Rainbow over the Window(s)

... more colors than you could expect
$whoami

- Peter
- @zer0mem
- Windows kernel research at KeenLab, Tencent
- fuzzing focus : state
- wushu player

- Daniel
- @long123king
- Windows kernel research at KeenLab, Tencent
- pwn2own winner (2016)
- fuzzing focus : data 'format'
- windbg guy
agenda

w32k
- prevalence
- references
- patch tuesday
- attack surface
- filtering
- extensions
- fuzzing

p2o 2016
- directx
- universal bug
- details
- exploitation
why we are interested

• resides in ring 0
  • i pretty much enjoy at level 0 and bellow ;)

• huge attack surface
  • huge in comparrison to ntoskrnl counterpart or in-ring3-sandbox interface
  • necessary to cover that, by white hats, as it exposes big impact for security

• accessible from sandbox-es
  • nowadays more or less => big success!

• field to train your fuzzer!
  • on this, bit later in this talk
previous (& ongoing) work in this area

• nils to p0
  • just follow his bucket of bugs and you got pretty much idea what's going on in w32k
• mwr labs defcon
  • 3 teams to cover w32k, different approaches & results
• p2o - from vulnerability to exploit
  • 2015 - 2x TTF [KEEN]
  • 2016 - DirectX [KEEN]
  • 2016 - chrome - flash - w32k breakdown [360]
• j00ru:
  • TTF
  • EMF
attack surface

• once i said big one, i meanded it!

C:\>cat w32k@subsurface | grep "Nt" | wc -l
1042
what is going on in w32k?

• huge numbers of syscalls
• lot of objects
• lot of hardcore graphics stuffs
• lot of things i dunno
DC - lets paint

vector 1:
includes fair amount of functions

vector 2:
interconnects nice number of different objects
fonts - did i mention it?

- various prevalent (mis)usage of different actors
  - stuxnet, duqu, ...
- our p2o 2015 target (2 x TTF to kernel code exec)
- j00ru heroic cleaning

- ahh ... from last year it is moved to user mode, is it over?
- but still for fonts loading you going to kernel, exposed syscalls
- found & reported nice bug recently

- takeaway : not all problems vanish by moving things around
  - but to be honest, it solves a lot ...
recent bugs

• just including one semi complete part, without targeting syscalls, mainly for tuning fuzzer infrastructure:
  • logic
  • mutator
  • generator
  • interconnections
  • additional algorithms

![Diagram showing bug distribution]

- Collisions: 23%
- ReadVa, nullptr: 24%
- reported: 35%
- queue: 18%

bugz so far #13~16
win32k - surface

- mentioned earlier, huge arsenal of syscalls
- condrv
- directx
- user mode callbacks
- ioctl alike not so 'hidden syscalls'
  - ntusermessagecall
  - apfn
  - and more
NtUserMessageCall

• used at p2o 2015
• very powerfull for exploitation
• more accessible “power” behind one syscall!
NtUserCall * => apfn table

• nice 'hidden' ioctl-alike attack surface:
• notice CreateMenu, and others
• + > 0x80 syscalls
Qilin <- win32kfull!apfnSimpleCall

class CfnSysCall :
    public Csyscall
{
    // win 1511, temporary hardcoding
    static SyscallTable::SyscallId ResolveWrapperId(
        inX SyscallTable::SyscallId syscallId
    )
    {
        if (syscallId < 0x27)
            return SyscallTable::SyscallId::NtUserCallNoParamAPI;
        if (syscallId < 0x27 + 0x2a)
            return SyscallTable::SyscallId::NtUserCallOneParamAPI;
        if (syscallId < 0x52 + 0x87)
            return SyscallTable::SyscallId::NtUserCallHwndAPI;
        if (syscallId < 0x59 + 0x01)
            return SyscallTable::SyscallId::NtUserCallHwndOptAPI;
        if (syscallId < 0x5B + 0x06)
            return SyscallTable::SyscallId::NtUserCallHwndParamAPI;
        if (syscallId < 0x66 + 0x64)
            return SyscallTable::SyscallId::NtUserCallHwndLockAPI;
        if (syscallId < 0x73 + 0x09)
            return SyscallTable::SyscallId::NtUserCallHwndParamLockAPI;
        if (syscallId < 0x7C + 0x11)
            return SyscallTable::SyscallId::NtUserCallOneParamAPI;
        return SyscallTable::SyscallId::Undefined;
    }
directx

• another nice example of w32k extension:

• interesting takeaways:
  • state alike fuzzing
    • less prone to bug
    • not so much code involved
    • basically wrappers and memory / locking mechanism
    • however universal bugs, independent of graphic
  • data fuzzing
    • mostly related to graphic drivers nvidia / intel
    • therefore not universal bugs
    • prone to bugs, lot ...
w32k filter
w32k filter

• introduced to limit unnecessary access to w32k
• more benevolent that w32k lockdown
• limit attack surface for bug hunting
• limit exploitation techniques
• bitmap of allowed syscalls
• wrapped in win32k: “x win32k!stub* “
w32k filter

- bitmap of allowed w32k
- edge example (part):

```plaintext
fff9784`-e97a4800 1020 MicrosoftEdge
fff9784`-a974c800 1764 browser_broker
fff9784`-d1bb800 1624 MicrosoftEdgeC
0: kd) dt _e proc ess ffff9784`-d1bb800 EnableFilteredWin32kAPIS DisallowWin32kSystemCalls AuditFilteredWin32kAPIS
ntl_EPROCESS
+0x300 DisableWin32kSystemCalls : 0y0
+0x6c4 EnableFilteredWin32kAPIS : 0y1
+0x6c4 AuditFilteredWin32kAPIS : 0y1
```
w32k indirect ways

• condrv.sys -> conhost.exe
  • aka console
  • issue ioctl to condrv
  • driver will forward those w32k alike command to conhost.exe
  • conhost.exe will issue w32k syscalls
  • through condrv ioctl you can fuzz / exploit w32k indirectly

• active at p2o 2016, penetrated by 360 vulcan team
  • escape through plugin
  • requires additional bug in plugin
  • in new environment where is no lockdown anymore!
Fuzzing?
data format(TTF, ..)
object state(dc, ..)
Documentation

- actually very well documented - msdn
- find your particular object
- get set of related api's
- understand api
- skip gdi workaround (locks, temporary memory and handles databases) and go directly for syscalls
  - although syscalls not documented, use api knowledge + RE
How

- templates
- examples of template fuzzers: trinity, syzkaller, mwr fuzzer
- our internal Qilin fuzzer
- grab api (REconstruct Nt* ones) definitions from msdn
- fill patterns with reasonable value ranges
- generate patterns
sophisticate it little bit

• sort patterns per object
  • Dc, Region, Bitmap, Font, ..
  • Window, Menu, UserMessage, afn, ..

• get meaningful connections
  • get from database active handle of particular type

• get interrupted at user mode callbacks
  • involve some meaningful system calls then

• scope create - delete
  • dont let it goes wild
code coverage

- essential to do (semi)meaningfull actions
  - approx good and bad parameters
  - good ration (40%+) of success ratio
- this alone can get you reasonable code coverage info

<table>
<thead>
<tr>
<th>Overall</th>
<th>#Kobjs</th>
<th>#syscalls</th>
</tr>
</thead>
<tbody>
<tr>
<td>elapsed time: 1:24</td>
<td>server-not-found</td>
<td>ConDrvConLocked0r =&gt; total: 179, ratio: 100.00</td>
</tr>
<tr>
<td>Total Process Count: 1</td>
<td>server-not-found</td>
<td>ConWriteLockedOutput =&gt; total: 232, ratio: 100.00</td>
</tr>
<tr>
<td>Process Killed: 2</td>
<td>server-not-found</td>
<td>ConLaunchServerProcess =&gt; total: 99, ratio: 100.00</td>
</tr>
<tr>
<td>Total Server # : 0</td>
<td>server-not-found</td>
<td>ConHostAPR =&gt; total: 165, ratio: 73.39</td>
</tr>
<tr>
<td>Total Client # : 0</td>
<td>server-not-found</td>
<td>ConDrvConFastToctl =&gt; total: 39, ratio: 100.00</td>
</tr>
<tr>
<td>Average Syscall Total #: 55</td>
<td>server-not-found</td>
<td>NCreateFile =&gt; total: 1023, ratio: 0.00</td>
</tr>
<tr>
<td>Total KObj count : 5</td>
<td>server-not-found</td>
<td></td>
</tr>
</tbody>
</table>
code coverage

• qemu
  • ola, runs win10!
  • you can even ssh to win10 ;)

• kvm vs tsg switching
• do minimalistic patch
• grab code coverage

• use powerfull static analysis arsenal : binary ninja!
• lead / help your fuzzer
• more on this topic another time, soon :)
Edge EoP for Pwn2Own 2016

• Bug: CVE-2016-0176
• Bug Type: Kernel Heap Overflow
• Bug Driver: dxgkrnl.sys
typedef struct _D3DKMT_PRESENTHISTORYTOKEN
{
    D3DKMT_PRESENT_MODEL Model; //D3DKMT_PM_REDIRECTED_FLIP = 2,
    // The size of the present history token in bytes including Model.
    // Should be set to zero by when submitting a token.
    // It will be initialized when reading present history and can be used to
    // go to the next token in the present history buffer.
    UINT TokenSize; // 0x438
    #if (DXGKDDI_INTERFACE_VERSION >= DXGKDDI_INTERFACE_VERSION_WIN8)
        // The binding id as specified by the Composition Surface
        UINT64 CompositionBindingId;
    #endif

    union
    {
        D3DKMT_FLIPMODEL_PRESENTHISTORYTOKEN Flip; // happen to be the largest union component
        D3DKMT_BLTMODEL_PRESENTHISTORYTOKEN Blt;
        D3DKMT_VISTABLTMODEL_PRESENTHISTORYTOKEN VistaBlt;
        D3DKMT_GDIMODEL_PRESENTHISTORYTOKEN Gdi;
        D3DKMT_FENCE_PRESENTHISTORYTOKEN Fence;
        D3DKMT_GDIMODEL_SYSMEM_PRESENTHISTORYTOKEN GdiSysMem;
        D3DKMT_COMPOSITION_PRESENTHISTORYTOKEN Composition;
    }
    Token;
} D3DKMT_PRESENTHISTORYTOKEN;
typedef struct _D3DKMT_FLIPMODEL_PRESENTHISTORYTOKEN
{
    UINT64    FenceValue;
    ULONG64   hLogicalSurface;
    UINT_PTR  dxgContext;
    D3DDDI_VIDEO_PRESENT_SOURCE_ID    VidPnSourceId;
    ...

    D3DKMT_HANDLE     hSyncObject; // The local handle of a sync object from D3D runtimes.
    // The global handle of the sync object coming to DWM.
    RECT              SourceRect;
    UINT              DestWidth;
    UINT              DestHeight;
    RECT              TargetRect;
    // DXGI_MATRIX_3X2_F: _11 _12 _21 _22 _31 _32
    FLOAT             Transform[6];
    UINT              CustomDuration;
    D3DDDI_FLIPINTERVAL_TYPE       CustomDurationFlipInterval;
    UINT              PlaneIndex;
} D3DKMT_FLIPMODEL_PRESENTHISTORYTOKEN;
typedef struct tagRECT
{
    LONG left;
    LONG top;
    LONG right;
    LONG bottom;
} RECT, *PRECT, NEAR *NPRECT, FAR *LPRECT; // 0x10 bytes

typedef struct _D3DKMT_DIRTYREGIONS
{
    UINT NumRects;

    RECT Rects[D3DKMT_MAX_PRESENT_HISTORY_RECTS]; // 0x10 * 0x10 = 0x100 bytes

    // #define D3DKMT_MAX_PRESENT_HISTORY_RECTS 16
}
} D3DKMT_DIRTYREGIONS;
Overflow Code (Disassembly)

loc_1C009832A: DXGCONTEXT::SubmitPresentHistoryToken(......) + 0x67B
  cmp    dword ptr[r15 + 334h], 10h // NumRects
  jbe   short loc_1C009834B; Jump if Below or Equal(CF = 1 | ZF = 1)
  call  cs : __imp_WdLogNewEntry5_WdAssertion
  mov   rcx, rax
  mov   qword ptr[rax + 18h], 38h
  call  cs : __imp_WdLogEvent5_WdAssertion

loc_1C009834B: DXGCONTEXT::SubmitPresentHistoryToken (......) + 0x6B2
  mov    eax, [r15 + 334h]
  shl    eax, 4
  add    eax, 338h
  jmp    short loc_1C00983BD

loc_1C00983BD: DXGCONTEXT::SubmitPresentHistoryToken (......) + 0x6A5
  lea    r8d, [rax + 7]
  mov    rdx, r15; Src
  mov    eax, 0FFFFFFF8h;
  mov    rcx, rsi; Dst
  and    r8, rax; Size
  call   memmove
Overflow Code (C++)

```cpp
D3DKMT_PRESENTHISTORYTOKEN* hist_token_src = BufferPassedFromUserMode(...);
D3DKMT_PRESENTHISTORYTOKEN* hist_token_dst = ExpInterlockedPopEntrySList(...);

if(hist_token_src->dirty_regions.NumRects > 0x10)
{
    // log via watch dog assertion, NOT work in free/release build
}

auto size = (hist_token_src->dirty_regions.NumRects * 0x10 + 0x338 + 7) / 8;
auto src = (uint8_t*)hist_token_src;
auto dst = (uint8_t*)hist_token_dst;
memcpy(dst, src, size);
```
Lookaside-like Singly-Linked List of Hist Token

FreeList: Head → A → B → C → D → E → F → G → H
Pop one node out for use (Pop A)

FreeSList: Head → B → C → D → E → F → G → H
Pop another node out for use (Pop B)

FreeSList: Head → C → D → E → F → G → H
Overflow Scenario 1

FreeSList: Head -> B -> C -> D -> E -> F -> G -> H
Overflow Scenario 2

FreeSList: Head -> C -> D -> E -> F -> G -> H
Push node B back after overflow scenario 2

FreeSList: Head -> B -> C -> D -> E -> F -> G -> H
Will this overflow lead to arbitrary write?

FreeSList: Head -> A -> B -> C -> D -> E -> F -> G -> H
Unfortunately!

FreeSList: Head -> A -> B -> C -> D -> E -> F -> G -> H
The overwritten ‘Next’ field will be recovered

FreeSList: Head -> A -> B -> C -> D -> E -> F -> G -> H
Back to where after 2 pops

FreeSList: Head → C → D → E → F → G → H
Push in different orders with pop

FreeSList: Head -> A -> C -> D -> E -> F -> G -> H
Overflow Scenario 3

FreeSList: Head -> A -> ?

FreeTokensHead
The gap between ideal and reality

• Till now, it is nothing but theory!
  • 1st try
    • Action: Loop calling into D3DKMTPresent(), which will trigger overflow scenario 1
    • Failed: Can not reach to overflow scenario 2 or 3
    • Reason: every request is served by the node A, and then release it.
  • 2nd try
    • Action: Loop calling into D3DKMTPresent() from multithread
    • Failed: Can not reach to overflow scenario 2 or 3
    • Reason: protected by a lock
The gap between ideal and reality

- Doubt: is it doable for a double pop?
- In theory: Yes, otherwise the lookaside list is meaningless
- Guess: there should other callstacks trigger pop
- Target: graphics intensive applications
- Detection: windbg script logging push and pop

**Solitaire** is the hero
- BitBlt() can trigger pop with a different call stack
- Multithread loop with a mix of D3DKMTPresent() and BitBlt() lead to double pop
- Double pop eventually lead to overflow scenario 2 and 3
Arbitrary read and write into kernel memory

• With the help of *Bitmap* object
• Spray bitmap objects into 4GB ranges
  • First hold space by array of 256MB big bitmap objects
  • Then replace with 1MB small bitmap objects
• Redirect overflow write to 1 bitmap object
• Need hint of location of bitmap arrays, info leak needed
  • Info leak by `user32! gSharedInfo`
• 2-steps manipulation of Bitmap object succeed arbitrary read/write to kernel
Steal token of SYSTEM process

- Info leak of nt base addr
  - Info leaked by sidt
- nt!PspCidTable
  - Same entry structures as handle table
  - Get SYSTEM’s _KPROCESS and _TOKEN addr
  - Get current process’s _KPROCESS _TOKEN addr
  - Steal it
  - Enjoy SYSTEM privilege now!
Q & A
references

• nils: his p0 bucket of bugs (example)
  • https://bugs.chromium.org/p/project-zero/issues/detail?id=746

• mwr: defcon slides & fuzzers
  • https://github.com/mwrlabs/KernelFuzzer

• nikita: directx zeronights talk
  • http://2015.zeronights.org/assets/files/11-Tarakanov.pdf

• tiraniddo: conhost p0 block post
  • https://googleprojectzero.blogspot.com/2015_05_01_archive.html?m=0

• j00ru: font blog post series
  • http://googleprojectzero.blogspot.com/2016/06/a-year-of-windows-kernel-font-fuzzing-1_27.html

• keenlab: recon 2015 ttf slides

• microsoft: mitigations

• qemu: win10 'fix' blog post
  • https://www.invincealabs.com/blog/2016/07/running-windows-64bit-qemu/