$hell on Earth:
From Browser to System Compromise

MATT MOLINYAWE | ABDUL-AZIZ HARIRI | JASIEL SPELMAN

Introduction

- Security Researchers
  - Matt Molinyawe - Twitter: @djmanilaice
  - Abdul-Aziz Hariri - Twitter: @abdhariri
  - Jasiel Spelman – Twitter: @WanderingGlitch
- Trend Micro – Zero Day Initiative
  - World's Largest Vendor Agnostic Bug Bounty Program
  - Focused on Vulnerability Discovery and Remediation
  - Research Advanced Exploitation Techniques
  - #1 in Vulnerabilities/CVEs for Microsoft and Adobe in 2015

Evolution of Exploit Development

PWN2OWN CASE STUDIES

Exploit Mitigations Prior to P2O 2013

- ASLR (Address Space Layout Randomization)
- DEP (Data Execution Prevention)
- Sandboxing technology
- Stack cookies
- Low Fragmentation Heap (Heap hardening)
- JS JIT mitigations
- SEHOP

Exploit Mitigations P2O 2013 - Present

- 2013
  - VTGuard – Virtual Table
  - ForceASLR
  - AppContainer
  - Pool Integrity Checks
  - Kernel ASLR
- 2014
  - EMET
  - PartitionAlloc
  - Java click to play
- 2015
  - Control Flow Guard
  - Isolated Heap
  - Memory Protection
  - win32k access prevention in Chrome
- 2016
  - Adobe Flash Isolated Heap
  - Adobe Flash Memory Protection

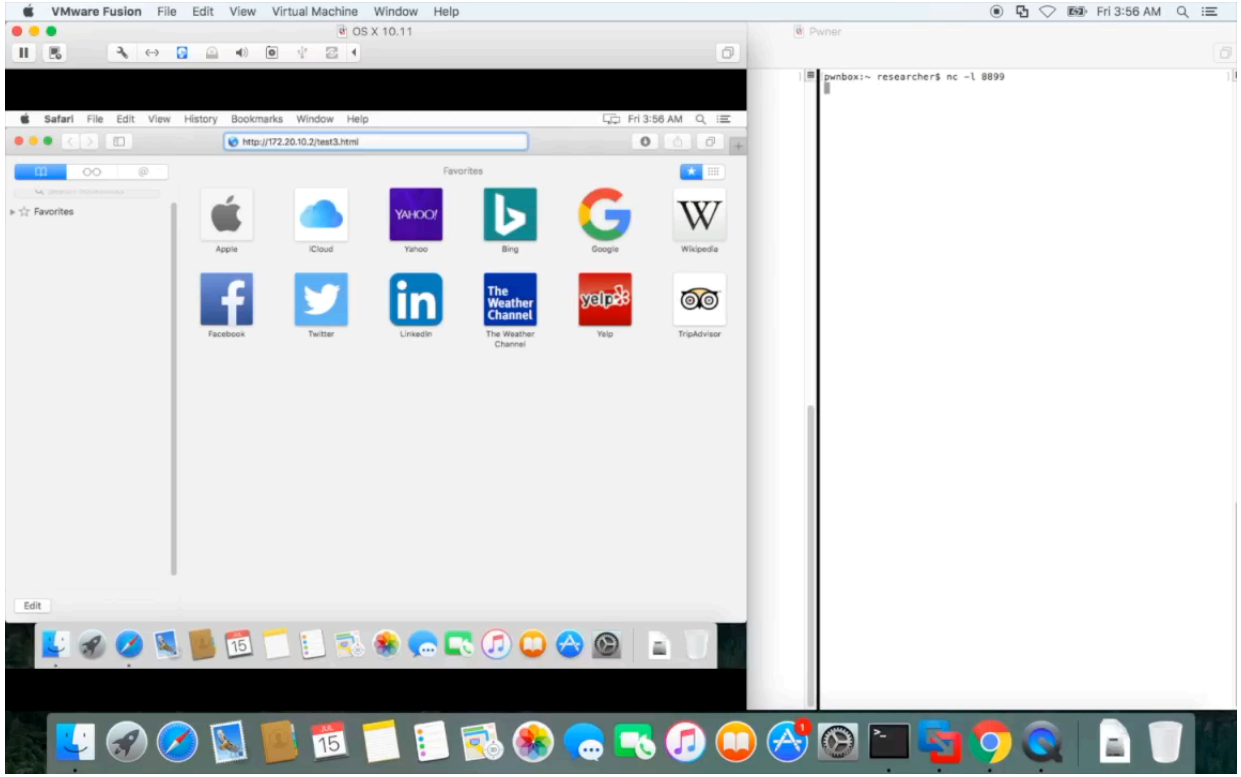
History of Pwn2Own Remote Browser to Kernel Exploits

- **Pwn2Own 2013 - Jon Butler and Nils**
  - Used a type confusion vulnerability in Google Chrome within SVG processing
  - Kernel vulnerability - NtUserMessageCall - exploiting a pool overflow in win32k.sys
- **Pwn2Own 2014 - Sebastian Apelt and Andreas Schmidt**
  - Two Internet Explorer 11 use-after-frees, which evaded ASLR/DEP
  - Kernel vulnerability - IOCTL 0X1207F and IOCTL 0X120C3 AFD.sys dangling pointer vulnerability
- **Pwn2Own 2015 - Peter Hlvaty, Jihui Lu, Zeguang Zhao (Team509), wushi (KeenTeam), Wen Xu, and Jun Mao (Tencent PCMgr)**
  - Exploited heap buffer overflows in Adobe Flash
  - Integer overflow and achieved pool corruption through True Type Font vulnerabilities
- **Pwn2Own 2015 – Mariusz Mlynski**
  - Same-origin policy violation and resource:// URL loading privileged pages
  - Used EMET's Windows Installer to uninstall EMET, which led to SYSTEM level privileges
- **Pwn2Own 2015 – Jung Hoon Lee (lokihardt)**
  - Buffer overflow race condition in Google Chrome and Google Chrome beta
  - Info disclosure and race condition within two Windows kernel drivers

Apple Safari to Kernel

TENCENT KEENLAB – TEAM SHIELD

Team Shield Exploit Demo



Apple Safari GraphicsContext UAF

- Proof of concept results in an access violation in the debugger

```
WebCore`WebCore::GraphicsContext::setPlatformTextDrawingMode:
0x10dec437d <+13>: mov rdi, qword ptr [rax]
```

- In pseudocode, the crash is here:

```
result = **(WebCore::GraphicsContext ***)this;
```

Exploitation Steps

1. Attained a write primitive from the UAF vulnerability
   - Initial spray to reclaim Graphics Context object of size 0x100
   - Primitive comes from WebCore::GraphicsContext::save() in Safari

```
qmemcpy((void *)(v3 + v4 + 16), (char *)this + 20, 0x4Du);
```

2. Second spray with the following memory layout

   `[String][Frame] [String][Frame] [String][Frame] [String][Frame]`

3. Bypassed ASLR utilizing a write primitive to enlarge the string object's length and leak a Frame object's vtable
4. Achieved execution with ROP and writing a vtable for the frame object
   - Calls .blur() which calls Webcore::jsElementPrototypeFunctionBlur

Apple OS X WindowServer MultiTouch UaF

- `_mthid_unserializeGestureConfiguration` function
  - `CFData` is checked and a free can occur
  - Race condition can occur where the `CFData` can be reclaimed by another thread

```
if ( v2 )
  {
    if ( !(unsigned __int8)_mthid_isGestureConfigurationValid(v2) )
      CFRelease(a1);
    v1 = v2;
  }
```

Race Condition Details

- Invoked by `_XSetGlobalForceConfig` which resides in `CoreGraphics` which the `WindowServer` utilizes
- Shortly after `_mthid_unserializeGestureConfiguration` is invoked, there is another call to a `CFRelease`

```
0x7fff88ec6ebd <+111>:  call _mthid_unserializesGestureConfiguration;
                        mov    r15, rax
                        test   r12, r12
                        je     0x7fff88ec6ed2  ; <+132>
                        mov    rdi, r12
                        call   0x7fff892c7c12  ; symbol stub for CFRelease
```

- If you are able to reclaim the data in time, you can fake out the method call within `objc_msgSend` when `CFRelease` is called and get direct code execution

Exploitation Steps

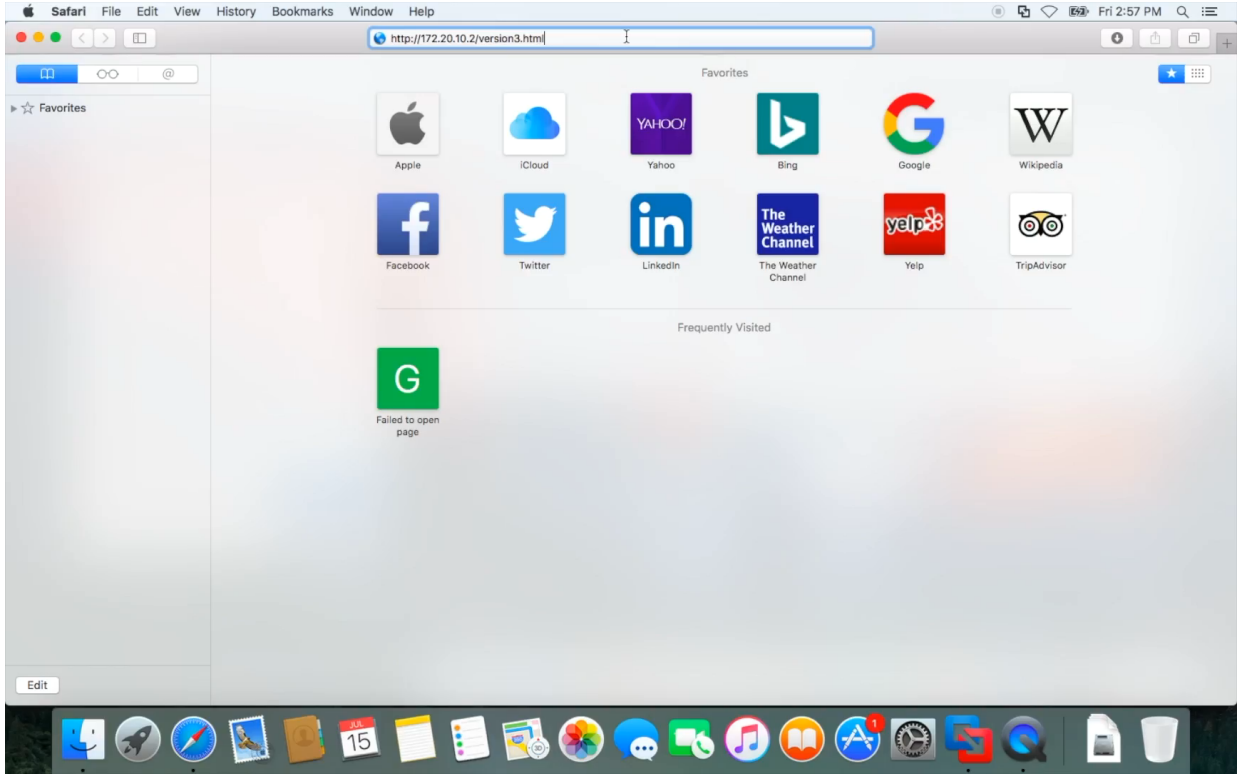
1. `QuartzCore` has a server that is running within `WindowServer`
2. The server port is acquired through calling `CGSGetConnectionPortById`
3. Spray memory with `CFDataCreateWithBytesNoCopy` and `CGSPropertyListCreateSerializedData`
4. In order to win exploit the race condition, call `CGSPropertyListCreateSerializedData` which sends to `_XRegisterClientOptions`
5. Utilize a buffer of size 0x30 and send several times
6. Buffer contains the ROP chain and shellcode
7. Once the race condition is won, code is running inside of `WindowServer`, and root execution is achieved simply by calling `setuid(0)`

Apple Safari to Kernel

TENCENT KEENLAB – TEAM SNIPER

TREND MICRO

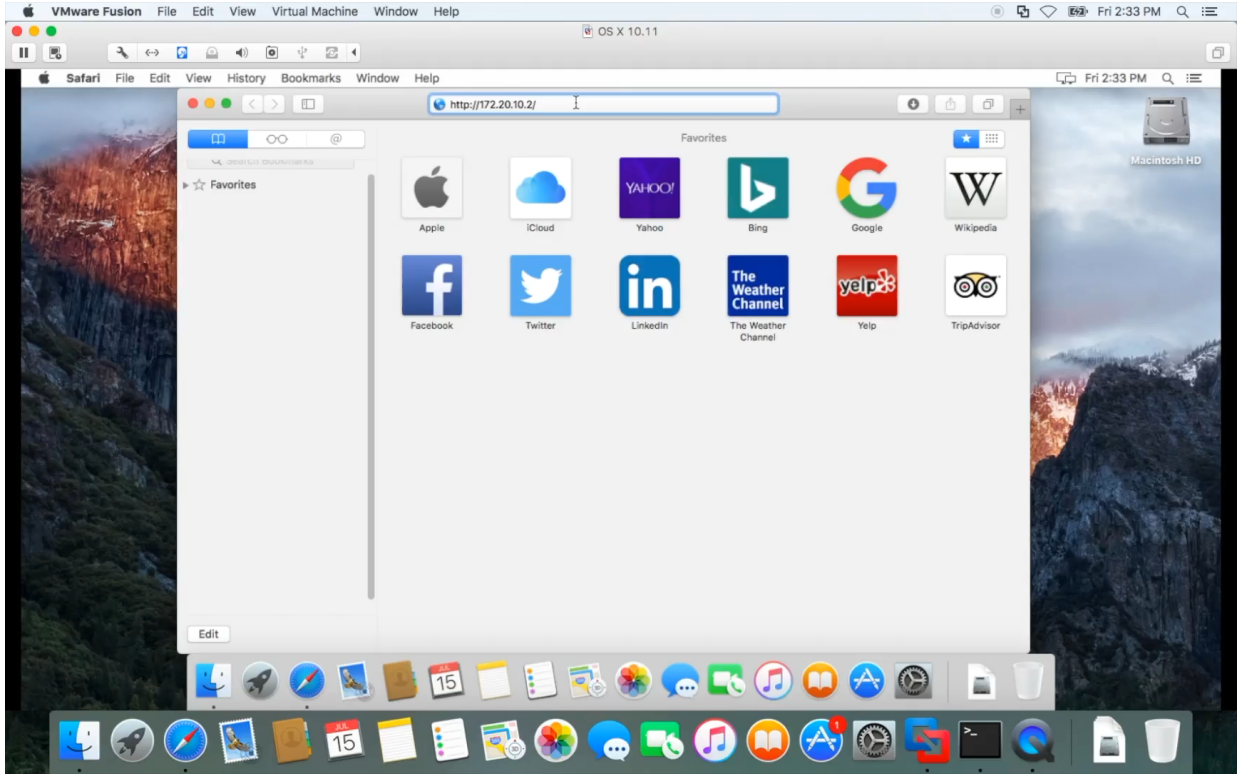
Team Sniper Exploit Demo



Apple Safari to Kernel

LOKIHARDT

Lokihardt's Exploit Demo



Apple Safari TextTrack Object UaF

- If `m_cues` is NULL and only `m_regions` exists, `setTrack(0)` on the items in `m_regions` never gets called.

```
TextTrack::~TextTrack()
{
if (m_cues) {
    if (m_client)
        m_client->textTrackRemoveCues(this, m_cues.get());
    for (size_t i = 0; i < m_cues->length(); ++i)
        m_cues->item(i)->setTrack(0);
    if (m_regions) {
        for (size_t i = 0; i < m_regions->length(); ++i)
            m_regions->item(i)->setTrack(0);
    }
}
clearClient();
}
```

Exploitation Steps

1. Leak a heap address off of `m_regions` in the `VTTRegionList` object
2. A series of string objects are allocated around a track element
3. Leak arbitrary addresses with the mode attribute from the track element
4. Spray `ArrayBuffer` objects and corrupt the aforementioned `m_list` attribute
   - Achieves write primitive allowing for out-of-bounds read and write access
5. Execution achieved through the JavaScript interpreter
   - Assigns a function to `oncuechange` from the controlled object
   - Runtime evaluates the shellcode

```
var jit_func = new Function("return 0x1234;");
jit_func();
for (var i = 0, n = tracks.length; i < n; i++)
  tracks[i].oncuechange = jit_func
```

Apple OS X fontd Heap-based Buffer Overflow

- `fontd` process provides the `com.apple.FontObjectsServer`
- Send a 0x2e message to reach a function which allocates memory of the size it reads from a controlled pointer
- Data passed to `GetUncompressedBitmapRepresentation`
- `GetUncompressedBitmapRepresentation` function contains no bounds checking
- Data copied into the buffer resulting in a heap-based buffer overflow

Exploitation Steps

1. Heap spray by sending a series of `mach_msg` to the `fontd` process
2. Trigger the heap-based buffer overflow
3. Utilizing leaks of `CoreFoundation` and `libsystem_c` with ROP techniques to call `mprotect` and gain code execution

Apple OS X fontd Sandbox Escape

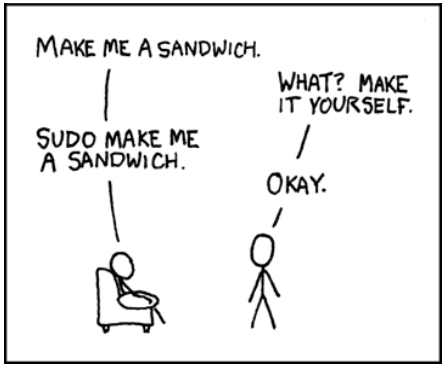
- `fontd` process is sandboxed
- `com.apple.fontd.internal.sb` contains a list of non-sandboxed applications
  - Located in `/usr/share/sandbox`
- `FontValidator` is not sandboxed
  - Uses `XT_FRAMEWORK_RESOURCES_PATH` environment variable as a path to look for libFontValidation.dylib
- Exploits the environment variable to escape the sandbox

Apple OS X SubmitDiagInfo Arbitrary Directory Creation

- `/System/Library/CoreServices/SubmitDiagInfo` runs as root
  - Provides and XPC service called: `com.apple.SubmitDiagInfo`
- Method `[Submitter sendToServerData:overrides:]` reads values from the following configuration file:
  - `/LibraryApplication Support/CrashReporter/DiagnosticMessagesHistory.plist`
- `SubmitToLocalFolder` key allows for an attacker to write an arbitrary directory

Exploitation Steps

1. Create the following directory
   - `/var/db/sudo/{USER_NAME}`
2. Allows the sudo command to run without any password authentication

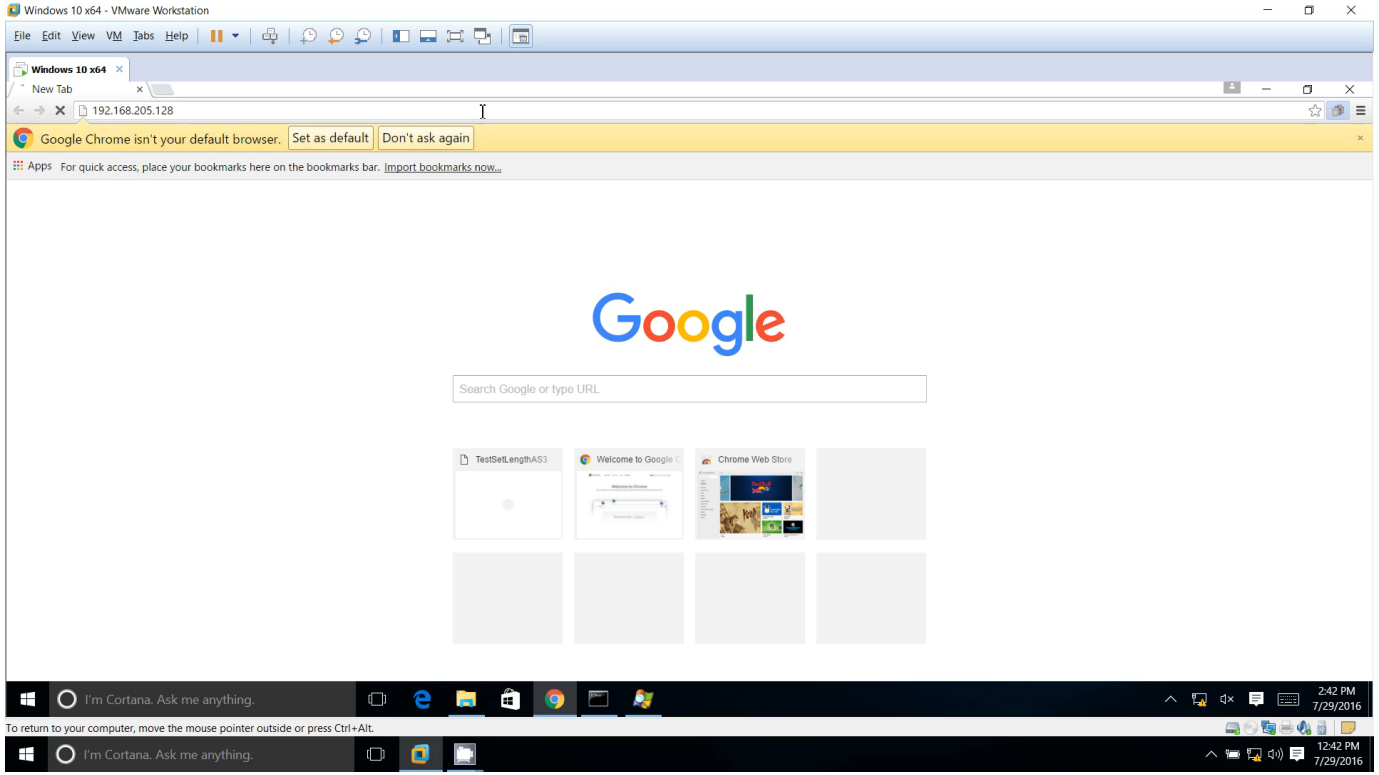


https://xkcd.com/149/

Google Chrome to SYSTEM

360 VULCAN TEAM

360 Vulcan Chrome Demo



Google Chrome V8 Out-Of-Bounds Access

- IterateElements
  - Responsible for visiting elements with an array
  - Assumed element access had no side effects
- Custom accessor for array
  - Modify array size while it is being iterated
- Fixed in the source code repository
  - Just three days before Pwn2Own

Google Chrome V8 Out-Of-Bounds Access

```
function evil_callback() {
    delete Array.prototype[0];
    this.length=1;  // Free the old storage

    return 0.1;
}

Array.prototype.__defineGetter__("0",evil_callback);

var arr=[];
for(var i=1;i<3;i++)
    arr[i] = 0.1;

arr = arr.concat();
alert(arr);
```

Exploitation Steps

1. Allocate an `ArrayBuffer` object in the freed storage
   - Contains an array of 20 floats that can be modified
2. Craft a custom `ArrayBuffer` object
   - Arbitrary reads and writes
3. Leak address to Text object
   - Used to find base address of chrome_child.dll
4. Leak address of kernel32

Adobe Flash AS2 Transform Use-After-Free

- `flash.geom.Transform`
  - Contains a matrix property
    - Instance of `flash.geom.Matrix`
- Custom accessor on `flash.geom`
  - *Modify the object before returning the original* `Matrix`
- Only useful as an information leak

Adobe Flash AS2 Transform Use-After-Free

```
var mc:MovieClip = movies[movies.length - 0x100];
var t = new Transform(mc);

var geom = _global["flash"]["geom"];
var OriginalMatrix = flash.geom.Matrix;

geom.addProperty('Matrix',function() {
    for ( var i = movies.length - 0x200; i < movies.length; ++ i )
        movies[i].removeMovieClip();
    return OriginalMatrix;
}, function() {} );

var m = t.matrix;
```

Exploitation Steps

1. Trigger the vulnerability
2. Within the custom accessor
    1. Free the `MovieClip` object
    2. Reclaim freed memory by allocating custom objects
3. Read values from the matrix property

Adobe Flash AS2 LoadVars Use-After-Free

- `LoadVars`
  - Contains a decode method
    - Takes URL-encoded string
    - Set properties on an object
- `Object.watch`
  - Set a callback on a property
  - Called when the property is modified

Adobe Flash AS2 LoadVars Use-After-Free

```
var mc:MovieClip = movies[movies.length - 0x200];
var my_lv:LoadVars = new LoadVars();

mc.watch("aaa", function() {
    for ( var i = 200; i < movies.length; ++ i )
        movies[i].removeMovieClip();
});

try {
my_lv.decode.call(mc, "aaa=1&bbb=2");
} catch (e) { }
```

Exploitation Steps

1. Trigger the vulnerability
2. Use the kernel32 information leak to bypass a dynamic call
3. Decrement arbitrary address
    1. Decrement reference count from leaked custom object
    2. Reclaim freed memory by allocating ByteArrays
4. Modify size of ByteArray
    1. Arbitrary read and write access

Microsoft Windows win32kfull.sys Surface Object UaF

- Issue arises because the code does not properly handle the reference count of a CompatibleDC object
- Window object creation handled by win32k
- Associated Surface object handled by Desktop Window Manager
- is createdCreationmethod takes a `HWND`
  - Does not verify that it is still valid
  - Typically done by calling `ValidateHwnd`

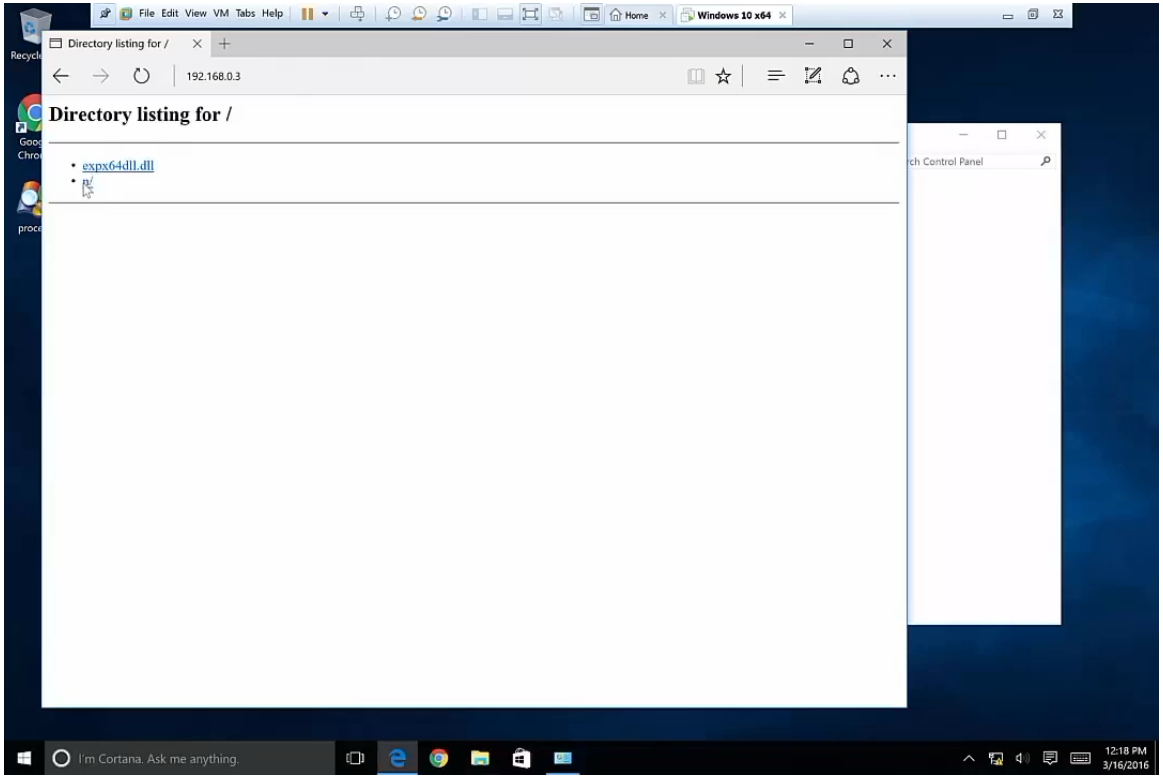
Exploitation Steps

1. Trigger the vulnerability
2. Allocate `AcceleratorTable` objects
3. Bitmap object used for memory access
   - `GetBitmapBits` used to read
   - `SetBitmapBits` used to write

Adobe Flash to SYSTEM

360 VULCAN TEAM

360 Vulcan Team Exploit Demo



Adobe Flash AS2 NetConnection Type Confusion

- AS2 `ScriptObjects` contain:
  - Type ID
  - Private data specific to the type
- Both typically set during object construction
  - *Not the case with `NetConnection` objects, which are set during connection*
- Implicit type conversions during function call
  - *Allow for object modification*

Adobe Flash AS2 NetConnection Type Confusion

```
var nc:NetConnection = new NetConnection()
var o = this;
o.toString = function() { super(); return "WIN 10,2,153,2"; }

var xml:XML = new XML("<mytag name='Val'> item </mytag>");
xml.firstChild.attributes.aaa = o;

nc.connect.call(this, url, xml);
```

Exploitation Steps

1. `ColorMatrixFilter` objects interpreted as a `NetConnection` objects
   - Contains an array of 20 floats that can be modified
2. Leak a pointer to a string
3. Trigger an arbitrary use-after-free
4. Modify a `ByteArray` object
   - Read/write access to memory

Microsoft Windows xxxEndDeferWindowPosEx Window UaF

- Issue arises because the code does not increment a reference count prior to calling userland code
- `PostIAMShellHookMessageEx` method takes a `HWND`
  - Does not verify that it is still valid
  - Typically done by calling `ValidateHwnd`

```
void __stdcall PostIAMShellHookMessageEx(int a1, int a2, int a3) {
  if ( !a1 ) return;
  v3 = _gpsi;
  if ( !(*(_BYTE *)(*(_DWORD *)_gpsi + 1712) & 8) || !*(_DWORD *)(a1 + ?)? )
    return;
  if ( a2 == 35 ) {
    v5 = *(_DWORD *)(*(_DWORD *)(a1 +4) +92);
    if ( v5 )
      _PostMessage(v5, *(_DWORD *)(*(_DWORD *)v3 + 520), a2, a3);
    return;
  }
  v4 = *(_DWORD *)(*(_DWORD *)(_gSharedInfo + 8) * (unsigned __int16)a3);
```

Microsoft Windows xxxEndDeferWindowPosEx Window UaF

- xxxEndDeferWindowPosEx method
  - Calls PostIAMShellHookMessageEx
  - Does not increment the WND reference count

```
if ( v52 ) {
  v28 = *(_DWORD *)(v26 + v27 + 24);
  if ( v28 & 0xF0000000 ) {
    if ( v28 & 0x10000000 ) {
      if ( *(_BYTE *)(v26 + v27 + 120) & 8 ) {
        PostIAMShellHookMessageEx(*(_DWORD *)(*(_DWORD *)_gptiCurrent + 216), 21, v52);
      }
      //...
      if ( *(_DWORD *)(v45 + *((_DWORD *)v2 + 6) + 24) & 0x80000000 )
        xxxSetTrayWindow(v3[54], 1);
      v26 = v45;
```

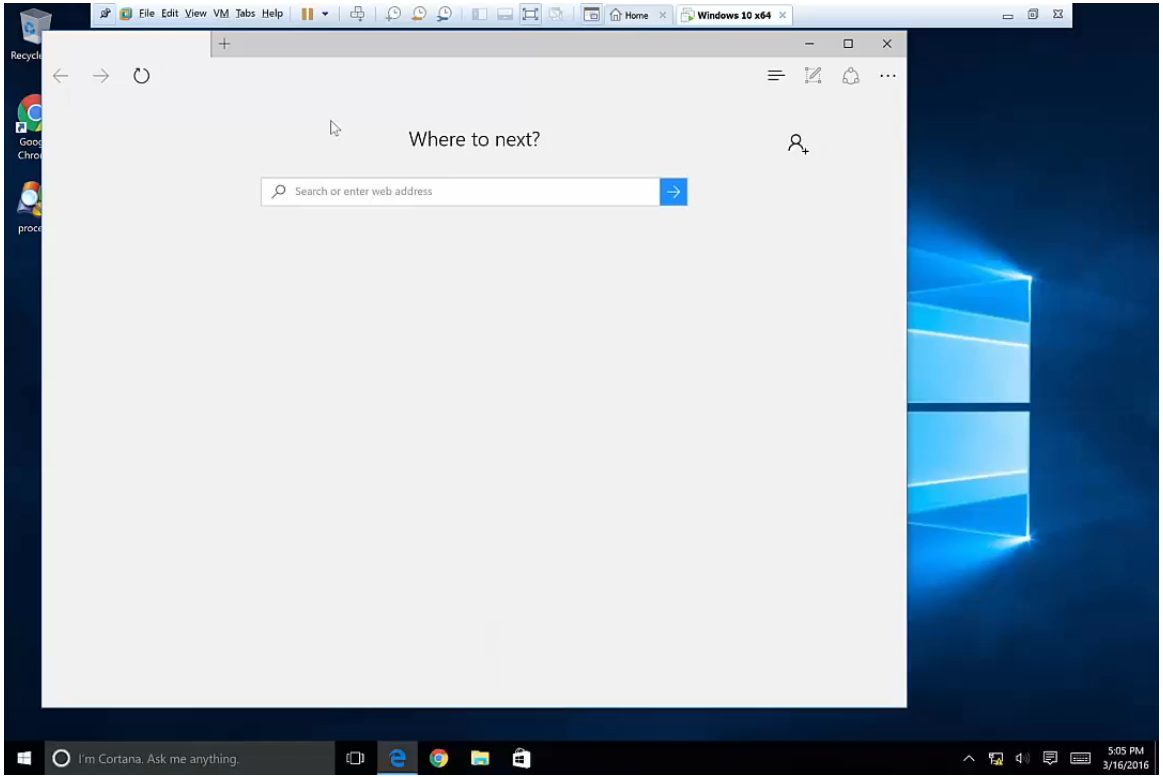
Exploitation Steps

1. `WND` replaced by `AcceleratorTable` object
2. Bitmap objects allocated adjacent
   - First Bitmap used to modify pointer in second Bitmap
   - `GetBitmapBits` used to read
   - `SetBitmapBits` used to write

Adobe Flash to SYSTEM

TENCENT KEENLAB – TEAM SNIPER

Team Sniper Exploit Demo



Adobe Flash JPEG-XR Parsing Stack-based Buffer Overflow

- Variant of a Google Project Zero discovery
  - *Nobody checked the 'else' branch of a condition*

```
int k, i = 1;
for (k = 0; k < num_nonzero; k+=1) {
  i += RLCoeffs[k*2];
  AdaptiveLPScan(image, LPInput[ndx], i, RLCoeffs[k*2+1]);
  i += 1;
}
```

Vulnerability in AdaptiveLPScan

```
void __cdecl sub_108A208B(_DWORD *a1, int a2, signed int a3, int a4)
{
  unsigned int v4; // ecx@4
  unsigned int v5; // eax@5
  int v6; // eax@6
  int v7; // ecx@6

  if ( !*a1 ) {
    if ( a3 > 0 ) {
      *(_DWORD *)(a2 + 4 * a1[a3 + 345]) = a4;
      v4 = ++a1[a3 + 360];
      if ( a3 > 1 ) {
        v5 = a1[a3 + 359];
        if ( v4 > v5 ) {
          a1[a3 + 360] = v5;
          v6 = a1[a3 + 344];
          a1[a3 + 359] = v4;
          v7 = a1[a3 + 345];
          a1[a3 + 345] = v6;
          a1[a3 + 344] = v7;
```

Exploitation Steps

1. Trigger vulnerability several times
2. Leak stack addresses
   - Used to craft a fake object pointer
3. Modify two objects on the stack
   - One to update pointers in the second object
   - The other to perform arbitrary reads and writes
4. Return

Microsoft Windows NtGdiGetEmbUFI Info Disclosure

- Simply calling `NtGdiGetEmbUFI` is sufficient
- `NtGdiGetEmbUFI` returns a pointer to a `PFFOBJ`
  - *Supposed to return an ID*
  - Issue is due to `GreGetUFI`

```
if ( PFFOBJ::bInPrivatePFT((PFFOBJ *)&v23) )
{
  *v8 |= 1u;
  if ( a7 )
    *(_DWORD *)a7 = *v16;
}
```

Microsoft Windows PFFOBJ::bDeleteLoadRef Font UaF

- `bDeleteLoadRef` frees `PFFOBJ` resources
- Return code depends on reference count
- Caller wouldn't know if resources freed or not

```
if ( v9 )
{
  PFFOBJ::vKill(v5);
  v4 = 1;
}
return *(_DWORD *)(*(_DWORD *)v5 + 44) == 0 ? v4 : 0;
```

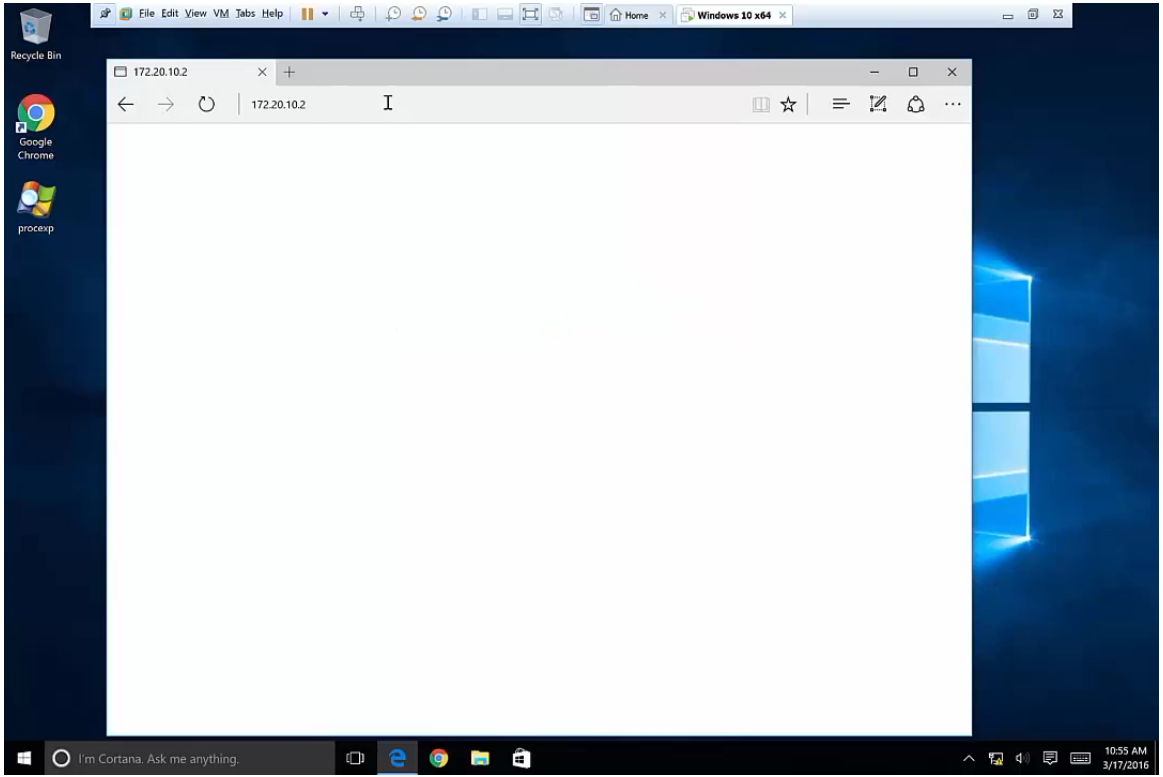
Exploitation Steps

1. Free the `PFFOBJ` object
2. `NtUserConvertMemHandle` used to reclaim freed memory
3. Results in a primitive to OR a value with 2
   - Used to increase a `WND` object's extra byte size
4. `SetWindowLongPtr` to set pointer in other `WND`
   - `InternalGetWindowText` to read arbitrary memory
   - `NtUserDefSetText` to write arbitrary memory

Microsoft Edge to SYSTEM

LOKIHARDT

Lokihardt Exploit Demo



Microsoft Edge JavaScript Concat Method Uninitialized Variable

- From JavascriptArray::ConcatArgs

```
var subItem;
uint32 lengthToUin32Max = length.IsSmallIndex() ? length.GetSmallIndex() : MaxArrayLength;
for (uint32 idxSubItem = 0u; idxSubItem < lengthToUin32Max; ++idxSubItem)
{
    if (JavascriptOperators::HasItem(itemObject, idxSubItem))
    {
        JavascriptOperators::GetItem(itemObject, idxSubItem, &subItem, scriptContext);
        if (pDestArray)
        {
            pDestArray->DirectSetItemAt(idxDest, subItem);
}
```

Exploitation Steps

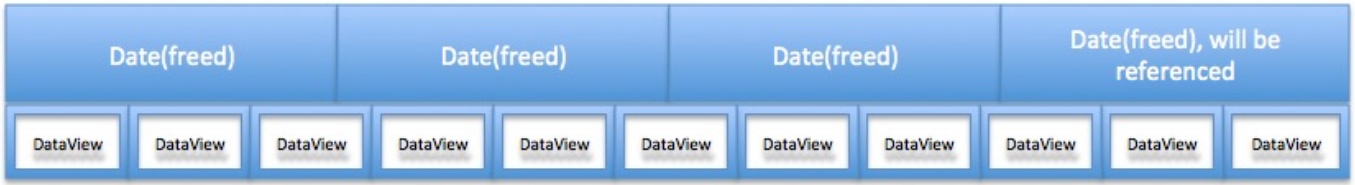
- Simple Trigger

```
var bug = new Proxy(new Array(1),{has: ()=> true});
alert(bug.concat());
```

1. Spray `JavascriptDate` objects of size 0x90
2. Push one of the `JavascriptDate` objects, which resided in the middle of the spray deep in the stack using `Array.slice`
3. Free memory
4. Spray `DataView` object at the freed space
5. Trigger the vulnerability to get the reference to the pointer that we pushed in step 2

Exploitation Steps

- Object that will be referenced using the vulnerability is the last `JavascriptDate` object which points to `DataView+0x30`



- The following is a representation of the `DataView` object structure:

	+ 0x00	+ 0x08
0x00	*vtable	*type
0x10	N/A	N/A
0x20	length	*arrayBuffer
0x30	byteOffset	*data

Exploitation Steps

- Read and write primitives
  - `DataView` objects memory is freed and filled with `NativeFloatArray` objects
  - As long as the size of `NativeFloatArray` is small, elements for the array are created next to each other
  - Fake `DataView` objects are created to perform read/write from process memory
- Control Flow Guard bypass
  - Use a `setjmp` call to obtain the stack address and overwrite the return address

Microsoft Windows Diagnostics Hub Standard Collector Directory Traversal

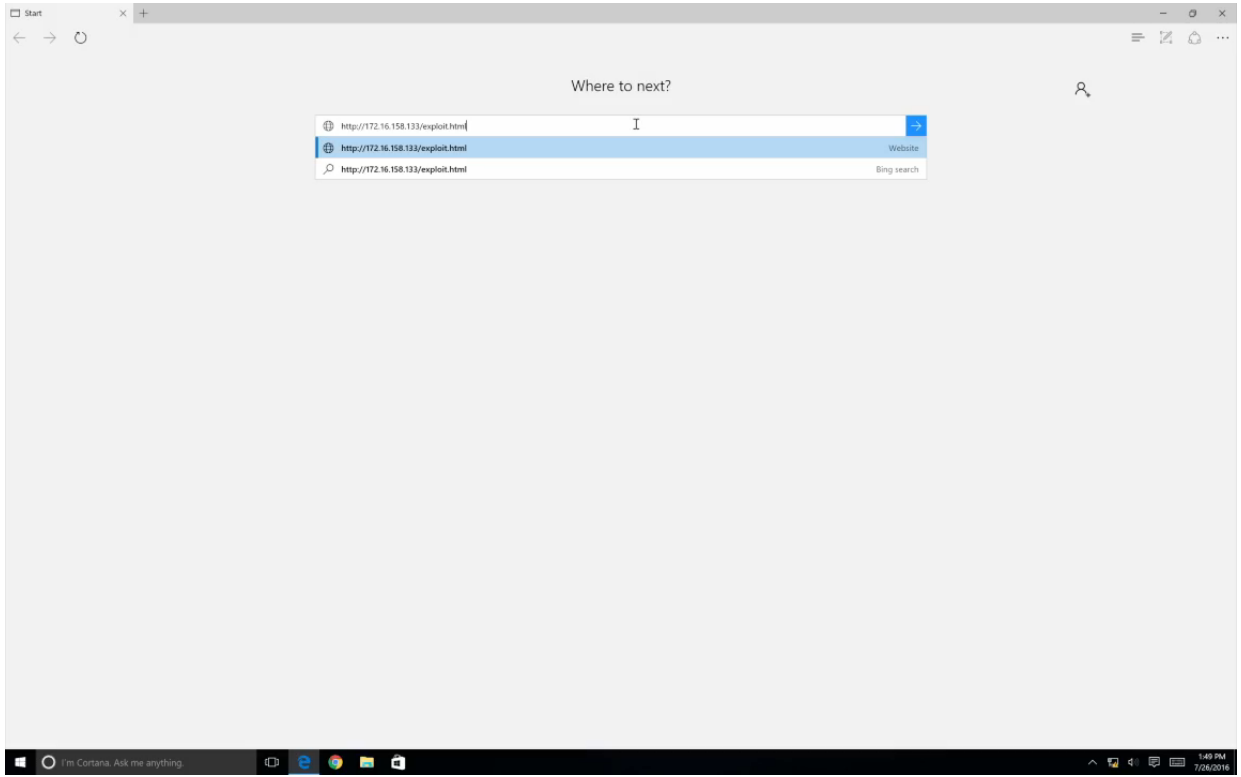
- Standard Collector is SYSTEM-level COM service
  - Communication is possible from within the Microsoft Edge's sandbox
- COM interface exposes the `AddAgent` function that takes two arguments
  - DLL path
  - GUID

DLL Path Not Validated

Microsoft Edge with SYSTEM

TENCENT KEENLAB – TEAM SNIPER

Team Sniper Exploit Demo



Microsoft Edge JavaScript `fill` Out-of-Bounds Access

- Length is fetched from the length property
- Length property can be overwritten in certain cases

```
Var JavascriptArray::EntryFill(RecyclableObject* function, CallInfo callInfo, ...)
{
  //....

  Var lenValue = JavascriptOperators::OP_GetLength(obj, scriptContext);
  length = JavascriptConversion::ToLength(lenValue, scriptContext);
}

return JavascriptArray::FillHelper(pArr, nullptr, obj, length, args, scriptContext);
```

Execution Flow

```
Var JavascriptArray::FillHelper(JavascriptArray* pArr, Js::TypedArrayBase* typedArrayBase,
RecyclableObject* obj, int64 length, Arguments& args, ScriptContext* scriptContext)
  {
    //...
    int64 end = mxin<int64>(finalVal, MaxArrayLength);
    uint32 u32k = static_cast<uint32>(k);

    while (u32k < end)
    {
      if (typedArrayBase) {
        typedArrayBaseypedArrayBase->DirectSetItem(u32k, fillValue, false);
      }
    else if (pArr)
    {
      pArr->SetItem(u32k, fillValue, PropertyOperation_ThrowIfNotExtensible);
    }
```

Execution Flow

```
__inline BOOL BaseTypedDirectSetItem(__in uint32 index, __in Js::Var value, __in bool skipSetElement, TypeName
(*convFunc)(Var value, ScriptContext* scriptContext))
 {
 // This call can potentially invoke user code, and may end up detaching the underlying array (this).
 // Therefore it was brought out and above the IsDetached check
 TypeName typedValue = convFunc(value, GetScriptContext());

 if (this->IsDetachedBuffer()) // 9.4.5.9 IntegerIndexedElementSet {
 JavascriptError::ThrowTypeError(GetScriptContext(), JSERR_DetachedTypedArray);
 }

 if (skipSetElement) { return FALSE; }

 AssertMsg(index < GetLength(), "Trying to set out of bound index for typed array.");
 Assert((index + 1)* sizeof(TypeName) + GetByteOffset() <= GetArrayBuffer()->GetByteLength());
 TypeName* typedBuffer = (TypeName*)buffer;
 typedBuffer[index] = typedValue;
```

Exploitation Steps

- Simple trigger

```
var ua = new Uint32Array(0x10);
ua.__proto__= new Array(0xffffffff);
ua.fill(0x41, 0x41414141, 0x41414141 + 1);
```

- Corrupt the length of a `JavascriptNativeIntArray`
  - Achieves arbitrary read and write
- Two heap sprays were used during the exploitation process
  - First one to stabilize the heap
  - Second one to fetch the index

Exploitation Steps

- Read and write primitive
  - Once the index of the target object was found, the bug was triggered against the object to overwrite the length with `0x7FFFFFFC`
  - Using this corrupted object, arbitrary read/write is possible against a large chunk of the Chakra heap
- Control Flow Guard bypass
  - Use a `setjmp` call to obtain the stack address and overwrite the return address

Microsoft Windows Dxgkrnl Driver Buffer Overflow

- dxgkrnl.sys kernel driver has a structure called _D3DKMT_PRESENTHISTORYTOKEN
  - Token and TokenSize members vary according to the Model member
  - There is a model called Flip model
  - Flip model's corresponding token structure is D3DKMT_FLIPMODEL_PRESENTHISTORYTOKEN
- _D3DKMT_PRESENTHISTORYTOKEN structure contains a structure called D3DKMT_DIRTYREGIONS
- D3DKMT_DIRTYREGIONS contains an array of RECTS

```
typedef struct _D3DKMT_DIRTYREGIONS
 {
 UINT NumRects;
 RECT Rects[D3DKMT_MAX_PRESENT_HISTORY_RECTS];
 } D3DKMT_DIRTYREGIONS;
```

Microsoft Windows Dxgkrnl Driver Buffer Overflow

- Driver checks whether NumRects is greater than D3DKMT_MAX_PRESENT_HISTORY_RECTS:

```
    cmp  dword ptr [r15+334h], 10h ; jumptable 00000001C0098BBB case 1
    jbe  short loc_1C0098C0B
    call cs:__imp_WdLogNewEntry5_WdAssertion
    mov  rcx, rax
    mov  qword ptr [rax+18h], 38h
    call cs:__imp_WdLogEvent5_WdAssertion
loc_1C0098C0B:
    mov  eax, [r15+334h]
    shl  eax, 4
    add  eax, 338h
    jmp  short loc_1C0098C7D
```

Microsoft Windows Dxgkrnl Driver Buffer Overflow

- Execution continues after logging the violation
- Code reaches `loc_1C0098C7D` regardless of the value specified for `NumRects`:

```
loc_1C0098C7D:
    lea  r8d, [rax+7]
    mov  rdx, r15          ; Src
    mov  eax, 0FFFFFF8h
    mov  rcx, rsi          ; Dst
    and  r8,  rax          ; Size
    call memmove
```

Exploitation Steps

1. Trigger overflow vulnerability to overwrite a subsequent history token record
2. When the record is read, the address that was specified (during the overwrite) will actually be read
3. `BITMAP` objects are sprayed into the kernel memory pool
   - Address of these `BITMAP` objects can be guessed using the GDI handle table, which can be accessed with `user32!gSharedInfo`
4. Using that address, arbitrary data can be written
5. The specific `BITMAP` object that was written to, can be found by traversing the sprayed `BITMAP` objects
6. Kernel-mode read and write can be achieved using `SetBitmapBits` and `GetBitmapBits` APIs on this corrupted `BITMAP` object
7. IDT table can be dumped and the `ntoskrnl` base address can be fetched
8. System process and the current process' `EPROCESS` structures can be fetched by traversing the `PspCidTable`

Conclusion

Conclusion

- Application sandboxing is a step in the right direction
- Kernel attack surface remains expansive and exposed
- Exploitation is getting harder…but still very possible
  - Write exploit primitives are a mainstay in exploits
  - Logic bugs can be high impact
- Vulnerability/Exploit research is becoming increasingly popular ;-)
  - New researchers setting a new standard
  - Becoming a team sport

Questions?

ZERO DAY INITIATIVE

www.zerodayinitiative.com

@thezdi

TREND MICRO