

風水

Heap Feng Shui in JavaScript

Alexander Sotirov
asotirov@determina.com

- What is Heap Feng Shui?
 - the ancient art of arranging heap blocks in order to redirect the program control flow to the shellcode
- Heap Feng Shui in JavaScript
 - precise application data overwrites
 - reliable browser exploitation

- State of the art in browser exploitation
- Internet Explorer heap internals
- HeapLib JavaScript library
- Heap manipulation
- Mitigation

Part I

State of the art in browser exploitation

Stack overflows

Very hard to exploit in most cases:

Target	Protection
return address	stack cookies (/GS flag)
SEH frame	SafeSEH exception handler table
local variables	local variable reordering in the Visual C++ compiler

Heap overflows

Generic heap exploitation is also difficult:

Target	Protection
doubly-linked list of free chunks	safe unlinking
heap chunk header	8-bit header cookie in XP, XOR of the header data in Vista
lookaside linked list	removed in Vista

What's left?

- Non-array stack overflows
 - very rare
- Use of uninitialized variables
 - stack variables
 - use after free
- Application data on the heap
 - application specific memory allocators
 - function pointers
 - C++ object pointers

WebView setSlice exploit

- Uses heap spraying to fill the browser heap with shellcode
- Overwrites application data in the previous heap chunk
- Multiple attempts until it either hits an object pointer, or crashes

Heap spraying

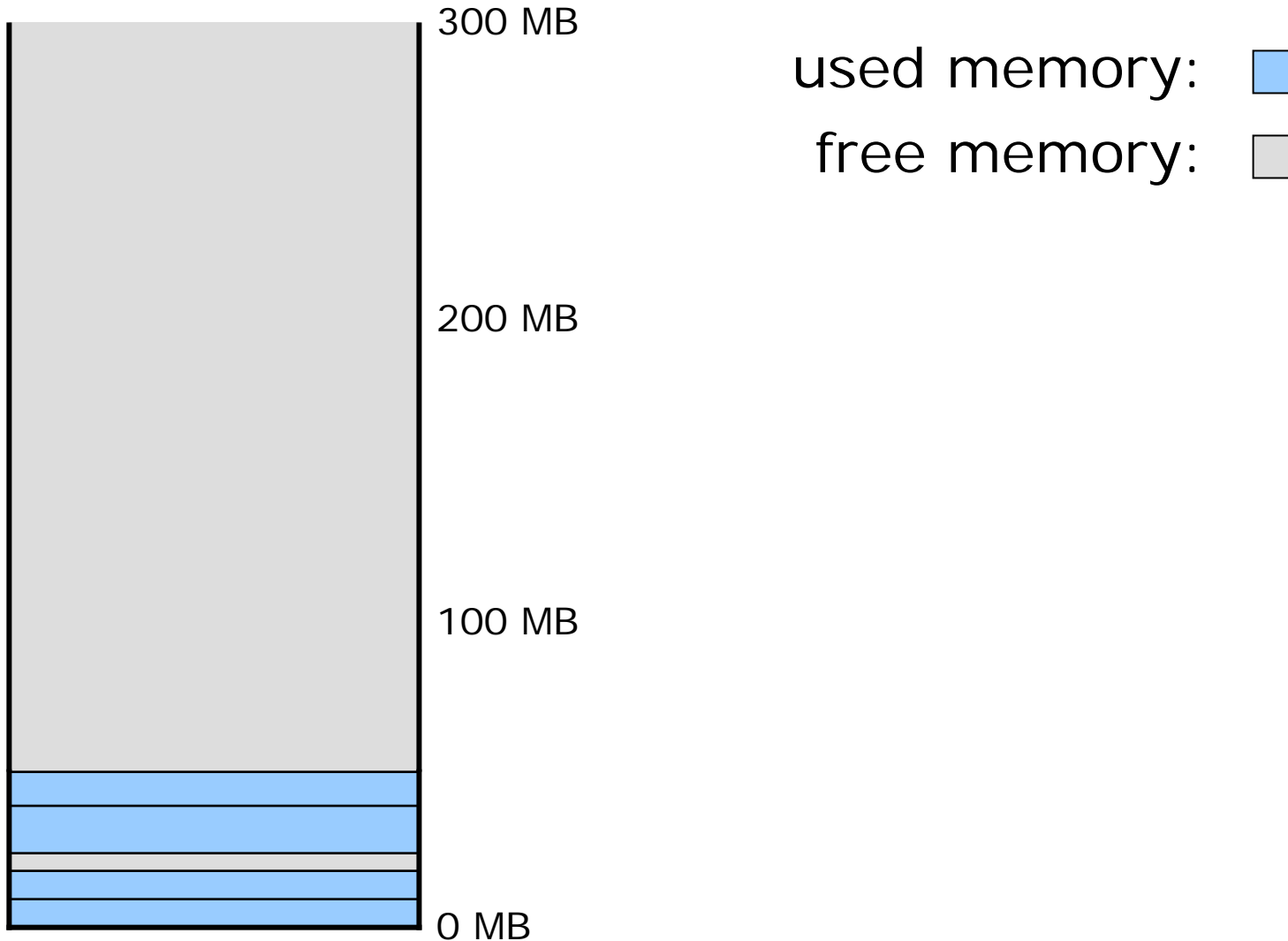
Developed by Blazde and SkyLined, used by most browser exploits since 2004.

```
var x = new Array();

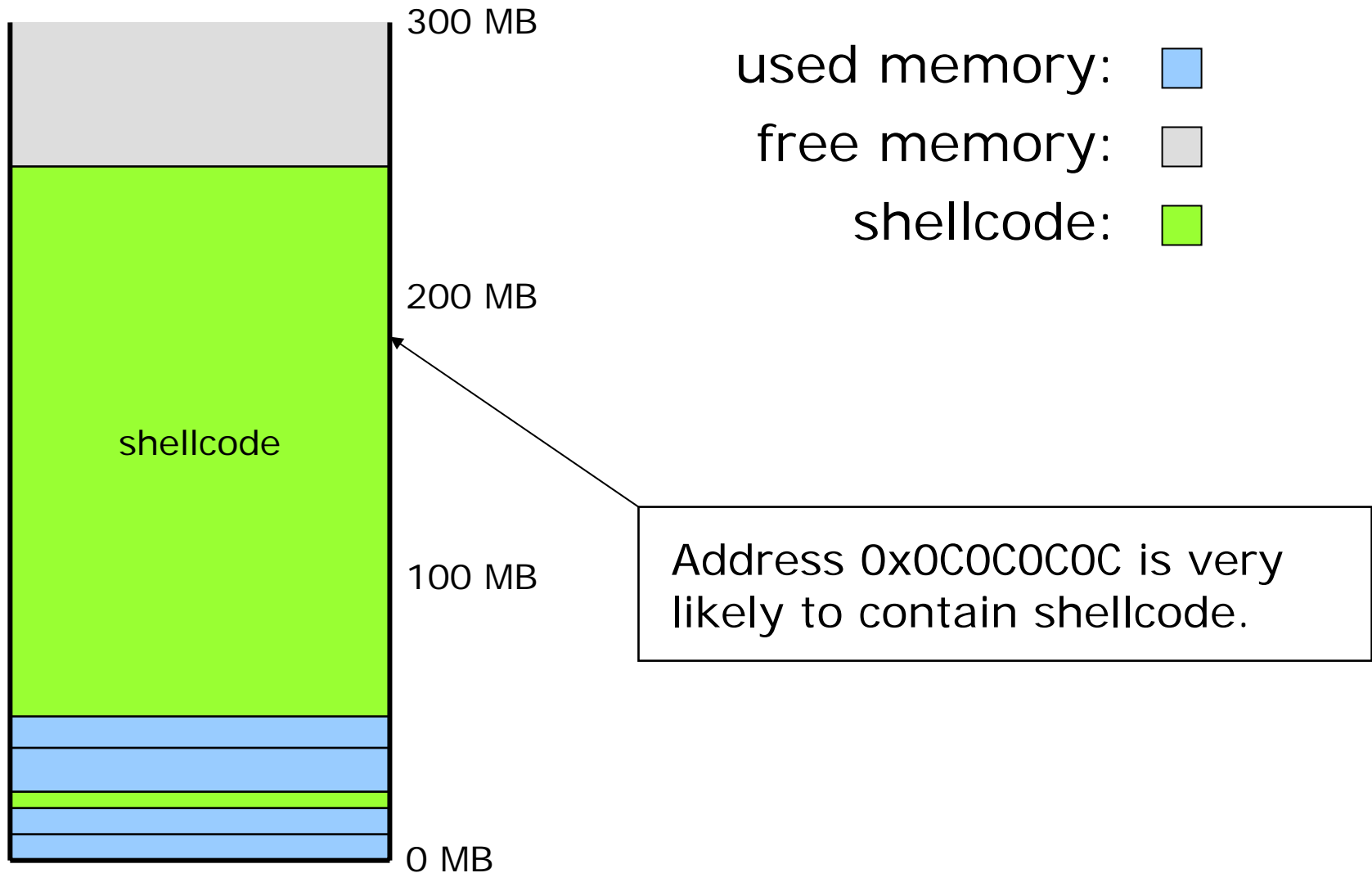
// Fill 200MB of memory with copies of the
// NOP slide and shellcode

for (var i = 0; i < 200; i++) {
    x[i] = nop + shellcode;
}
```

Normal heap layout

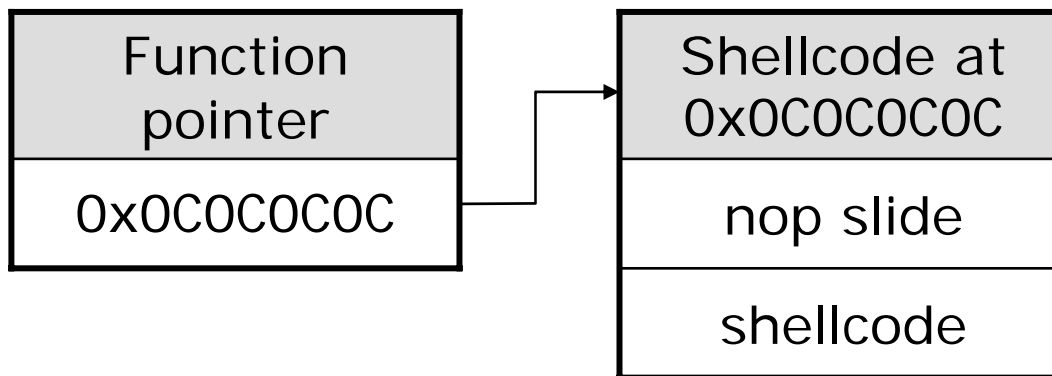


After heap spraying



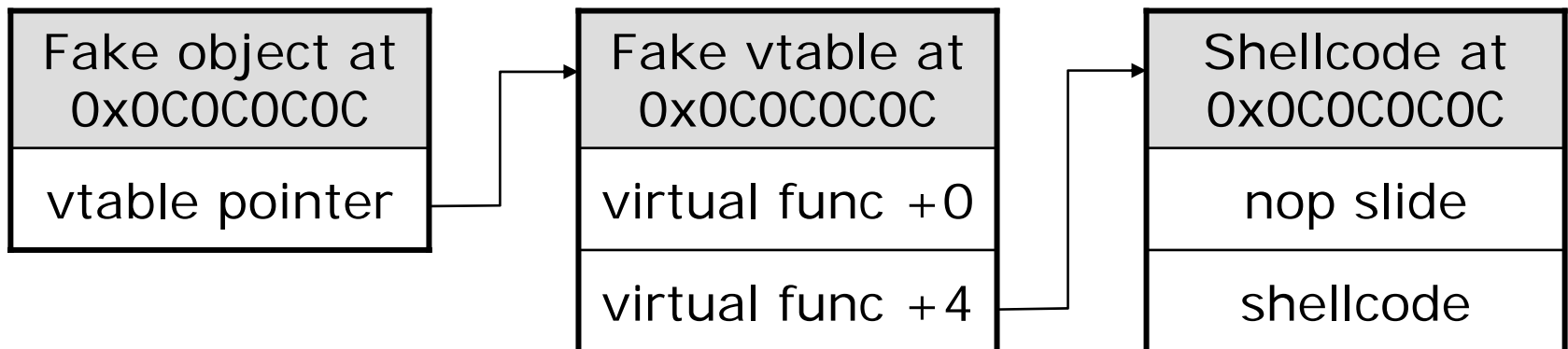
Function pointer overwrite

1. Spray the heap with 200MB of shellcode
2. Overwrite a function pointer with 0x0C0C0C0C
3. Call the function pointer



Object pointer overwrite

1. Spray the heap with 200MB of shellcode, using byte 0xC as a nop slide
2. Overwrite an object pointer with 0x0C0C0C0C
3. Call a virtual function of the object



Unreliable exploitation

- Heap spraying is a great technique, but the setSlice exploit is still not reliable
- Overwriting application data requires a specific layout of heap chunks
- We need to control the heap state

Part II

Heap Feng Shui

Heap Feng Shui

- The heap allocator is deterministic
- Specific sequences of allocations and frees can be used to control the layout



used: 

free: 

Heap Feng Shui

- The heap allocator is deterministic
- Specific sequences of allocations and frees can be used to control the layout



used: 

We allocate two 4KB blocks

free: 

our data: 

Heap Feng Shui

- The heap allocator is deterministic
- Specific sequences of allocations and frees can be used to control the layout



used: 

We free the first 4KB block

free: 

our data: 

Heap Feng Shui

- The heap allocator is deterministic
- Specific sequences of allocations and frees can be used to control the layout



used: 

The application allocates a 4KB
block and reuses our data

free: 

our data: 

Heap Feng Shui

- The heap allocator is deterministic
- Specific sequences of allocations and frees can be used to control the layout



used: 

We just exploited an uninitialized data vulnerability

free: 

our data: 

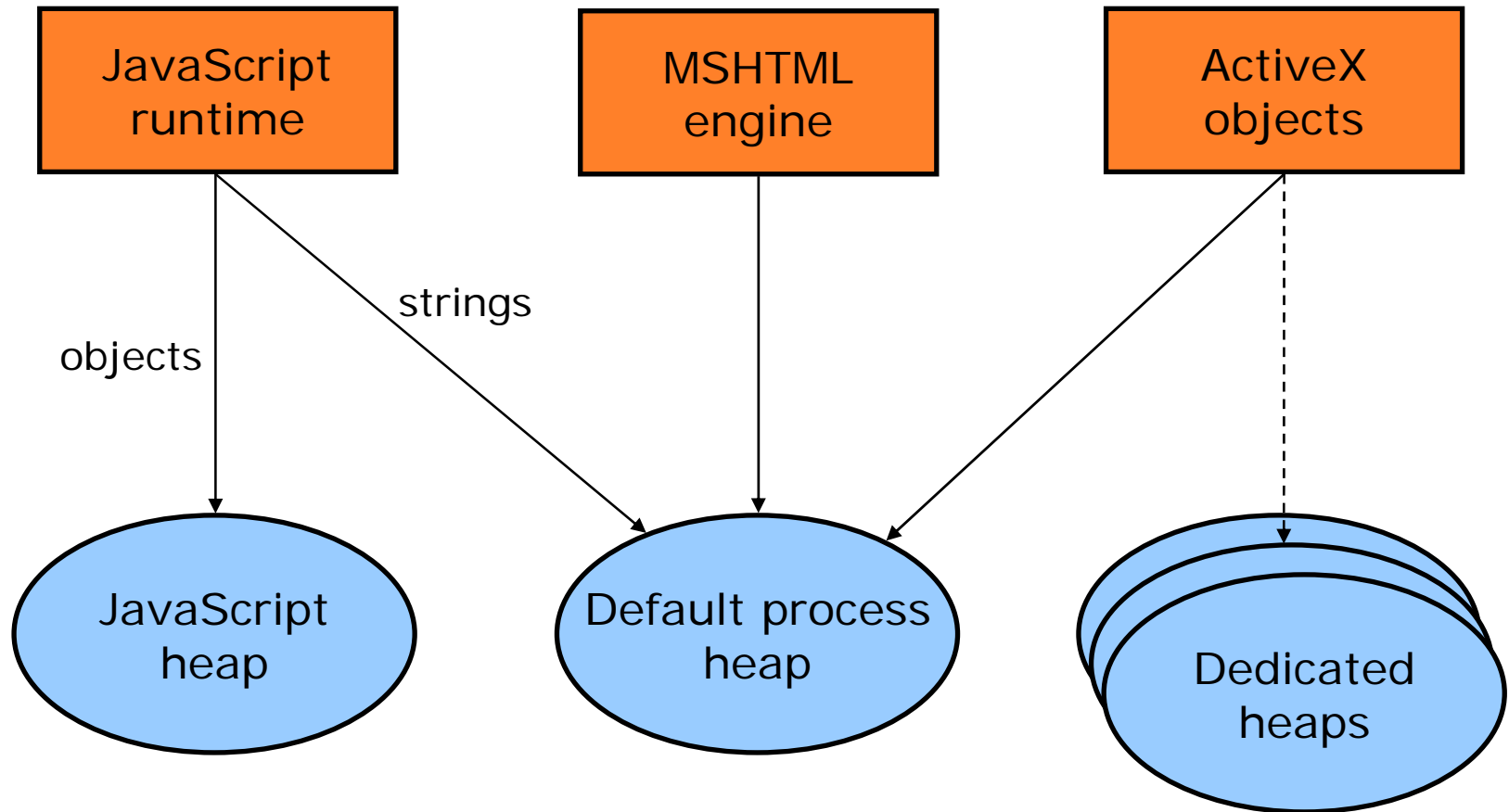
Heap Feng Shui in JavaScript

- We want to set the heap state before triggering a vulnerability
- Heap spraying proves that JavaScript can access the system heap
- We need a way to allocate and free blocks of an arbitrary size

Part III

Internet Explorer heap internals

Internet Explorer heap usage



JavaScript strings

The string "AAAA" is stored as:

string size	string data	null terminator
4 bytes	length / 2 bytes	2 bytes
08 00 00 00	41 00 41 00 41 00 41 00	00 00

We can calculate its size in bytes with:

$$\text{bytes} = \text{len} * 2 + 6$$

$$\text{len} = (\text{bytes} - 6) / 2$$

String allocation

```
var str1 = "AAAAAAAAAA"; // no allocation
```

```
// allocates a 10 character string
```

```
var str2 = str1.substr(0, 10);
```

```
// allocates a 20 character string
```

```
var str3 = str1 + str2;
```

String garbage collection

- Mark-and-sweep algorithm, frees all unreferenced objects
- Triggered by a number of heuristics
- Explicitly by the `CollectGarbage()` call in Internet Explorer

JavaScript alloc and free

```
var padding = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..."
var str;

function alloc(bytes) {
    str = padding.substr(0, (bytes-6)/2);
}

function free() {
    str = null;
    CollectGarbage();
}

alloc(0x10000); // allocate 64KB memory block
free();        // free memory block
```

OLEAUT32 allocator

Not all string allocations and frees reach the system memory allocator

- custom memory allocator in OLEAUT32
- caching of free memory blocks
- 4 bins for blocks of different sizes
- up to 6 free blocks stored in each bin

OLEAUT32 alloc function

bin = the right bin for the requested size

if (bin not empty)

 find a block in the bin > requested size

 if (found)

 return block

 else

 return sysalloc(size)

else

 return sysalloc(size)

OLEAUT32 free function

bin = the right bin for the block size

if (bin not full)

 add block to bin

else

 find the smallest block in the bin

 if (smallest block < new block)

 sysfree(smallest block)

 add new block to bin

 else

 sysfree(new block)

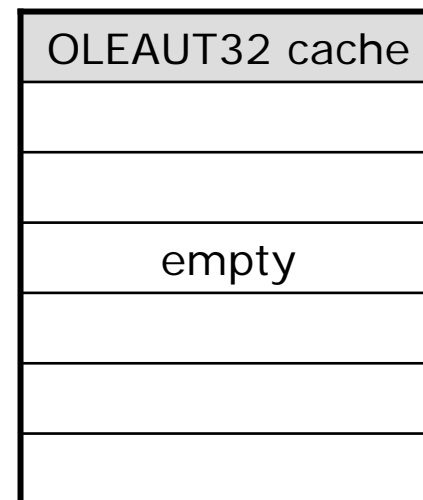
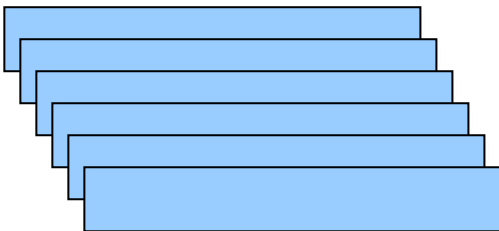
Bypassing the cache

- Our freed blocks will go into the cache
- Freeing 6 maximum sized blocks for each bin will push all smaller blocks out
- Allocating the 6 blocks again will leave the cache empty
- When the cache is empty, allocations will come from the system heap

Plunger Technique

- ▶ 1. Allocate 6 maximum size blocks
- 2. Allocate our blocks
- 3. Free our blocks
- 4. Free 6 maximum size blocks
- 5. Allocate 6 maximum size blocks

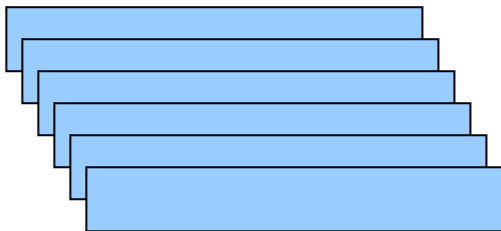
maximum size blocks



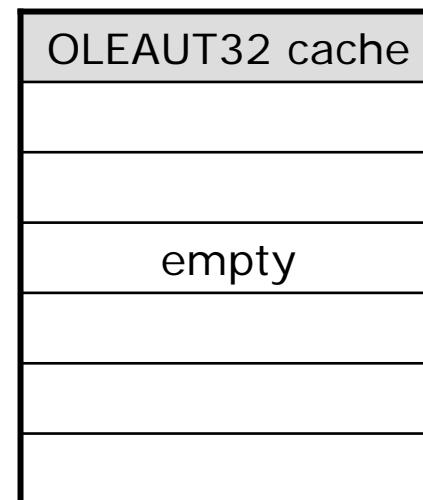
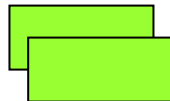
Plunger Technique

1. Allocate 6 maximum size blocks
- ▶ 2. Allocate our blocks
3. Free our blocks
4. Free 6 maximum size blocks
5. Allocate 6 maximum size blocks

maximum size blocks



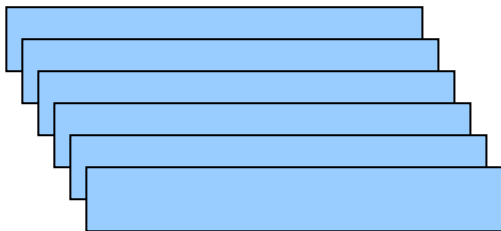
our blocks



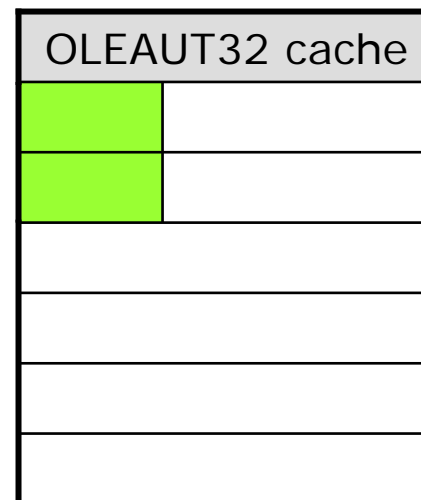
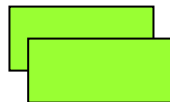
Plunger Technique

1. Allocate 6 maximum size blocks
2. Allocate our blocks
- ▶ 3. Free our blocks
4. Free 6 maximum size blocks
5. Allocate 6 maximum size blocks

maximum size blocks

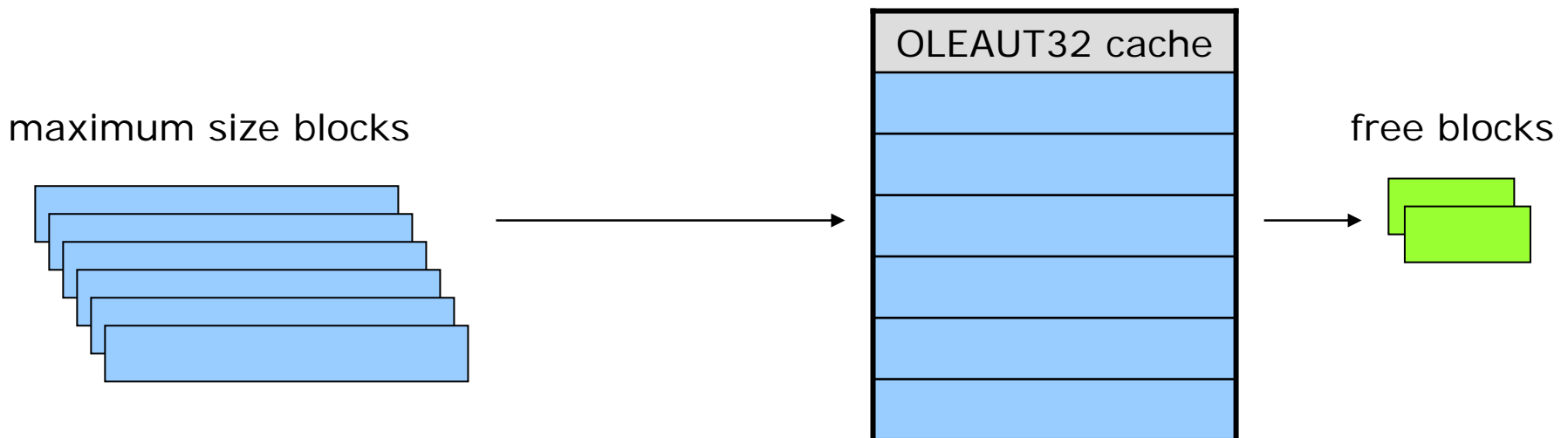


our blocks



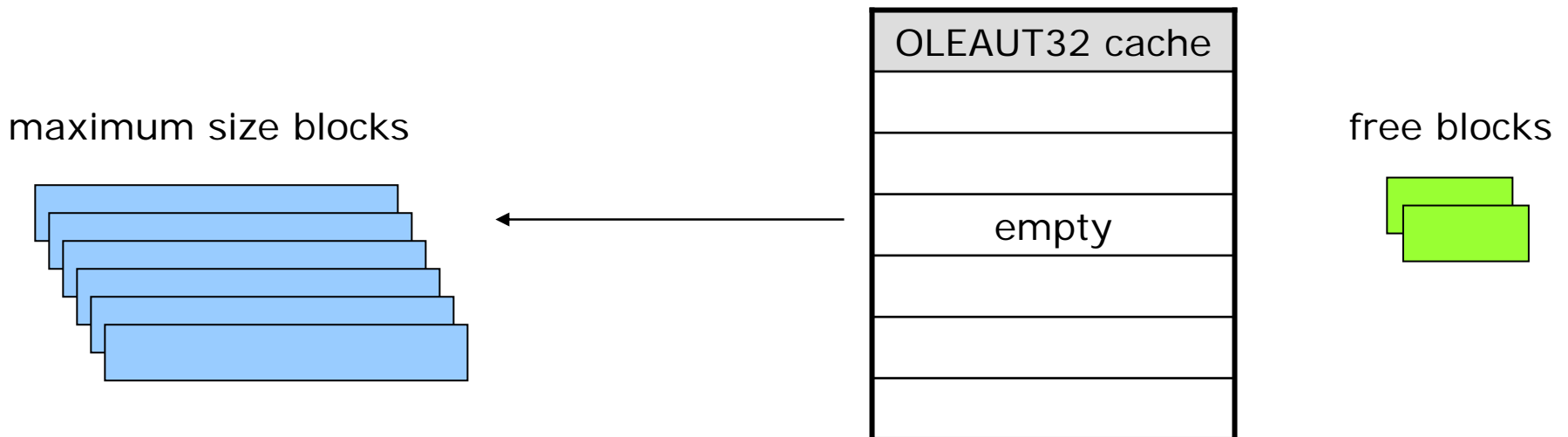
Plunger Technique

1. Allocate 6 maximum size blocks
2. Allocate our blocks
3. Free our blocks
- ▶ 4. Free 6 maximum size blocks
5. Allocate 6 maximum size blocks



Plunger Technique

1. Allocate 6 maximum size blocks
2. Allocate our blocks
3. Free our blocks
4. Free 6 maximum size blocks
- ▶ 5. Allocate 6 maximum size blocks



Part IV

HeapLib - JavaScript heap manipulation library

Introducing HeapLib

- Supports Internet Explorer 5-7
- Object oriented API
- Functions for:
 - heap logging and debugging
 - allocation and freeing of blocks with arbitrary size and contents
 - high-level heap manipulation function (not yet supported on Vista)

Hello world!

```
<scri pt src="heapLi b. j s" ></scri pt >
```

```
<scri pt >
```

```
  var heap = new heapLi b. i e();
```

```
  heap. gc();
```

```
  heap. debugHeap(true);
```

```
  heap. al l oc(512);
```

```
  heap. al l oc("BBBBB" ,  "foo");
```

```
  heap. free("foo");
```

```
  heap. debugHeap(fal se);
```

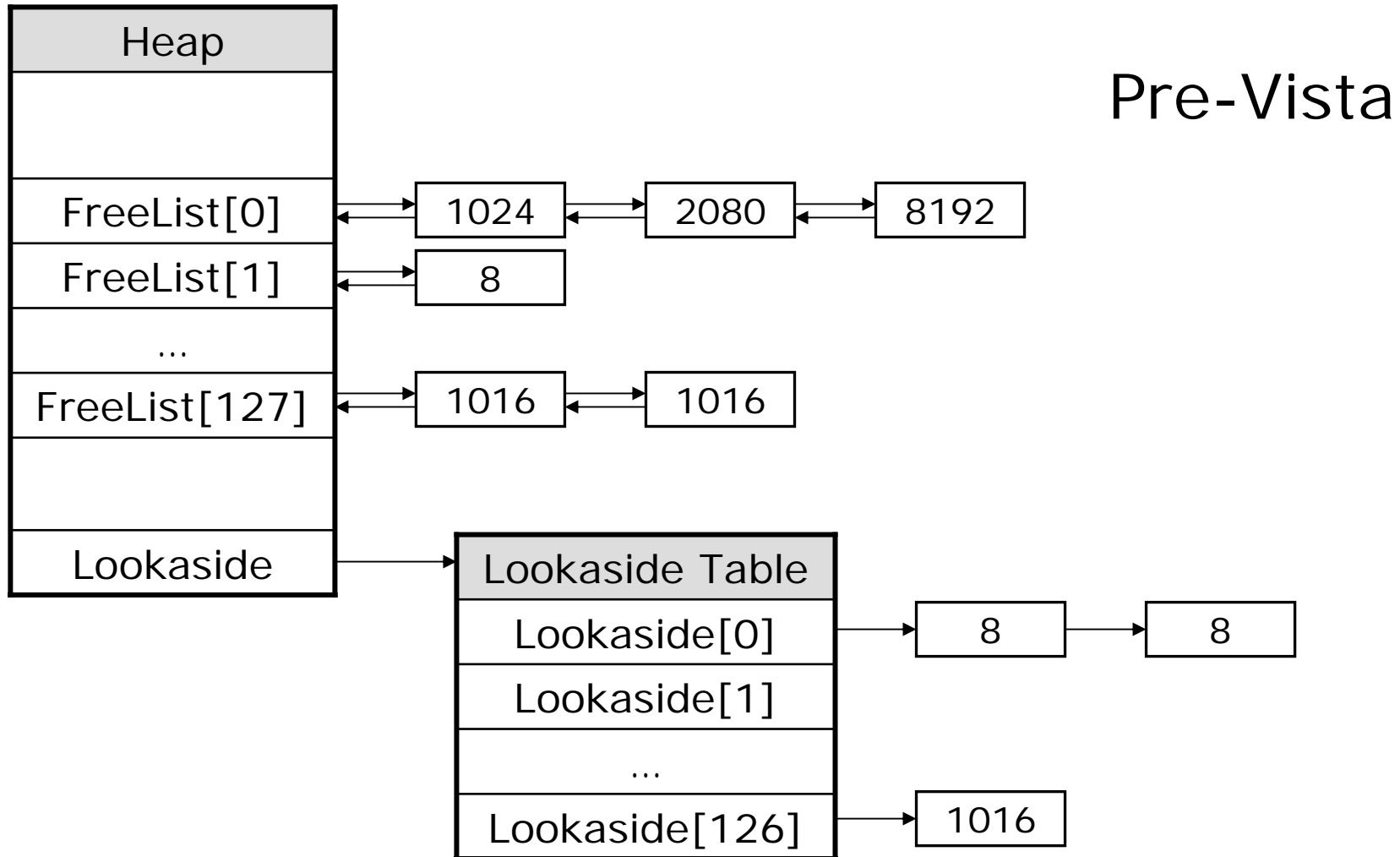
```
</scri pt >
```

HeapLib Demo

Part V

Windows Heap Manipulation

Windows Heap Overview



Free Algorithm

```
if size >= 512KB
```

```
    free with VirtualFree
```

```
    return
```

```
if size < 1KB and Lookaside not full
```

```
    add to Lookaside list
```

```
    return
```

```
coalesce block with free blocks around it
```

```
if size < 1KB
```

```
    add to FreeList[size/8]
```

```
else
```

```
    add to FreeList[0]
```

Allocate Algorithm

```
if size >= 512KB
    alloc with VirtualAlloc
    return

if size < 1KB
    if lookaside not empty
        return a block from the lookaside
    if FreeList[size/8] not empty
        return a block from FreeList[size/8]

if FreeList[0] not empty
    return a block from FreeList[0]

allocate more memory with VirtualAlloc
```

Defragmenting the heap

To allocate two consecutive blocks, we need to defragment the heap.

```
for (var i = 0; i < 1000; i++)  
    heap.alloc(0x2010);
```

used: 

free: 



Defragmenting the heap

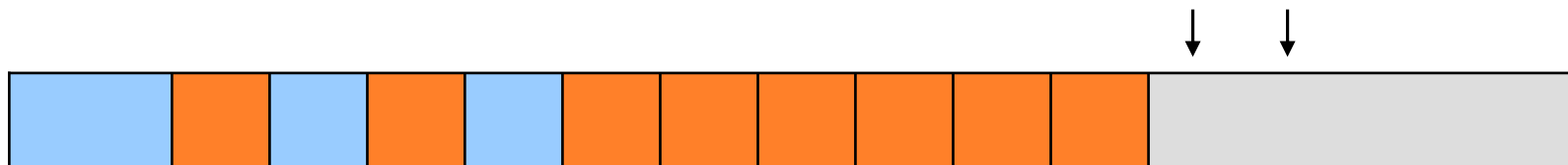
To allocate two consecutive blocks, we need to defragment the heap.

```
for (var i = 0; i < 1000; i++)  
    heap.alloc(0x2010);
```

used: 

free: 

our blocks: 



Putting a block on the FreeList

To put a block on the free list, we need to ensure that it is not coalesced.

```
heap. alloc(0x2010, "foo");
```

```
heap. alloc(0x2010);
```

```
heap. alloc(0x2010, "foo");
```

```
heap. free("foo");
```

used: 

free: 

our blocks: 



Putting a block on the FreeList

To put a block on the free list, we need to ensure that it is not coalesced.

```
heap. alloc(0x2010, "foo");
```

```
heap. alloc(0x2010);
```

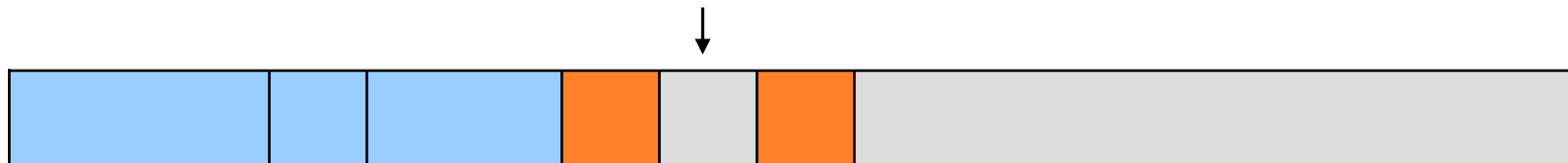
```
heap. alloc(0x2010, "foo");
```

```
heap. free("foo");
```

used: 

free: 

our blocks: 



Emptying the lookaside

To empty the lookaside, allocate enough blocks of the same size.

```
for (var i = 0; i < 100; i++)  
    heap.alloc(512);
```

Freeing to the lookaside

To put a block on the lookaside, empty it and free the block.

```
for (var i = 0; i < 100; i++)  
    heap.alloc(512);  
  
heap.alloc(512, "foo");  
heap.free("foo");
```

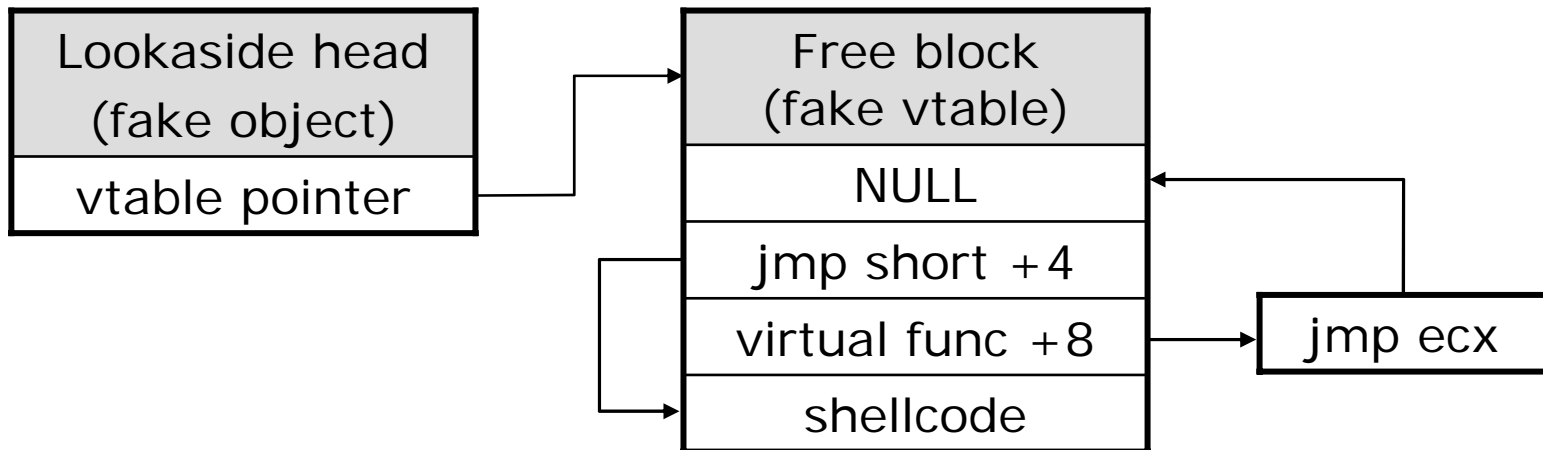
Object pointer overwrite

The lookaside linked list can be used to exploit object pointer overwrites without heap spraying.

1. Empty the lookaside
2. Build a fake vtable block
3. Free the fake vtable to the lookaside
4. Overwrite an object pointer with the address of the lookaside head
5. Call a virtual function of the object

Object pointer overwrite

```
mov ecx, dword ptr [eax] ; get the vtable address
push eax                 ; push the 'this' pointer
call dword ptr [ecx+08h] ; call virtual func
```



NULL disassembles as two `sub [eax], al` instructions

Exploit Demo

Mitigation

determina™

- Heap isolation
- Non-determinism in the heap allocator

Questions?

asotirov@determina.com