Some Notes on VIRTUALGATE

norfolkinfosec.com/some-notes-on-virtualgate/

norfolk October 3, 2022

Late last week, Mandiant researchers published findings from an incident response engagement detailing an attacker workflow that took place in a VMWare ESXI environment. In this workflow, the attackers placed malware or persistence mechanisms on each layer of this environment:

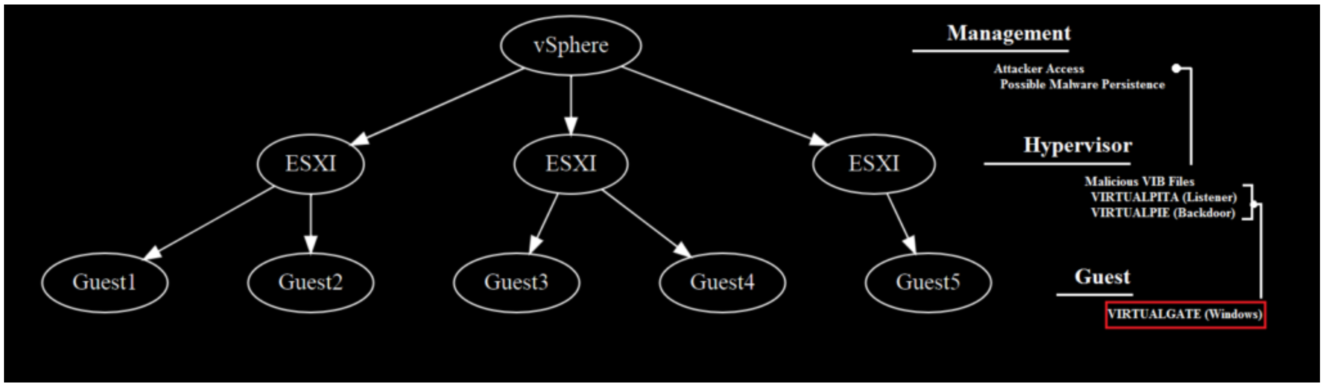
1. vSphere layer, which can manage multiple ESXI environments
2. ESXI hypervisor layer, which can manage virtualized “guest” machines
3. Virtualized guest machines

A key function of several of the attacker tools placed at the ESXI and guest levels in this environment was reportedly the ability to exchange attacker commands and data between the two layers.

This blog post examines a likely sample of VIRTUALGATE, a reported malware family that sits at the guest machine layer of this workflow. The post will provide additional technical details regarding its underlying mechanisms and provides context for how an attacker might operate in this environment.

vSphere, ESXI, and Attacker Malware

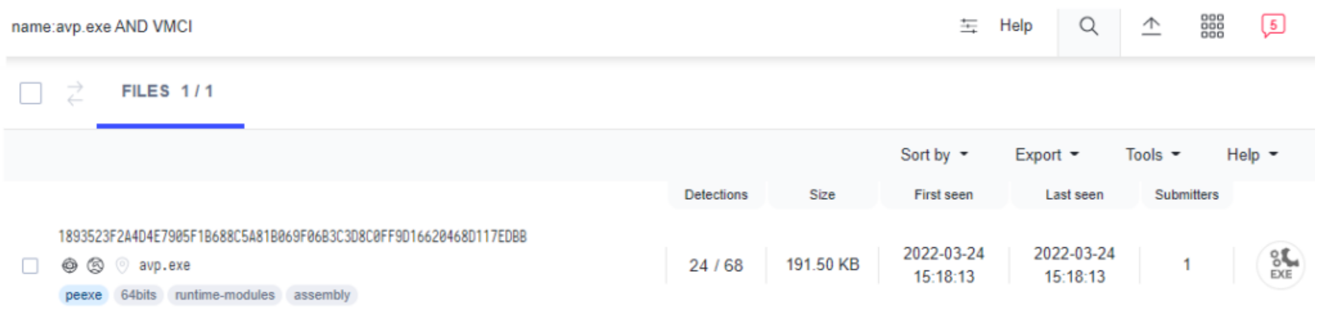
Mandiant’s blog post detailed several layers of activity for the attackers. At the top level, the attackers gained access to – and likely created persistence mechanisms within – the organization’s vSphere infrastructure. This in turn gave the attackers access to Virtual Machine hypervisors, which manage virtualized guest machines for the environment.



Although Mandiant provided hashes for some of these files, the files were not publicly available at the time of this writing; however, Mandiant does provide a file name (*avp.exe)* and a description for one of these files, which the authors named VIRTUALGATE, a malware family designed to run on Windows operating systems:

```
The memory only dropper... uses VMware's virtual machine communication interface
(VMCI) sockets to run commands on a guest virtual machine from a hypervisor host
```

These two datapoints are sufficient to find a possible candidate on VirusTotal:



name:avp.exe AND VMCI

FILES 1 / 1

	Detections	Size	First seen	Last seen	Submitters
1893523F2A4D4E7905F1B688C5A81B069F06B3C3D8C0FF9D16620468D117EDBB avp.exe peexe 64bits runtime-modules assembly	24 / 68	191.50 KB	2022-03-24 15:18:13	2022-03-24 15:18:13	1

As the remainder of this post will show, this file's functions align well with Mandiant's description. This post focuses more on how this file works rather than on further attribution efforts (although the technical relationships are implied), as the internal mechanisms would likely apply to other malware families that use the same communication techniques.

Technical Information

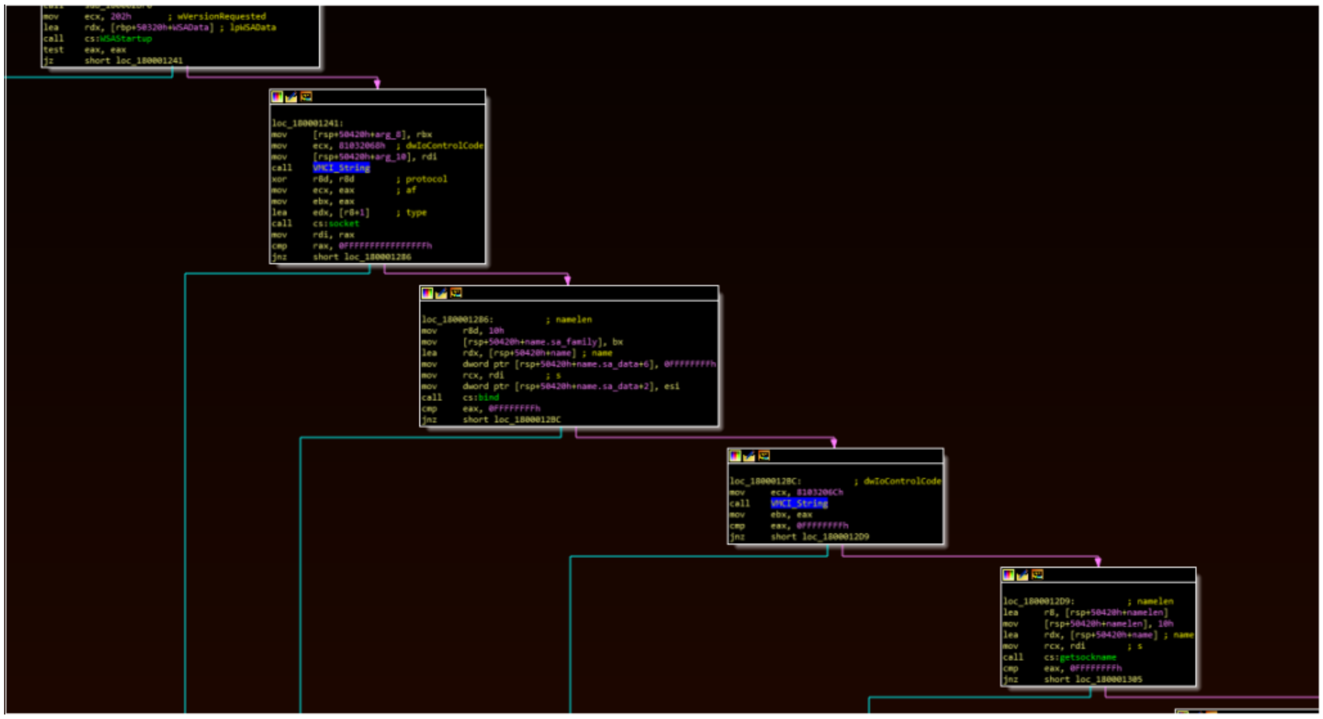
Main Workflow

Filename: avp.exe
MD5: 3c7316012cba3bbfa8a95d7277cda873
SHA1: d6a57b9aaa20fe4f3330f5979979081af09a4232
SHA256: 1893523f2a4d4e7905f1b688c5a81b069f06b3c3d8c0ff9d16620468d117edbb

As Mandiant noted, the file consists of a loader and DLL. Once loaded in memory, the payload uses three main components:

– A function containing API calls typically seen in files that use Sockets (Winsock)
– A function referencing cmd.exe
– A reference to a log file, with format strings suggesting timestamped entries

The socket workflow is responsible for launching the other two components.



```
mov     ecx, 202h       ; wVersionRequested
lea     rdx, [rbp+50320h+WSAData] ; lpWSAData
call    cs:WSAStartup
test    eax, eax
jz      short loc_180001241
```

```
loc_180001241:
mov     [rsp+50420h+arg_8], rbx
mov     ecx, 81032068h  ; dwIoControlCode
mov     [rsp+50420h+arg_10], rdi
call    VMCI_String
xor     r8d, r8d        ; protocol
mov     ecx, eax        ; af
mov     ebx, eax
lea     edx, [r8+1]     ; type
call    cs:socket
mov     rdi, rax
cmp     rax, 0FFFFFFFFFFFFFFFFh
jnz     short loc_180001286
```

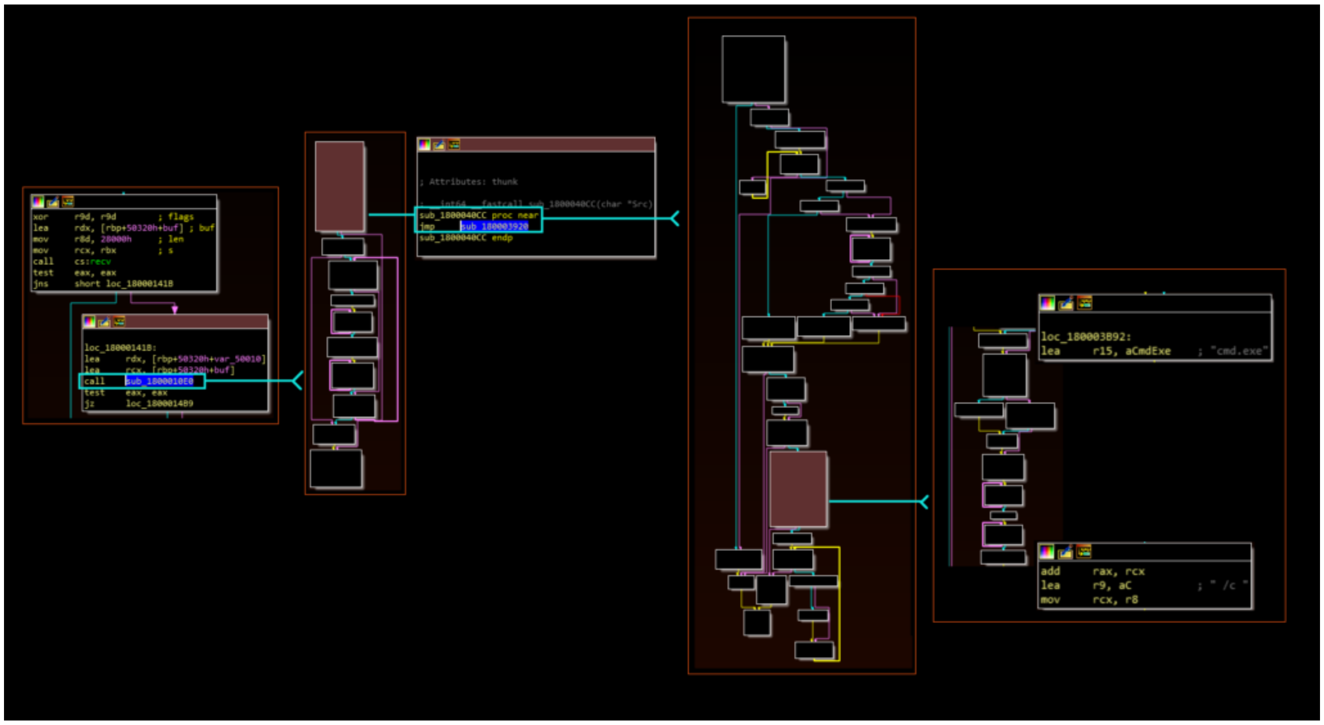
```
loc_180001286:          ; namelen
mov     r8d, 10h
mov     [rsp+50420h+name.sa_family], bx
lea     rdx, [rsp+50420h+name] ; name
mov     dword ptr [rsp+50420h+name.sa_data+6], 0FFFFFFFFh
mov     rcx, rdi        ; s
mov     dword ptr [rsp+50420h+name.sa_data+2], esi
call    cs:bind
cmp     eax, 0FFFFFFFFh
jnz     short loc_1800012BC
```

```
loc_1800012BC:          ; dwIoControlCode
mov     ecx, 8103206Ch
call    VMCI_String
mov     ebx, eax
cmp     eax, 0FFFFFFFFh
jnz     short loc_1800012D9
```

```
loc_1800012D9:          ; namelen
lea     r8, [rsp+50420h+namelen]
mov     [rsp+50420h+namelen], 10h
lea     rdx, [rsp+50420h+name] ; name
mov     rcx, rdi        ; s
call    cs:getsockname
cmp     eax, 0FFFFFFFFh
jnz     short loc_180001305
```

Socket API calls

Immediately following the "recv" API call, there is a series of function calls that eventually leads to the execution of cmd.exe with the "/c" option:



```
xor     r9d, r9d        ; flags
lea     rdx, [rbp+50320h+buf] ; buf
mov     r8d, 20000h     ; len
mov     rcx, rbx        ; s
call    cs:recv
test    eax, eax
jns     short loc_180001418
```

```
loc_180001418:
lea     rdx, [rbp+50320h+var_50010]
lea     rcx, [rbp+50320h+buf]
call    sub_1800010E0
test    eax, eax
jz      loc_180001489
```

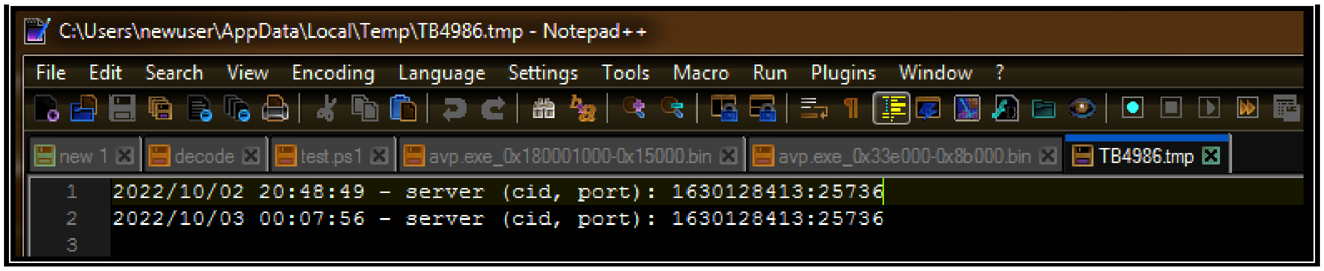
```
; Attributes: thunk

; __int64 __fastcall sub_1800040CC(char *Src)
sub_1800040CC proc near
jmp     sub_180003920
sub_1800040CC endp
```

```
loc_180003B92:
lea     r15, aCmdExe    ; "cmd.exe"
```

```
add     rax, rcx
lea     r9, aC          ; " /c "
mov     rcx, r8
```

Recv followed by branch leading to cmd.exe

Running the malware dynamically also creates a log entry in the user's AppData\Temp folder, the contents of which conform to the format string mentioned above:


C:\Users\newuser\AppData\Local\Temp\TB4986.tmp - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 1
decode
test.ps1
avp.exe_0x180001000-0x15000.bin
avp.exe_0x33e000-0x8b000.bin
TB4986.tmp

```
2022/10/02 20:48:49 - server (cid, port): 1630128413:25736
2022/10/03 00:07:56 - server (cid, port): 1630128413:25736
```

Log file generated during the malware's execution

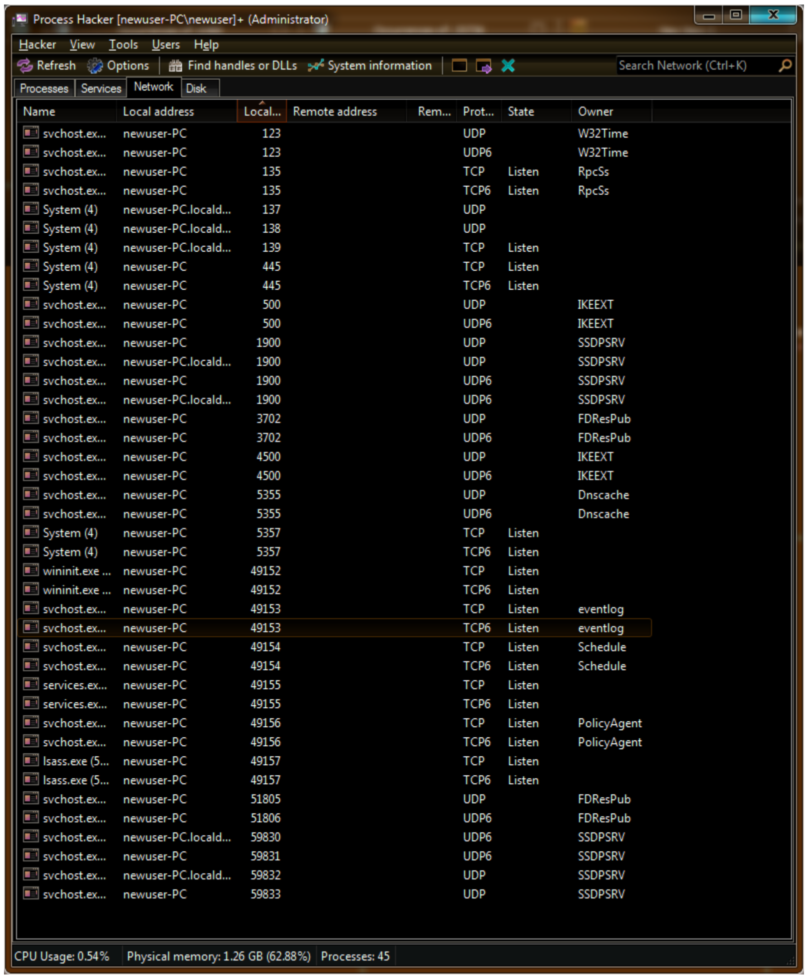
Notably, this log entry closely resembles the format of the logs produced by another file described in the Mandiant report, lending additional weight to the likelihood that this file is part of the same attacker toolset. The log entry implies that the malware is configured to interact using port 25736.

A reasonable hypothesis would be that the malware opens a listener (likely on port 25736) and executes data that it receives through this listener via cmd.exe, given the order of these observed functions as well as the contents of the log file and the description of VIRTUALGATE in the Mandiant report.

A Detour to VMCI Sockets

At this stage, we would want to test this hypothesis by sending data over this port.

Typically, during dynamic analysis, researchers can use tools such as Process Hacker to help identify network connections and processes that are listening on unusual ports. Unfortunately, the port does not appear in Process Hacker's capture:


Process Hacker [newuser-PC\newuser]+ (Administrator)
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Search Network (Ctrl+K)
Processes Services Network Disk

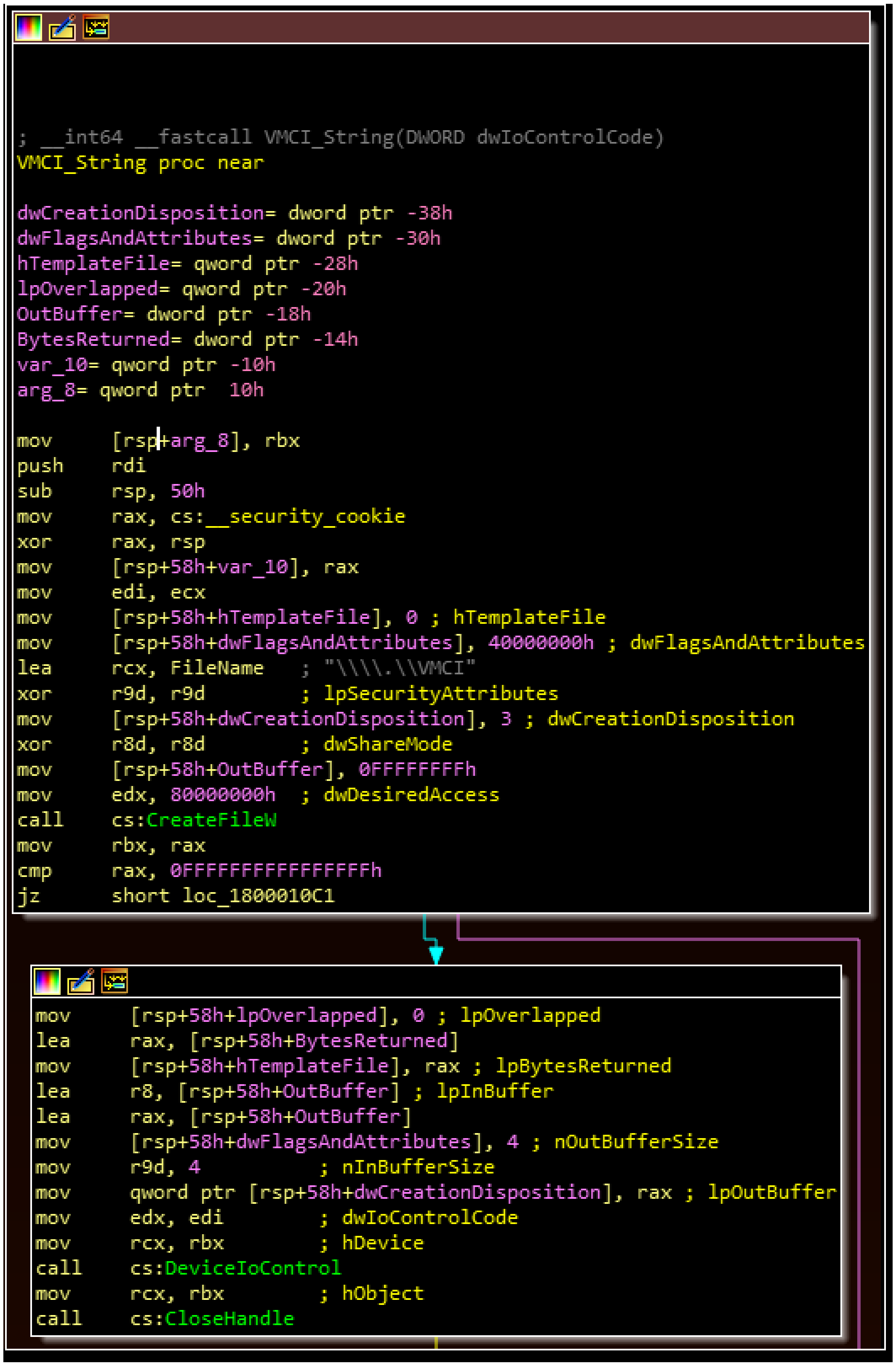
Name	Local address	Local...	Remote address	Rem...	Prot...	State	Owner
svchost.ex...	newuser-PC	123			UDP		W32Time
svchost.ex...	newuser-PC	123			UDP6		W32Time
svchost.ex...	newuser-PC	135			TCP	Listen	RpcSs
svchost.ex...	newuser-PC	135			TCP6	Listen	RpcSs
System (4)	newuser-PC.locald...	137			UDP		
System (4)	newuser-PC.locald...	138			UDP		
System (4)	newuser-PC.locald...	139			TCP	Listen	
System (4)	newuser-PC	445			TCP	Listen	
System (4)	newuser-PC	445			TCP6	Listen	
svchost.ex...	newuser-PC	500			UDP		IKEEXT
svchost.ex...	newuser-PC	500			UDP6		IKEEXT
svchost.ex...	newuser-PC	1900			UDP		SSDPSRV
svchost.ex...	newuser-PC.locald...	1900			UDP		SSDPSRV
svchost.ex...	newuser-PC	1900			UDP6		SSDPSRV
svchost.ex...	newuser-PC.locald...	1900			UDP6		SSDPSRV
svchost.ex...	newuser-PC	3702			UDP		FDResPub
svchost.ex...	newuser-PC	3702			UDP6		FDResPub
svchost.ex...	newuser-PC	4500			UDP		IKEEXT
svchost.ex...	newuser-PC	4500			UDP6		IKEEXT
svchost.ex...	newuser-PC	5355			UDP		Dnscache
svchost.ex...	newuser-PC	5355			UDP6		Dnscache
System (4)	newuser-PC	5357			TCP	Listen	
System (4)	newuser-PC	5357			TCP6	Listen	
wininit.exe ...	newuser-PC	49152			TCP	Listen	
wininit.exe ...	newuser-PC	49152			TCP6	Listen	
svchost.ex...	newuser-PC	49153			TCP	Listen	eventlog
svchost.ex...	newuser-PC	49153			TCP6	Listen	eventlog
svchost.ex...	newuser-PC	49154			TCP	Listen	Schedule
svchost.ex...	newuser-PC	49154			TCP6	Listen	Schedule
services.ex...	newuser-PC	49155			TCP	Listen	
services.ex...	newuser-PC	49155			TCP6	Listen	
svchost.ex...	newuser-PC	49156			TCP	Listen	PolicyAgent
svchost.ex...	newuser-PC	49156			TCP6	Listen	PolicyAgent
lsass.exe (5...	newuser-PC	49157			TCP	Listen	
lsass.exe (5...	newuser-PC	49157			TCP6	Listen	
svchost.ex...	newuser-PC	51805			UDP		FDResPub
svchost.ex...	newuser-PC	51806			UDP6		FDResPub
svchost.ex...	newuser-PC.locald...	59830			UDP6		SSDPSRV
svchost.ex...	newuser-PC	59831			UDP6		SSDPSRV
svchost.ex...	newuser-PC.locald...	59832			UDP		SSDPSRV
svchost.ex...	newuser-PC	59833			UDP		SSDPSRV

CPU Usage: 0.54% Physical memory: 1.26 GB (62.88%) Processes: 45

Commands such as "netstat" also yield no results. Revisiting Mandiant's description offers a hint as to why this might be through its reference to VMCI sockets:

The memory only dropper... uses VMware's virtual machine communication interface (VMCI) sockets to run commands on a guest virtual machine from a hypervisor host

The following additional branch off of the "socket function" above also offers some clues:



```
; __int64 __fastcall VMCI_String(DWORD dwIoControlCode)
VMCI_String proc near

dwCreationDisposition= dword ptr -38h
dwFlagsAndAttributes= dword ptr -30h
hTemplateFile= qword ptr -28h
lpOverlapped= qword ptr -20h
OutBuffer= dword ptr -18h
BytesReturned= dword ptr -14h
var_10= qword ptr -10h
arg_8= qword ptr  10h

mov     [rsp+arg_8], rbx
push    rdi
sub     rsp, 50h
mov     rax, cs:__security_cookie
xor     rax, rsp
mov     [rsp+58h+var_10], rax
mov     edi, ecx
mov     [rsp+58h+hTemplateFile], 0 ; hTemplateFile
mov     [rsp+58h+dwFlagsAndAttributes], 40000000h ; dwFlagsAndAttributes
lea     rcx, FileName   ; "\\\\.\\VMCI"
xor     r9d, r9d        ; lpSecurityAttributes
mov     [rsp+58h+dwCreationDisposition], 3 ; dwCreationDisposition
xor     r8d, r8d        ; dwShareMode
mov     [rsp+58h+OutBuffer], 0FFFFFFFFh
mov     edx, 80000000h  ; dwDesiredAccess
call    cs:CreateFileW
mov     rbx, rax
cmp     rax, 0FFFFFFFFFFFFFFFFh
jz      short loc_1800010C1
```

```
mov     [rsp+58h+lpOverlapped], 0 ; lpOverlapped
lea     rax, [rsp+58h+BytesReturned]
mov     [rsp+58h+hTemplateFile], rax ; lpBytesReturned
lea     r8, [rsp+58h+OutBuffer] ; lpInBuffer
lea     rax, [rsp+58h+OutBuffer]
mov     [rsp+58h+dwFlagsAndAttributes], 4 ; nOutBufferSize
mov     r9d, 4          ; nInBufferSize
mov     qword ptr [rsp+58h+dwCreationDisposition], rax ; lpOutBuffer
mov     edx, edi        ; dwIoControlCode
mov     rcx, rbx        ; hDevice
call    cs:DeviceIoControl
mov     rcx, rbx        ; hObject
call    cs:CloseHandle
```

These API calls and Mandiant's description lead to two useful resources:

– VMWare's documentation regarding "VMCI sockets"
– A snippet on Github that can help an analyst understand how these work at a programming-level

– A second page that includes a CreateFile call for “vmci” which provides another code example

Looking first at VMWare’s documentation, we learn that:

```
VMCI sockets are similar to other socket types... VMCI sockets work on an individual
physical machine, and like UNIX sockets, they can perform interprocess communication
on the local system. [They] allow different virtual machines to communicate with each
other, provided they reside on the same VMware host.
```

This documentation also provides an overview of the functions involved, including:

– Bind() – “Associates the stream socket with the network settings in the sockaddr_vm structure
– Listen () – “Prepares to accept incoming client connections”
– Accept() – “Waits indefinitely for an incoming connection to arrive”
– Recv() – “Reads data from the client application”
– Send() – “Writes data to the client application”

This helps explain why the we saw familiar function names, but did not observe the behavior we expected. At this point, we have enough information to conclude that we are likely dealing with a VMCI socket, and not a “normal” socket.

Testing VMCI Connections

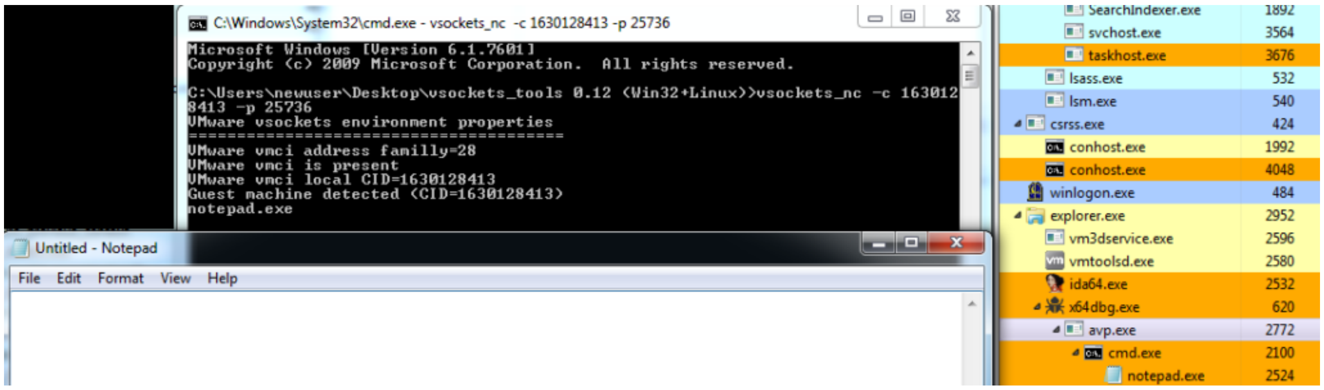
The final piece to this puzzle is finding a way to view a VMCI connection (also referred to as vsockets). Fortunately, there is outstanding research publicly available from Pedro Mendes da Silva. This whitepaper explains why the virtual sockets may not be visible using traditional tools:

“vsockets” can be used much the same way as TCP/IP sockets but using a different address family. “vsockets” can also be called “vmci sockets” because they use a “virtual vmci device” in the lower level to communicate with the host.

Fortunately, the author provides a set of tools available here to help work with these vsockets. Using one of these tools, we can:

– Test a connection over port 25736 to see if it is open
– Test sending a cmd.exe command (e.g. “notepad.exe”) over this port to validate the execution workflow
– Repeat the test over a port expected to be closed, to validate that this only works for this port

This test is relatively straightforward:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\newuser\Desktop\vsockets_tools 0.12 (Win32+Linux)>vsockets_nc -c 163012
8413 -p 25736
VMware vsockets environment properties
======================================
VMware vmci address familly=28
VMware vmci is present
VMware vmci local CID=1630128413
Guest machine detected (CID=1630128413)
notepad.exe
```

SearchIndexer.exe	1892
svchost.exe	3564
taskhost.exe	3676
lsass.exe	532
lsm.exe	540
csrss.exe	424
conhost.exe	1992
conhost.exe	4048
winlogon.exe	484
explorer.exe	2952
vm3dservice.exe	2596
vmtoolsd.exe	2580
ida64.exe	2532
x64dbg.exe	620
avp.exe	2772
cmd.exe	2100
notepad.exe	2524

C:\Windows\System32\cmd.exe - vsockets_nc -c 1630128413 -p 25736
Untitled - Notepad
File Edit Format View Help

avp.exe spawning cmd.exe and notepad.exe following the vsocket connection

The success of the test is evident both in the creation of notepad.exe and the process tree evidence showing avp.exe spawning the command shell. At this stage, the malware's functionality is validated.

Concluding Thoughts

The novelty and complexity of the malware examined in this post does not reside in its functionality; at face value, the malware receives data over a socket and executes a command. Instead, the complexity lies in the (somewhat) esoteric nature of the ecosystem it is installed in. As Mandiant notes in its blog post, ESXI environments are less likely to have EDR products present to detect the steps leading to the installation of this malware in the first place.