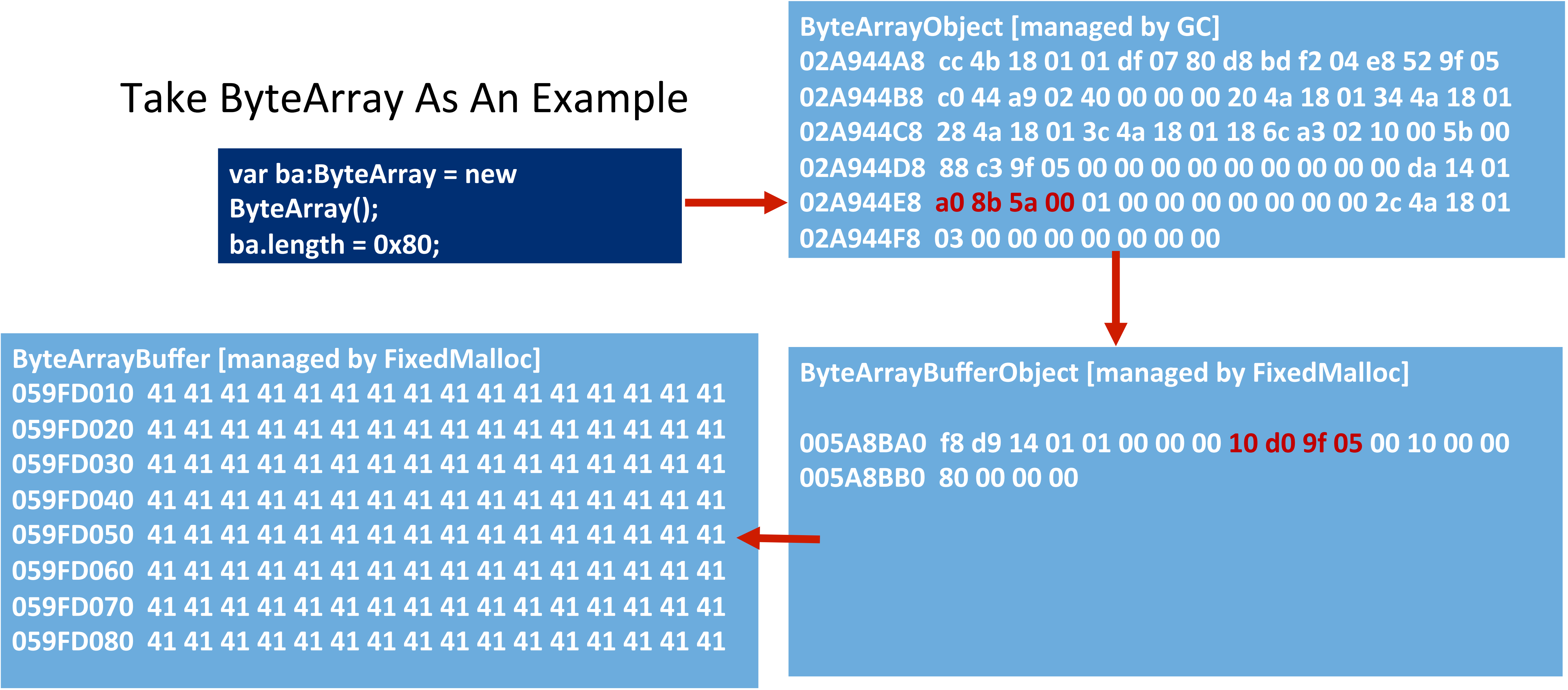
```
void FASTCALL ByteArray::Grower::ReallocBackingStore(uint32_t newCapacity)
{
        …
        m_oldArray = m_owner->m_buffer->array;
        m_oldLength = m_owner->m_buffer->length;
        m_oldCapacity = m_owner->m_buffer->capacity;
        uint8_t* newArray = mmfx_new_array_opt(uint8_t, newCapacity, MMgc::kCanFail);
        …
        m_owner->TellGcNewBufferMemory(newArray, newCapacity);
        if (m_oldArray){
                VMPI_memcpy(newArray, m_oldArray, min(newCapacity, m_oldLength));
                if (newCapacity > m_oldLength)
                        VMPI_memset(newArray+m_oldLength, 0, newCapacity-m_oldLength);
        }else{
                VMPI_memset(newArray, 0, newCapacity);
        }
        m_owner->m_buffer->array = newArray;
        m_owner->m_buffer->capacity = newCapacity;
        …
}
```

Mmfx_ is a series Macro in FixedMalloc

ByteArrayDataBuffer is managed by FixedMalloc

From AS3 To Memory

Take ByteArray As An Example



```
var ba:ByteArray = new
ByteArray();
ba.length = 0x80;
```

ByteArrayObject [managed by GC]

```
02A944A8  cc 4b 18 01 01 df 07 80 d8 bd f2 04 e8 52 9f 05
02A944B8  c0 44 a9 02 40 00 00 00 20 4a 18 01 34 4a 18 01
02A944C8  28 4a 18 01 3c 4a 18 01 18 6c a3 02 10 00 5b 00
02A944D8  88 c3 9f 05 00 00 00 00 00 00 00 00 00 da 14 01
02A944E8  a0 8b 5a 00 01 00 00 00 00 00 00 00 2c 4a 18 01
02A944F8  03 00 00 00 00 00 00 00
```

ByteArrayBufferObject [managed by FixedMalloc]

```
005A8BA0  f8 d9 14 01 01 00 00 00 10 d0 9f 05 00 10 00 00
005A8BB0  80 00 00 00
```

ByteArrayBuffer [managed by FixedMalloc]

```
059FD010  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
059FD020  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
059FD030  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
059FD040  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
059FD050  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
059FD060  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
059FD070  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
059FD080  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
```

A Lightweight Page Heap For FixedMalloc

- Change all of Heap Allocators in FixedMalloc to HeapAlloc in ProcessHeap
- Turn On Page Heap on Windows Process Heap

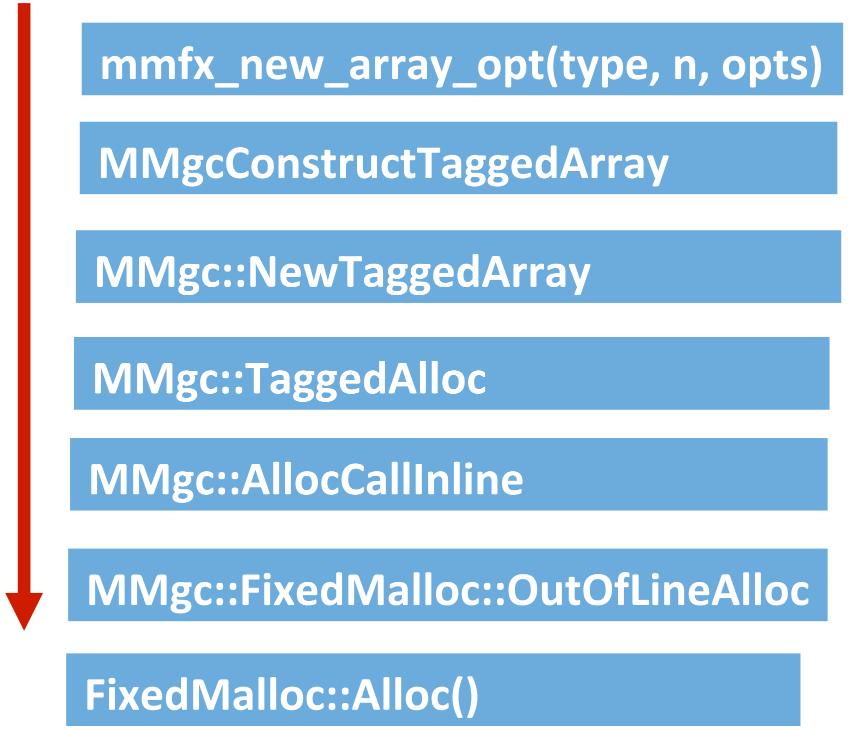
Heap Allocators in FixedMalloc/FixedAlloc

AllocationMaros.h in Avmplus/MMgc

```
// Used for allocating/deallocating memory with MMgc's fixed allocator.
// The memory allocated using these macros will be released when the MMgc aborts due to
// an unrecoverable out of memory situation.
#define mmfx_new(new_data)                new (MMgc::kUseFixedMalloc) new_data
#define mmfx_new0(new_data)               new (MMgc::kUseFixedMalloc, MMgc::kZero) new_data

#define mmfx_new_array(type, n)           ::MMgcConstructTaggedArray((type*)NULL, n, MMgc::kNone)

#define mmfx_new_opt(new_data, opts)      new (MMgc::kUseFixedMalloc, opts) new_data
#define mmfx_new_array_opt(type, n, opts)   ::MMgcConstructTaggedArray((type*)NULL, n, opts)

#define mmfx_delete(p)                    ::MMgcDestructTaggedScalarChecked(p)
#define mmfx_delete_array(p)              ::MMgcDestructTaggedArrayChecked(p)

#define mmfx_alloc(_siz)                  MMgc::AllocCall(_siz)
#define mmfx_alloc_opt(_siz, opts)        MMgc::AllocCall(_siz, opts)
#define mmfx_free(_ptr)                   MMgc::DeleteCall(_ptr)
```

Heap Allocators in FixedMalloc

- Take mmfx_new_array_opt as an example



mmfx_new_array_opt(type, n, opts)

MMgcConstructTaggedArray

MMgc::NewTaggedArray

MMgc::TaggedAlloc

MMgc::AllocCallInline

MMgc::FixedMalloc::OutOfLineAlloc

FixedMalloc::Alloc()

```
REALLY_INLINE void* FixedMalloc::Alloc(size_t size, FixedMallocOpts flags)
{
        if (size <= (size_t)kLargestAlloc)
                return FindAllocatorForSize(size)->Alloc(size, flags);
        else
                return LargeAlloc(size, flags);
}
```

```
REALLY_INLINE FixedAllocSafe* FixedMalloc::FindAllocatorForSize(size_t size)
{
        unsigned const index = (size <= 4) ? 0 : kSizeClassIndex[((size+7)>>3)];
        GCAssert(size <= m_allocs[index].GetItemSize());
        GCAssert(index == 0 || size > m_allocs[index-1].GetItemSize());
        return &m_allocs[index];
}
```

Heap Allocators in FixedMalloc

- Take mmfx_new_array_opt as an example

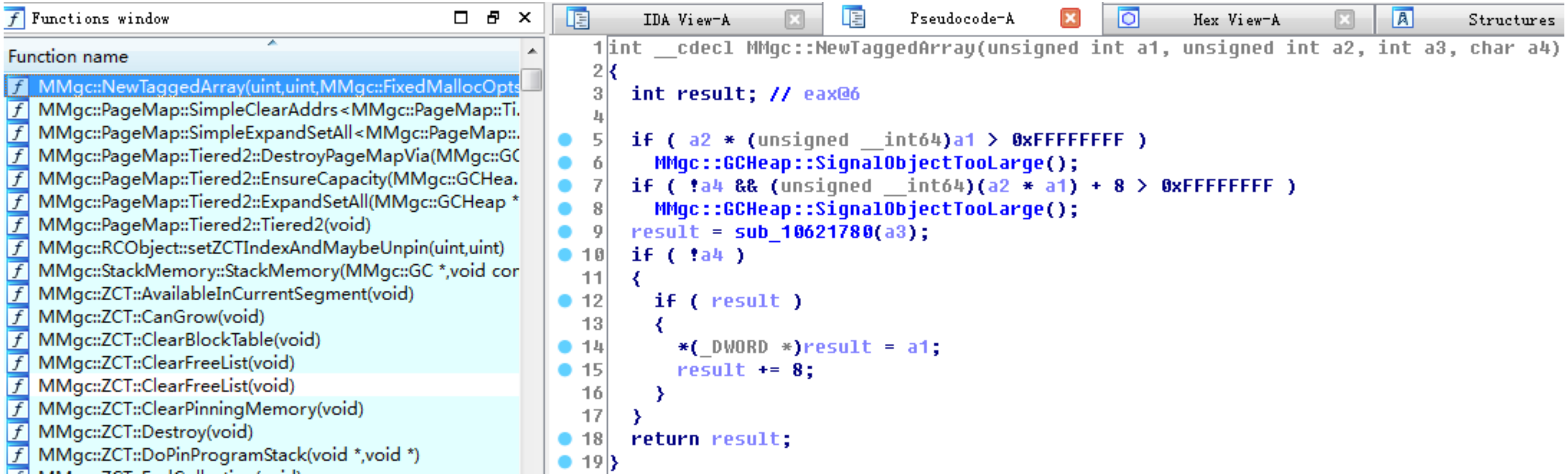
```
REALLY_INLINE void* FixedMalloc::Alloc(size_t size, FixedMallocOpts flags)
{
        if (size <= (size_t)kLargestAlloc)
                return FindAllocatorForSize(size)->Alloc(size, flags);
        else
                return LargeAlloc(size, flags);
}
```

Hook and change Fixed Heap Allocators to HeapAlloc in ProcessHeap

```
REALLY_INLINE void* FixedMalloc::Alloc(size_t size, FixedMallocOpts flags)
{
        …
        return HeapAlloc(GetProcessHeap(), 0, size);
}
```

A Lightweight Page Heap For FixedMalloc

- Find Heap Allocators in FixedMalloc(Simplest Way – AVM.sig)



MMgc::NewTaggedArray(uint,uint,MMgc::FixedMallocOpts
MMgc::PageMap::SimpleClearAddrs<MMgc::PageMap::Ti.
MMgc::PageMap::SimpleExpandSetAll<MMgc::PageMap::.
MMgc::PageMap::Tiered2::DestroyPageMapVia(MMgc::GC
MMgc::PageMap::Tiered2::EnsureCapacity(MMgc::GCHea.
MMgc::PageMap::Tiered2::ExpandSetAll(MMgc::GCHeap *
MMgc::PageMap::Tiered2::Tiered2(void)
MMgc::RCObject::setZCTIndexAndMaybeUnpin(uint,uint)
MMgc::StackMemory::StackMemory(MMgc::GC *,void cor
MMgc::ZCT::AvailableInCurrentSegment(void)
MMgc::ZCT::CanGrow(void)
MMgc::ZCT::ClearBlockTable(void)
MMgc::ZCT::ClearFreeList(void)
MMgc::ZCT::ClearFreeList(void)
MMgc::ZCT::ClearPinningMemory(void)
MMgc::ZCT::Destroy(void)
MMgc::ZCT::DoPinProgramStack(void *,void *)

```
int __cdecl MMgc::NewTaggedArray(unsigned int a1, unsigned int a2, int a3, char a4)
{
  int result; // eax@6

  if ( a2 * (unsigned __int64)a1 > 0xFFFFFFFF )
    MMgc::GCHeap::SignalObjectTooLarge();
  if ( !a4 && (unsigned __int64)(a2 * a1) + 8 > 0xFFFFFFFF )
    MMgc::GCHeap::SignalObjectTooLarge();
  result = sub_10621780(a3);
  if ( !a4 )
  {
    if ( result )
    {
      *(_DWORD *)result = a1;
      result += 8;
    }
  }
  return result;
}
```

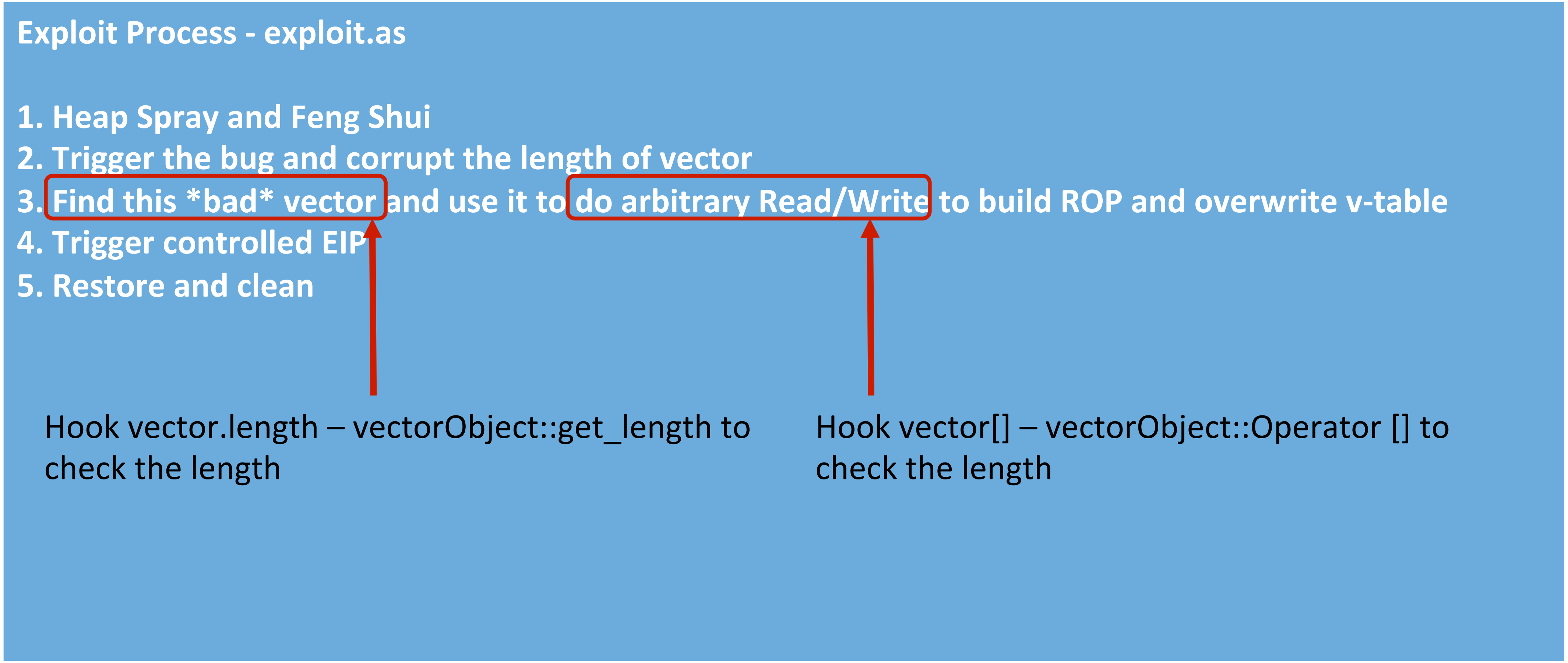
Part 3 : Detect Exploit

Find *bad* vector

3 Layer Exploit Detection

- Check the length of vectors when vector length/read/write operation in exploits use methods in flash module
- Monitor the length of vectors in Args of JIT function when vector length/read/write operation in exploits use JIT code
- God Mode: Monitor all of vector length when DoABC2 Tag is parsed.

Bad Vector Detection



Exploit Process - exploit.as

1. **Heap Spray and Feng Shui**
2. **Trigger the bug and corrupt the length of vector**
3. **Find this *bad* vector and use it to do arbitrary Read/Write to build ROP and overwrite v-table**
4. **Trigger controlled EIP**
5. **Restore and clean**

Hook vector.length – vectorObject::get_length to check the length

Hook vector[] – vectorObject::Operator [] to check the length

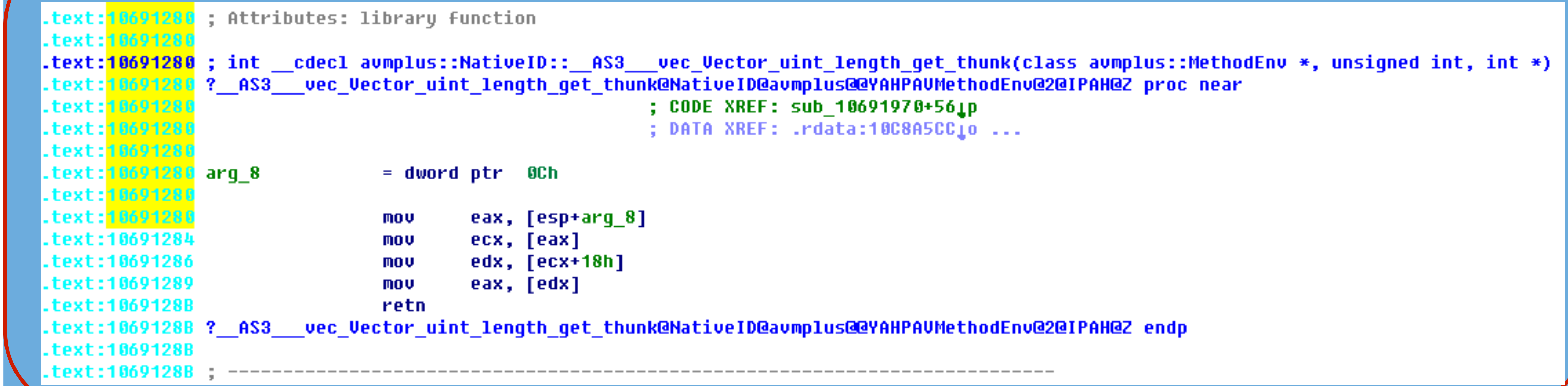
Not JIT-ed Length/Write/Read

Take vector.length as an example:

Exploit.as

```
for(_loc1_=0; _loc1_<cnt; _locl_++)
{
            if(vectors[_locl_].length > orig_length)
                        break;
}
```

We can set a hookpoint at Vector::get_length



```
.text:10691280 ; Attributes: library function
.text:10691280
.text:10691280 ; int __cdecl avmplus::NativeID::__AS3___vec_Vector_uint_length_get_thunk(class avmplus::MethodEnv *, unsigned int, int *)
.text:10691280 ?__AS3___vec_Vector_uint_length_get_thunk@NativeID@avmplus@@YAHPAVMethodEnv@2@IPAH@Z proc near
.text:10691280                                         ; CODE XREF: sub_10691970+56↓p
.text:10691280                                         ; DATA XREF: .rdata:10C8A5CC↓o ...
.text:10691280
.text:10691280 arg_8           = dword ptr  0Ch
.text:10691280
.text:10691280                 mov     eax, [esp+arg_8]
.text:10691284                 mov     ecx, [eax]
.text:10691286                 mov     edx, [ecx+18h]
.text:10691289                 mov     eax, [edx]
.text:1069128B                 retn
.text:1069128B ?__AS3___vec_Vector_uint_length_get_thunk@NativeID@avmplus@@YAHPAVMethodEnv@2@IPAH@Z endp
.text:1069128B
.text:1069128B ; ---------------------------------------------------------------------------
```

JIT-ed Length/Write/Read

Take vector.length as an example:

Exploit.as

```
for(_loc1_=0; _loc1_<cnt; _locl_++)
{
            if(vectors[_locl_].length > orig_length)
                        break;
}
```

We can't set a hookpoint at dynamic JIT-ed code

```
03d6042c mov     edx,dword ptr [ebp-90h] ; edx is address of arg3
03d60432 mov     eax,dword ptr [ebp-94h] ; eax is VectorObject address
03d60438 and     eax,0FFFFFFF8h ; atom type address
03d6043b mov     dword ptr [ebp-94h],eax
03d60441 je      <Unloaded_oy.dll>+0x3d60671 (03d60672)
03d60447 mov     ecx,dword ptr [eax+18h] ; [eax+0x18] is VectorBuffer
03d6044a mov     eax,dword ptr [ecx] ; ecx is VectorBuffer and [ecx] is the length of vector
03d6044c mov     dword ptr [ebp-98h],eax
03d60452 lea     esp,[esp]
03d60455 mov     ecx,dword ptr <Unloaded_oy.dll>+0xa7 (000000a8)[edx] ds:0023:03d44158=00000072 ; <- here
03d6045b mov     dword ptr [ebp-9Ch],ecx ; ecx is orig_length now
03d60461 cmp     eax,ecx ; compare vector.length and orig_length
03d60463 sete    dl
```

Arg3[base+0xa8] = orig_length

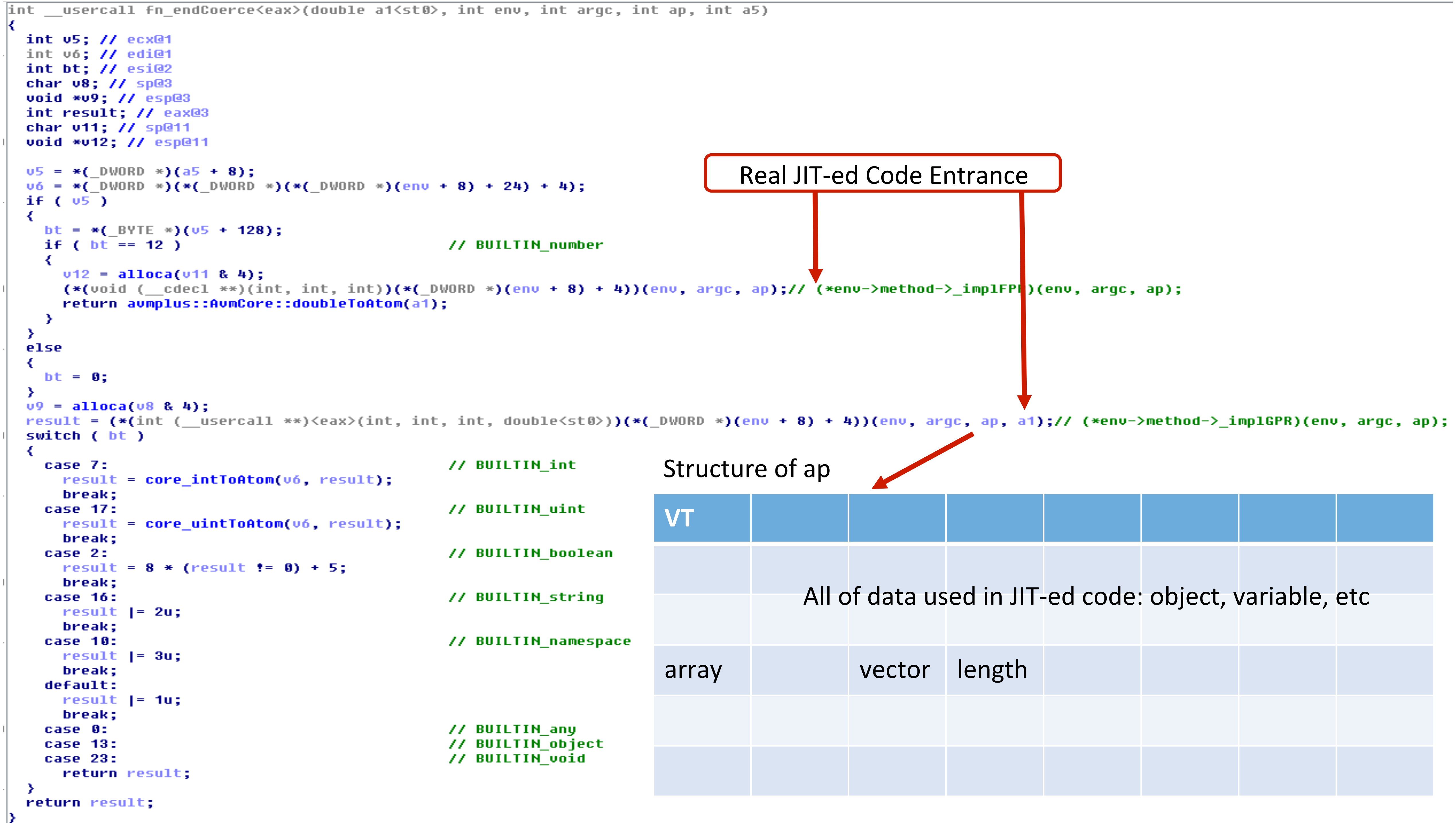
Where does the “VectorObject” in JIT-ed code come from?

JIT-ed ASM Code Fragment:

```
03d602be mov    edi,dword ptr [ebp+10h] <- edi is from the arg3
03d602f7 mov    ebx,dword ptr [edi]
03d602f9 mov    dword ptr [ebp-50h],ebx
03d60342 mov    eax,dword ptr [ebp-50h]
03d60345 mov    dword ptr [ebp-90h],eax
03d6034b mov    esi,dword ptr <Unloaded_oy.dll>+0x1d3 (000001d4)[eax] <- here
03d60351 mov    dword ptr [ebp-94h],esi
03d60370 mov    eax,dword ptr [ebp-94h]
03d60376 test   eax,eax
03d6037e lea    eax,[eax+1]
03d60387 push   eax <- from arg3
03d60399 mov    dword ptr [ebp-98h],eax <-eax is address of VectorObject
03d603f7 mov    eax,dword ptr [ebp-98h]
03d6040d mov    dword ptr [ebp-94h],eax
03d6042c mov    edx,dword ptr [ebp-90h]
03d60432 mov    eax,dword ptr [ebp-94h] <- eax is address of VectorObject
03d60438 and    eax,0FFFFFFF8h
03d6043b mov    dword ptr [ebp-94h],eax
03d60447 mov    ecx,dword ptr [eax+18h]
```

Arg3 is the 3rd argument of endCoerce and _implGPR.

```
Atom BaseExecMgr::endCoerce(MethodEnv* env, int32_t argc, uint32_t *ap, MethodSignaturep ms)
(*env->method->_implGPR)(env, argc, ap);
```



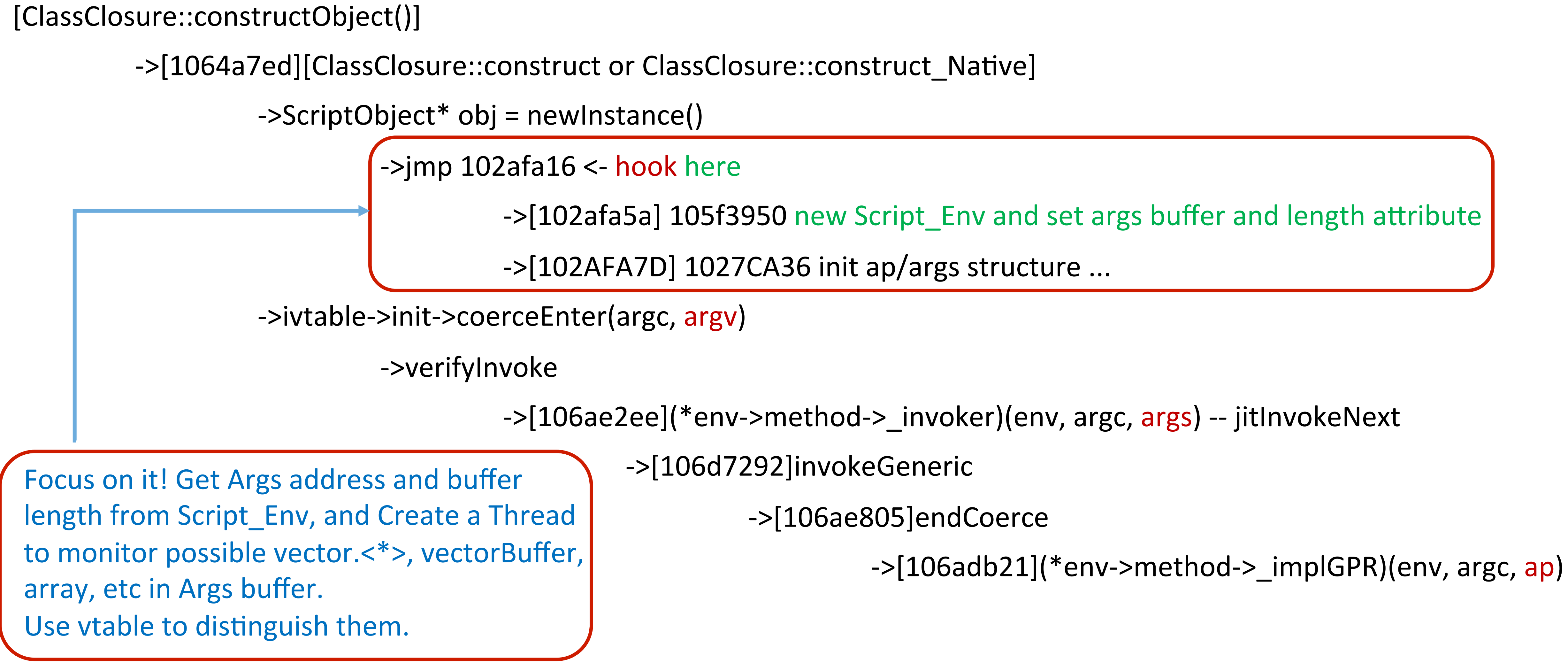
```
int __usercall fn_endCoerce<eax>(double a1<st0>, int env, int argc, int ap, int a5)
{
  int v5; // ecx@1
  int v6; // edi@1
  int bt; // esi@2
  char v8; // sp@3
  void *v9; // esp@3
  int result; // eax@3
  char v11; // sp@11
  void *v12; // esp@11

  v5 = *(_DWORD *)(a5 + 8);
  v6 = *(_DWORD *)(*(_DWORD *)(*(_DWORD *)(env + 8) + 24) + 4);
  if ( v5 )
  {
    bt = *(_BYTE *)(v5 + 128);
    if ( bt == 12 )                          // BUILTIN_number
    {
      v12 = alloca(v11 & 4);
      (*(void (__cdecl **)(int, int, int))(*(_DWORD *)(env + 8) + 4))(env, argc, ap);// (*env->method->_implFPR)(env, argc, ap);
      return avmplus::AvmCore::doubleToAtom(a1);
    }
  }
  else
  {
    bt = 0;
  }
  v9 = alloca(v8 & 4);
  result = (*(int (__usercall **)<eax>(int, int, int, double<st0>))(*(_DWORD *)(env + 8) + 4))(env, argc, ap, a1);// (*env->method->_implGPR)(env, argc, ap);
  switch ( bt )
  {
    case 7:                                  // BUILTIN_int
      result = core_intToAtom(v6, result);
      break;
    case 17:                                 // BUILTIN_uint
      result = core_uintToAtom(v6, result);
      break;
    case 2:                                  // BUILTIN_boolean
      result = 8 * (result != 0) + 5;
      break;
    case 16:                                 // BUILTIN_string
      result |= 2u;
      break;
    case 10:                                 // BUILTIN_namespace
      result |= 3u;
      break;
    default:
      result |= 1u;
      break;
    case 0:                                  // BUILTIN_any
    case 13:                                 // BUILTIN_object
    case 23:                                 // BUILTIN_void
      return result;
  }
  return result;
}
```

Real JIT-ed Code Entrance

Structure of ap

VT							
All of data used in JIT-ed code: object, variable, etc							
array		vector	length				

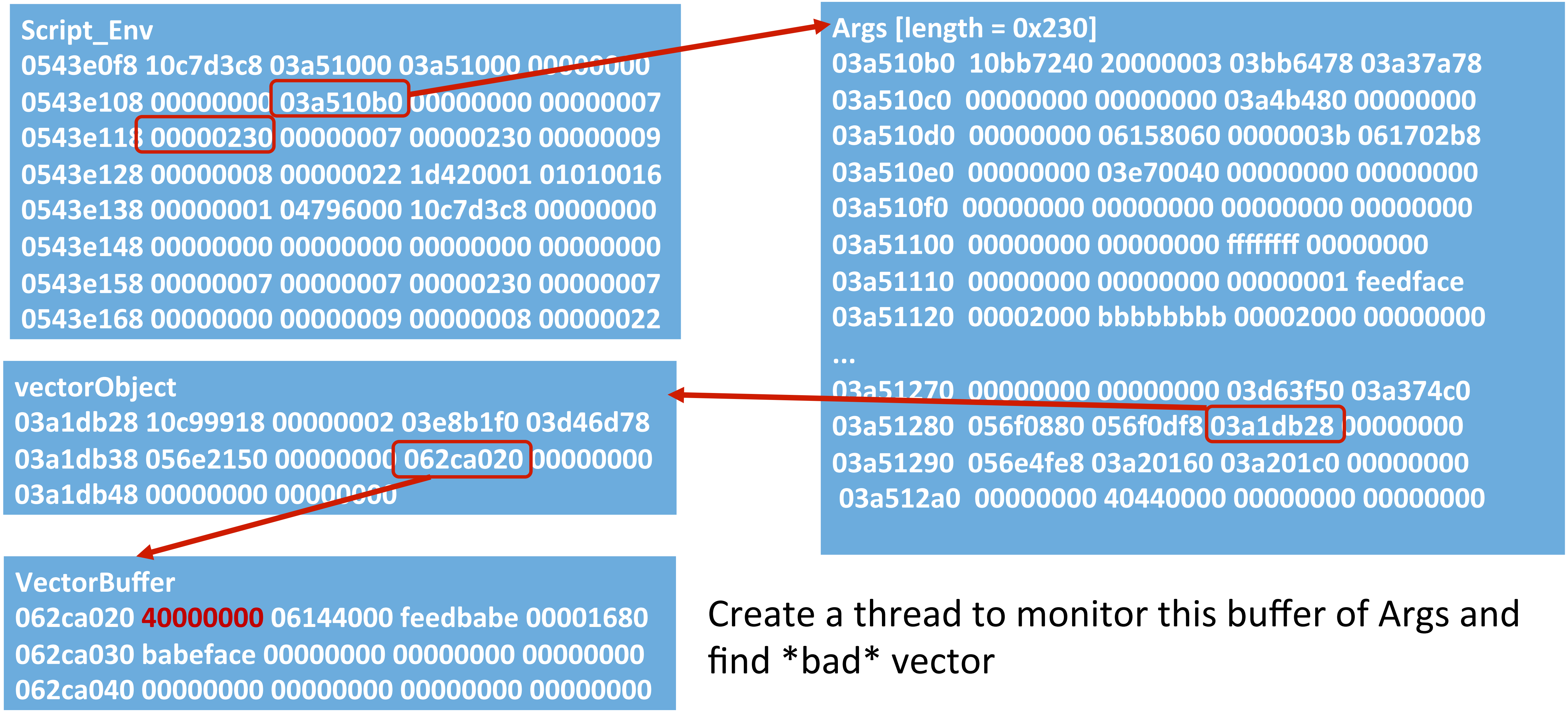
A Typical JIT Procedure in Flash



```
[ClassClosure::constructObject()]
        ->[1064a7ed][ClassClosure::construct or ClassClosure::construct_Native]
                ->ScriptObject* obj = newInstance()
                        ->jmp 102afa16 <- hook here
                                ->[102afa5a] 105f3950 new Script_Env and set args buffer and length attribute
                                ->[102AFA7D] 1027CA36 init ap/args structure ...
                ->ivtable->init->coerceEnter(argc, argv)
                        ->verifyInvoke
                                ->[106ae2ee](*env->method->_invoker)(env, argc, args) -- jitInvokeNext
                                        ->[106d7292]invokeGeneric
                                                ->[106ae805]endCoerce
                                                        ->[106adb21](*env->method->_implGPR)(env, argc, ap)
```

Focus on it! Get Args address and buffer length from Script_Env, and Create a Thread to monitor possible vector.<*>, vectorBuffer, array, etc in Args buffer.
Use vtable to distinguish them.

Memory Dump of Script_Env and Args



Script_Env

```
0543e0f8 10c7d3c8 03a51000 03a51000 00000000
0543e108 00000000 03a510b0 00000000 00000007
0543e118 00000230 00000007 00000230 00000009
0543e128 00000008 00000022 1d420001 01010016
0543e138 00000001 04796000 10c7d3c8 00000000
0543e148 00000000 00000000 00000000 00000000
0543e158 00000007 00000007 00000230 00000007
0543e168 00000000 00000009 00000008 00000022
```

Args [length = 0x230]

```
03a510b0  10bb7240 20000003 03bb6478 03a37a78
03a510c0  00000000 00000000 03a4b480 00000000
03a510d0  00000000 06158060 0000003b 061702b8
03a510e0  00000000 03e70040 00000000 00000000
03a510f0  00000000 00000000 00000000 00000000
03a51100  00000000 00000000 ffffffff 00000000
03a51110  00000000 00000000 00000001 feedface
03a51120  00002000 bbbbbbbb 00002000 00000000
…
03a51270  00000000 00000000 03d63f50 03a374c0
03a51280  056f0880 056f0df8 03a1db28 00000000
03a51290  056e4fe8 03a20160 03a201c0 00000000
 03a512a0  00000000 40440000 00000000 00000000
```

vectorObject

```
03a1db28 10c99918 00000002 03e8b1f0 03d46d78
03a1db38 056e2150 00000000 062ca020 00000000
03a1db48 00000000 00000000
```

VectorBuffer

```
062ca020 40000000 06144000 feedbabe 00001680
062ca030 babeface 00000000 00000000 00000000
062ca040 00000000 00000000 00000000 00000000
```

Create a thread to monitor this buffer of Args and find *bad* vector

Whole process of detecting *bad* vector operation in JIT-ed Code

- Find and hook “new Script_Env”
- Get Args buffer and length, Create a thread to monitor this buffer
- Use vtable to distinguish possible exploit object[vector.<*>, array, etc]

Turn on God Mode of Detection

Exploit Process - exploit.as

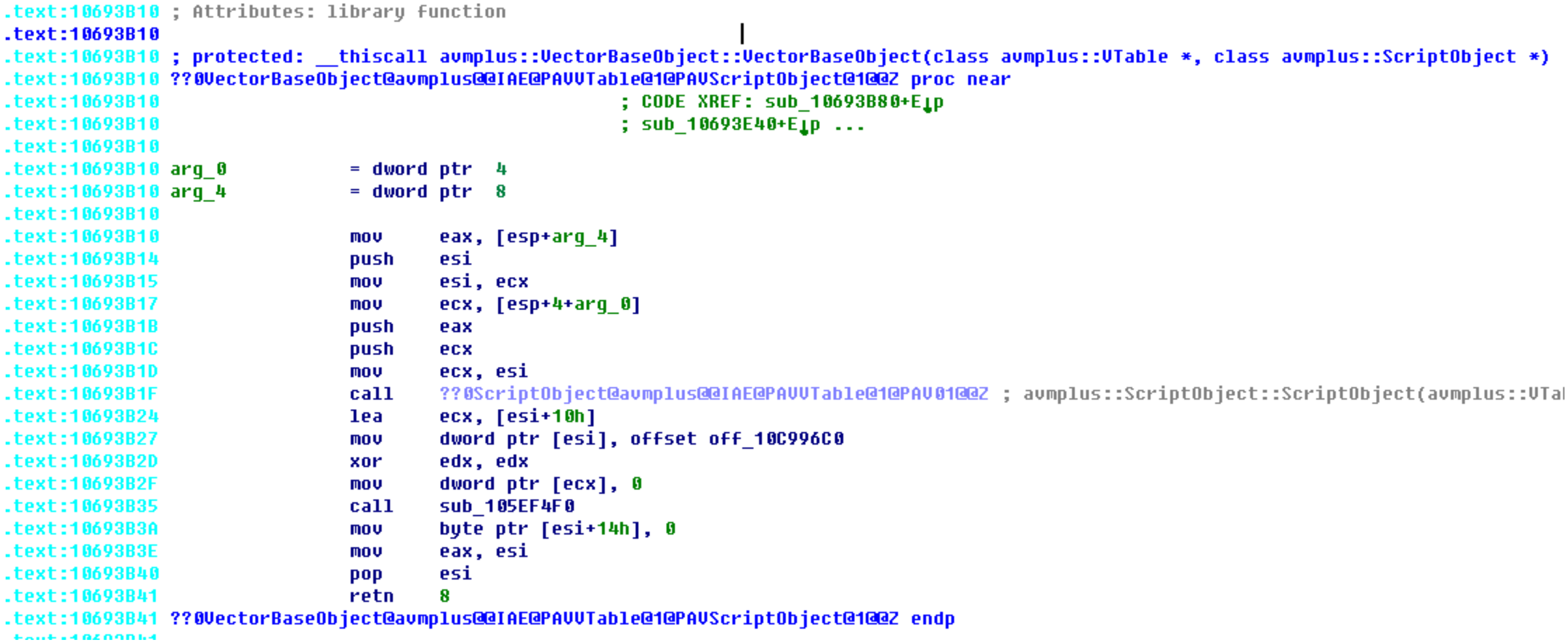
1. Heap Spray and Feng Shui
2. Trigger the bug and corrupt the length of vector
3. Find this *bad* vector and use it to do arbitrary Read/Write to build ROP and overwrite v-table
4. Trigger controlled EIP
5. Restore and clean

Hook VectorBaseObject::VectorBaseObject to Record all of allocated Vectors.<int>/<uint>/<double>

Create a Thread to Monitor every length change of vectors at the beginning of ActionScript was parsed

Hook Where 1#

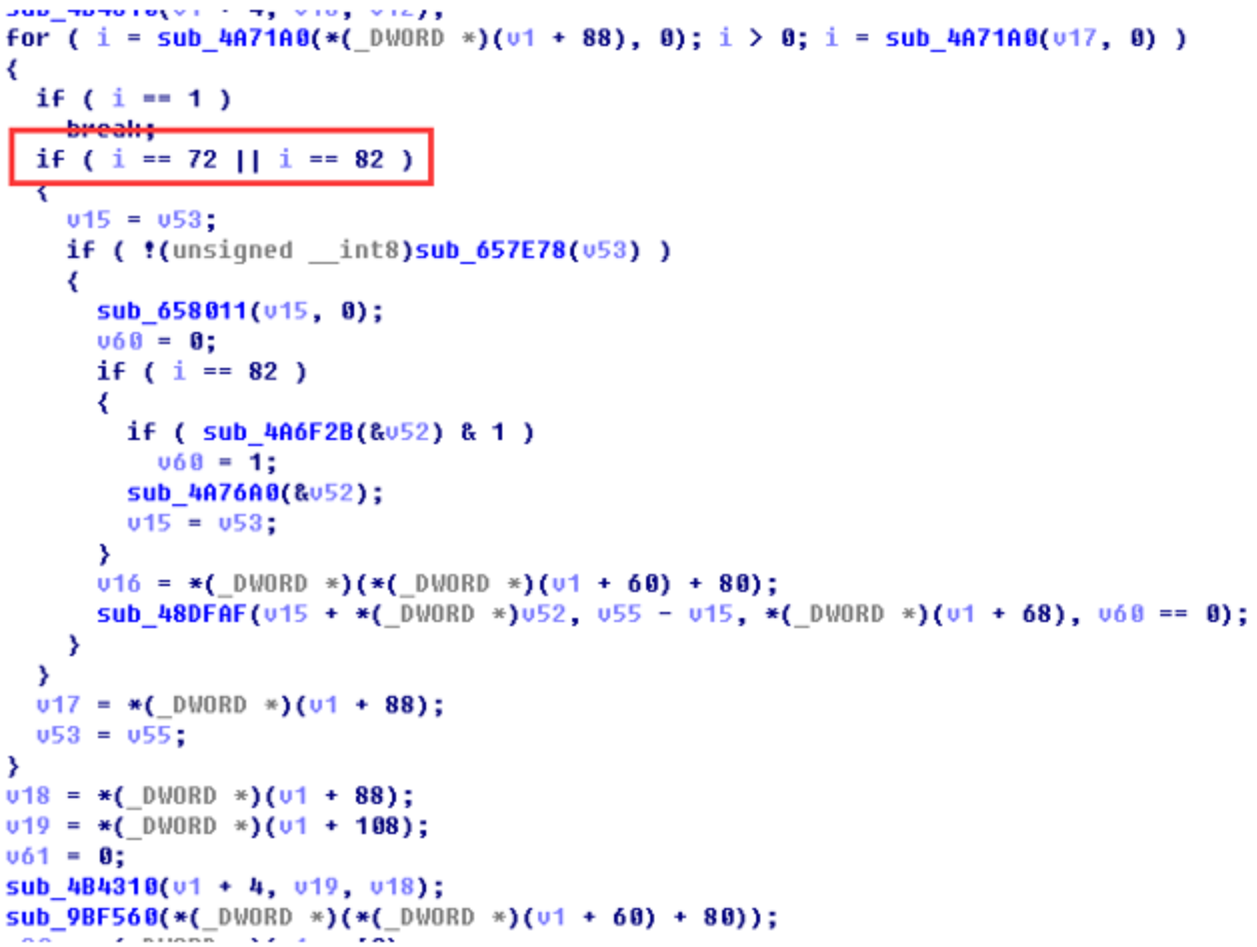
- Find VectorBaseObject::VectorBaseObject(Simplest Way – AVM.sig)



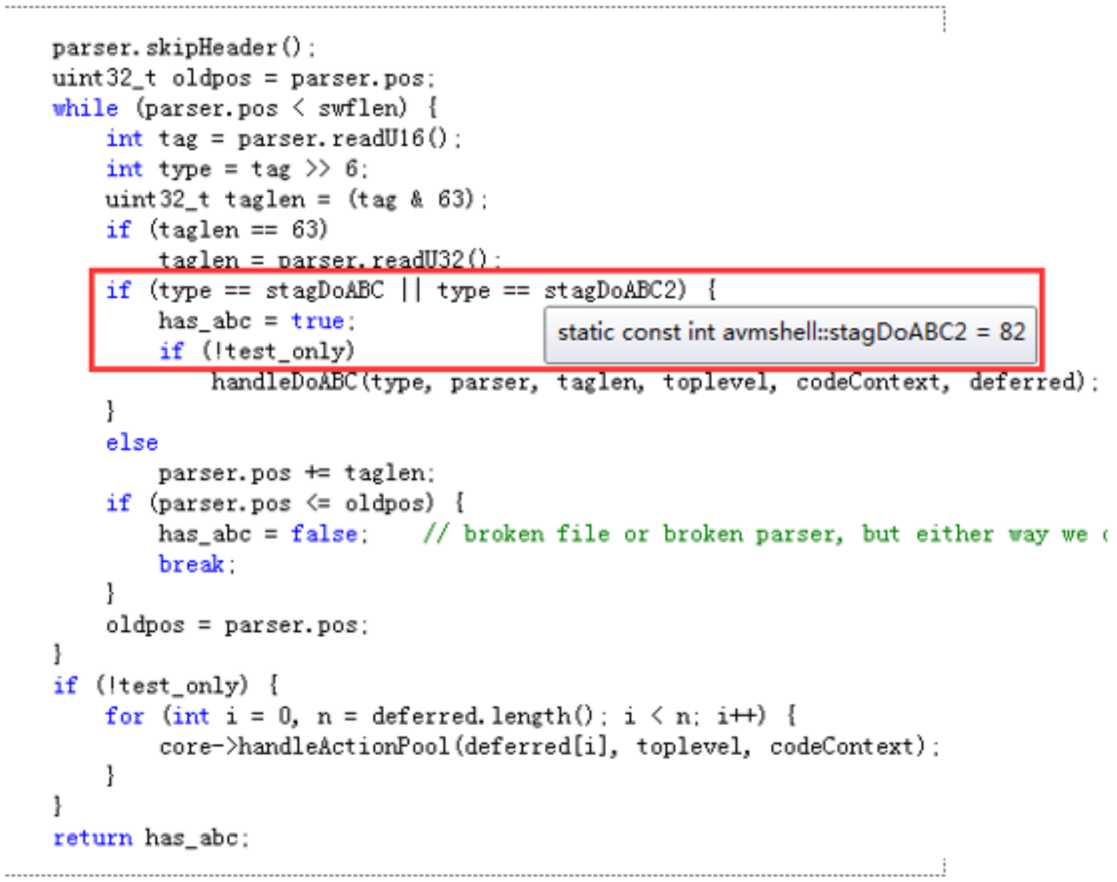
```
.text:10693B10 ; Attributes: library function
.text:10693B10
.text:10693B10 ; protected: __thiscall avmplus::VectorBaseObject::VectorBaseObject(class avmplus::VTable *, class avmplus::ScriptObject *)
.text:10693B10 ??0VectorBaseObject@avmplus@@IAE@PAVVTable@1@PAVScriptObject@1@@Z proc near
.text:10693B10                                         ; CODE XREF: sub_10693B80+E↓p
.text:10693B10                                         ; sub_10693E40+E↓p ...
.text:10693B10
.text:10693B10 arg_0           = dword ptr  4
.text:10693B10 arg_4           = dword ptr  8
.text:10693B10
.text:10693B10                 mov     eax, [esp+arg_4]
.text:10693B14                 push    esi
.text:10693B15                 mov     esi, ecx
.text:10693B17                 mov     ecx, [esp+4+arg_0]
.text:10693B1B                 push    eax
.text:10693B1C                 push    ecx
.text:10693B1D                 mov     ecx, esi
.text:10693B1F                 call    ??0ScriptObject@avmplus@@IAE@PAVVTable@1@PAV01@@Z ; avmplus::ScriptObject::ScriptObject(avmplus::VTa
.text:10693B24                 lea     ecx, [esi+10h]
.text:10693B27                 mov     dword ptr [esi], offset off_10C996C0
.text:10693B2D                 xor     edx, edx
.text:10693B2F                 mov     dword ptr [ecx], 0
.text:10693B35                 call    sub_105EF4F0
.text:10693B3A                 mov     byte ptr [esi+14h], 0
.text:10693B3E                 mov     eax, esi
.text:10693B40                 pop     esi
.text:10693B41                 retn    8
.text:10693B41 ??0VectorBaseObject@avmplus@@IAE@PAVVTable@1@PAVScriptObject@1@@Z endp
.text:10693B41
```

Hook Where 2#

- Find DoABCTag Function which is responsible for parse DoABC Tag.



```
sub_4B4310(v1 + 4, v18, v12);
for ( i = sub_4A71A0(*(_DWORD *)(v1 + 88), 0); i > 0; i = sub_4A71A0(v17, 0) )
{
  if ( i == 1 )
    break;
  if ( i == 72 || i == 82 )
  {
    v15 = v53;
    if ( !(unsigned __int8)sub_657E78(v53) )
    {
      sub_658011(v15, 0);
      v60 = 0;
      if ( i == 82 )
      {
        if ( sub_4A6F2B(&v52) & 1 )
          v60 = 1;
        sub_4A76A0(&v52);
        v15 = v53;
      }
      v16 = *(_DWORD *)(*(_DWORD *)(v1 + 60) + 80);
      sub_48DFAF(v15 + *(_DWORD *)v52, v55 - v15, *(_DWORD *)(v1 + 68), v60 == 0);
    }
  }
  v17 = *(_DWORD *)(v1 + 88);
  v53 = v55;
}
v18 = *(_DWORD *)(v1 + 88);
v19 = *(_DWORD *)(v1 + 108);
v61 = 0;
sub_4B4310(v1 + 4, v19, v18);
sub_9BF560(*(_DWORD *)(*(_DWORD *)(v1 + 60) + 80));
```



```
parser.skipHeader();
uint32_t oldpos = parser.pos;
while (parser.pos < swflen) {
    int tag = parser.readU16();
    int type = tag >> 6;
    uint32_t taglen = (tag & 63);
    if (taglen == 63)
        taglen = parser.readU32();
    if (type == stagDoABC || type == stagDoABC2) {
        has_abc = true;
        if (!test_only)
            handleDoABC(type, parser, taglen, toplevel, codeContext, deferred);
    }
    else
        parser.pos += taglen;
    if (parser.pos <= oldpos) {
        has_abc = false;    // broken file or broken parser, but either way we
        break;
    }
    oldpos = parser.pos;
}
if (!test_only) {
    for (int i = 0, n = deferred.length(); i < n; i++) {
        core->handleActionPool(deferred[i], toplevel, codeContext);
    }
}
return has_abc;
```

static const int avmshell::stagDoABC2 = 82

Life cycle of *bad* vector

- Why do we have to monitor the every length change, not check all of vectors once at the end of swf finish ?

Exploit Process - exploit.as

Exploiters can make the life cycle of *bad* vector very short !

1. Heap Spray and Feng Shui
2. Trigger the bug and corrupt the length of vector
3. Find this *bad* vector and use it to do arbitrary Read/Write to build ROP and overwrite c_cleaner of bad vector itself
4. Trigger controlled EIP with “bad_vector.length = new_length”
5. Restore 0and clean

Bad_vector will be free and reallocate. The length of this bad vector will be set to new_length, if we check after all this happened, everything will be normal.

Life cycle of *bad* vector

- Why do we have to monitor the every length change, not check all of vectors once at the end of swf finish ?

VectorBuffer structure

Length	C_Cleaner	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	…

bad VectorBuffer with normal c_cleaner

40000000	ABCDEFGH	data	data	data	data	data	data	data

bad VectorBuffer with *bad* c_cleaner
bad_vector[3fffffff] = bad_vector[base+3fffffff*4+8] = bad_vector[base+4] = DEADBEEF

40000000	DEADBEEF	data	data	data	data	data	data	data

Bad_vector.length = 0x72, bad c_cleaner[DEADBEEF] will trigger controlled EIP, and bad vector change to normal vector

00000072	DEADBEEF	data	data	data	data	data	data	data

Details about how c_cleaner[DEADBEEF] trigger controlled EIP -- http://researchcenter.paloaltonetworks.com/2015/05/the-latest-flash-uaf-vulnerabilities-in-exploit-kits/

Exploit Mitigation in flash_18_0_0_209

- Kill the vector-like object with length validation and isolated heap
- Raise the *bar* of exploit

Summary

- Dissect and unclose some undocumented and uncovered internals inside flash for detecting flash exploits.
- Multiple Dimensional Exploit Detection Based on the Deep Understanding of Exploit Essence
- Find Other Possible/Potential Exploit Object in Flash In the future

Thanks

- Thanks to Yamata Li and others in IPS Team
- Special thanks to @guhe120, @promised_lu

Questions ?