Exploit Adobe Flash Under the Latest Mitigation

Yuki Chen (@guhe120)

Qihoo 360Vulcan Team

360Vulcan

live long and pwn

Agenda

- Who am I
- Background
- Flash Exploit Mitigations
- Conclusion

About 360Vulcan Team

- ✓ Security Researchers
- ✓ Pwn2Own 2015 Internet Explorer 11
- ✓ Pwn2Own 2016 Google Chrome
- ✓ Pwn2Own 2016 Adobe Flash in Microsoft Edge
- ✓ 100+ CVE from Microsoft
- ✓ Syscan/BlackHat/HITCON /Syscan360/44Con/POC

Agenda

- Who am I
- Background
- Flash Exploit Mitigations
- Conclusion

Background

- Flash player is one of the hottest target in Apt/Target attacks these years
  - Remote
  - Multiple browsers
  - Many bugs
  - Easy to exploit

Hacking Team Leak – The Trigger?

- **3 0day exploits**, everyone can use it easily
- **Sophisticated** exploit template demonstrated
- Remaindered us again that **how easy it was** to exploit a flash bug
- **Adobe decided** to do something **to fight** against such in-the-wild 0day **exploits**

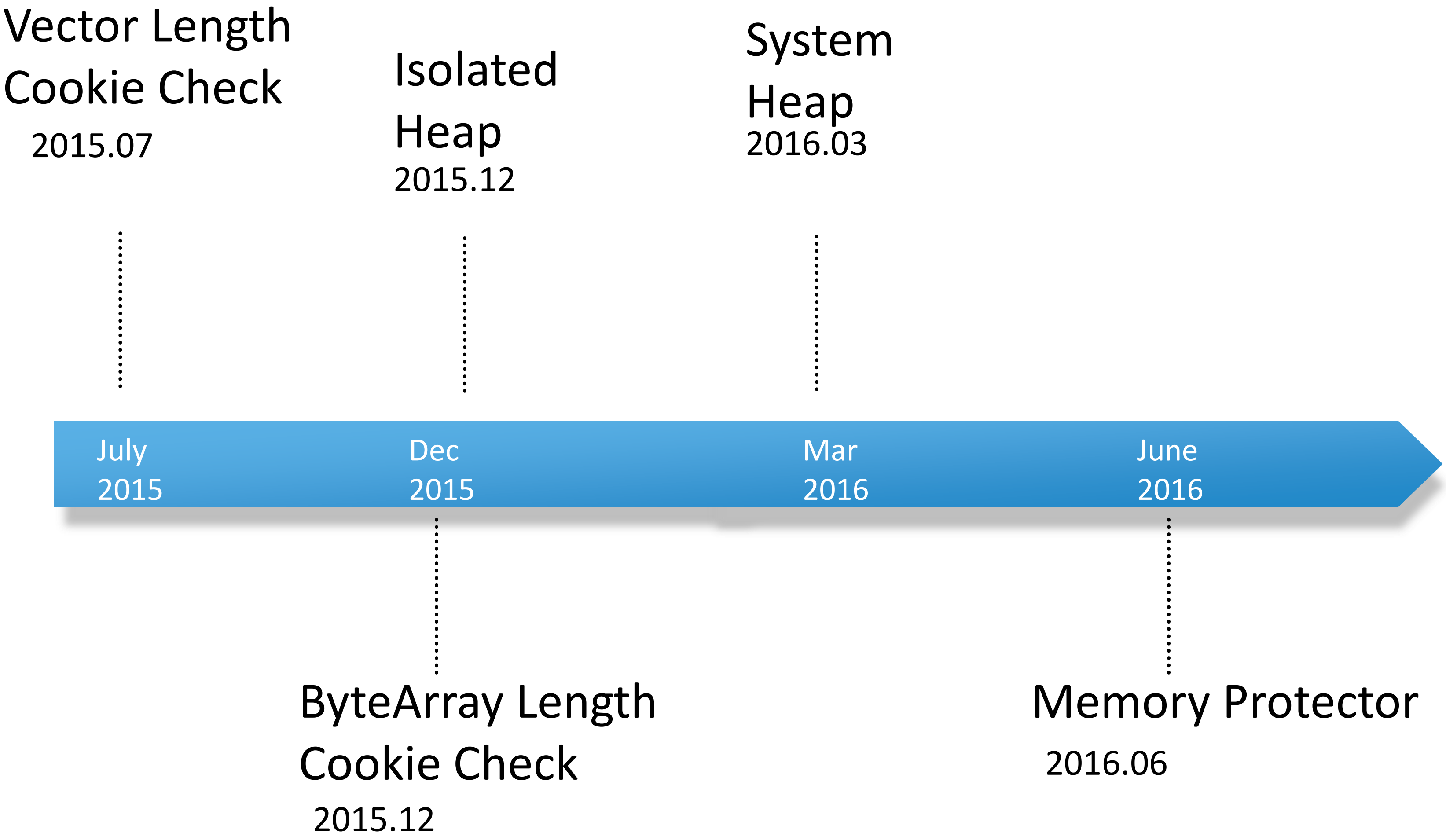
Adobe is Serious, So are We

- They added some really good mitigations
- We also researched these mitigations carefully
  - For the pwn2own contest
  - Made several flash exploits under the mitigations
  - 2 used in pwn2own 2016
    - One for Microsoft Edge Browser
    - One for Google Chrome sandbox bypass
- Some share about our research today

Agenda

- Who am I
- Background
- Flash Exploit Mitigations
- Conclusion

TimeLine of Import Flash Exploit Mitigation



Vector Length
Cookie Check
2015.07

Isolated
Heap
2015.12

System
Heap
2016.03

July
2015

Dec
2015

Mar
2016

June
2016

ByteArray Length
Cookie Check
2015.12

Memory Protector
2016.06

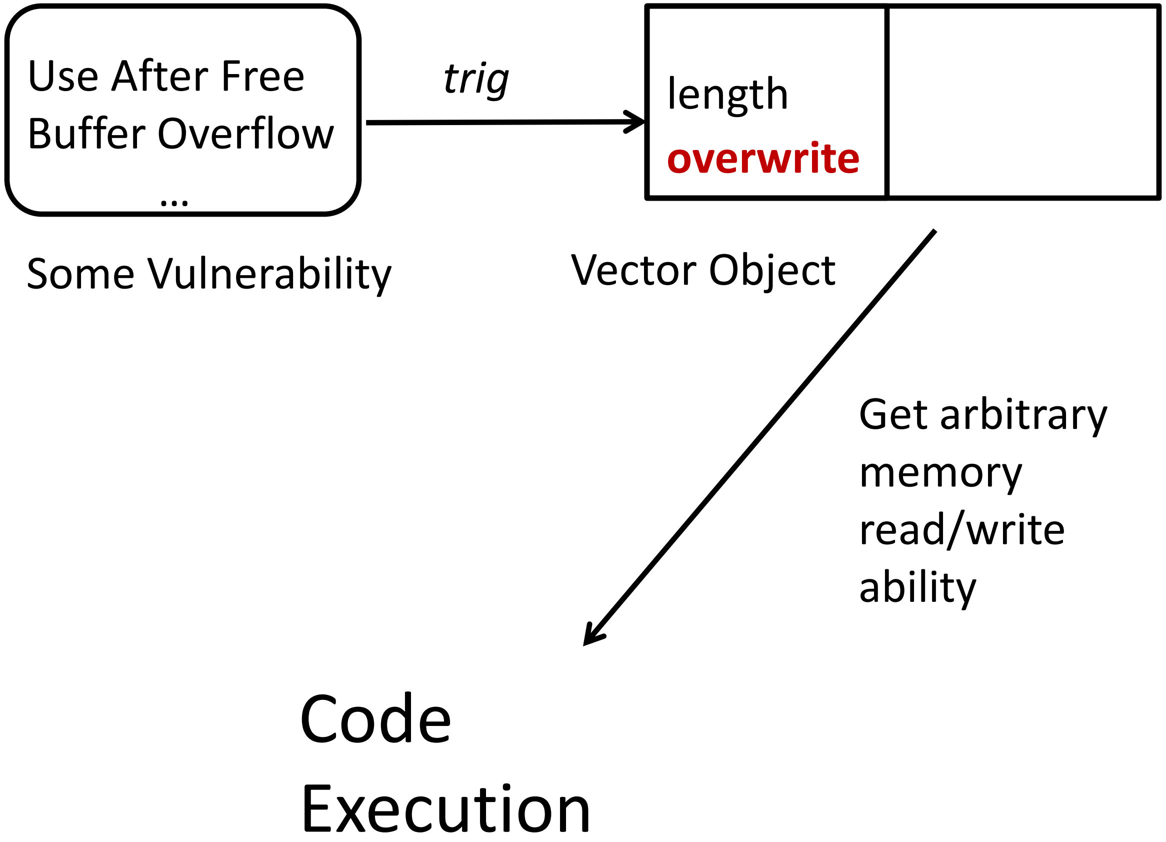
Length Cookie

- First introduced in July 2015
- Add extra checks when using some array-like objects
  - Vector
  - ByteArray
  - BitmapData

The Array-Type Object and Exploits

- Good friends of Exploit Writer
- JS
  - TypedArray, NativeArray, Array, String
- Java
  - Java Primitive Array
- Actionscript
  - Vector, ByteArray, BitmapData, String

Exploit Abusing Vector Before the Mitigation



Use After Free
Buffer Overflow
…

Some Vulnerability

trig

length
overwrite

Vector Object

Get arbitrary memory read/write ability

Code Execution

Length Cookie Mitigation

- Stored a XORed cookie of important fields in the array-like object
  - Vector: length
  - ByteArray: length, capacity, m_array
  - BitmapData: length, data

- Check the cookie when use the object
- The XOR key is initialized randomly when module is loaded

Length Cookie Mitigation - Example

```
var v:Vector.<uint> = new Vector.<uint>(0x100);
```

XORed Length

```
0:025> dd 09ecd020

09ecd020  b71a6a6f 1ca16666 1ca17777 00000000
09ecd030  00000000 00000000 00000000 00000000
09ecd040  00000000 00000000 00000000 00000000
```

```
0a6a4e19        mov   eax, ecx
0a6a4e1b        xor   edx, b71a6b6f
```

Key

b71a6b6f ^ b71a6a6f = 0x100

Length Cookie Mitigation - Efficiency

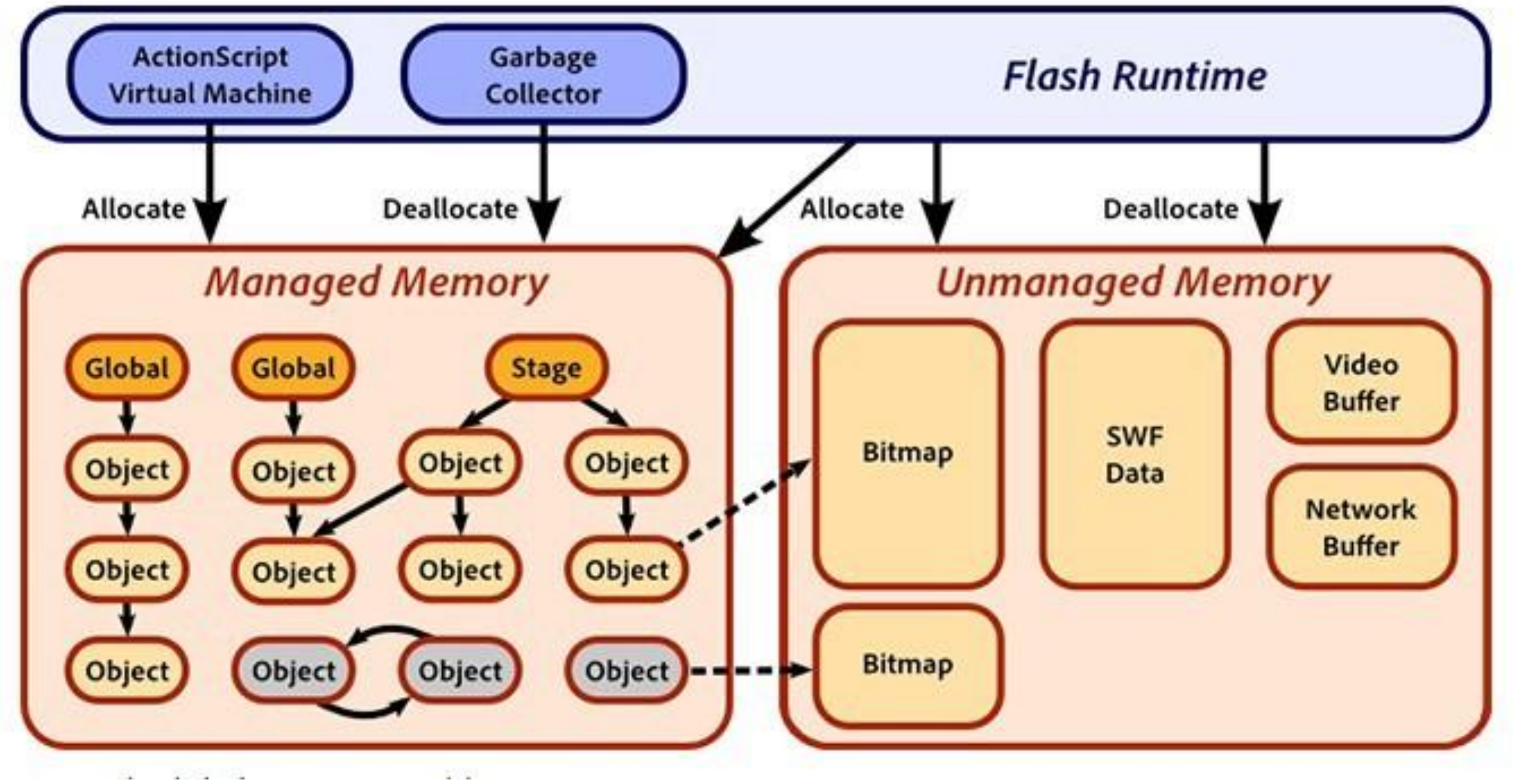
- Very powerful mitigation, significantly raises the difficulty of exploiting flash bugs

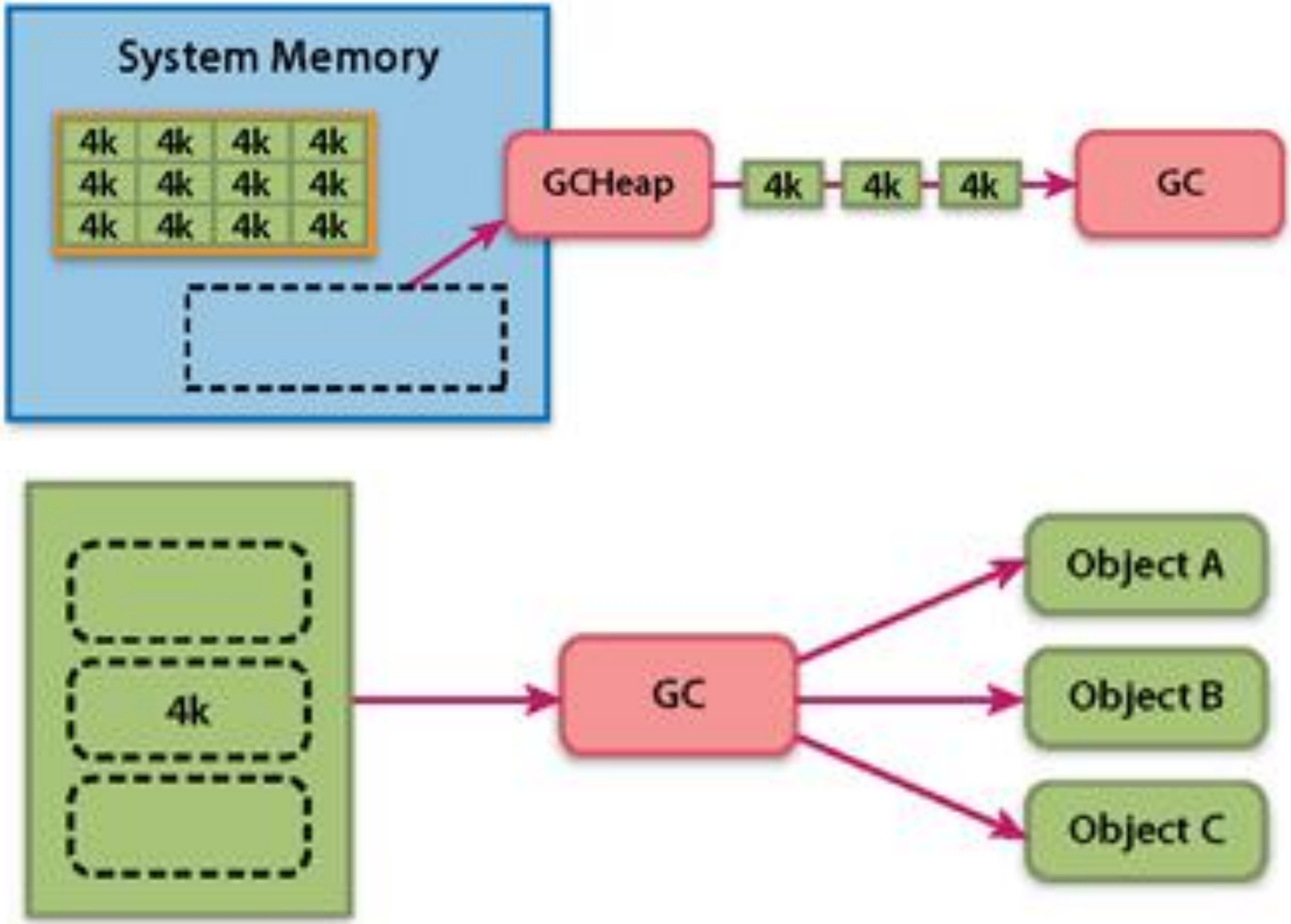
- Some other choices (but not as good as)
  - JS array in browser
  - Leak the cookie first, then overwrite
  - Other not protected objects

Mitigation in Heap Management

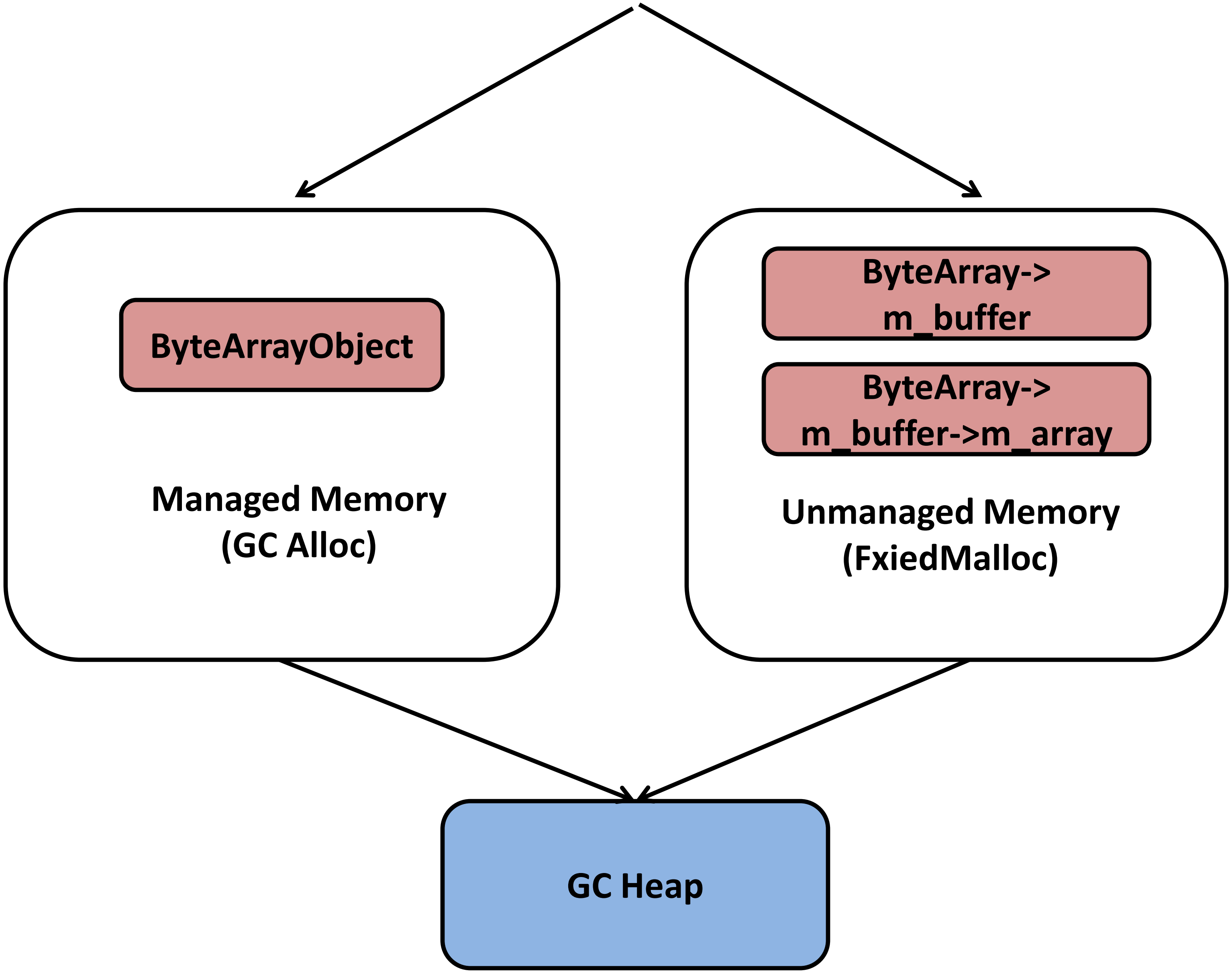
- Isolated Heap
- System Heap

An Overview of the Flash Heap (Before Dec 2015)




System Memory
4k 4k 4k 4k
4k 4k 4k 4k
4k 4k 4k 4k
GCHeap
4k
4k
4k
GC
4k
GC
Object A
Object B
Object C

```
var ba:ByteArray = new ByteArray();
ba.length = 0x1000;
```



ByteArrayObject

Managed Memory
(GC Alloc)

ByteArray->
m_buffer

ByteArray->
m_buffer->m_array

Unmanaged Memory
(FxiedMalloc)

GC Heap

The Problem of the Flash Heap

- All memory blocks are allocated with the same underline GC Heap (**No Isolation**)
  - GC/No-GC objects are allocated together
  - Object (class object, array, …) and Data (buffer, …) are allocated together
- No front-end randomization in both allocators (**Predictable**)
- Heap meta data (header, free list,…) lack of protection (**Vulnerable**)

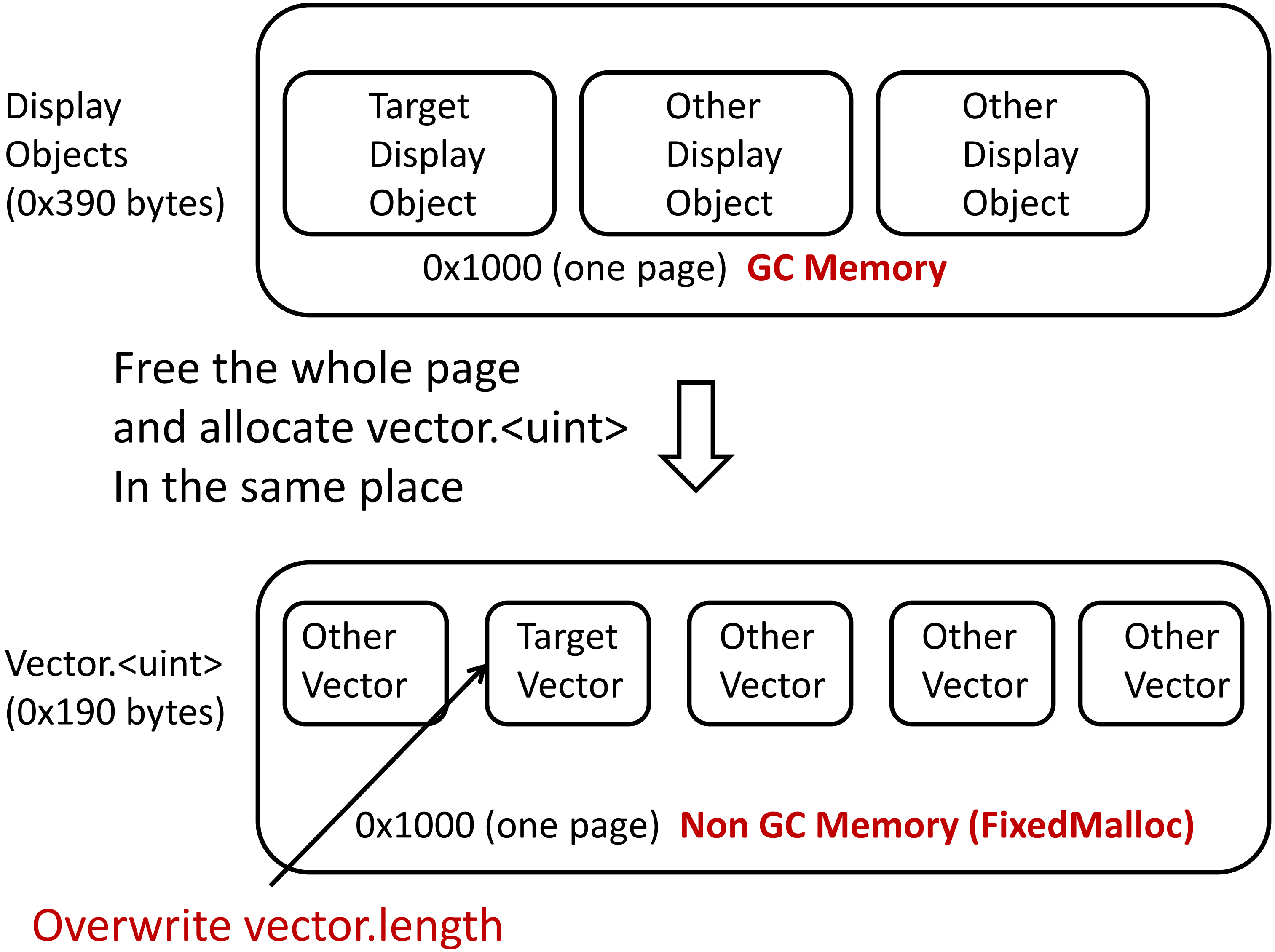
Example: CVE-2015-5122

- The hacking team 0day
- Use after free in Display Object
  - In the GC Heap

CVE-2015-5122 - Exploit

- Abuse vector.<uint>
- Free the problematic object and the whole page (**GC Heap**)
- Allocate vector.<uint> in the place of freed page (**No GC Heap**)
- Overwrite vector.length

The problem: Mix different objects in same heap makes exploit easy



Display Objects (0x390 bytes)

Target Display Object

Other Display Object

Other Display Object

0x1000 (one page) **GC Memory**

Free the whole page
and allocate vector.<uint>
In the same place

Vector.<uint> (0x190 bytes)

Other Vector

Target Vector

Other Vector

Other Vector

Other Vector

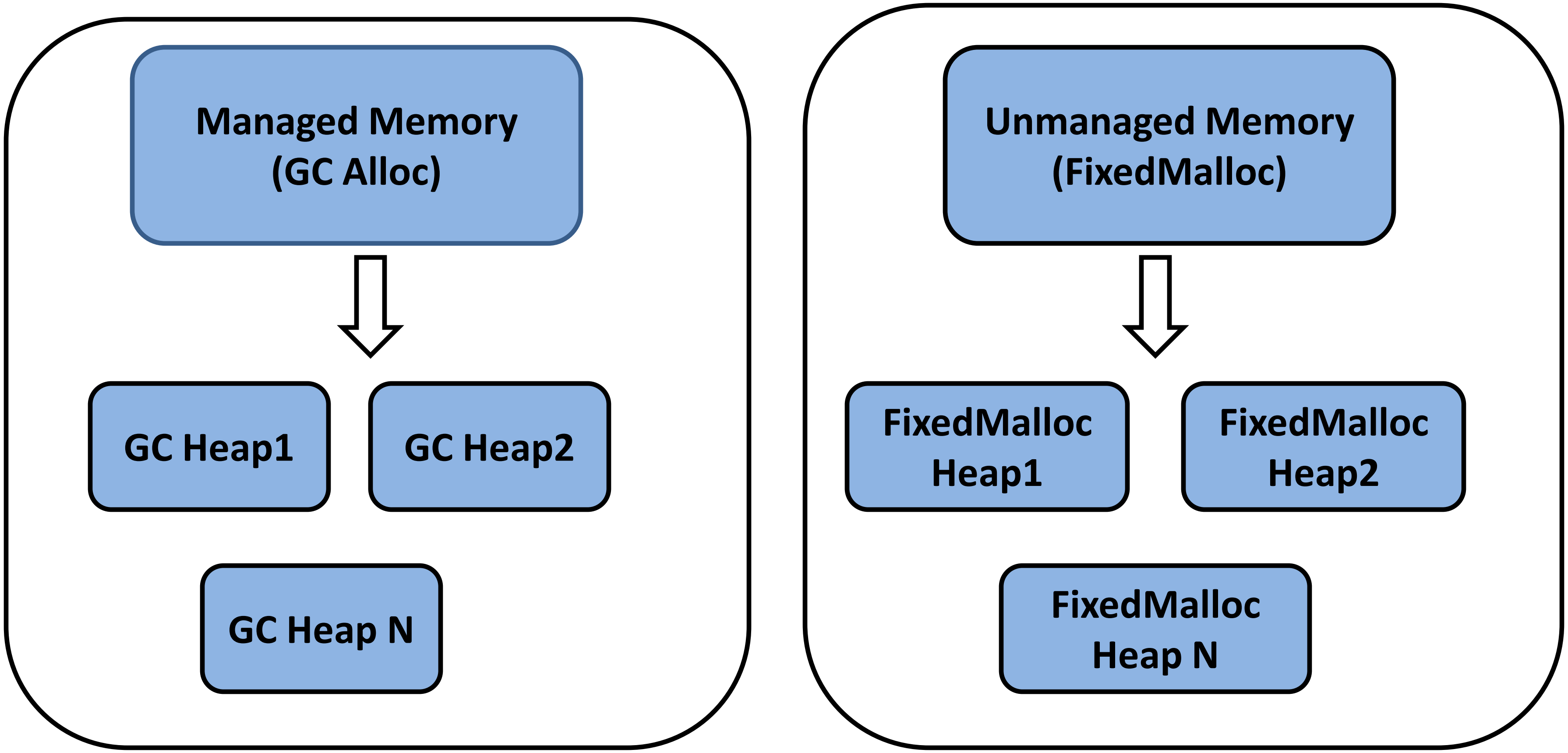
0x1000 (one page) **Non GC Memory (FixedMalloc)**

Overwrite vector.length

Isolated Heap

- Introduced in Dec 2015 CPU
- Aimed to address the biggest problem of flash memory management:
  - The problem that all objects share the same low-level heap

Isolated Heap Overview



Managed Memory (GC Alloc)

GC Heap1

GC Heap2

GC Heap N

Unmanaged Memory (FixedMalloc)

FixedMalloc Heap1

FixedMalloc Heap2

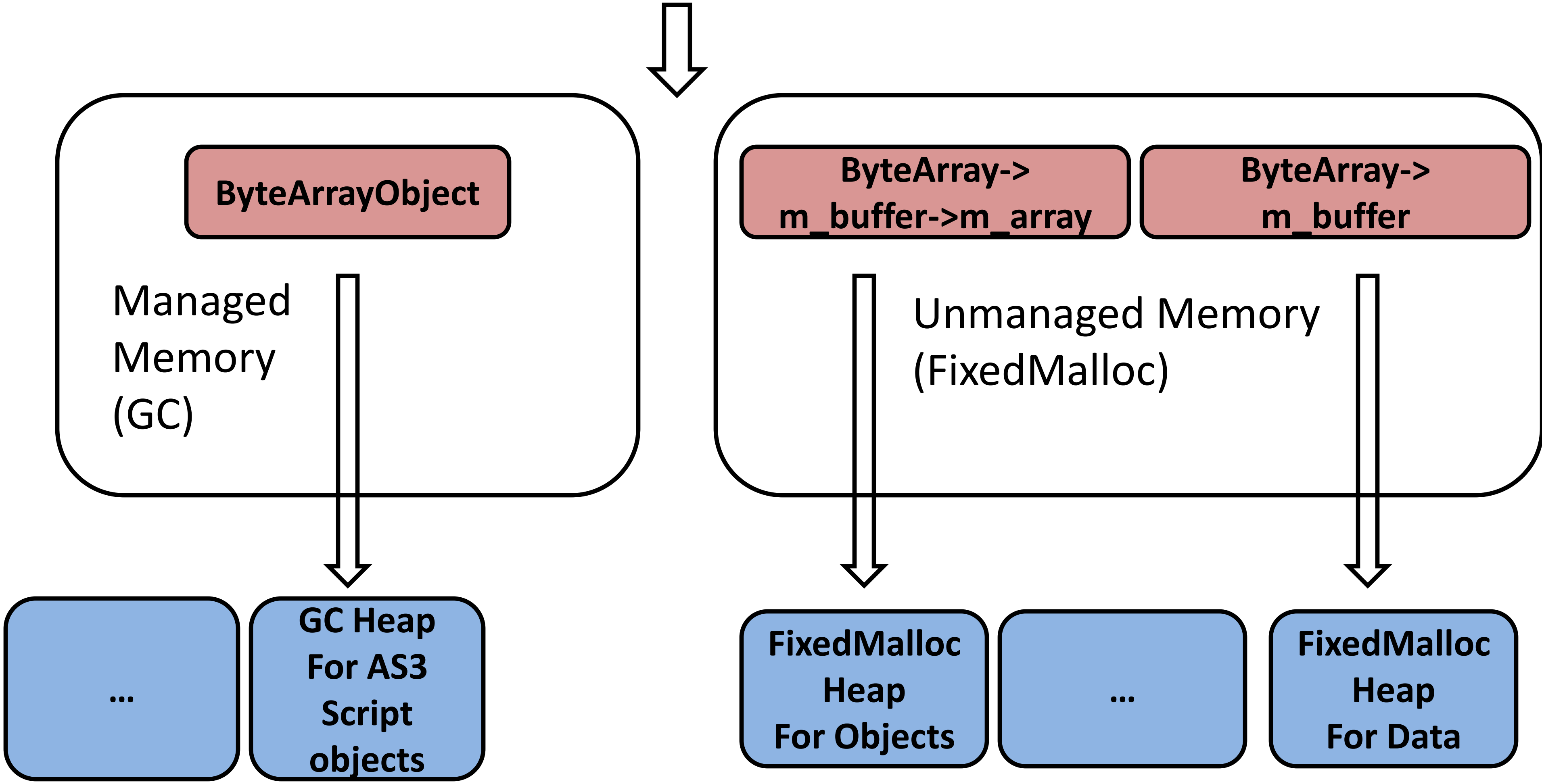
FixedMalloc Heap N

Isolated Heap - Highlight

- GC allocation and non-GC allocations (FixedMalloc) are now separated

- Different objects inside GC/Non GC allocations are also separated
  - GC/FixedMalloc contains several different heaps for different purpose (extensible)
  - e.g. In FixedMalloc, data and objects are separated

```
var ba:ByteArray = new ByteArray();
ba.length = 0x1000;
```



ByteArrayObject

Managed Memory (GC)

…

GC Heap For AS3 Script objects

ByteArray-> m_buffer->m_array

ByteArray-> m_buffer

Unmanaged Memory (FixedMalloc)

FixedMalloc Heap For Objects

…

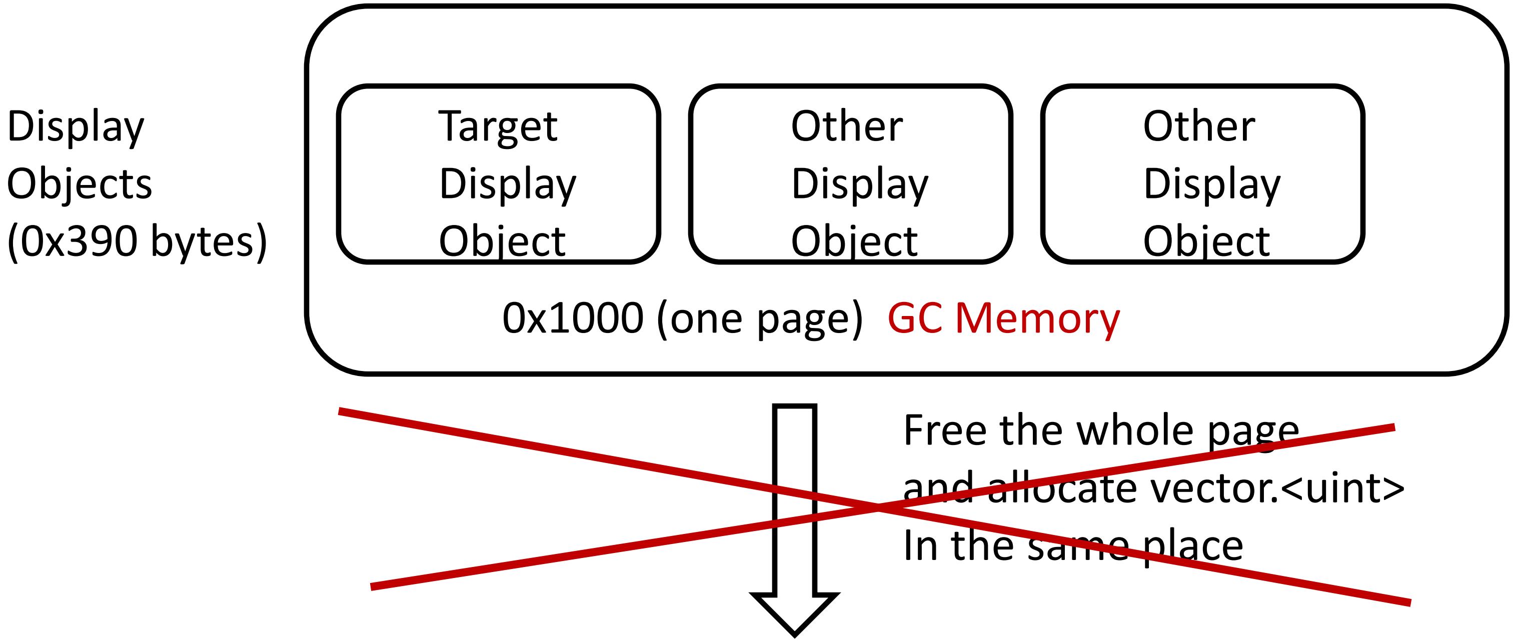
FixedMalloc Heap For Data

Isolated Heap - Efficiency

- The flash isolated heap mitigation is actually a very powerful mitigation
  - The data and objects are separated
  - High risk object and other objects are separated

- Consider the example of CVE-2015-5122

CVE-2015-5122 Exploit under Isolated Heap



Display Objects (0x390 bytes)

Target Display Object

Other Display Object

Other Display Object

0x1000 (one page) GC Memory

~~Free the whole page and allocate vector.<uint> In the same place~~

The reuse does not work, because now display object and Vector.<unit> are in different heaps

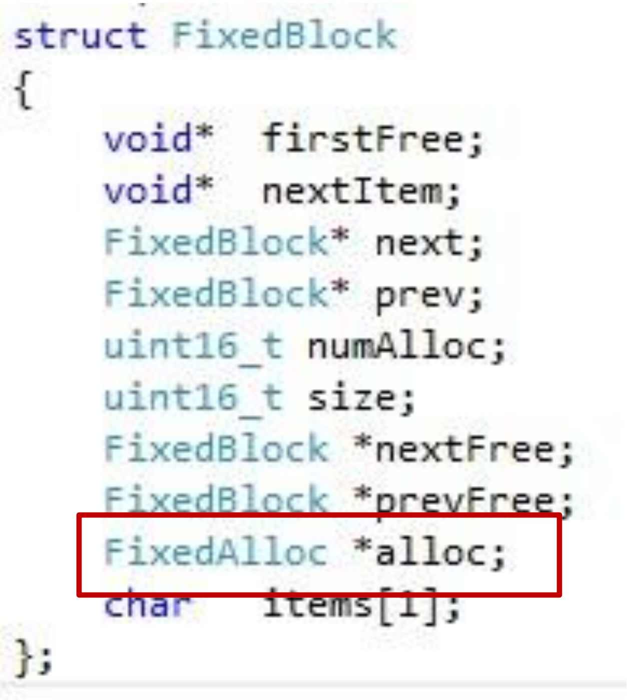
Isolated Heap – Enough?

- The number of separated heaps are still too little, especially in GC memory

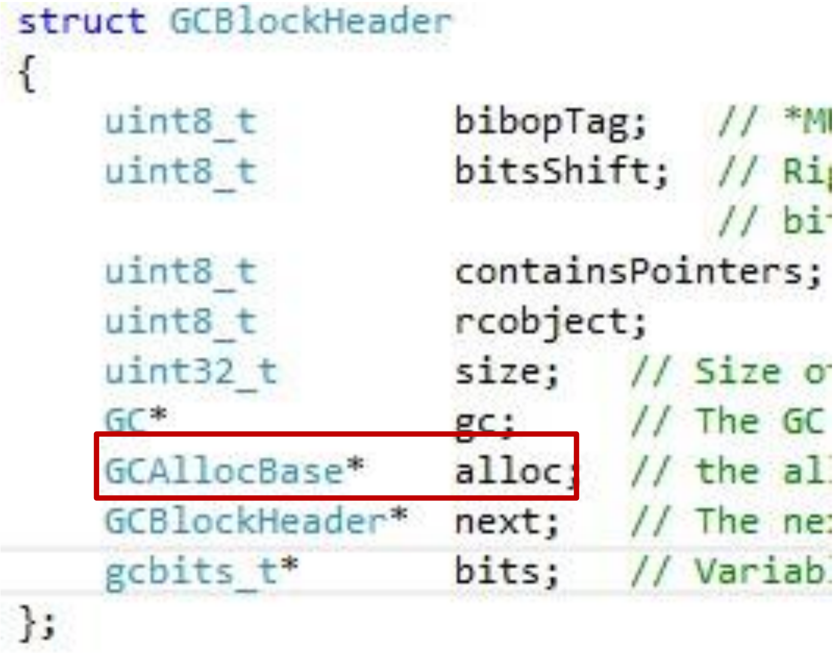
- Objects are separated by type, not by size
  - Object with different size can still be allocated together
  - Partially solved by the system heap mitigation

Isolated Heap – Enough?

- Travel between different isolated heaps
  - By overwriting the allocator in the block header



```
struct FixedBlock
{
    void*  firstFree;
    void*  nextItem;
    FixedBlock* next;
    FixedBlock* prev;
    uint16_t numAlloc;
    uint16_t size;
    FixedBlock *nextFree;
    FixedBlock *prevFree;
    FixedAlloc *alloc;
    char   items[1];
};
```



```
struct GCBlockHeader
{
    uint8_t          bibopTag;    // *M
    uint8_t          bitsShift;   // Ri
                                  // bi
    uint8_t          containsPointers;
    uint8_t          rcobject;
    uint32_t         size;    // Size o
    GC*              gc;      // The GC
    GCAllocBase*     alloc;   // the al
    GCBlockHeader*   next;    // The ne
    gcbits_t*        bits;    // Variab
};
```

System Heap

- Introduced in Mar 2016 CPU

- Aimed to address the problem that:
  - The flash heap allocation is too predictable
  - The flash heap block metadata has little protection

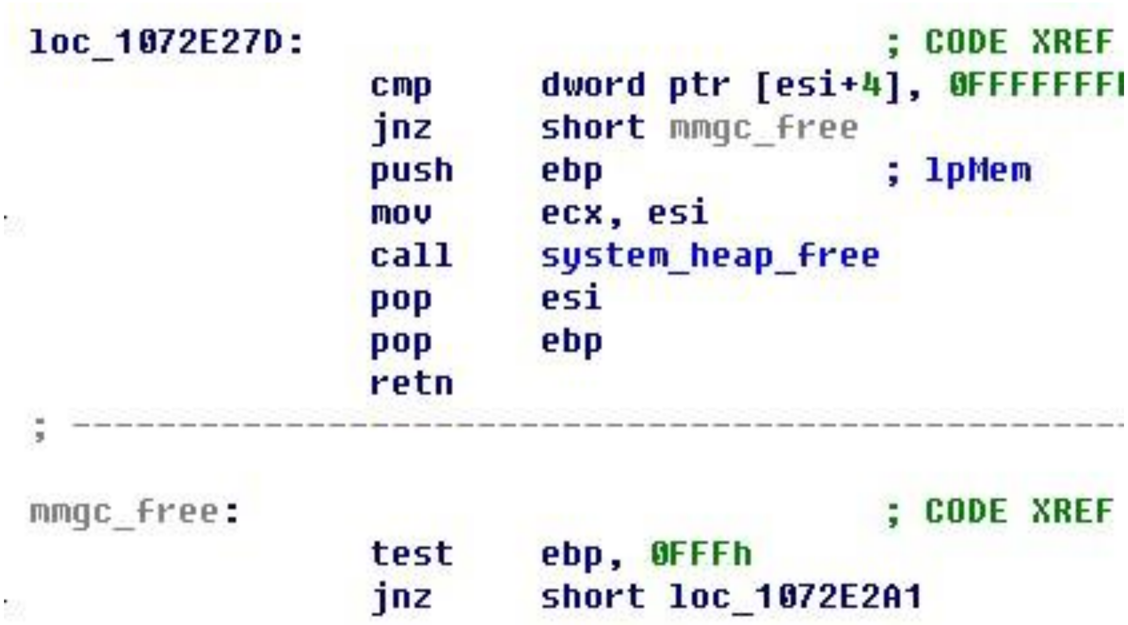
- Only works for MMGC heap (unmanaged memory)

System Heap

- Released 1 week before Pwn2Own 2016
  - Delayed patch

System Heap - Implementation

- The concept is simple:
  - Use system heap (HeapAlloc) directly in MMGC (unmanaged memory) allocation



```
loc_1072E27D:                           ; CODE XREF
                cmp     dword ptr [esi+4], 0FFFFFFF
                jnz     short mmgc_free
                push    ebp             ; lpMem
                mov     ecx, esi
                call    system_heap_free
                pop     esi
                pop     ebp
                retn
; ---------------------------------------------------

mmgc_free:                              ; CODE XREF
                test    ebp, 0FFFh
                jnz     short loc_1072E2A1
```

System Heap - Efficiency

- Front end randomization in windows 8+
  - Gives more random memory layout
- The system heap metadata is protected
  - The old heap metadata (block header, free list entry) could be easily attacked

Before system heap: allocate 10 objects, 0x38 bytes each

```
rax=000002bef2db8388
rax=000002bef2db83c0
rax=000002bef2db83f8
rax=000002bef2db8430
rax=000002bef2db8468
rax=000002bef2db84a0
rax=000002bef2db84d8
rax=000002bef2db8510
rax=000002bef2db8548
rax=000002bef2db8580
```

After system heap: allocate 10 objects, 0x38 bytes each

```
rax=000001f559513710
rax=000001f559513190
rax=000001f5595133d0
rax=000001f559513610
rax=000001f559513150
rax=000001f5595132d0
rax=000001f559513390
rax=000001f559513410
rax=000001f559513550
rax=000001f559513450
```

System Heap - Problem

- The biggest problem is that it is only used for mmgc allocation
  - The GC memory still uses flash's heap management
    - Still predictable
    - Attack heap metadata still possible
    - Memory reuse is easy
- Also some objects/buffer in mmgc still use the old allocation
  - Vector, ByteArray
  - We will demonstrate an attack on such object later

Use After Free Mitigation - Memory Protector

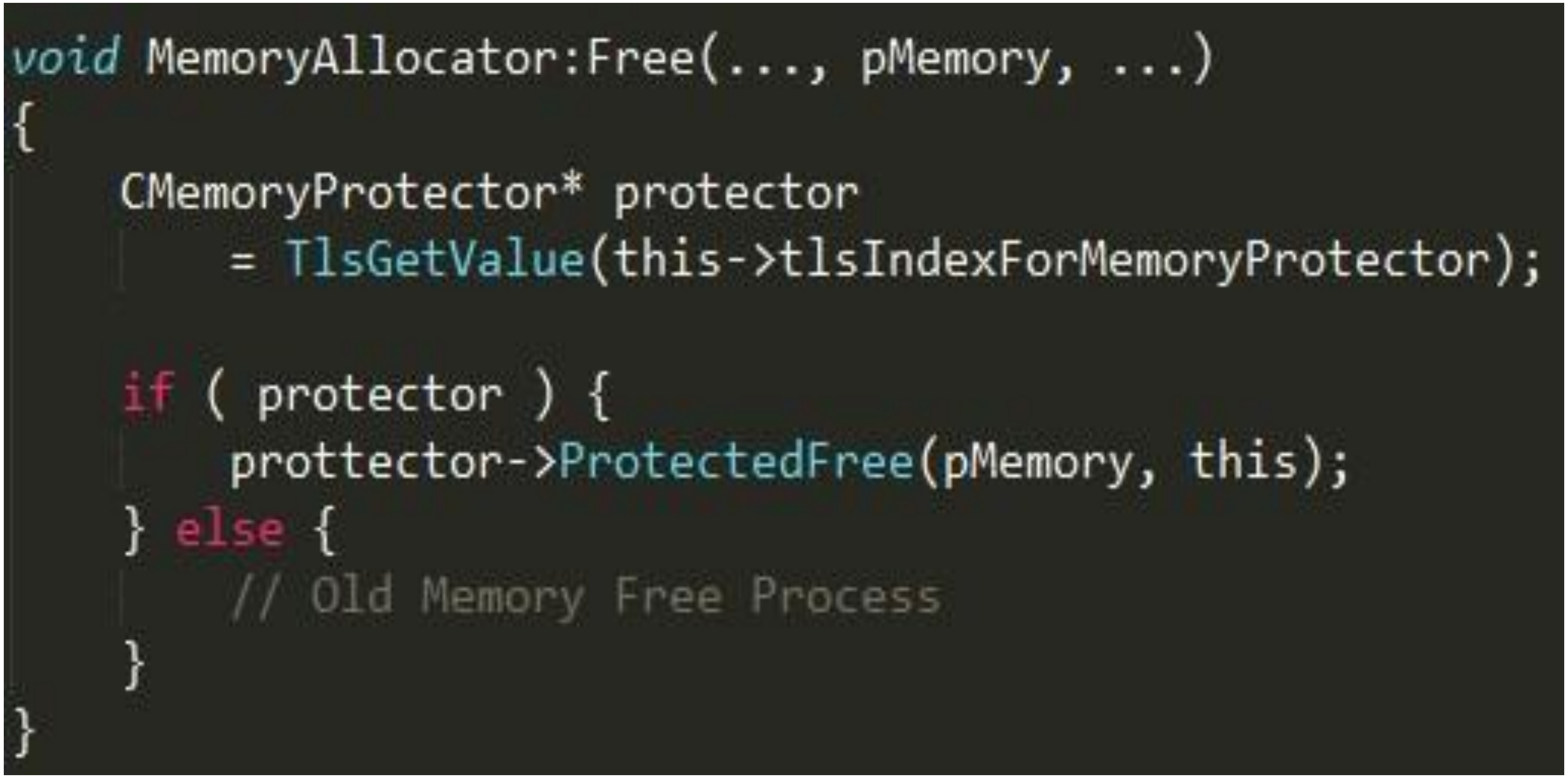
- Used first by Microsoft IE/Edge to mitigate use after free exploits
  - Aka. Deferred Free
  - Proven very effective

- Why memory protector in flash?
  - Many exploitable (exploited) vulnerabilities in flash player are use after free vulnerabilities

Memory Protector

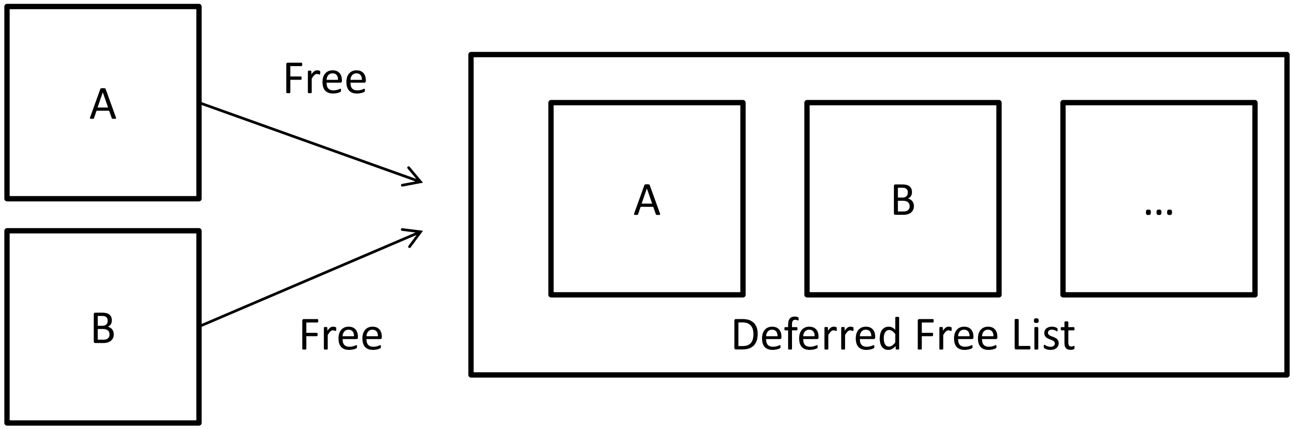
- When an element is freed
  - It’s memory is not freed immediately
  - Instead it is added to a deferred free list
  - The list will be iterated later (when newly freed memory size > threshold)
  - Memory block which meets the free criteria will be freed
- The free criteria
  - There must not be any reference to the memory block on the stack

Flash Memory Protector

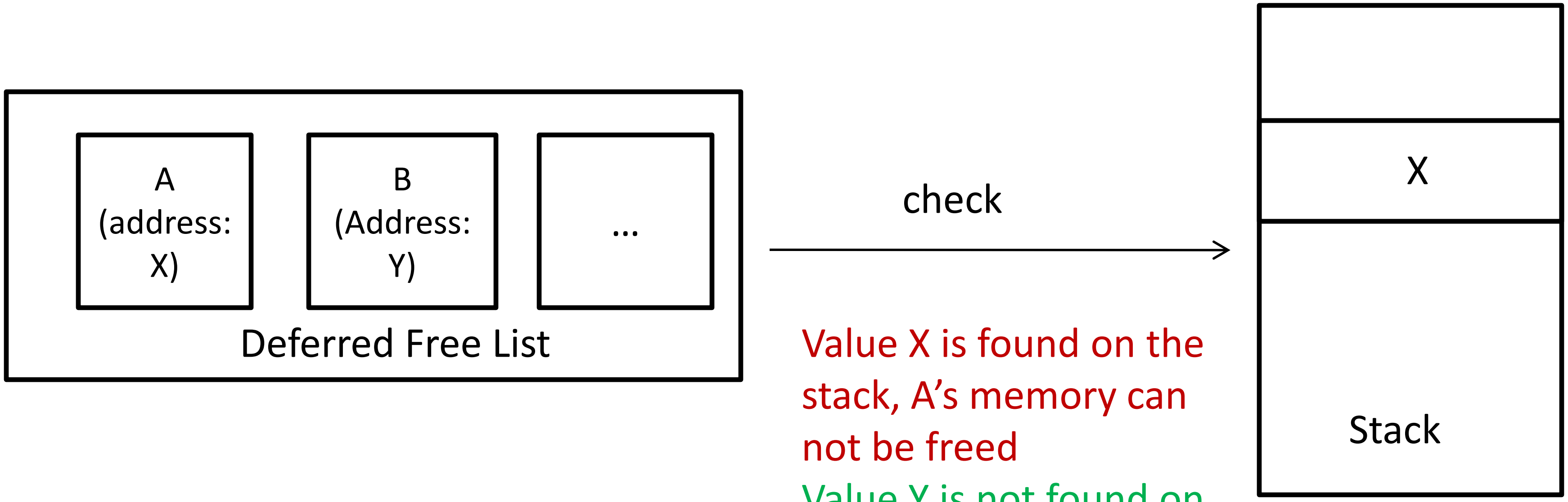


```
void MemoryAllocator:Free(..., pMemory, ...)
{
    CMemoryProtector* protector
        = TlsGetValue(this->tlsIndexForMemoryProtector);

    if ( protector ) {
        prottector->ProtectedFree(pMemory, this);
    } else {
        // Old Memory Free Process
    }
}
```



Free Stage

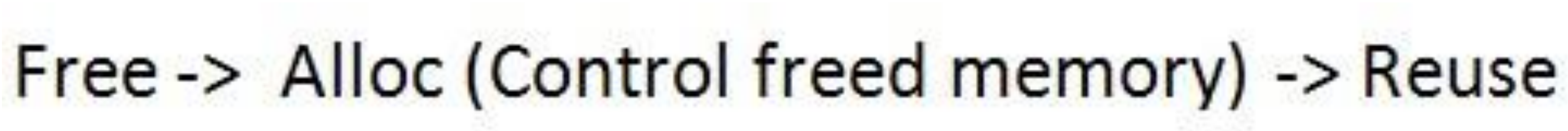


Value X is found on the stack, A’s memory can not be freed

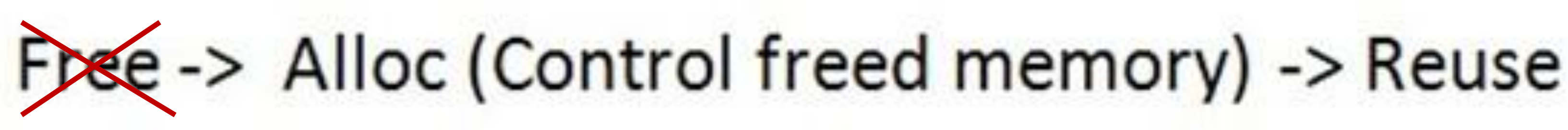
Value Y is not found on The stack, B’s memory will Be freed

Recycle Stage

Memory Protector Mitigation



Free -> Alloc (Control freed memory) -> Reuse



~~Free~~ -> Alloc (Control freed memory) -> Reuse

Flash Memory Protector – Effective?

- It would be OK **if** adobe **just make a full copy of** Microsoft's implementation directly

- **But** they made some changes in their own implementation

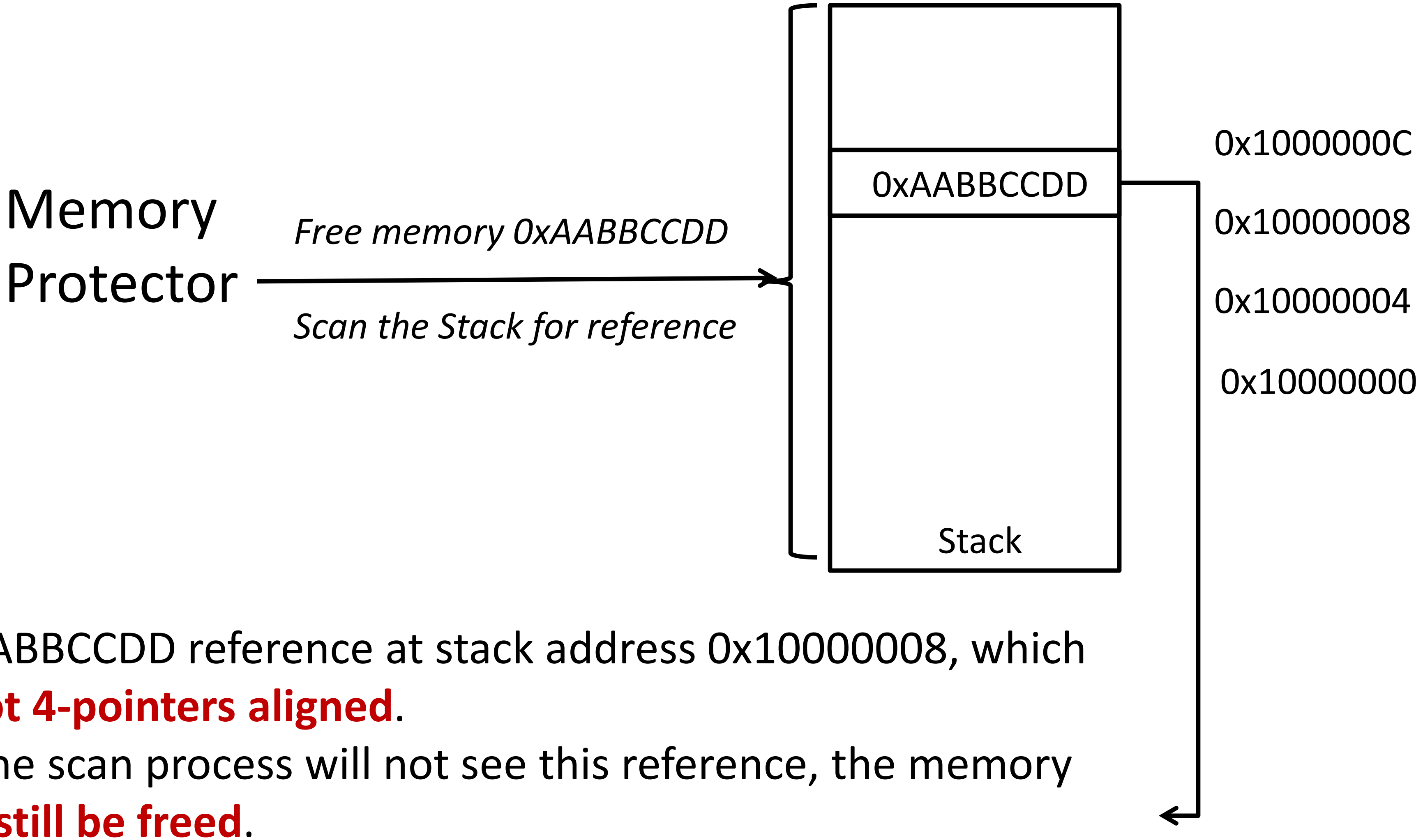
Problem of Flash Memory Protector

- Implementation contains trade-off
- Can help attacker to bypass ASLR
- Security Vulnerability

Implementation contains trade-off

```
.text:10724BDD          add     edi, 10h
.text:10724BE0          cmp     edi, [esi]
.text:10724BE2          jb      short
```

The stack scan checks **every 4 pointer** (not every pointer)
Why adobe implements it like this is mystery



Memory Protector

Free memory 0xAABBCCDD

Scan the Stack for reference

0xAABBCCDD

Stack

0x1000000C

0x10000008

0x10000004

0x10000000

0xAABBCCDD reference at stack address 0x10000008, which is **not 4-pointers aligned**.

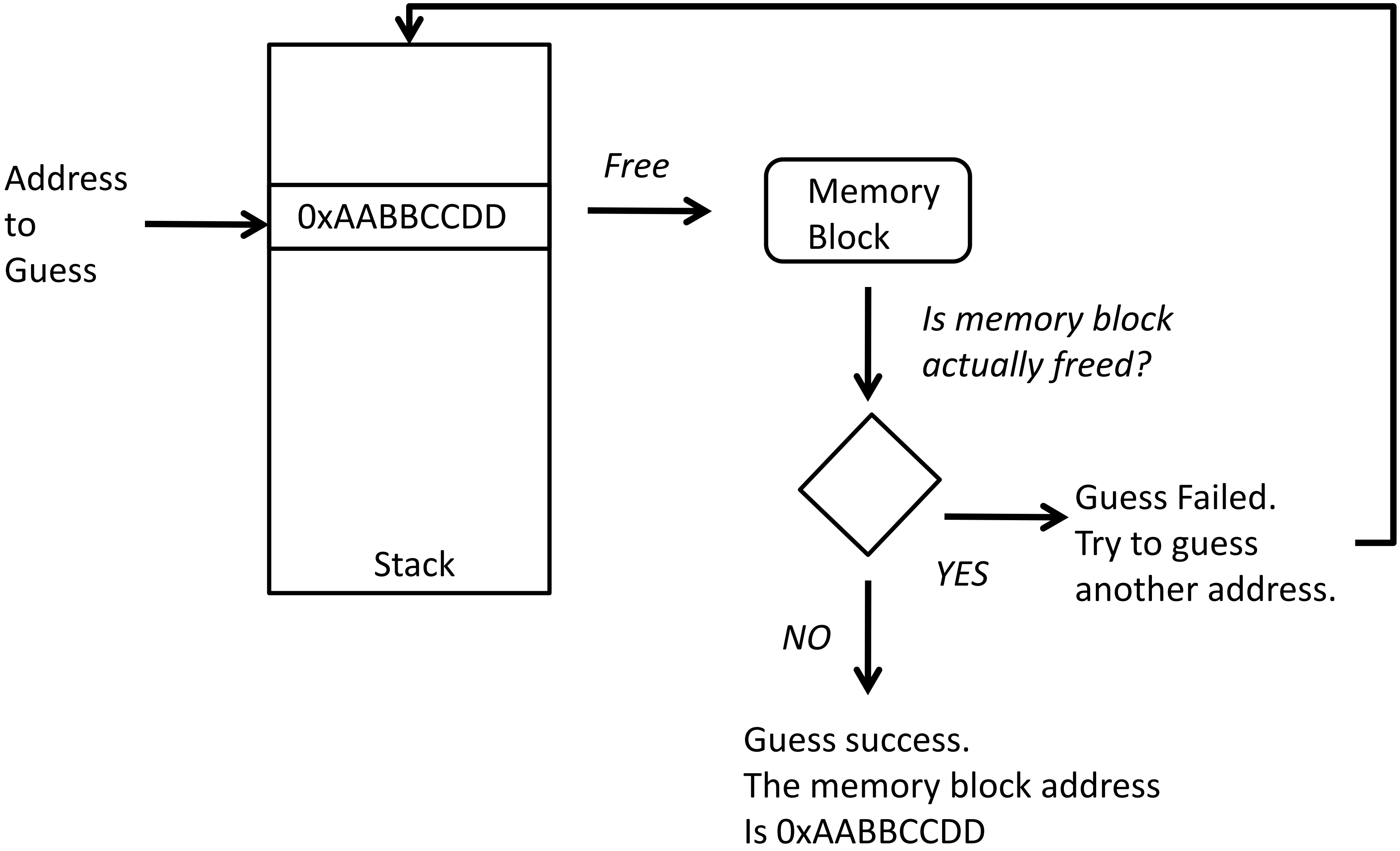
So the scan process will not see this reference, the memory **will still be freed**.

The use after free **vulnerability will still be triggered**.

ASLR Bypass using Memory Protector

- The stack scan process can not distinguish between pointer and data

- We can guess the address of a memory block:
  - Put the guess address (e.g. 0xaabbccdd) on stack
  - Free the memory block and trig reclaim
  - Check whether the memory block is actually freed, if it is not freed, then 0xaabbccdd should be the address of this block



Address to Guess

0xAABBCCDD

Stack

Free

Memory Block

Is memory block actually freed?

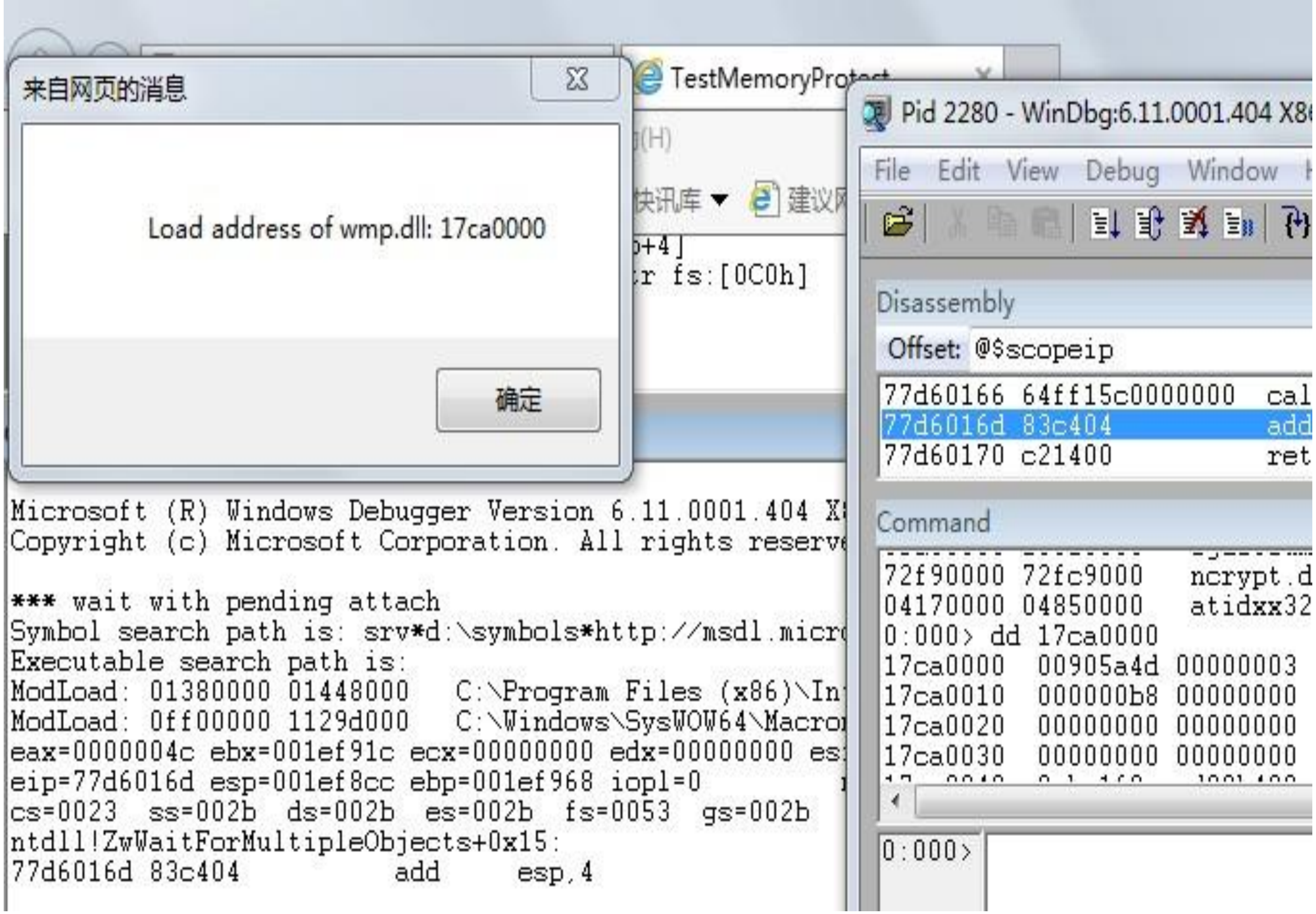
YES

Guess Failed.
Try to guess another address.

NO

Guess success.
The memory block address
Is 0xAABBCCDD

ASLR Bypass - Demo



Load address of wmp.dll: 17ca0000

```
b+4]
tr fs:[0C0h]
```

```
77d60166 64ff15c0000000   cal
77d6016d 83c404           add
77d60170 c21400           ret
```

```
72f90000 72fc9000   ncrypt.d
04170000 04850000   atidxx32
0:000> dd 17ca0000
17ca0000  00905a4d 00000003
17ca0010  000000b8 00000000
17ca0020  00000000 00000000
17ca0030  00000000 00000000
```

```
0:000>
```

```
Microsoft (R) Windows Debugger Version 6.11.0001.404 X8
Copyright (c) Microsoft Corporation. All rights reserve

*** wait with pending attach
Symbol search path is: srv*d:\symbols*http://msdl.micro
Executable search path is:
ModLoad: 01380000 01448000   C:\Program Files (x86)\In
ModLoad: 0ff00000 1129d000   C:\Windows\SysWOW64\Macro
eax=0000004c ebx=001ef91c ecx=00000000 edx=00000000 es
eip=77d6016d esp=001ef8cc ebp=001ef968 iopl=0
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
ntdll!ZwWaitForMultipleObjects+0x15:
77d6016d 83c404          add     esp,4
```

Security Vulnerability

- Memory protector uses a fixed size (0x400 items) array to store memory blocks

```
if (this->dwCount >= 0x400 || this->totalSize >= 0x186a0) {
        // Reclaim memory blocks in this->pBlocks
}

this->pBlocks[this->dwCount ++] = newBlock;
```

Figure out where the bug is, you have 5 seconds

Security Vulnerability

- Consider the following situation

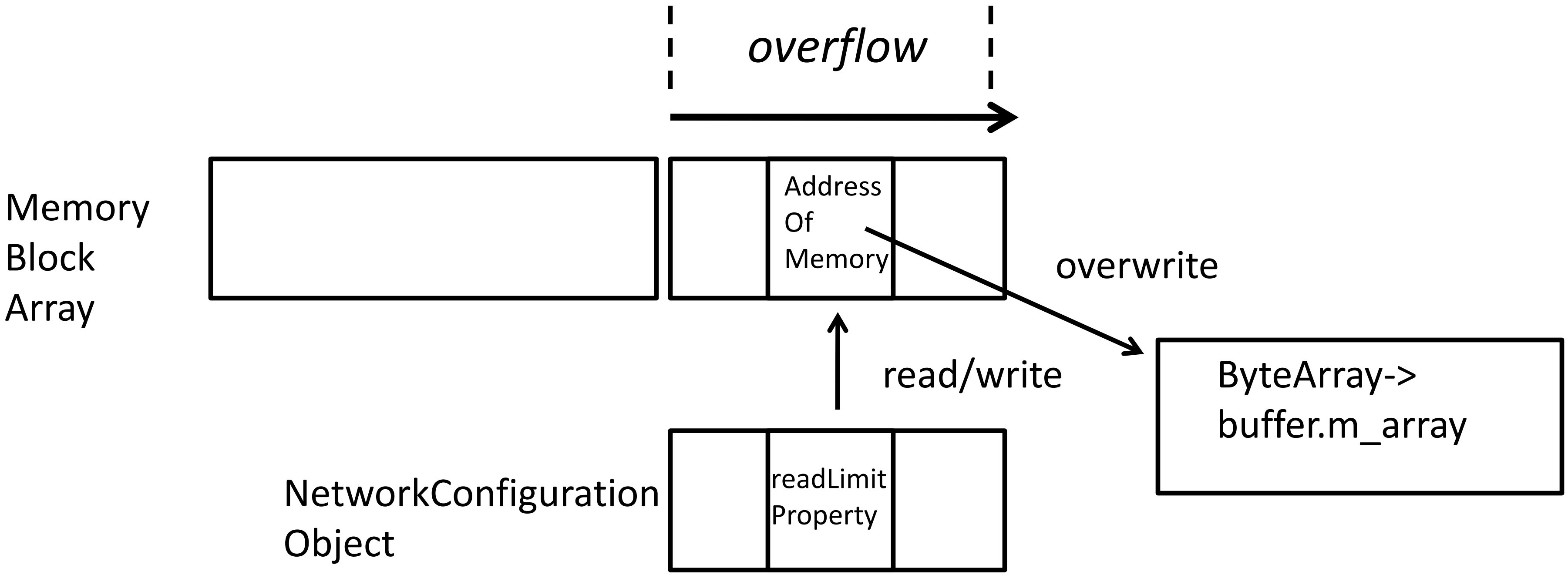
```
if (this->dwCount >= 0x400 || this->totalSize >= 0x186a0) {
        // Reclaim memory blocks in this->pBlocks
        // if all 0x400 blocks in the array has reference
        on the stack, then non of them will be reclaimed
}

this->pBlocks[this->dwCount ++] = newBlock; // overflow!
```

A buffer overflow in the exploit mitigation?

Exploit the Exploit Mitigation (Step 1)

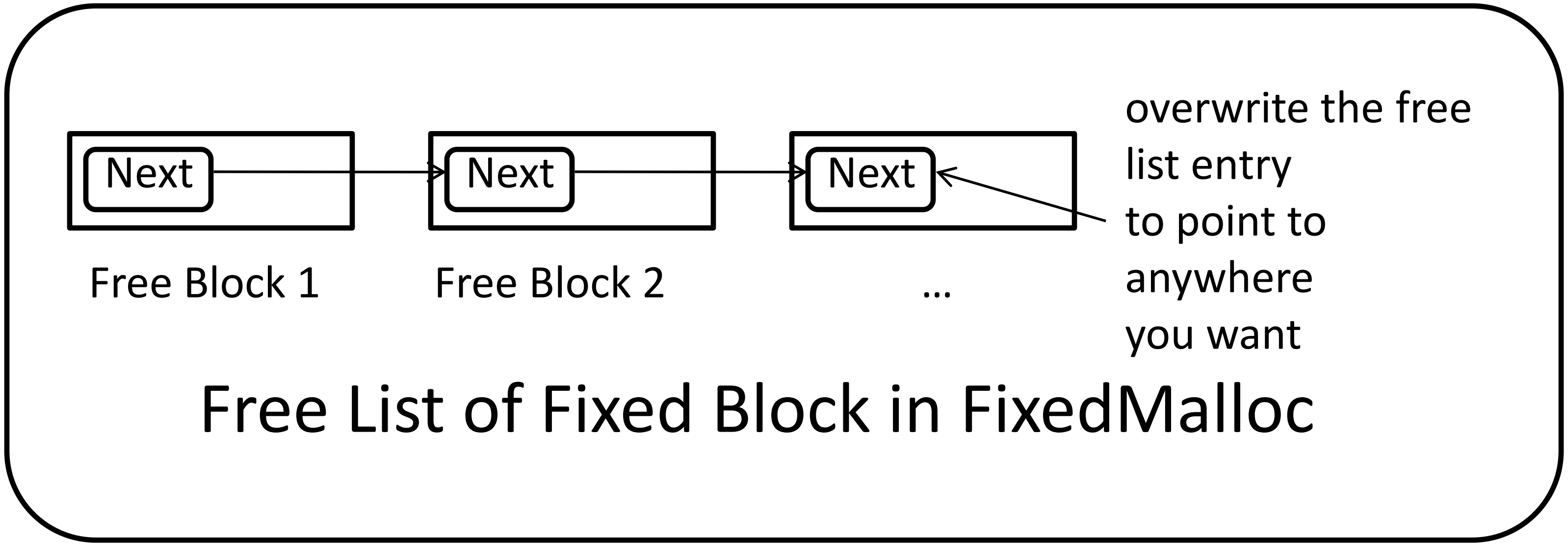
- Heap Overflow -> Use After Free



By overwriting the memory address in the memory protector array, we can make memory protector to free arbitrary address we want.

Exploit the Exploit Mitigation (Step 2)

- Use After Free -> Memory Overlapping
- ByteArray->buffer.m_array is allocated with FixedMalloc (not system heap)



Next

Next

Next

Free Block 1

Free Block 2

…

overwrite the free list entry to point to anywhere you want

Free List of Fixed Block in FixedMalloc

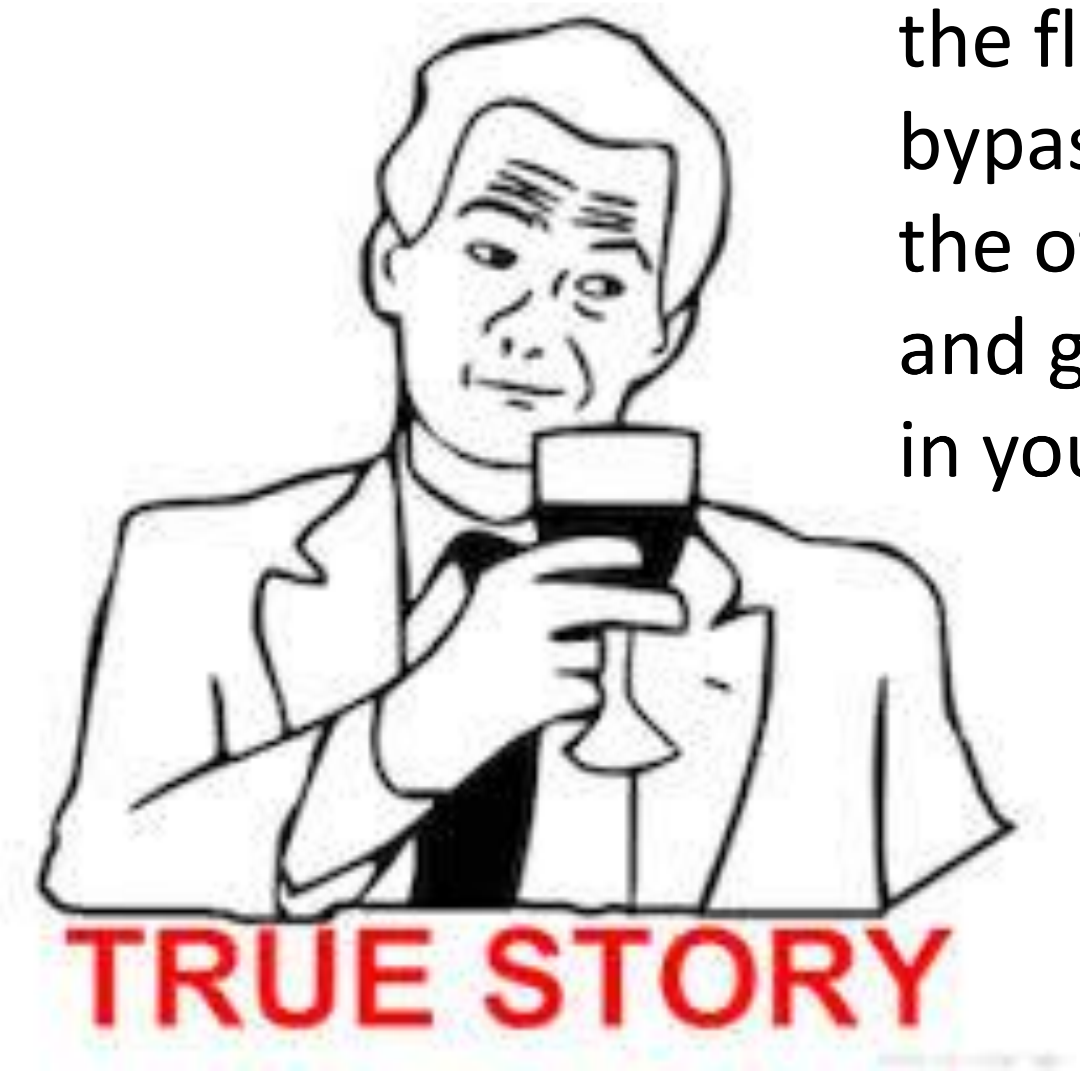
Exploit the Exploit Mitigation (Step 3)

- Allocate a new ByteArray whose length is the same with the free block

- You get a ByteArray which can read/write the arbitrary address pointed by the fake free list entry

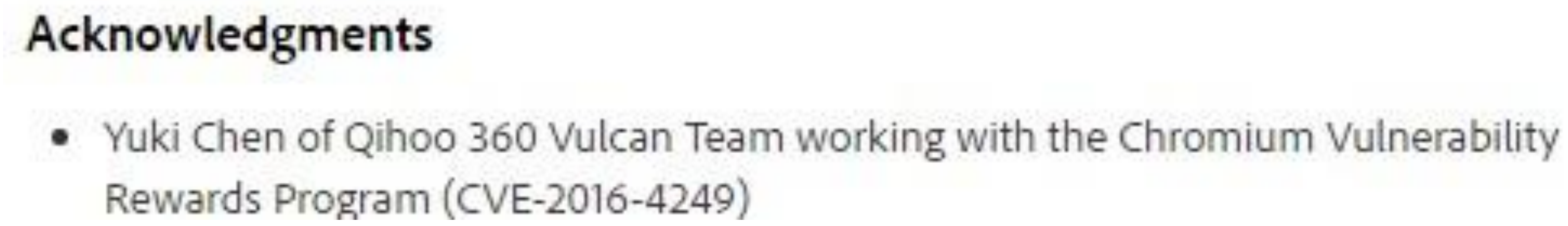
Exploit the Exploit Mitigation (Demo)

So I exploited a bug in
the flash exploit mitigation,
bypassed all of
the other mitigations,
and got RCE
in your browser.



Adobe’s Fix on this Bug

- Reported to adobe at 17th June
- Fixed in July security update as CVE-2016-4249



Acknowledgments

- Yuki Chen of Qihoo 360 Vulcan Team working with the Chromium Vulnerability Rewards Program (CVE-2016-4249)

- The End of the Story?
  - No

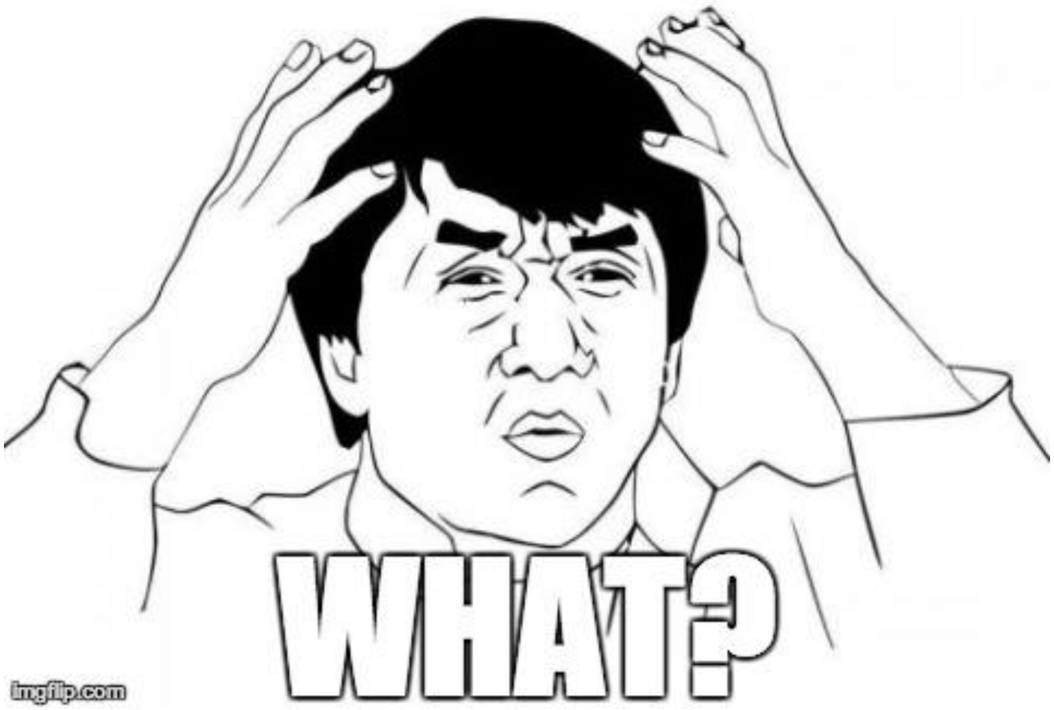
```
if (this->dwCount >= 0x400 || this->totalSize >= 0x186a0) {
    // Reclaim memory blocks in this->pBlocks
}

If (this->dwCount >= 0x400) {
    // Just free the memory
    return;
}

this->pBlocks[this->dwCount ++] = newBlock;
```

Just free the memory directly?

But you are memory protector right ?



Just free the memory…

Free the memory…

The memory…

Memory …

Adobe’s Fix on this Bug

- This fix just makes memory protector **useless in some condition**
- We only need to **make the blocks array full** while all of the blocks in the array have references on the stack
- After that, **any memory block will be directly freed** just like there is no memory protector at all

Future of Flash Exploits Under the latest Mitigation

- The percentage of useable bugs decreased
  - Especially for 64-bits target
- But high quality bugs can still survive
  - Type Confusion
  - Out-of-bounds array R/W

CVE-2016-1015

- The exploit we demonstrated in pwn2own 2016
- Type confusion
  - A NetConnection object could be confused to **any other** object
  - Could be easily converted to out-of-bounds r/w, uaf, …

CVE-2016-1016 + CVE-2016-1017

- Another exploit we used in pwn2own 2016
- Combination of 2 use after free bugs
  - Info Leak + Arbitrary Write
- Less affected by the heap mitigations
  - Because they are in GC Memory

CVE-2016-4117

- 0day exploited in the wild
- Type confusion bug
  - Type confuse a script object to another type
  - Exploit process:
    - Confuse a sub-class of ByteArray to another class
    - Leak the XOR key
    - Make a fake ByteArray with length 0xffffffff with the leaked key
    - Get arbitrary memory R/W

Agenda

- Who am I
- Background
- Flash Exploit Mitigations
- Conclusion

Conclusion

- Adobe added many good mitigations into flash player since July 2015
  - Length cookie
  - Isolated heap
  - System heap
  - Memory protector

- Although neither of them is perfect, these mitigations really raised the difficulty of writing a working flash exploit in the latest OS

Join Us

- Security Researcher
  - Brower/Kernel/Virtualization
  - Vulnerability/Exploiting Technique
- Full Time/Internship/Remote

360Vulcan

live long and pwn