Tencent

Rainbow over the Window(s)

... more colors than you could expect

KEEN security lab

$whoami

- Peter
- @zer0mem
- Windows kernel research at KeenLab, Tencent
- pwn2own winner (2015 / 2016), pwnie nominee (2015)
- 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

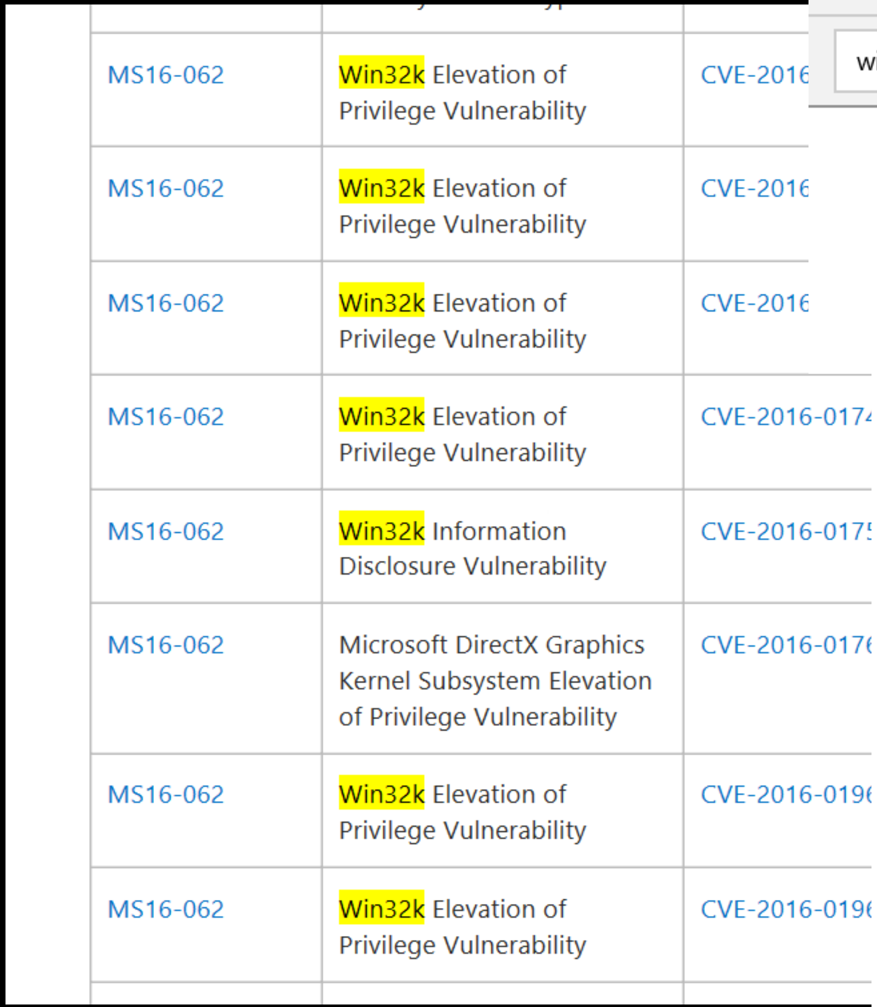
why we are interested

- resides in ring 0
  - i pretty much enjoy at level 0 and bellow ;)
- huge attack surface
  - huge in comparsion 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
  - nowdays more or less => big success!
- field to train your fuzzer!
  - on this, bit later in this talk

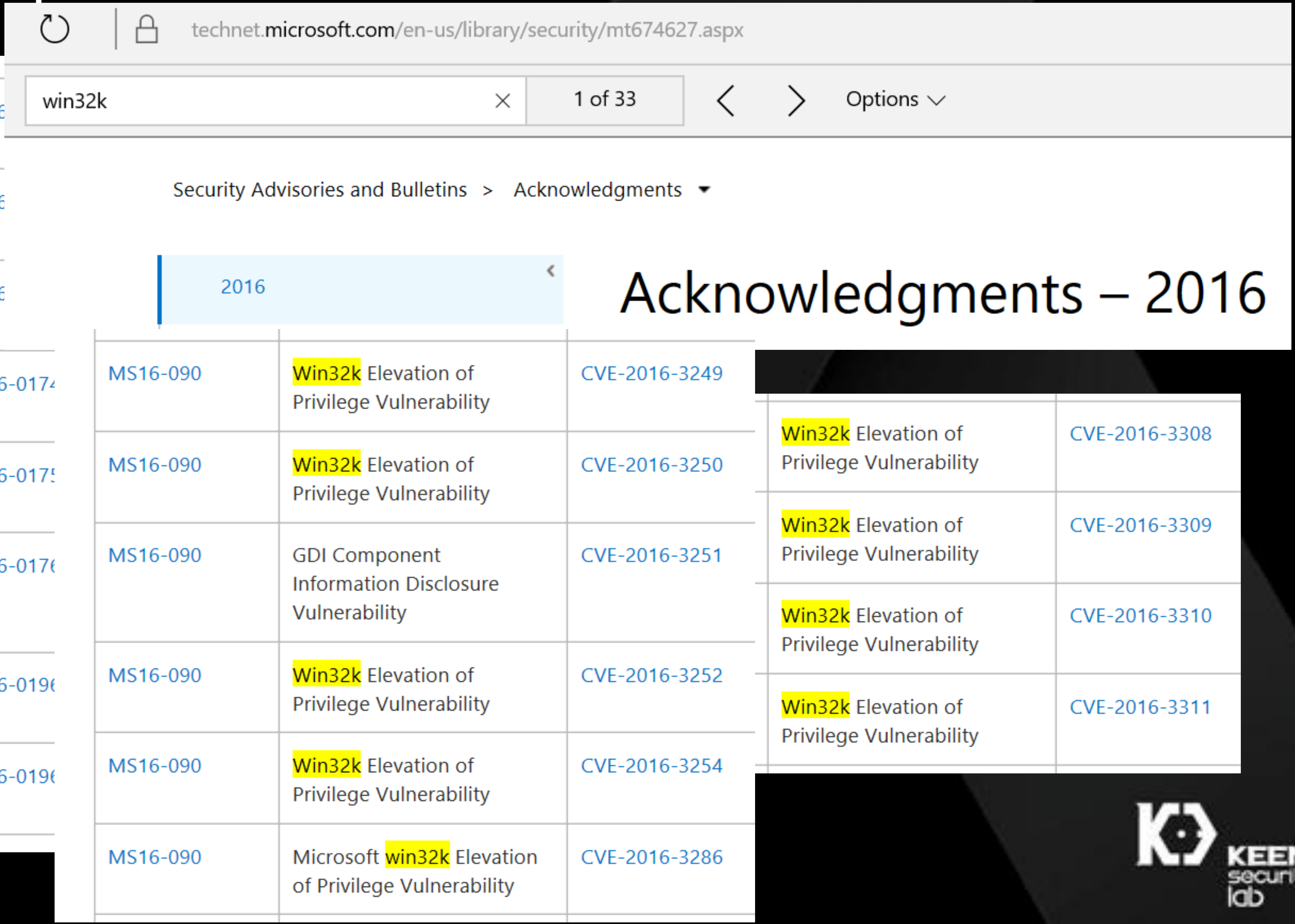
previous (& ongoing) work in this area

- nils to p0
  - just follow his bucket of bugs and you got pretty much idea whats 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

bulletins example



MS16-062	Win32k Elevation of Privilege Vulnerability	CVE-2016
MS16-062	Win32k Elevation of Privilege Vulnerability	CVE-2016
MS16-062	Win32k Elevation of Privilege Vulnerability	CVE-2016
MS16-062	Win32k Elevation of Privilege Vulnerability	CVE-2016-017
MS16-062	Win32k Information Disclosure Vulnerability	CVE-2016-017
MS16-062	Microsoft DirectX Graphics Kernel Subsystem Elevation of Privilege Vulnerability	CVE-2016-017
MS16-062	Win32k Elevation of Privilege Vulnerability	CVE-2016-019
MS16-062	Win32k Elevation of Privilege Vulnerability	CVE-2016-019



technet.microsoft.com/en-us/library/security/mt674627.aspx

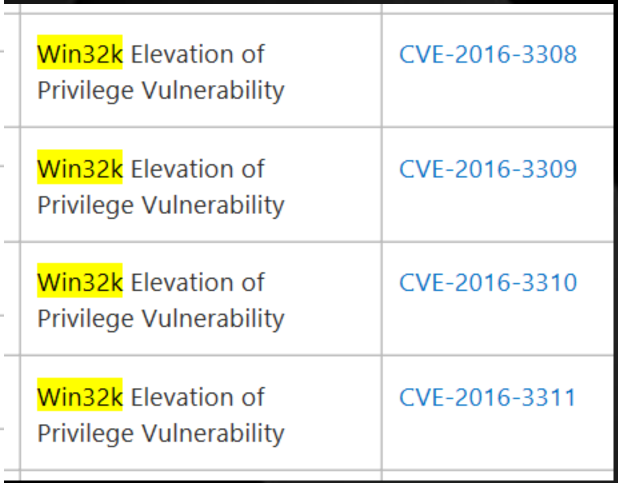
win32k　1 of 33　Options

Security Advisories and Bulletins > Acknowledgments

2016

Acknowledgments – 2016

MS16-090	Win32k Elevation of Privilege Vulnerability	CVE-2016-3249
MS16-090	Win32k Elevation of Privilege Vulnerability	CVE-2016-3250
MS16-090	GDI Component Information Disclosure Vulnerability	CVE-2016-3251
MS16-090	Win32k Elevation of Privilege Vulnerability	CVE-2016-3252
MS16-090	Win32k Elevation of Privilege Vulnerability	CVE-2016-3254
MS16-090	Microsoft win32k Elevation of Privilege Vulnerability	CVE-2016-3286

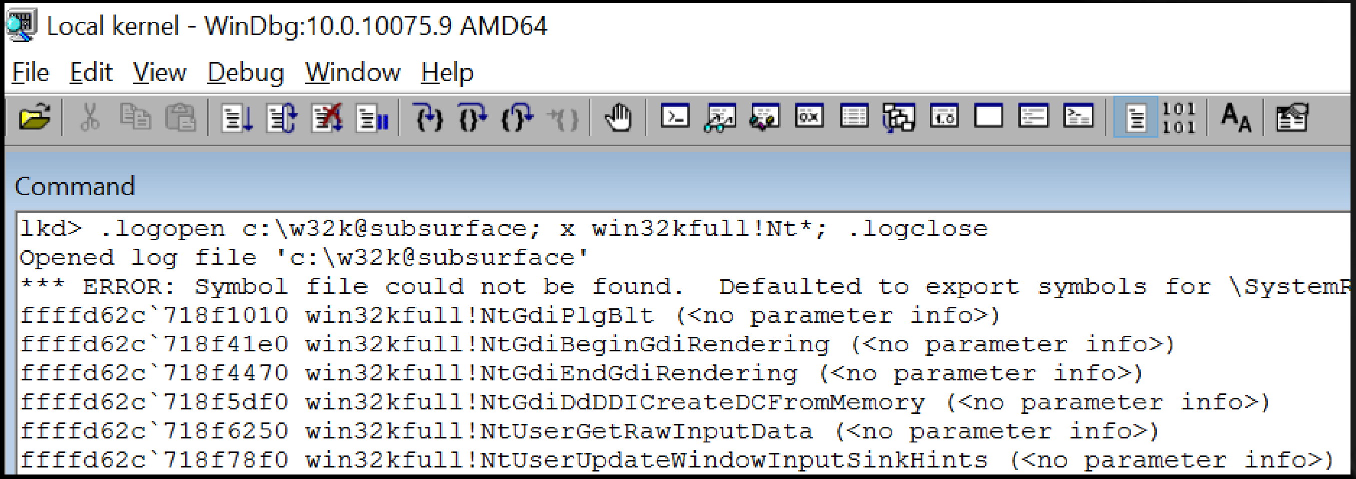


Win32k Elevation of Privilege Vulnerability	CVE-2016-3308
Win32k Elevation of Privilege Vulnerability	CVE-2016-3309
Win32k Elevation of Privilege Vulnerability	CVE-2016-3310
Win32k Elevation of Privilege Vulnerability	CVE-2016-3311

attack surface

- once i said big one, i meaned it!

```
C:\>cat w32k@subsurface | grep "Nt" | wc -l
1042
```



```
lkd> .logopen c:\w32k@subsurface; x win32kfull!Nt*; .logclose
Opened log file 'c:\w32k@subsurface'
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for \SystemR
ffffd62c`718f1010 win32kfull!NtGdiPlgBlt (<no parameter info>)
ffffd62c`718f41e0 win32kfull!NtGdiBeginGdiRendering (<no parameter info>)
ffffd62c`718f4470 win32kfull!NtGdiEndGdiRendering (<no parameter info>)
ffffd62c`718f5df0 win32kfull!NtGdiDdDDICreateDCFromMemory (<no parameter info>)
ffffd62c`718f6250 win32kfull!NtUserGetRawInputData (<no parameter info>)
ffffd62c`718f78f0 win32kfull!NtUserUpdateWindowInputSinkHints (<no parameter info>)
```

Local kernel - WinDbg:10.0.10075.9 AMD64
File Edit View Debug Window Help
Command

what is going on in w32k ?

- huge numbers of syscalls
- lot of objects
- lot of hardcore graphics stuffs
- lot of things i dunno

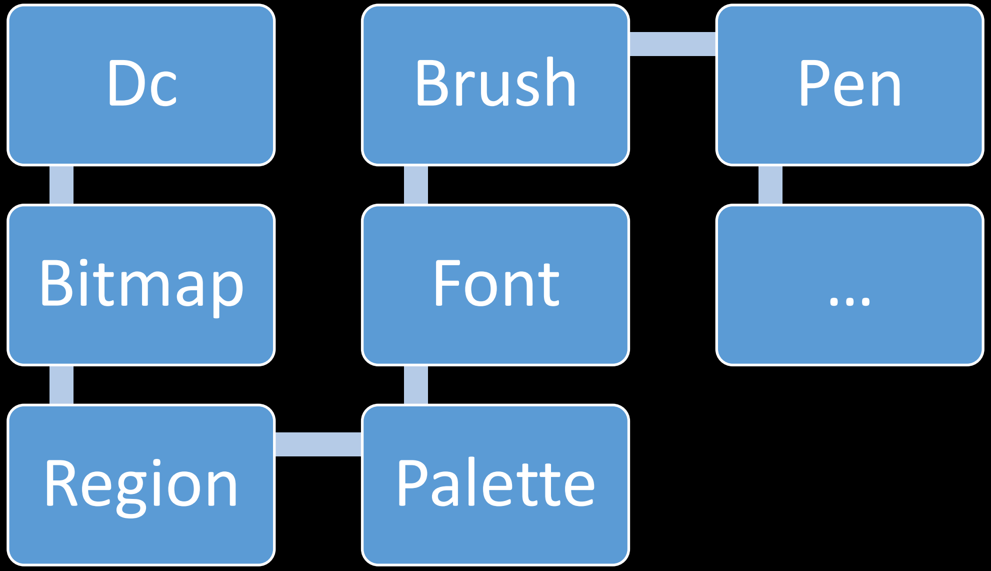
DC - lets paint

vector 1:

includes fair amount of functions

vector 2 :

interconnects nice number of different objects



fonts - did i mention it ?

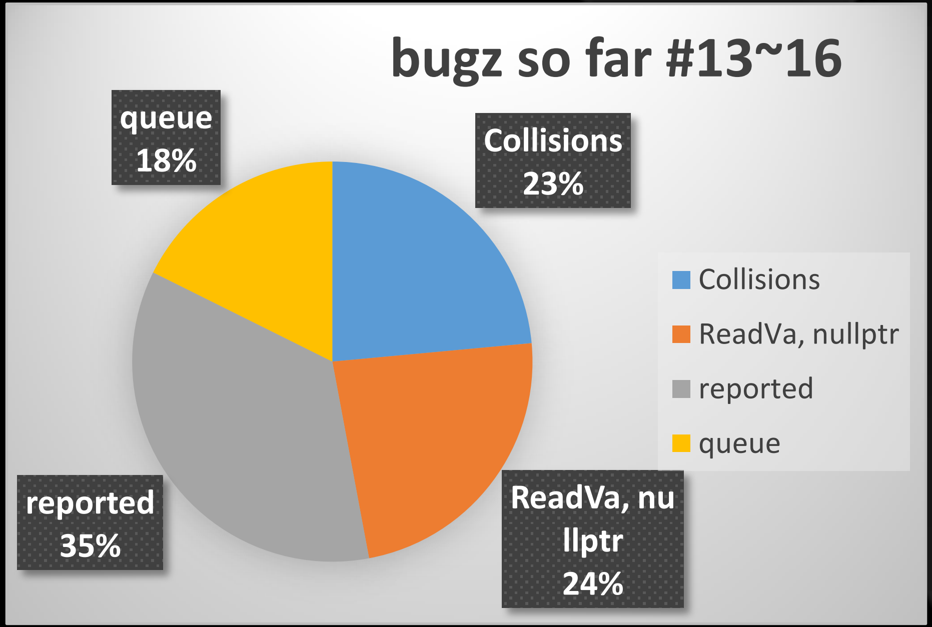
- various prelevant (mis)usage of different actors
  - stuxnet, duqu, ...
- our p2o 2015 target (2 x TTF to kernel code exec)
- j00ru heroic cleaning

- 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



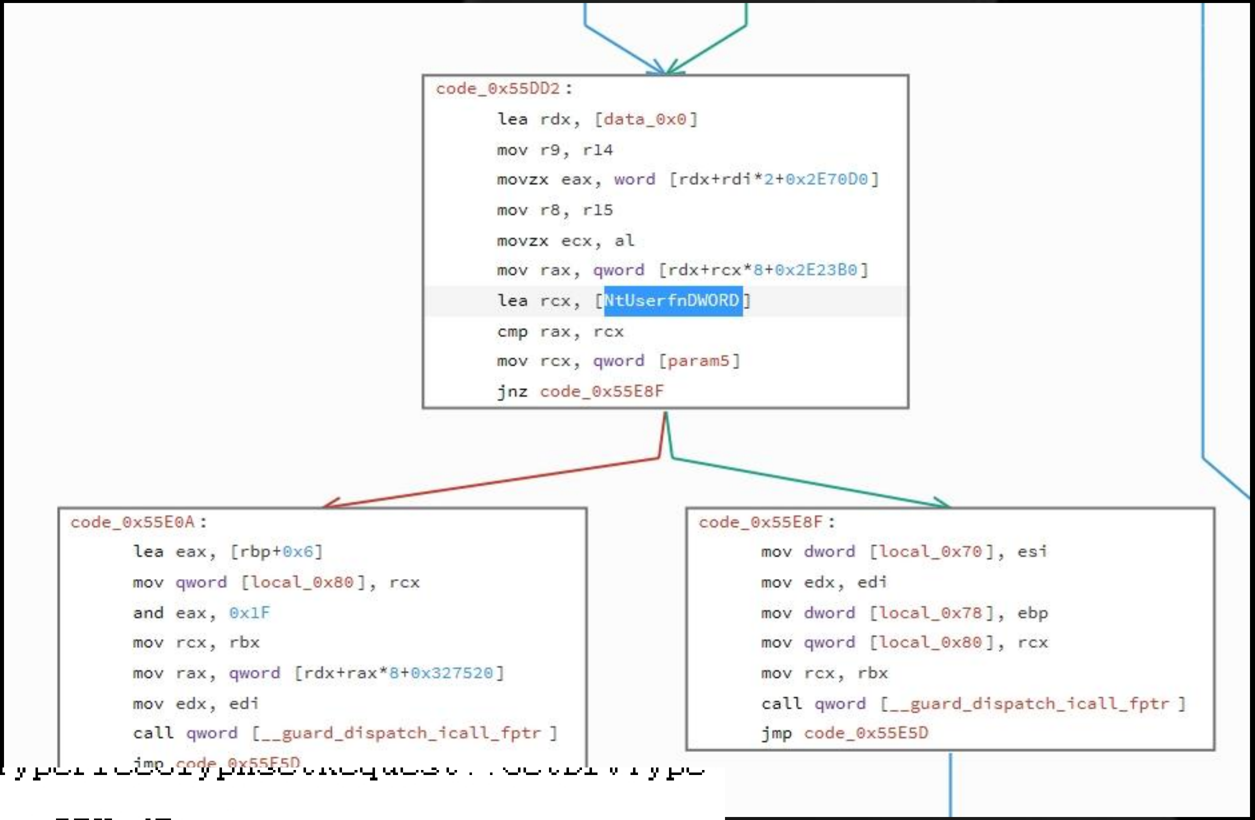
win32k - surface

- 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"

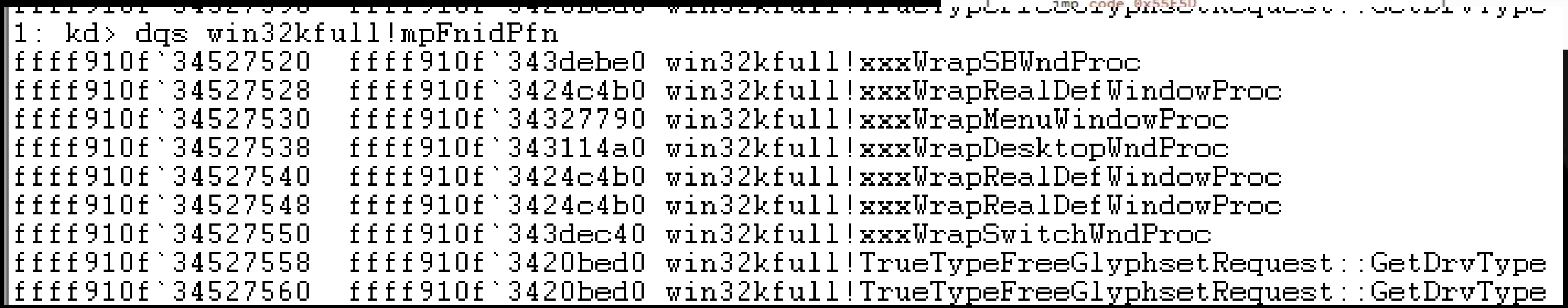
  behind one syscall!



```
code_0x55DD2:
        lea rdx, [data_0x0]
        mov r9, r14
        movzx eax, word [rdx+rdi*2+0x2E70D0]
        mov r8, r15
        movzx ecx, al
        mov rax, qword [rdx+rcx*8+0x2E23B0]
        lea rcx, [NtUserfnDWORD]
        cmp rax, rcx
        mov rcx, qword [param5]
        jnz code_0x55E8F
```

```
code_0x55E0A:
        lea eax, [rbp+0x6]
        mov qword [local_0x80], rcx
        and eax, 0x1F
        mov rcx, rbx
        mov rax, qword [rdx+rax*8+0x327520]
        mov edx, edi
        call qword [__guard_dispatch_icall_fptr]
        jmp code_0x55E5D
```

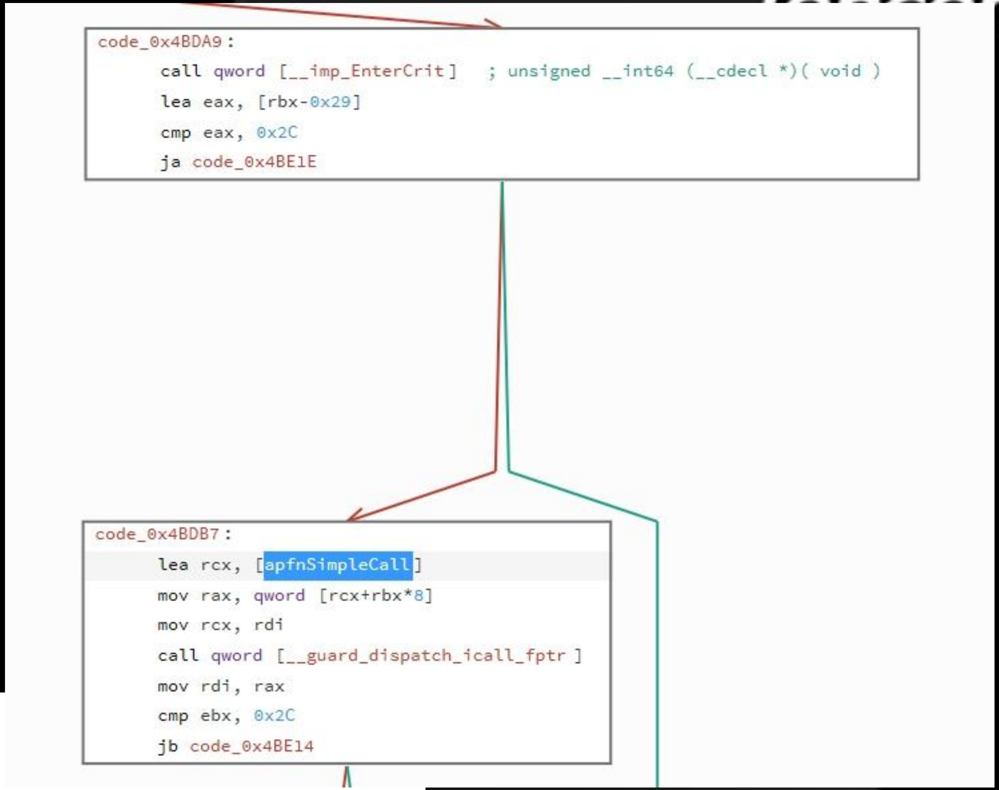
```
code_0x55E8F:
        mov dword [local_0x70], esi
        mov edx, edi
        mov dword [local_0x78], ebp
        mov qword [local_0x80], rcx
        mov rcx, rbx
        call qword [__guard_dispatch_icall_fptr]
        jmp code_0x55E5D
```



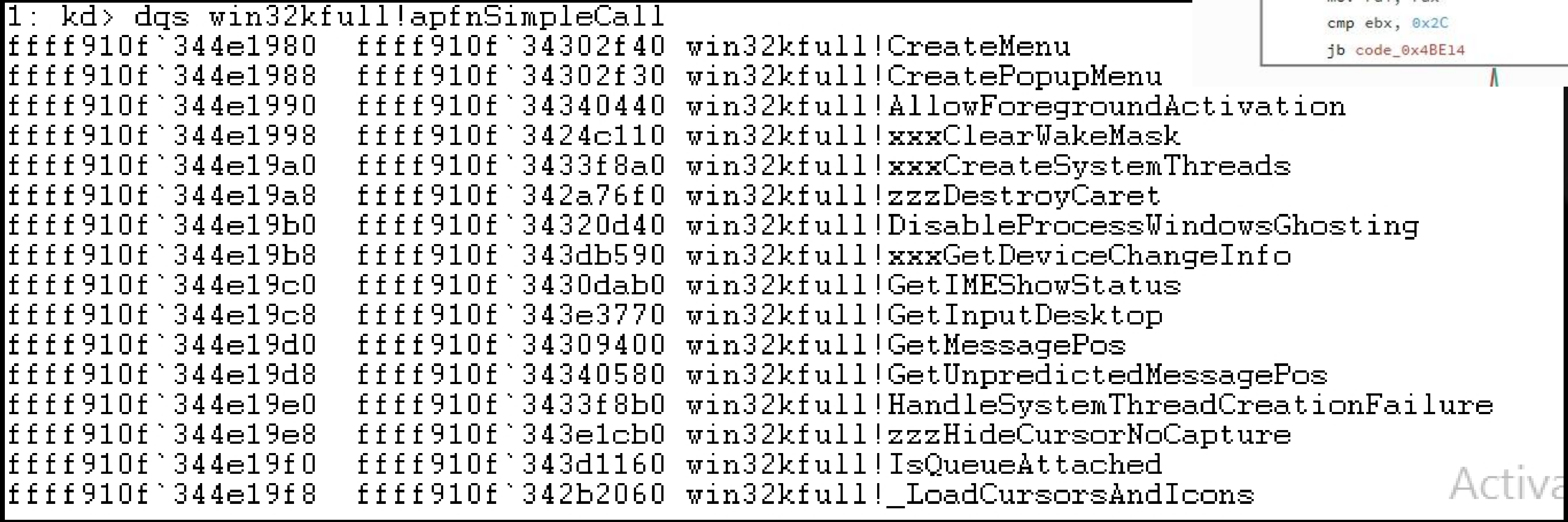
```
1: kd> dqs win32kfull!mpFnidPfn
ffff910f`34527520  ffff910f`343debe0 win32kfull!xxxWrapSBWndProc
ffff910f`34527528  ffff910f`3424c4b0 win32kfull!xxxWrapRealDefWindowProc
ffff910f`34527530  ffff910f`34327790 win32kfull!xxxWrapMenuWindowProc
ffff910f`34527538  ffff910f`343114a0 win32kfull!xxxWrapDesktopWndProc
ffff910f`34527540  ffff910f`3424c4b0 win32kfull!xxxWrapRealDefWindowProc
ffff910f`34527548  ffff910f`3424c4b0 win32kfull!xxxWrapRealDefWindowProc
ffff910f`34527550  ffff910f`343dec40 win32kfull!xxxWrapSwitchWndProc
ffff910f`34527558  ffff910f`3420bed0 win32kfull!TrueTypeFreeGlyphsetRequest::GetDrvType
ffff910f`34527560  ffff910f`3420bed0 win32kfull!TrueTypeFreeGlyphsetRequest::GetDrvType
```

NtUserCall * => apfn table

- nice 'hidden' ioctl-alike attack surface :
- notice CreateMenu, and others
- + > 0x80 syscalls

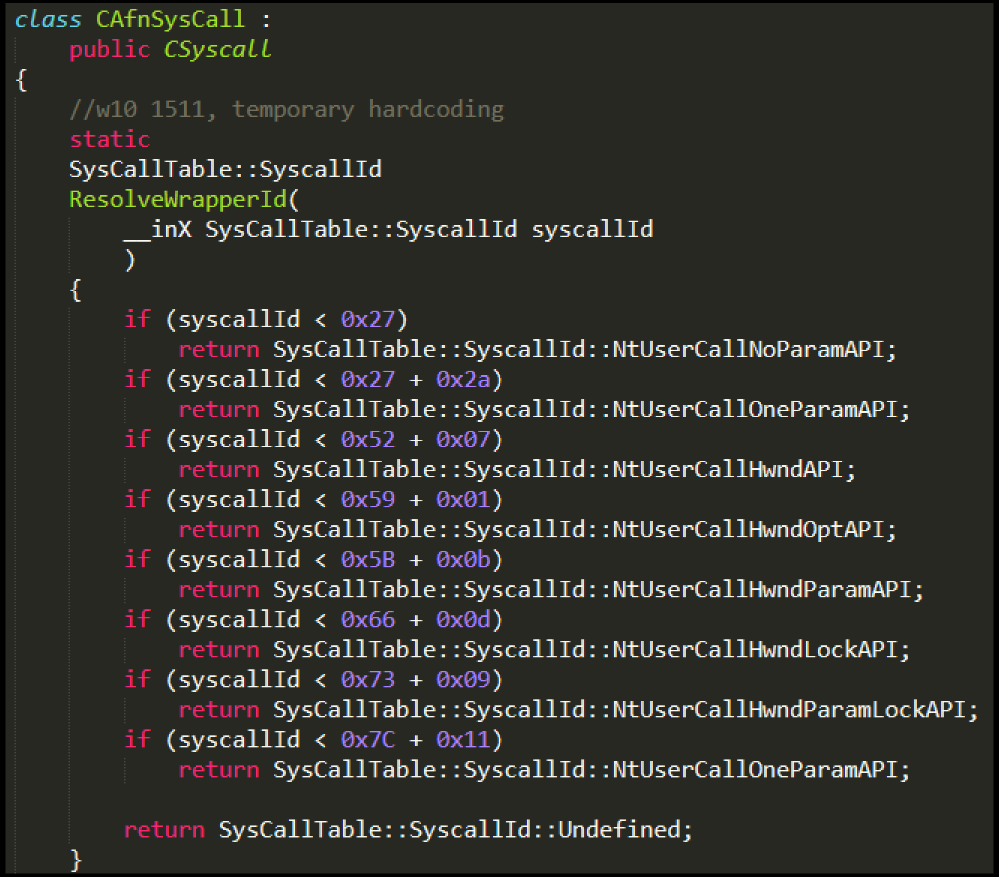


```
code_0x4BDA9:
        call qword [__imp_EnterCrit]   ; unsigned __int64 (__cdecl *)( void )
        lea eax, [rbx-0x29]
        cmp eax, 0x2C
        ja code_0x4BE1E

code_0x4BDB7:
        lea rcx, [apfnSimpleCall]
        mov rax, qword [rcx+rbx*8]
        mov rcx, rdi
        call qword [__guard_dispatch_icall_fptr ]
        mov rdi, rax
        cmp ebx, 0x2C
        jb code_0x4BE14
```



```
1: kd> dqs win32kfull!apfnSimpleCall
ffff910f`344e1980  ffff910f`34302f40 win32kfull!CreateMenu
ffff910f`344e1988  ffff910f`34302f30 win32kfull!CreatePopupMenu
ffff910f`344e1990  ffff910f`34340440 win32kfull!AllowForegroundActivation
ffff910f`344e1998  ffff910f`3424c110 win32kfull!xxxClearWakeMask
ffff910f`344e19a0  ffff910f`3433f8a0 win32kfull!xxxCreateSystemThreads
ffff910f`344e19a8  ffff910f`342a76f0 win32kfull!zzzDestroyCaret
ffff910f`344e19b0  ffff910f`34320d40 win32kfull!DisableProcessWindowsGhosting
ffff910f`344e19b8  ffff910f`343db590 win32kfull!xxxGetDeviceChangeInfo
ffff910f`344e19c0  ffff910f`3430dab0 win32kfull!GetIMEShowStatus
ffff910f`344e19c8  ffff910f`343e3770 win32kfull!GetInputDesktop
ffff910f`344e19d0  ffff910f`34309400 win32kfull!GetMessagePos
ffff910f`344e19d8  ffff910f`34340580 win32kfull!GetUnpredictedMessagePos
ffff910f`344e19e0  ffff910f`3433f8b0 win32kfull!HandleSystemThreadCreationFailure
ffff910f`344e19e8  ffff910f`343e1cb0 win32kfull!zzzHideCursorNoCapture
ffff910f`344e19f0  ffff910f`343d1160 win32kfull!IsQueueAttached
ffff910f`344e19f8  ffff910f`342b2060 win32kfull!_LoadCursorsAndIcons
```

Qilin <- win32kfull!apfnSimpleCall



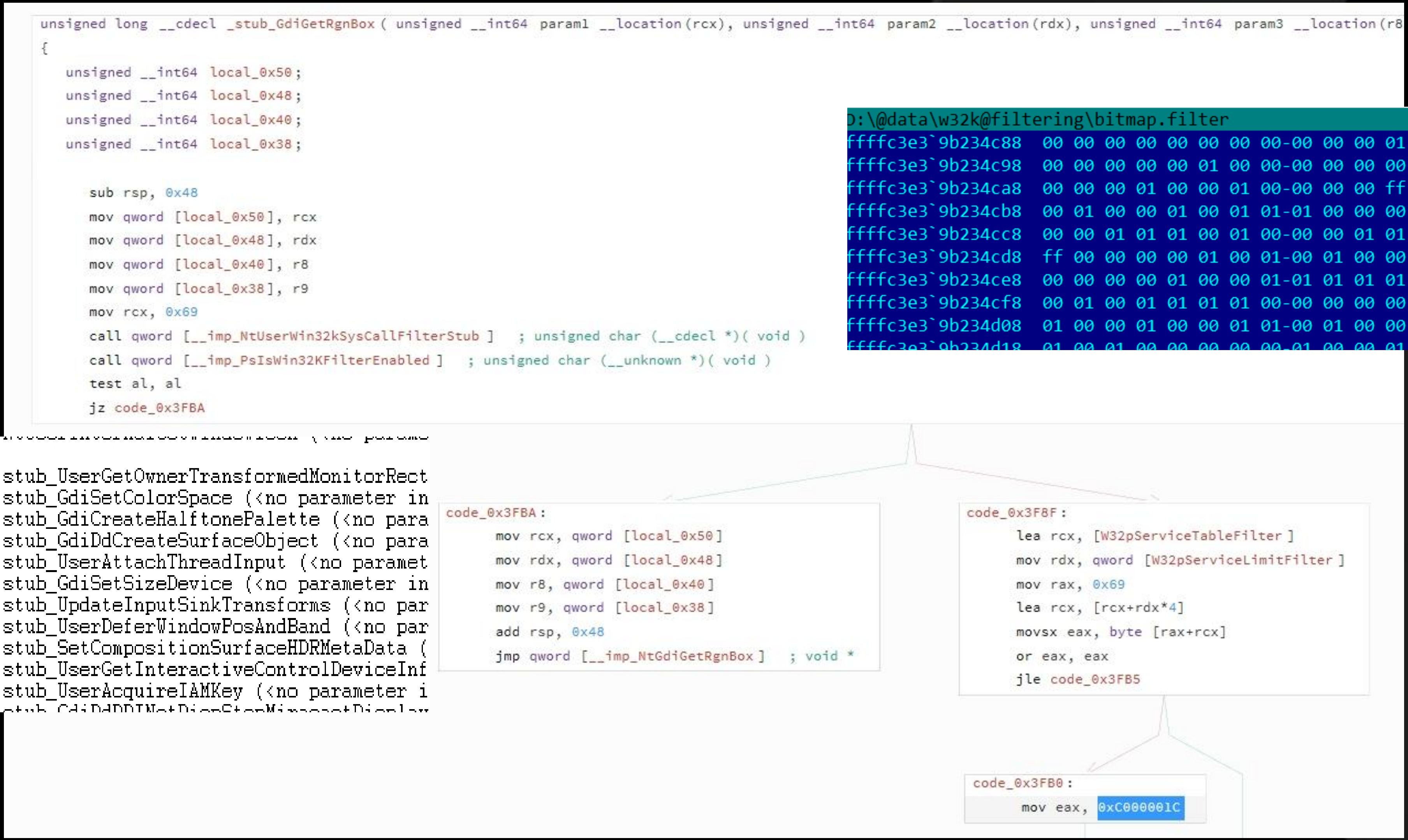
```
class CAfnSysCall :
    public CSyscall
{
    //w10 1511, temporary hardcoding
    static
    SysCallTable::SyscallId
    ResolveWrapperId(
        __inX SysCallTable::SyscallId syscallId
        )
    {
        if (syscallId < 0x27)
            return SysCallTable::SyscallId::NtUserCallNoParamAPI;
        if (syscallId < 0x27 + 0x2a)
            return SysCallTable::SyscallId::NtUserCallOneParamAPI;
        if (syscallId < 0x52 + 0x07)
            return SysCallTable::SyscallId::NtUserCallHwndAPI;
        if (syscallId < 0x59 + 0x01)
            return SysCallTable::SyscallId::NtUserCallHwndOptAPI;
        if (syscallId < 0x5B + 0x0b)
            return SysCallTable::SyscallId::NtUserCallHwndParamAPI;
        if (syscallId < 0x66 + 0x0d)
            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 takeways :
  - state alike fuzzing
    - less prone to bug
    - not so much code involved
    - basically wrappers and memory / locking mechanism
    - however universal bugs, independed of graphic
  - data fuzzing
    - mostly related to garphic drivers nvidia / intel
    - therefore not universal bugs
    - prone to bugs, lot ...

w32k filter

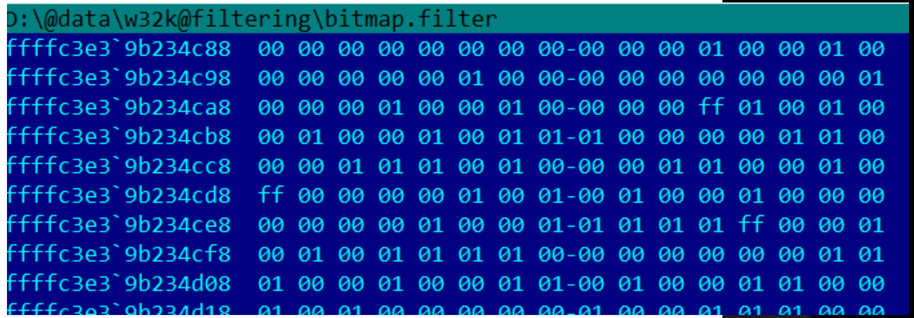


```
unsigned long __cdecl _stub_GdiGetRgnBox ( unsigned __int64 param1 __location(rcx), unsigned __int64 param2 __location(rdx), unsigned __int64 param3 __location(r8
{
  unsigned __int64 local_0x50;
  unsigned __int64 local_0x48;
  unsigned __int64 local_0x40;
  unsigned __int64 local_0x38;

    sub rsp, 0x48
    mov qword [local_0x50], rcx
    mov qword [local_0x48], rdx
    mov qword [local_0x40], r8
    mov qword [local_0x38], r9
    mov rcx, 0x69
    call qword [__imp_NtUserWin32kSysCallFilterStub ]   ; unsigned char (__cdecl *)( void )
    call qword [__imp_PsIsWin32KFilterEnabled ]   ; unsigned char (__unknown *)( void )
    test al, al
    jz code_0x3FBA
```

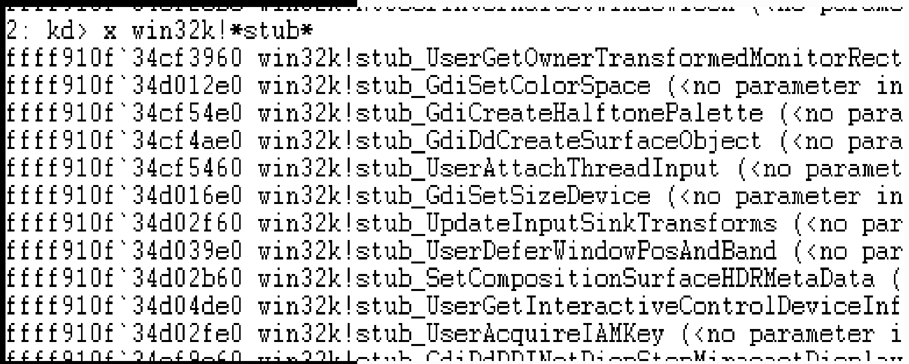
```
code_0x3FBA:
    mov rcx, qword [local_0x50]
    mov rdx, qword [local_0x48]
    mov r8, qword [local_0x40]
    mov r9, qword [local_0x38]
    add rsp, 0x48
    jmp qword [__imp_NtGdiGetRgnBox ]   ; void *
```

```
code_0x3F8F:
    lea rcx, [W32pServiceTableFilter ]
    mov rdx, qword [W32pServiceLimitFilter ]
    mov rax, 0x69
    lea rcx, [rcx+rdx*4]
    movsx eax, byte [rax+rcx]
    or eax, eax
    jle code_0x3FB5
```

```
code_0x3FB0:
    mov eax, 0xC000001C
```



```
D:\@data\w32k@filtering\bitmap.filter
ffffc3e3`9b234c88  00 00 00 00 00 00 00 00-00 00 00 01 00 00 01 00
ffffc3e3`9b234c98  00 00 00 00 00 01 00 00-00 00 00 00 00 00 00 01
ffffc3e3`9b234ca8  00 00 00 01 00 00 01 00-00 00 00 ff 01 00 01 00
ffffc3e3`9b234cb8  00 01 00 00 01 00 01 01-01 00 00 00 00 01 01 00
ffffc3e3`9b234cc8  00 00 01 01 01 00 01 00-00 00 01 01 00 00 01 00
ffffc3e3`9b234cd8  ff 00 00 00 00 01 00 01-00 01 00 00 01 00 00 00
ffffc3e3`9b234ce8  00 00 00 00 01 00 00 01-01 01 01 01 ff 00 00 01
ffffc3e3`9b234cf8  00 01 00 01 01 01 01 00-00 00 00 00 00 00 01 01
ffffc3e3`9b234d08  01 00 00 01 00 00 01 01-00 01 00 00 01 01 00 00
```



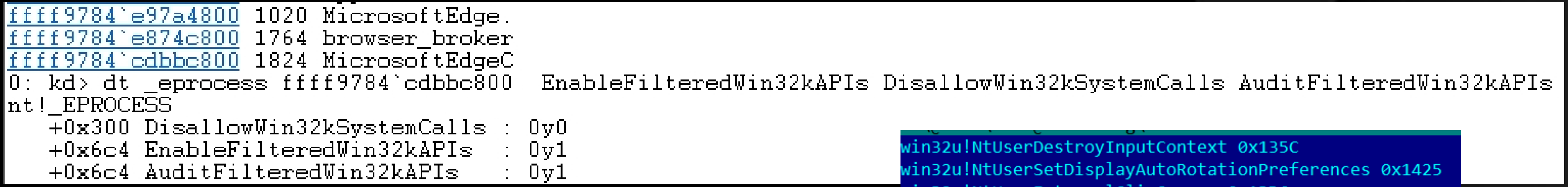
```
2: kd> x win32k!*stub*
ffff910f`34cf3960 win32k!stub_UserGetOwnerTransformedMonitorRect
ffff910f`34d012e0 win32k!stub_GdiSetColorSpace (<no parameter in
ffff910f`34cf54e0 win32k!stub_GdiCreateHalftonePalette (<no para
ffff910f`34cf4ae0 win32k!stub_GdiDdCreateSurfaceObject (<no para
ffff910f`34cf5460 win32k!stub_UserAttachThreadInput (<no paramet
ffff910f`34d016e0 win32k!stub_GdiSetSizeDevice (<no parameter in
ffff910f`34d02f60 win32k!stub_UpdateInputSinkTransforms (<no par
ffff910f`34d039e0 win32k!stub_UserDeferWindowPosAndBand (<no par
ffff910f`34d02b60 win32k!stub_SetCompositionSurfaceHDRMetaData (
ffff910f`34d04de0 win32k!stub_UserGetInteractiveControlDeviceInf
ffff910f`34d02fe0 win32k!stub_UserAcquireIAMKey (<no parameter i
```

w32k filter

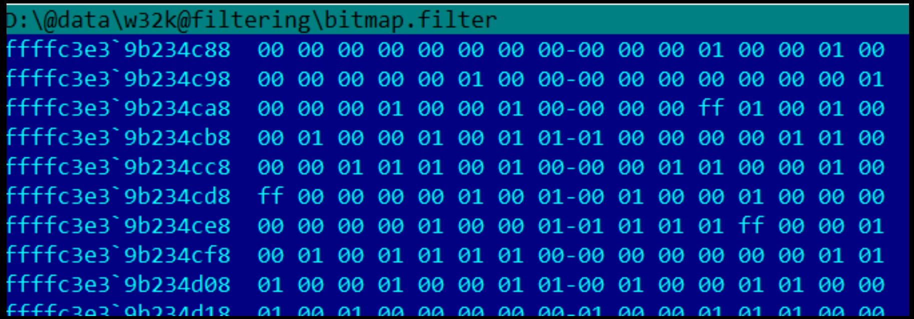
- introduced to limit unecessary access to w32k
- more benevolent that w32k lockdown
- limit attack surface for bug hunting
- limit exploitation techniques
- bitmap of allowed syscalls
- wrapped in win32k : “ x win32k!stub* “

w32k filter

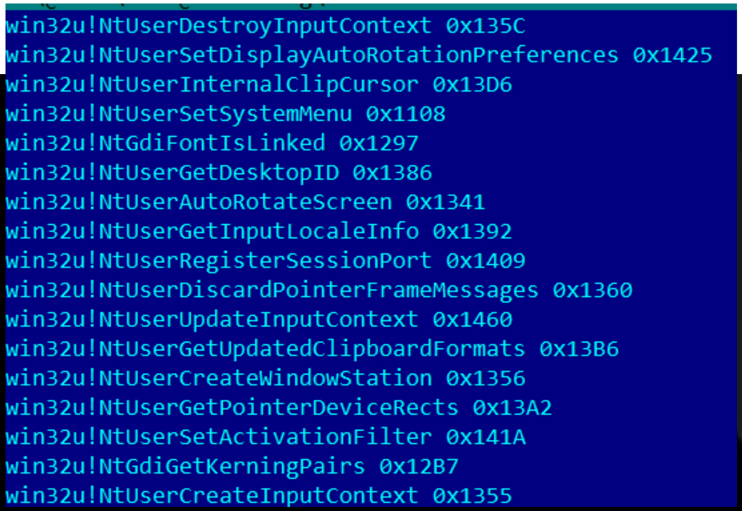
- bitmap of allowed w32k
  - edge example (part) :



```
ffff9784`e97a4800 1020 MicrosoftEdge.
ffff9784`e874c800 1764 browser_broker
ffff9784`cdbbc800 1824 MicrosoftEdgeC
0: kd> dt _eprocess ffff9784`cdbbc800  EnableFilteredWin32kAPIs DisallowWin32kSystemCalls AuditFilteredWin32kAPIs
nt!_EPROCESS
   +0x300 DisallowWin32kSystemCalls : 0y0
   +0x6c4 EnableFilteredWin32kAPIs  : 0y1
   +0x6c4 AuditFilteredWin32kAPIs   : 0y1
```



```
D:\@data\w32k@filtering\bitmap.filter
ffffc3e3`9b234c88  00 00 00 00 00 00 00 00-00 00 00 01 00 00 01 00
ffffc3e3`9b234c98  00 00 00 00 00 01 00 00-00 00 00 00 00 00 00 01
ffffc3e3`9b234ca8  00 00 00 01 00 00 01 00-00 00 00 ff 01 00 01 00
ffffc3e3`9b234cb8  00 01 00 00 01 00 01 01-01 00 00 00 00 01 01 00
ffffc3e3`9b234cc8  00 00 01 01 01 00 01 00-00 00 01 01 00 00 01 00
ffffc3e3`9b234cd8  ff 00 00 00 00 01 00 01-00 01 00 00 01 00 00 00
ffffc3e3`9b234ce8  00 00 00 00 01 00 00 01-01 01 01 01 ff 00 00 01
ffffc3e3`9b234cf8  00 01 00 01 01 01 01 00-00 00 00 00 00 00 01 01
ffffc3e3`9b234d08  01 00 00 01 00 00 01 01-00 01 00 00 01 01 00 00
ffffc3e3`9b234d18  01 00 01 00 00 00 00 00-01 00 00 01 01 01 00 00
```



```
win32u!NtUserDestroyInputContext 0x135C
win32u!NtUserSetDisplayAutoRotationPreferences 0x1425
win32u!NtUserInternalClipCursor 0x13D6
win32u!NtUserSetSystemMenu 0x1108
win32u!NtGdiFontIsLinked 0x1297
win32u!NtUserGetDesktopID 0x1386
win32u!NtUserAutoRotateScreen 0x1341
win32u!NtUserGetInputLocaleInfo 0x1392
win32u!NtUserRegisterSessionPort 0x1409
win32u!NtUserDiscardPointerFrameMessages 0x1360
win32u!NtUserUpdateInputContext 0x1460
win32u!NtUserGetUpdatedClipboardFormats 0x13B6
win32u!NtUserCreateWindowStation 0x1356
win32u!NtUserGetPointerDeviceRects 0x13A2
win32u!NtUserSetActivationFilter 0x141A
win32u!NtGdiGetKerningPairs 0x12B7
win32u!NtUserCreateInputContext 0x1355
```

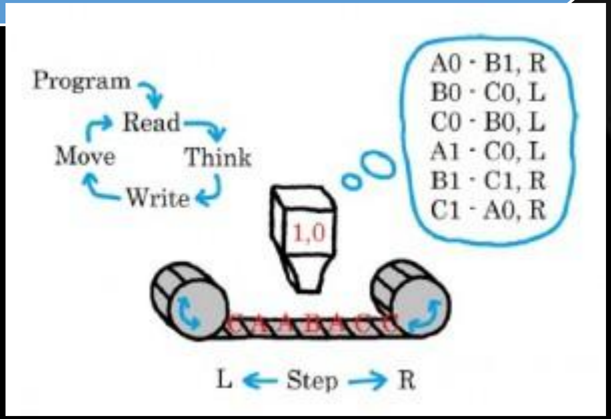
w32k indirect ways

- 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
  - trough condrv ioctls 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
    - altough syscalls not documented, use api knowledge + RE

How

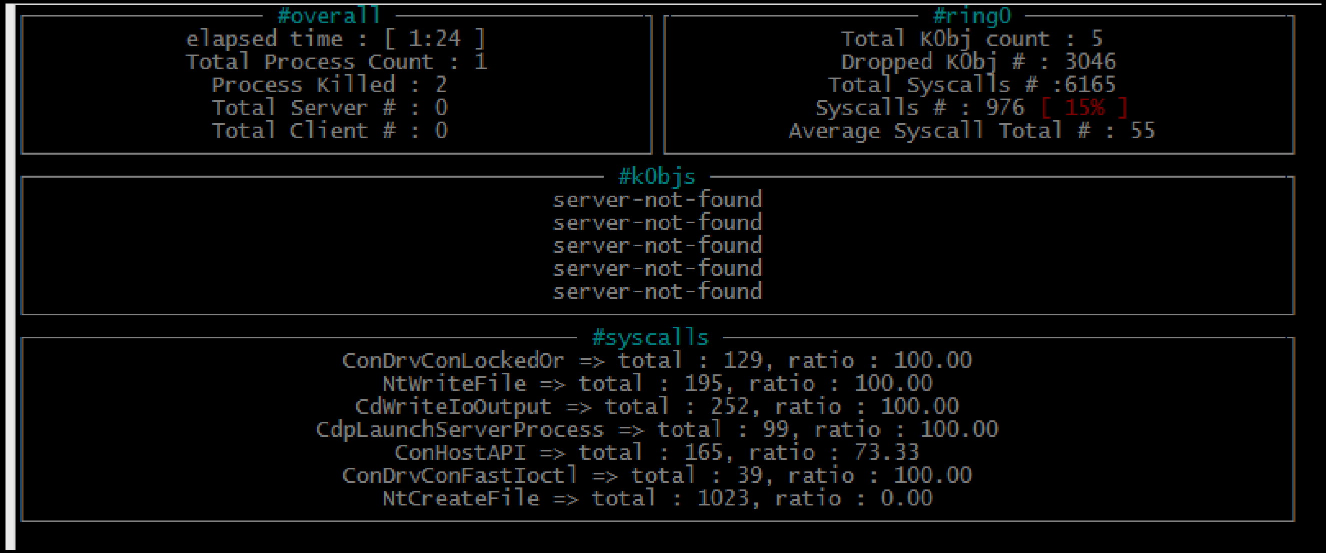
- templates
- examples of template fuzzers : trinity, syzkaller, mwr fuzzer
- our internal Qilin fuzzer
- grab api (REconstruct Nt* ones) definitions from msdn
- fill patterns with reasonable value ranges
- generate patterns

sophisticate it little bit

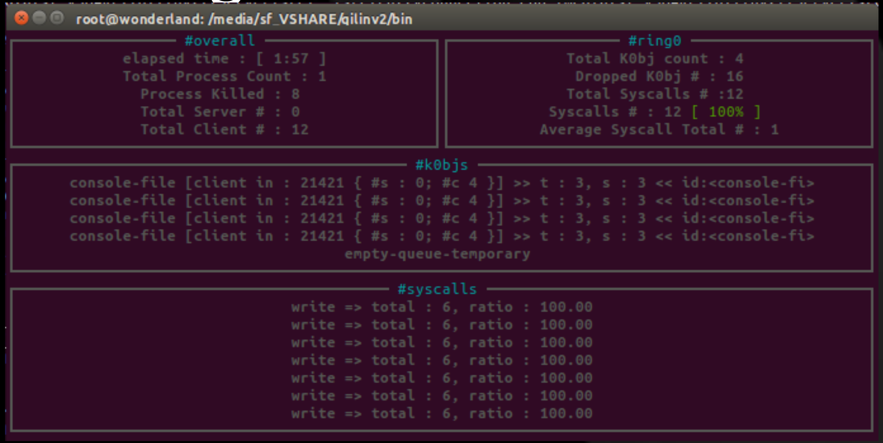
- sort patterns per object
  - Dc, Region, Bitmap, Font, ..
  - Window, Menu, UserMessage, afn, ..
- get meanigfull connections
  - get from database active handle of particular type
- get interrupted at user mode callbacks
  - involve some meaningfull syscalls then
- scope create - delete
  - dont let 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



```
#overall                                   #ring0
elapsed time : [ 1:24 ]                    Total K0bj count : 5
Total Process Count : 1                    Dropped K0bj # : 3046
Process Killed : 2                         Total Syscalls # :6165
Total Server # : 0                         Syscalls # : 976 [ 15% ]
Total Client # : 0                         Average Syscall Total # : 55

#k0bjs
server-not-found
server-not-found
server-not-found
server-not-found
server-not-found

#syscalls
ConDrvConLockedOr => total : 129, ratio : 100.00
NtWriteFile => total : 195, ratio : 100.00
CdWriteIoOutput => total : 252, ratio : 100.00
CdpLaunchServerProcess => total : 99, ratio : 100.00
ConHostAPI => total : 165, ratio : 73.33
ConDrvConFastIoctl => total : 39, ratio : 100.00
NtCreateFile => total : 1023, ratio : 0.00
```



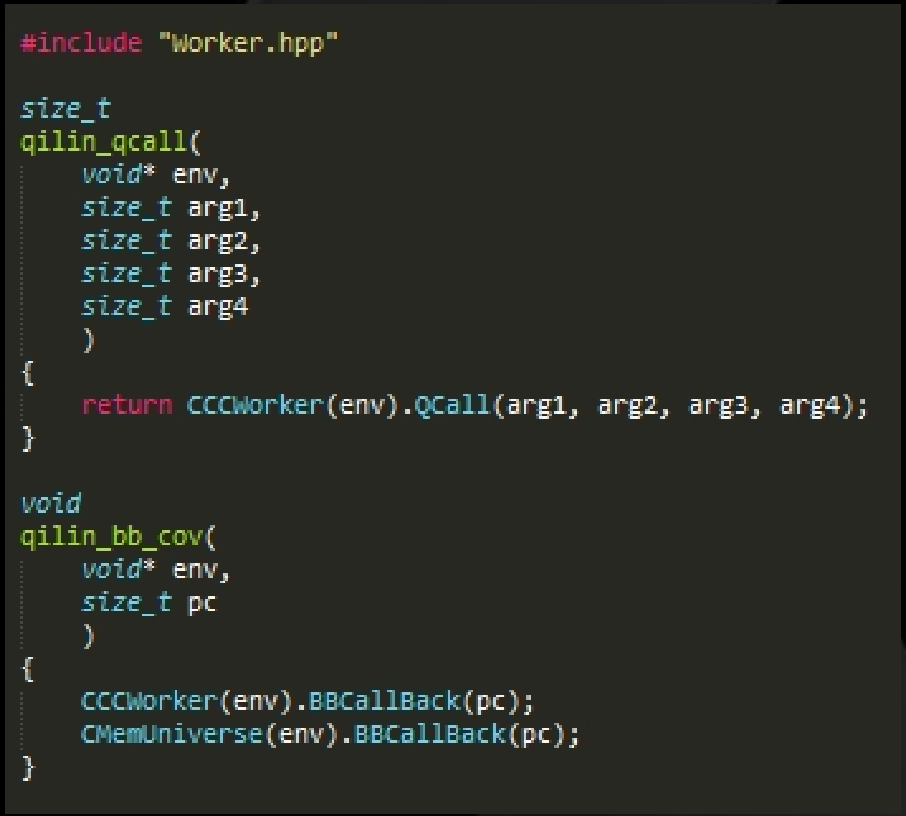
```
root@wonderland: /media/sf_VSHARE/qilinv2/bin

#overall                                   #ring0
elapsed time : [ 1:57 ]                    Total K0bj count : 4
Total Process Count : 1                    Dropped K0bj # : 16
Process Killed : 8                         Total Syscalls # :12
Total Server # : 0                         Syscalls # : 12 [ 100% ]
Total Client # : 12                        Average Syscall Total # : 1

#k0bjs
console-file [client in : 21421 { #s : 0; #c 4 }] >> t : 3, s : 3 << id:<console-fi>
console-file [client in : 21421 { #s : 0; #c 4 }] >> t : 3, s : 3 << id:<console-fi>
console-file [client in : 21421 { #s : 0; #c 4 }] >> t : 3, s : 3 << id:<console-fi>
console-file [client in : 21421 { #s : 0; #c 4 }] >> t : 3, s : 3 << id:<console-fi>
empty-queue-temporary

#syscalls
write => total : 6, ratio : 100.00
write => total : 6, ratio : 100.00
write => total : 6, ratio : 100.00
write => total : 6, ratio : 100.00
write => total : 6, ratio : 100.00
write => total : 6, ratio : 100.00
write => total : 6, ratio : 100.00
```

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 :)



```
#include "Worker.hpp"

size_t
qilin_qcall(
    void* env,
    size_t arg1,
    size_t arg2,
    size_t arg3,
    size_t arg4
    )
{
    return CCCWorker(env).QCall(arg1, arg2, arg3, arg4);
}

void
qilin_bb_cov(
    void* env,
    size_t pc
    )
{
    CCCWorker(env).BBCallBack(pc);
    CMemUniverse(env).BBCallBack(pc);
}
```

Edge EoP for Pwn2Own 2016

- Bug: CVE-2016-0176
- Bug Type: Kernel Heap Overflow
- Bug Driver: dxgkrnl.sys

_D3DKMT_PRESENTHISTORYTOKEN

```
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;
```

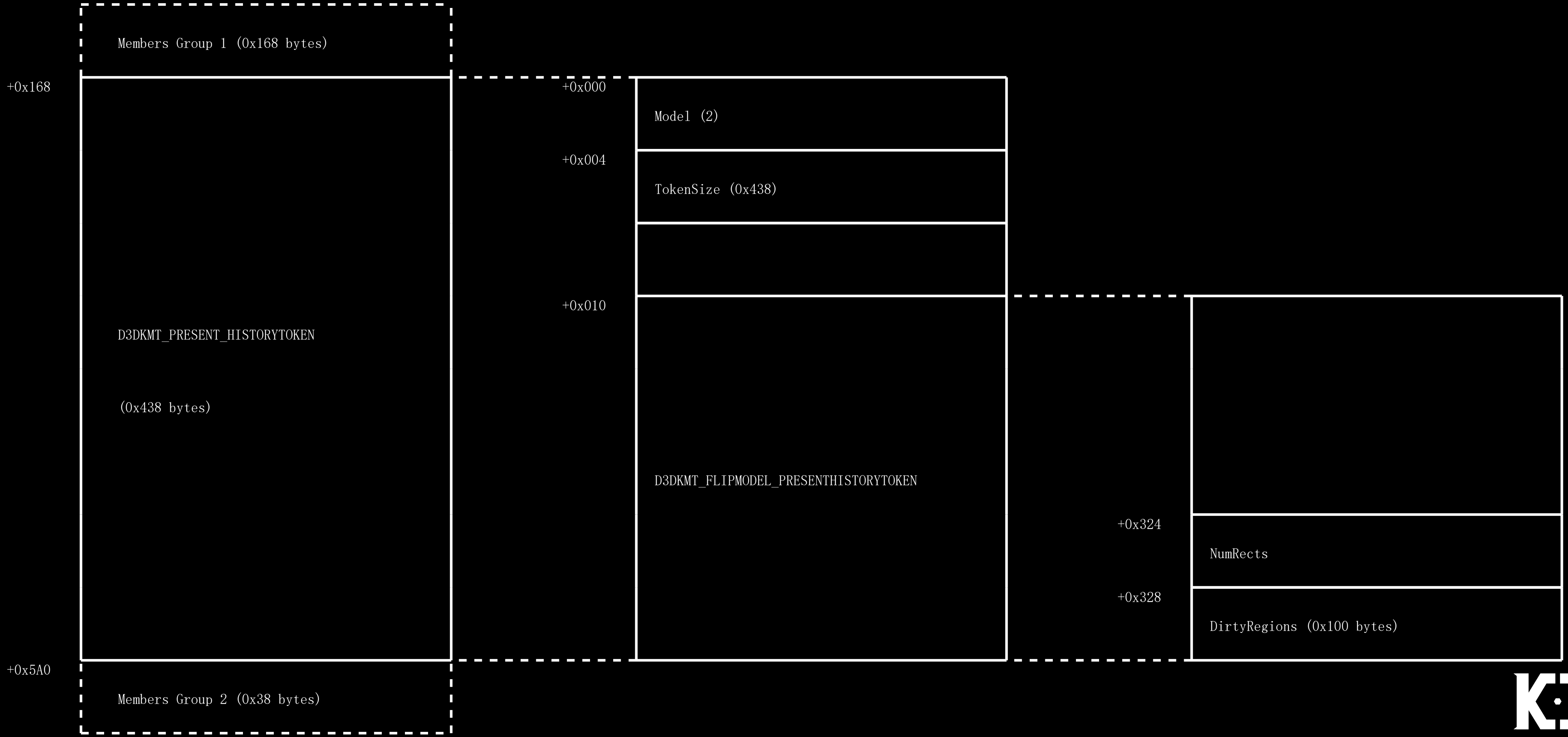
_D3DKMT_FLIPMODEL_PRESENTHISTORYTOKEN

```
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;
#endif
#if (DXGKDDI_INTERFACE_VERSION >= DXGKDDI_INTERFACE_VERSION_WDDM2_0)
    D3DDDI_COLOR_SPACE_TYPE                 ColorSpace;
#endif
    D3DKMT_DIRTYREGIONS                     DirtyRegions;
} D3DKMT_FLIPMODEL_PRESENTHISTORYTOKEN;
```

_D3DKMT_DIRTYREGIONS

```
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;
```

Layout


Members Group 1 (0x168 bytes)
+0x168
D3DKMT_PRESENT_HISTORYTOKEN
(0x438 bytes)
+0x5A0
Members Group 2 (0x38 bytes)
+0x000
Model (2)
+0x004
TokenSize (0x438)
+0x010
D3DKMT_FLIPMODEL_PRESENTHISTORYTOKEN
+0x324
NumRects
+0x328
DirtyRegions (0x100 bytes)

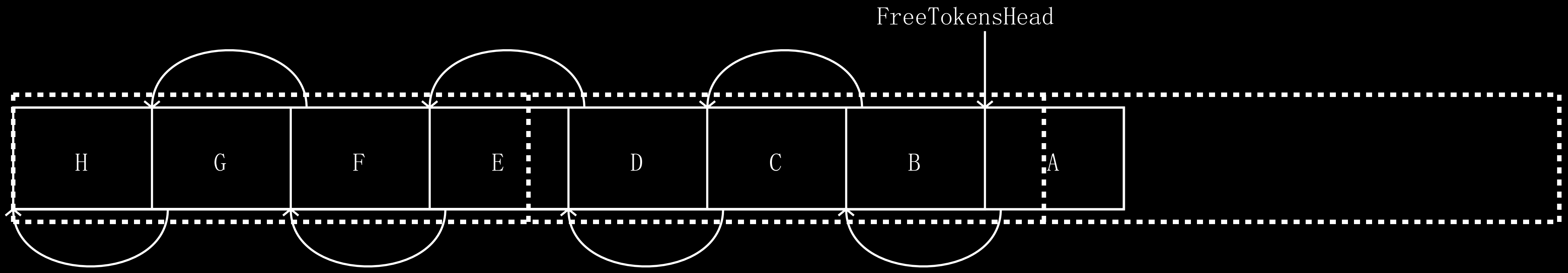
Overflow Code (Disassembly)

```
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++)

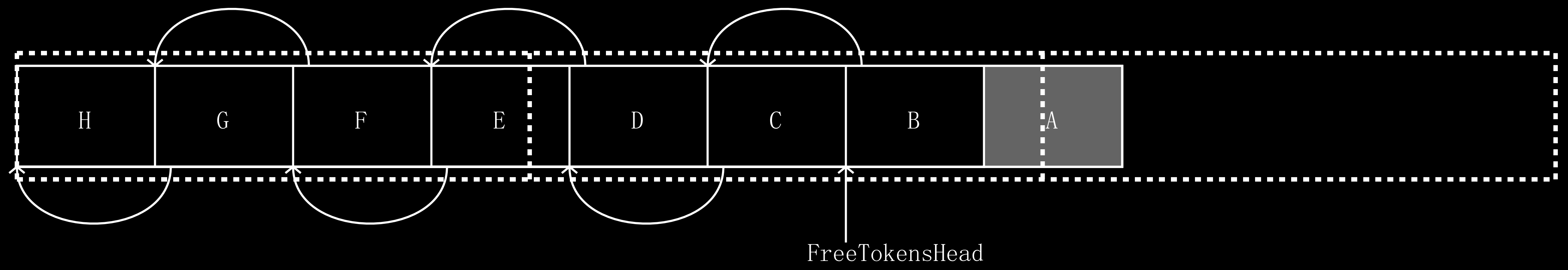
```
D3DKMT_PRESENTHISTORYTOKEN* hist_token_src = BufferPassedFromUserMode(…);
D3DKMT_PRESENTHISTORYTOKEN* hist_token_dst = ExpInterlockedPopEntrySList(…);

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



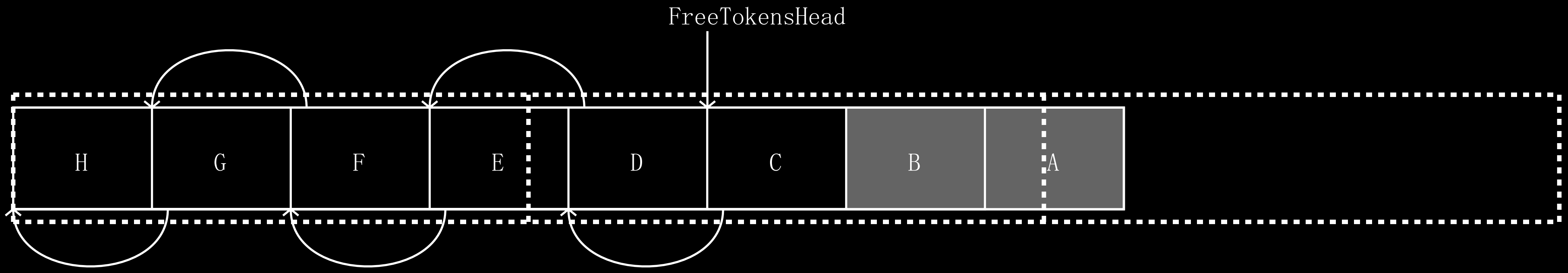
FreeSList: Head -> A -> B -> C -> D -> E -> F -> G -> H

Pop one node out for use (Pop A)



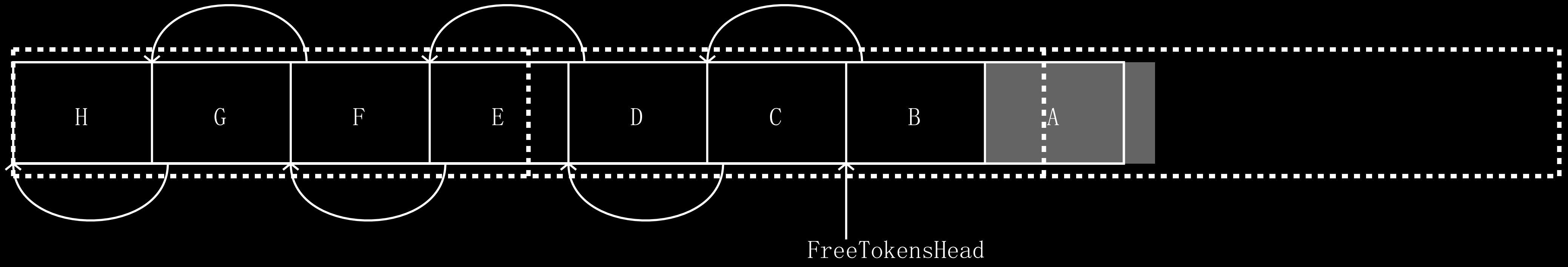
FreeSList: Head -> B -> C -> D -> E -> F -> G -> H

Pop another node out for use (Pop B)



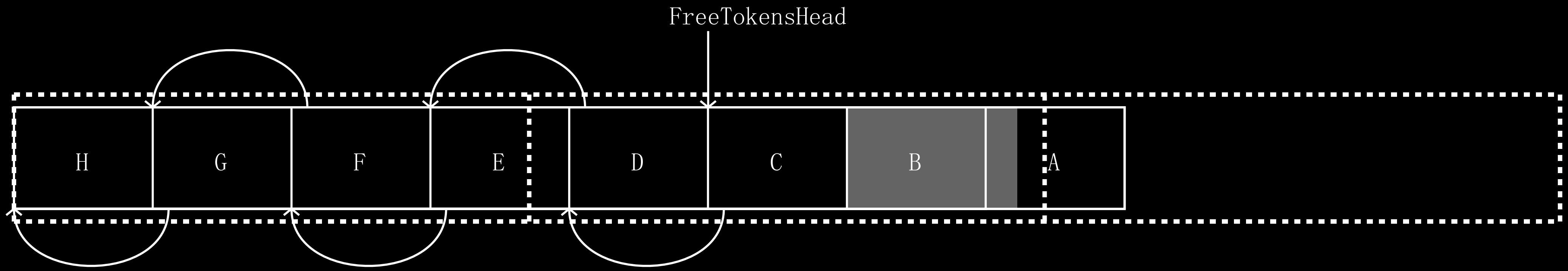
FreeSList: Head -> C -> D -> E -> F -> G -> H

Overflow Scenario 1



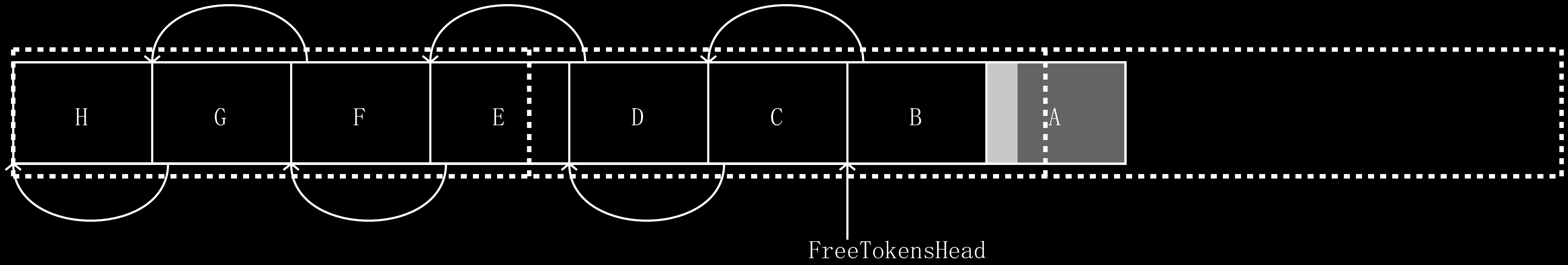
FreeSList: Head -> B -> C -> D -> E -> F -> G -> H

Overflow Scenario 2



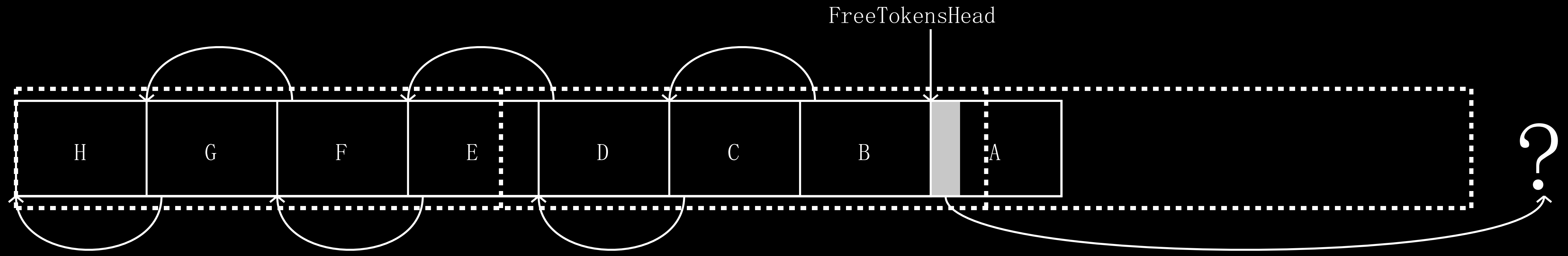
FreeSList: Head -> C -> D -> E -> F -> G -> H

Push node B back after overflow scenario 2



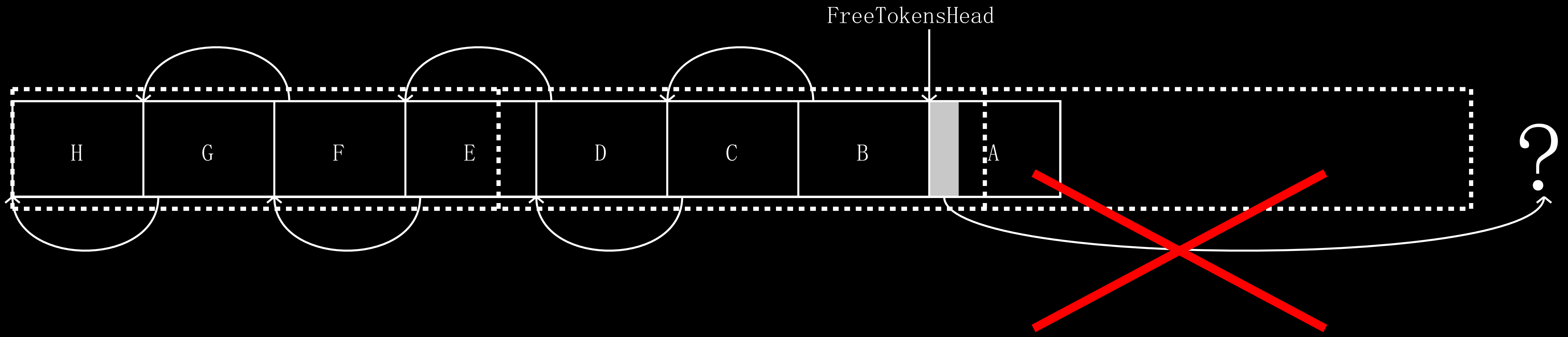
FreeSList: Head -> B -> C -> D -> E -> F -> G -> H

Will this overflow lead to arbitrary write?



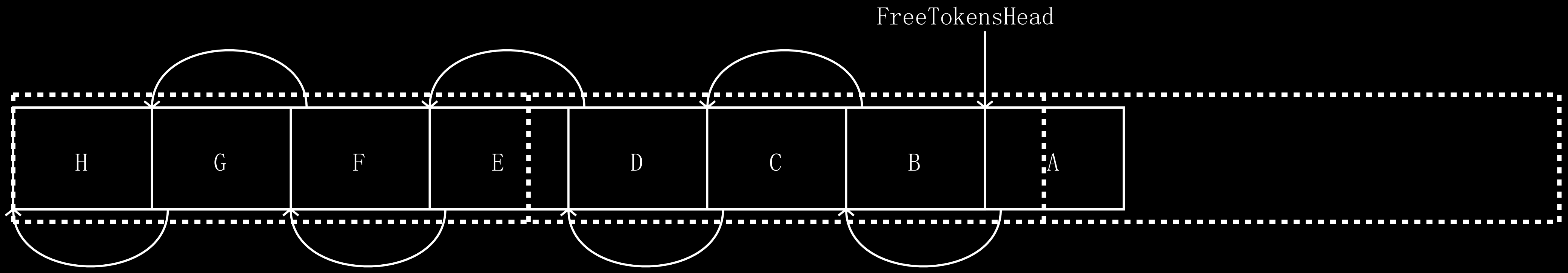
FreeSList: Head -> A -> B -> C -> D -> E -> F -> G -> H

Unfortunately!



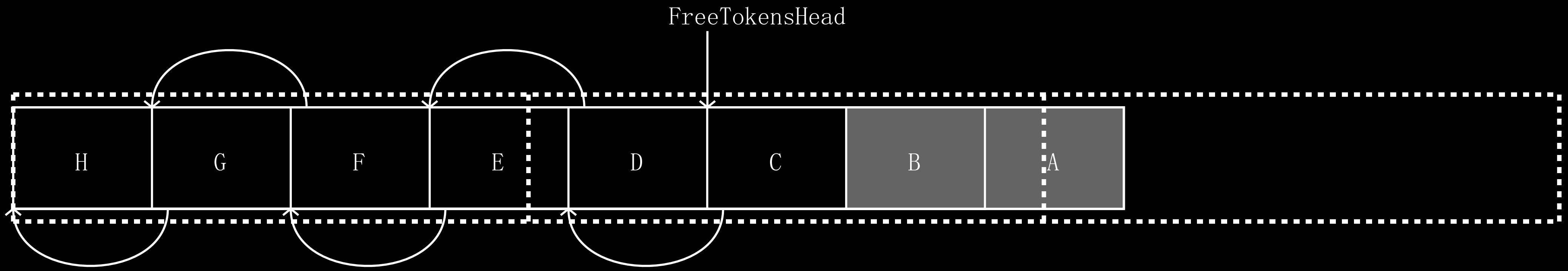
FreeSList: Head -> A -> B -> C -> D -> E -> F -> G -> H

The overwritten ‘Next’ field will be recovered



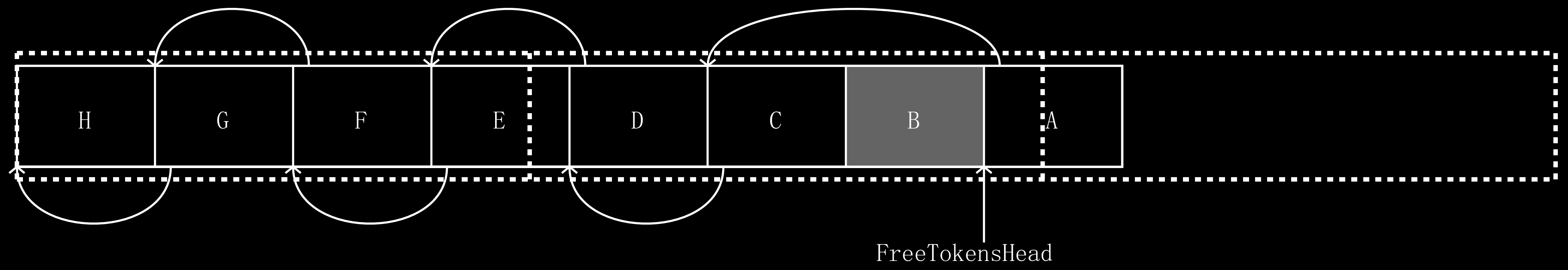
FreeSList: Head -> A -> B -> C -> D -> E -> F -> G -> H

Back to where after 2 pops



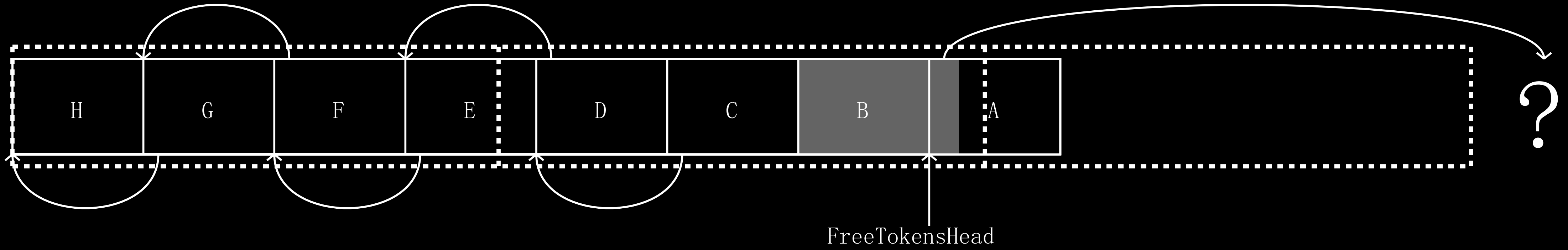
FreeSList: Head -> C -> D -> E -> F -> G -> H

Push in different orders with pop



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

Overflow Scenario 3



FreeSList: Head -> A -> ?

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
  - https://recon.cx/2015/slides/recon2015-05-peter-hlavaty-jihui-lu-This-Time-Font-hunt-you-down-in-4-bytes.pdf
- microsoft : mitigations
  - https://www.blackhat.com/docs/us-16/materials/us-16-Weston-Windows-10-Mitigation-Improvements.pdf
- qemu : win10 'fix' blog post
  - https://www.invincealabs.com/blog/2016/07/running-windows-64bit-qemu/

Tencent

KEEN
security
lab