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 embedded 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
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
    }
}
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!
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**

  ![OBA Diagram]

  - Heap block
  - CRASH!
  - NO_Access
  - Free

- **UAF**

  ![UAF Diagram]

  - CRASH!
Custom Heap in Flash MMgc

- GCHeap
- FixedMalloc
- m_allocs[]
- Same size in one bucket
- Different sizes
- Used buffer
- Free buffer
- Ptr to next free block
Custom Heap in Flash MMgc

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::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

```javascript
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
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;
    ...
}
From AS3 To Memory

Take ByteArray As An Example

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

```c
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

```cpp
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

```cpp
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)
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; _loc1_++)
{
    if(vectors[_loc1_].length > orig_length)
        break;
}

We can set a hookpoint at Vector::get_length
Take `vector.length` as an example:

```javascript
Exploit.as
for(_loc1_=0; _loc1_<cnt; _loc1_++)
{
  if(vectors[_loc1_].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,0FFFFFFFF8h ; 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 eax, dword ptr [ebp-50h], ebx
03d60342  mov eax, dword ptr [ebp-90h], eax
03d6034b  mov esi, dword ptr <Unloaded_oy.dll>+0x1d3 (000001d4)[eax] <- here
03d60351  mov eax, dword ptr [ebp-94h]
03d60370  mov eax, dword ptr [ebp-94h]
03d60376  test eax, eax
03d6037e  lea eax, [eax+1]
03d60387  push eax <- from arg3
03d60399  mov eax, dword ptr [ebp-98h], eax <- eax is address of VectorObject
03d603f7  mov eax, dword ptr [ebp-98h]
03d6040d  mov eax, 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, 0FFFFFFFF8h
03d6043b  mov eax, 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, MethodSignature* ms)
(*env->method->_implGPR)(env, argc, ap);
Real JIT-ed Code Entrance

Structure of ap

All of data used in JIT-ed code: object, variable, etc

<table>
<thead>
<tr>
<th>VT</th>
<th>array</th>
<th>vector</th>
<th>length</th>
</tr>
</thead>
</table>


A Typical JIT Procedure in Flash

```
[ClassClosure::constructObject()]
  ->[1064a7ed][ClassClosure::construct or ClassClosure::construct_Native]
    ->[102afa16] ScriptObject* obj = newInstance()
    ->[102afa5a] 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, argv) -- 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

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)

```assembly
.text:10693B10 ; Attributes: library function
.text:10693B10 ; protected: thiscall avmplus::VectorBaseObject::VectorBaseObject(class avmplus::UObject *, class avmplus::ScriptObject *)
.text:10693B10 ??0VectorBaseObject@avmplus@@AEAAEPAVUObjectA@HOPAU5ScriptObject@@H宁静 proc near
.text:10693B10 ; CODE XREF: sub_10693880+34p
.text:10693B10 ; sub_10693E40+34p ...
.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:10693B19 push eax
.text:10693B1C push ecx
.text:10693B1D mov ecx, esi
.text:10693B1F call ??0ScriptObject@avmplus@@IAEPAVUObjectA@HOPAU5ScriptObject@@H宁静 ; avmplus::ScriptObject::ScriptObject(avmplus::UObject *)
.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_1065EF40
.text:10693B3A mov byte ptr [esi+14h], 0
.text:10693B3E mov eax, esi
.text:10693B40 pop esi
.text:10693B41 retn 0
.text:10693B44 ??0VectorBaseObject@avmplus@@IAEPAVUObjectA@HOPAU5ScriptObject@@H宁静 endp
```
Hook Where 2#

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

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

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

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

|--------|-----------|---------|---------|---------|---------|---------|---------|-----|

*bad* VectorBuffer with normal c_cleaner

| 40000000 | ABCDEFGH  | data   | data   | data   | data   | data   | data   |     |

*bad* VectorBuffer with *bad* c_cleaner


| 40000000 | DEADBEEF  | 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   |     |

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 ?