Practical 2 Report

Joshua Main-Smith
joshuamainsmith@ufl.edu
Practical 2 - Malware Reverse Engineering

2021-03-18

Contents

Executive Summary

The binary exhibits behavior that is consistent with Trojan malware, with the majority of antivirus engines classifying it as such (VT). The binary has been identified as a variant of the AVE_MARIA family, which can be read further with Yoroi's analysis. The malware takes advantage of CVE-2017-11882, which is a Microsoft memory corruption vulnerability that allows an attacker to execute arbitrary code and has a high ranking in terms of severity. The goal of the attacker appears to be stolen system information, stolen credentials, and backdoor installation (for further susceptibility/infection). Further information can be read on Malwarebytes blog.

The sample malware analyzed here had various strings and imports that were deemed suspicious by PEStudio. Particularly interesting was the PE Header section, which contained a blacklisted section (.00cfg) and .tls, which is a section typical in anti-debugging techniques. There were several anti-debugging and anti-vm techniques found while debugging, which essentially consisted of recording timestamps of the victim machine then throwing an exception when execution appeared to be too slow.

The malware attempted to contact an IP address on 195.140.214.82:6703, which can be recognized due to the unusual port number associated with the IP address. Also, it deleted a file that holds meta information on file downloading activity, suggesting the malware was attempting to obscure its behavior by hiding the file it was attempting to download.

Finally, the malware opened several registry keys and manipulated the values of some that would allow the application to make several simultaneous connections to a host.

Static Analysis

Hashes and Antivirus Check

The following are the hashes generated from the VirusTotal report.

MD5	971a3320179e0494fdb70b138ada2446
SHA-1	b04bba3b8be297b6178e73d10f0380897e76464c
SHA-256	9633d0564a2b8f1b4c6e718ae7ab48be921d435236a403cf5e7ddfbfd4283382
Vhash	016086551d551d1515156045z200677z90baz1bfz
Authentihash	70e203d43f38c128fed15c70ec8479eb5989ab63f535bf190d9c1d54847e9b45
Imphash	f396b39dbfa473ab2b7180d955fdc740
Rich PE header hash	94adf1b937a03026d0489a22843d03cc
SSDEEP	12288:hkhSL4pH7FYiliicuueTh9yeJWrpDz29Wa+QB1t6gMvlTpa6NYjHhtkaJN:h72Z/8VWrpn2ZF1Ea1jBH
TLSH	T1E3654A07A7628111FCFE12F774FFB2B8552CABA2279892C352C5A5ECA7055F06D30E52
File type	Win32 EXE
Magic	PE32 executable for MS Windows (console) Intel 80386 32-bit
TrID	Win32 Executable MS Visual C++ (generic) (47.3%)
TrID	Win64 Executable (generic) (15.9%)
TrID	Win16 NE executable (generic) (10.6%)
TrID	Win32 Dynamic Link Library (generic) (9.9%)
TrID	Win32 Executable (generic) (6.8%)
File size	1.38 MB (1446912 bytes)
PEiD packer	Microsoft Visual C++ 8.0 [Debug]

There were a total of 52 engines that detected this binary as being malicious, the majority classifying it as a trojan.

Packing

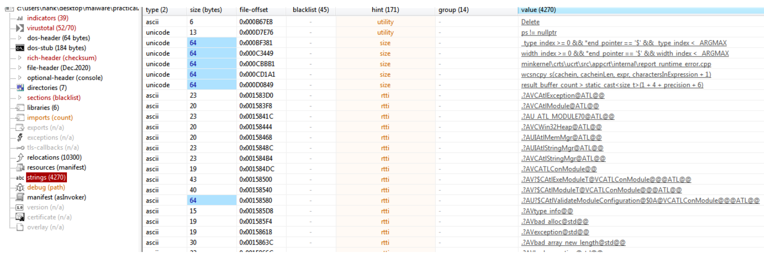
The binary appears to not be packed due to a few indicators discussed below. To start, the entropy of all the sections are around six and below.

property	value	value	value	value	value	value	value	value
name	.text	.rdata	.data	.idata	.tls	.00cfg	.rsrc	.reloc
md5	F4652CB5BEC6D3B287599D...	5BEE6DE30D069254DD3473D...	A164260C7E639AC06B537C2...	B93F40DB6CFB5866381BC50...	C573BD7CEA296A9C5D230C...	B113A4C4A7E762786CC1415...	EEA80AA728A3053C9C54C2...	4FDD24485D60B33CE44EB3...
entropy	5.417	3.656	4.611	4.583	0.011	0.061	2.141	6.094
file-ratio (99.93%)	51.27 %	11.75 %	34.47 %	0.39 %	0.07 %	0.04 %	0.11 %	1.84 %
raw-address	0x00000400	0x000B5600	0x000DEE00	0x00158A00	0x0015A000	0x0015A400	0x0015A600	0x0015AC00
raw-size (1445888 bytes)	0x000B5200 (741888 bytes)	0x00029800 (169984 bytes)	0x00079C00 (498688 bytes)	0x00001600 (5632 bytes)	0x00000400 (1024 bytes)	0x00000200 (512 bytes)	0x00000600 (1536 bytes)	0x00006800 (26624 bytes)
virtual-address	0x00401000	0x004B7000	0x004E1000	0x0055F000	0x00561000	0x00562000	0x00563000	0x00564000
virtual-size (1460414 bytes)	0x000B5071 (741489 bytes)	0x000297DC (169948 bytes)	0x0007DB5C (514908 bytes)	0x0000150E (5390 bytes)	0x00000309 (777 bytes)	0x00000104 (260 bytes)	0x0000043C (1084 bytes)	0x000067BE (26558 bytes)
entry-point	0x000045D4	-	-	-	-	-	-	-
characteristics	0x60000020	0x40000040	0xC0000040	0x40000040	0xC0000040	0x40000040	0x40000040	0x42000040

This is consistent with binaries that are not packed as binaries that are packed will generally have sections with an entropy above seven. Further, in the section header one can see that there is minimal deviation between the raw size and virtual size for each section is minimal.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	000B5071	00001000	000B5200	00000400	00000000	00000000	0000	0000	60000020
.rdata	000297DC	000B7000	00029800	000B5600	00000000	00000000	0000	0000	40000040
.data	0007DB5C	000E1000	00079C00	000DEE00	00000000	00000000	0000	0000	C0000040
.idata	0000150E	0015F000	00001600	00158A00	00000000	00000000	0000	0000	40000040
.tls	00000309	00161000	00000400	0015A000	00000000	00000000	0000	0000	C0000040
.00cfg	00000104	00162000	00000200	0015A400	00000000	00000000	0000	0000	40000040
.rsrc	0000043C	00163000	00000600	0015A600	00000000	00000000	0000	0000	40000040
.reloc	000067BE	00164000	00006800	0015AC00	00000000	00000000	0000	0000	42000040

Packed binaries tend to have much larger virtual sizes when compared to their raw size sectional counterpart. Finally, the string count numbers over four thousand and the imports number over 150.



Packed malicious binaries tend to have much fewer string and import counts as to obscure behavior.

Compilation Date and Subsystem

The apparent compilation date and subsystem of the program is December 09, 2020 and is a console application.

file-type	**executable**
cpu	**32-bit**
subsystem	console
compiler-stamp	0x5FD1540F (Wed Dec 09 16:47:43 2020)
debugger-stamp	0x5FD1540F (Wed Dec 09 16:47:43 2020)

Program Imports

There are a number of import modules, as listed below.

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	103	0015F4E4	00000000	00000000	0015FA82	0015F070
USER32.dll	27	0015F75C	00000000	00000000	0015FC4A	0015F2E8
GDI32.dll	2	0015F4B0	00000000	00000000	0015FC78	0015F03C
ADVAPI32.dll	4	0015F474	00000000	00000000	0015FCC4	0015F000
ole32.dll	9	0015F808	00000000	00000000	0015FD92	0015F394
OLEAUT32.dll	11	0015F6FC	00000000	00000000	0015FD9C	0015F288

Kernel32.dll is a common DLL that is primarily involved in core functionality in relation to memory, file, and hardware manipulation. *User32.dll* contains user-interface items, like buttons, scrolling, response to user interactions, etc. *Gdi32.dll* is primarily used to display and manipulate graphics. *Advapi32.dll* is used primarily to provide access to core Windows components like the Service Manager and Registry. *Ole32.dll* is used to initialize and uninitialize the COM library and the *Oleaut32.dll* is a subset of COM that is used for OLE technologies and IPC (Wiki).

name (156)	group (11)	type (1)	ordinal (11)	blacklist (23)	anti-debug (0)	undocumented (0)	deprecated (3)	library (6)
RegDeleteKeyA	registry	implicit	-	x	-	-	-	advapi32.dll
InterlockedPushEntrySList	synchronization	implicit	-	x	-	-	-	kernel32.dll
VirtualProtect	memory	implicit	-	x	-	-	-	kernel32.dll
HeapQueryInformation	memory	implicit	-	x	-	-	-	kernel32.dll
FindNextFileW	file	implicit	-	x	-	-	-	kernel32.dll
FindFirstFileExW	file	implicit	-	x	-	-	-	kernel32.dll
GetCurrentThreadId	execution	implicit	-	x	-	-	-	kernel32.dll
SetEnvironmentVariableW	execution	implicit	-	x	-	-	-	kernel32.dll
GetEnvironmentStringsW	execution	implicit	-	x	-	-	-	kernel32.dll
GetCurrentThread	execution	implicit	-	x	-	-	-	kernel32.dll
TerminateProcess	execution	implicit	-	x	-	-	-	kernel32.dll
GetCurrentProcessId	execution	implicit	-	x	-	-	-	kernel32.dll
RaiseException	exception-handling	implicit	-	x	-	-	-	kernel32.dll
GetModuleFileNameA	dynamic-library	implicit	-	x	-	-	-	kernel32.dll
GetModuleHandleExW	dynamic-library	implicit	-	x	-	-	-	kernel32.dll
GetModuleFileNameW	dynamic-library	implicit	-	x	-	-	-	kernel32.dll
OutputDebugStringW	diagnostic	implicit	-	x	-	-	-	kernel32.dll
ReadConsoleW	console	implicit	-	x	-	-	-	kernel32.dll
SetConsoleCtrlHandler	console	implicit	-	x	-	-	-	kernel32.dll
186 (UnRegisterTypeLib)	registry	implicit	x	x	-	-	-	oleaut32.dll
163 (RegisterTypeLib)	registry	implicit	x	x	-	-	-	oleaut32.dll
161 (LoadTypeLib)	dynamic-library	implicit	x	x	-	-	-	oleaut32.dll
PostThreadMessageA	execution	implicit	-	x	-	-	-	user32.dll

There were a few import functions that were blacklisted on PEStudio, as shown above. Several of these originated from the Kernel32.dll and appears to mostly be associated with memory manipulation, file handling, and execution. Advapi32.dll and Oleaut32.dll have API calls with the registry grouping.

Suspicious Strings

There were a number of blacklisted strings found by PEStudio, several of those being the ASCII value of the imported functions discussed earlier.

blacklist (45)	hint (171)	group (15)	value (4270)
x	file	-	c:\users\w7h64\desktop\vcsamples-master\vc2010samples\atl\general\atlcon\propbrowsectl.h
x	file	-	c:\users\w7h64\desktop\vcsamples-master\vc2010samples\atl\general\atlcon\atlcontrols.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\cstringt.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atlwin.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atlwin.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atltransactionmanager.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atltrace.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atlstr.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atlsimpstr.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atlsimpcoll.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atlmem.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atlexcept.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atlconv.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atlcomcli.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atlcomcli.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atlbase.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atlbase.h
x	file	-	c:\program files (x86)\microsoft visual studio\2017\community\vc\tools\msvc\14.16.27023\atlmfc\include\atlalloc.h
x	file	-	C:\Users\W7H64\Desktop\VCSamples-master\VC2010Samples\ATL\General\AtlCon\bitcoin coinjoin op.pdb
x	-	memory	VirtualProtect
x	-	execution	TerminateProcess
x	-	cryptography	SystemFunction036
x	-	execution	SetThreadStackGuarantee
x	-	execution	SetEnvironmentVariable
x	-	console	SetConsoleCtrlHandler
x	-	registry	RegisterTypeLibForUser
x	-	registry	RegDeleteKey
x	-	console	ReadConsole
x	-	exception-handling	RaiseException
x	-	execution	PostThreadMessage
x	-	diagnostic	OutputDebugString
x	-	synchronization	InterlockedPushEntrySList
x	-	memory	HeapQueryInformation
x	-	desktop	GetUserObjectInformation
x	-	desktop	GetProcessWindowStation
x	-	dynamic-library	GetModuleHandleEx
x	-	dynamic-library	GetModuleFileName
x	-	dynamic-library	GetModuleFileName
x	-	execution	GetEnvironmentStrings
x	-	execution	GetCurrentThreadId
x	-	execution	GetCurrentThread
x	-	execution	GetCurrentProcessId
x	-	file	FindNextFile
x	-	file	FindFirstFileEx
x	-	-	.00cfg

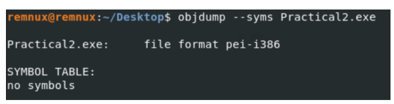
The rest appear to be directory paths to files that are either created or used. All but three of the blacklisted path strings navigate to a header file in Microsoft Visual Studio 2017. The last three navigate to a location under a user named *W7H64*, with the last being a database file (commonly used with MSVS).

PE Header Sections

A screenshot of the various sections for the binary has been included under the *Packing* section. The *text* section primarily contains executable code, *rdata* section has globally accessible read-only data, *data* section has globally accessible data, *.idata* contains import function information, *rsrc* section contains resources necessary for execution, *reloc* section has relocation information of library files, *.tls* contains TLS, a Windows storage class and is uncommon with most programs (it has code that is executed before the entry point, common in anti-debugging techniques), and *.00cfg*, which is commonly employed with MSVS and most likely contains call guard check information.

Anti-Disassembly Techniques

The binary attempts to obfuscate its contents using a few common techniques. One technique (which is common in the industry as well) is to remove the debugging information from the binary. This can be seen when attempting to view the debug symbols, as shown below.

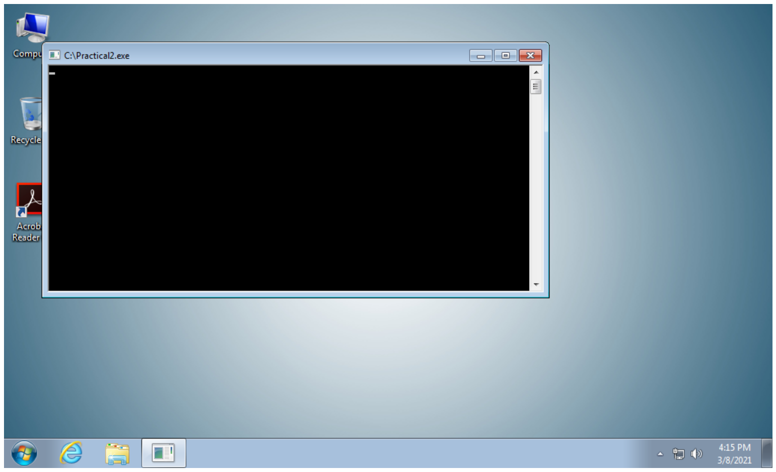


The OEP as shown in Ghidra is also different than when executed in a debugger, suggesting that the binary changes the OEP of the program upon execution. This can make it more difficult to reverse alongside debugging when comparing addresses.

Dynamic Analysis

Post-Execution Behavioral Analysis

Upon execution, the first thing one would initially notice is the command line program starting then terminating, as can be seen below.



Having process monitor open would show that Practical2 running as a process in the background. Hybrid analysis noted that there was a shell code injection into the running process, as indicated below.

Installs hooks/patches the running process

details	"Practical2.exe" wrote bytes "fae6b077e1a6b5772e71b577ee29b57785e2b0776da0b57726e4b077d16db577003db377804bb37700000000ad37c5778b2dc577b641c57700000000" to virtual address "0x74E81000" (part of module "WSHTCPIP.DLL") "Practical2.exe" wrote bytes "c04eb3772054b477e065b477b538b57700000000000d08a7700000000c5ea8a770000000088ea8a7700000000e968ac758228b577ee29b57700000000d269ac75000000007dbb8a770000000009beac7500000000ba188a7700000000" to virtual address "0x769B1000" (part of module "NSI.DLL")
source	Hook Detection
relevance	10/10
ATT&CK ID	T1179 (Show technique in the MITRE ATT&CK™ matrix)

Further, there were several attempts to contact an IP address using an unusual port number (discussed below).

Network Communication

When ran in VirusTotal, the binary attempted to contact the IP address 195.140.214.82 on port 6703.

Network Communication

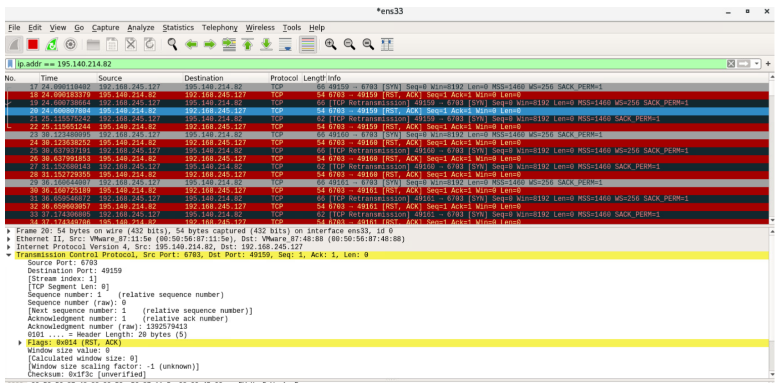
IP Traffic

195.140.214.82:6703 (TCP)

Memory Pattern Urls

tcp://195.140.214.82:6703

Wireshark was able to confirm this by attempting to contact the host at the above mentioned IP. The ports used on the host machine were iterated incrementally by one, starting at port 49159.



Something interesting to note is that the port used to contact 195.140.214.82 is atypical with most known services and may be an indicator of malicious behavior.

Registry Keys Created or Modified

Some registry keys that were opened or modified are shown below.

Registry Actions

Registry Keys Opened

HKCU\Software\Microsoft\ActiveMovie\devenum

HKCU\Software\Microsoft\ActiveMovie\devenum\Version

HKCU\Software\Microsoft\ActiveMovie\devenum\{860BB310-5D01-11D0-BD3B-00A0C911CE86}

HKCR\CLSID

HKCR\CLSID\{860BB310-5D01-11D0-BD3B-00A0C911CE86}\Instance

HKCR\DirectShow\MediaObjects

HKCR\DirectShow\MediaObjects\Categories\860bb310-5d01-11d0-bd3b-00a0c911ce86

HKLM\Software\Microsoft\Windows NT\CurrentVersion\VFW

HKLM\SYSTEM\CurrentControlSet\Control\MediaResources\msvideo

HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings

HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\OQWGJ0HVDJ

Registry Keys Set

– Software\Microsoft\Windows\CurrentVersion\Internet Settings\MaxConnectionsPer1_0Server

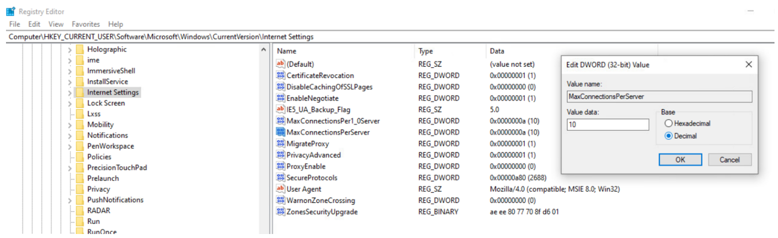
10

– Software\Microsoft\Windows\CurrentVersion\Internet Settings\MaxConnectionsPerServer

10

After running the binary in a virtual machine, the registry keys *MaxConnectionsPerServer* and *MaxConnectionsPer1_0Server* were set to the decimal value

10.



This allows an application (or multiple applications) to make up to ten simultaneous network connections for a given process.

Files Created or Modified

There were several files touched by the binary, most of which were DLL files.

Files Opened

C:\Windows\system32\bcrypt.dll
C:\Windows\system32\version.DLL
C:\Windows\system32\NETAPI32.dll
C:\Windows\system32\netutils.dll
C:\Windows\system32\srvcli.dll
C:\Windows\system32\wkscli.dll
C:\Windows\system32\SAMCLI.DLL
C:\Windows\SysWOW64\devenum.dll
C:\Windows\system32\ntmarta.dll
C:\Windows\system32\msdmo.dll
C:\Windows\system32\avicap32.dll
C:\Windows\system32\MSVFW32.dll
C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_5.82.7601.18837_none_ec86b8d6858ec0bc
C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_5.82.7601.18837_none_ec86b8d6858ec0bc\COMCTL32.dll
C:\Windows\system32\en-US\MSVFW32.dll.mui
C:\Windows\system32\en-US\avicap32.dll.mui
C:\Program Files\Microsoft DN1
C:\Windows\syswow64\SHELL32.dll
C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.18837_none_41e855142bd5705d
C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.18837_none_41e855142bd5705d\comctl32.dll
C:\Windows\WindowsShell.Manifest
C:\Users\<USER>\Downloads\Practical2.exe
C:\Users\<USER>\AppData\Local
C:\Users\<USER>\AppData\Local\Microsoft Vision\
:Zone.Identifier
C:\Windows\system32\mswsock.dll
C:\Windows\System32\wshtcpip.dll
\Device\Afd\Endpoint

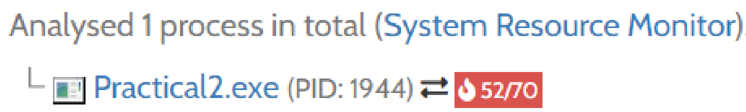
Files Deleted

:Zone.Identifier

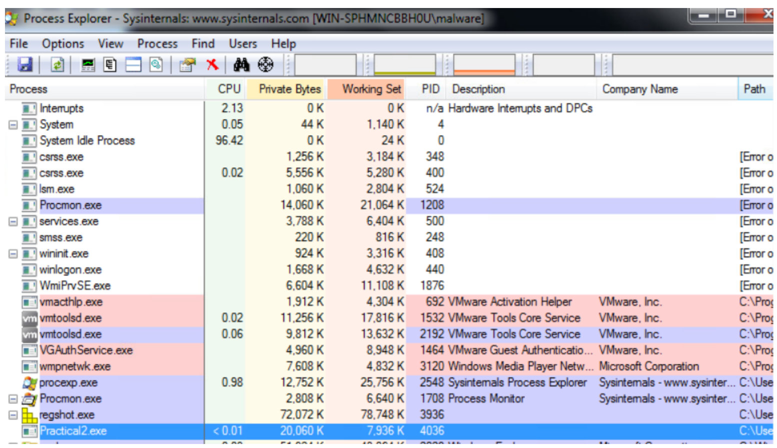
One file the Zone.identifier, was deleted from disk. This is a file that contains meta-information about the file, such as network file downloading activity. The binary deleting this file is an attempt to obscure its network behavior by hiding evidence of file downloads.

Processes Started

There was only one process started by the binary, which was its own process.



This was also observed during dynamic analysis.

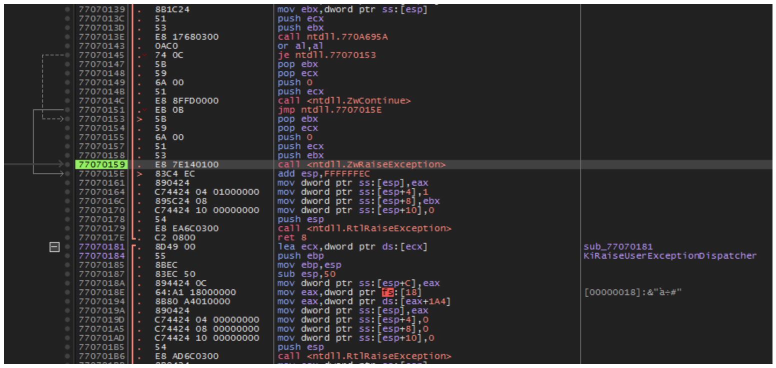


Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Path
Interrupts	2.13	0 K	0 K	n/a	Hardware Interrupts and DPCs		
System	0.05	44 K	1,140 K	4			
System Idle Process	96.42	0 K	24 K	0			
csrss.exe		1,256 K	3,184 K	348			[Error o
csrss.exe	0.02	5,556 K	5,280 K	400			[Error o
lsm.exe		1,060 K	2,804 K	524			[Error o
Procmon.exe		14,060 K	21,064 K	1208			[Error o
services.exe		3,788 K	6,404 K	500			[Error o
smss.exe		220 K	816 K	248			[Error o
wininit.exe		924 K	3,316 K	408			[Error o
winlogon.exe		1,668 K	4,632 K	440			[Error o
WmiPrvSE.exe		6,604 K	11,108 K	1876			[Error o
vmacthlp.exe		1,912 K	4,304 K	692	VMware Activation Helper	VMware, Inc.	C:\Prog
vmtoolsd.exe	0.02	11,256 K	17,816 K	1532	VMware Tools Core Service	VMware, Inc.	C:\Prog
vmtoolsd.exe	0.06	9,812 K	13,632 K	2192	VMware Tools Core Service	VMware, Inc.	C:\Prog
VGAuthService.exe		4,960 K	8,948 K	1464	VMware Guest Authenticatio...	VMware, Inc.	C:\Prog
wmpnetwk.exe		7,608 K	4,832 K	3120	Windows Media Player Netw...	Microsoft Corporation	C:\Prog
procexp.exe	0.98	12,752 K	25,756 K	2548	Sysinternals Process Explorer	Sysinternals - www.sysinter...	C:\Use
Procmon.exe		2,808 K	6,640 K	1708	Process Monitor	Sysinternals - www.sysinter...	C:\Use
regshot.exe		72,072 K	78,748 K	3936			C:\Use
Practical2.exe	< 0.01	20,060 K	7,936 K	4036			C:\Use

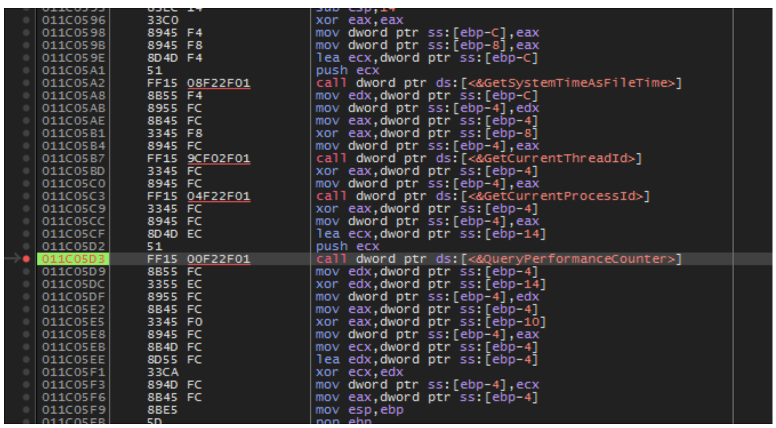
Anti-Debugging and Anti-Virtual-Machine Techniques

As mentioned in the static analysis section, there was a .tls section found in the binary. This is usually used in malware to execute code prior to a debugger pausing execution. The presence of .tls lends suspicious to the binary using anti-debugging techniques as this section is rarely present in legitimate applications. This is given further credibility with the presence of debussing strings like *IsDebuggerPresent* and *OutputDebugStringW*.

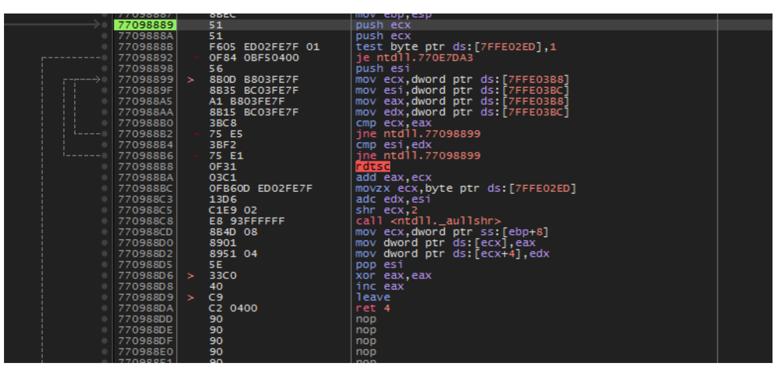
There was a point during debugging where the program got stuck in a loop where the program continually raised exceptions. This is one tactic used in anti-debugging techniques by misusing exception handling by recording timestamps to see how long it takes for the program to execute.



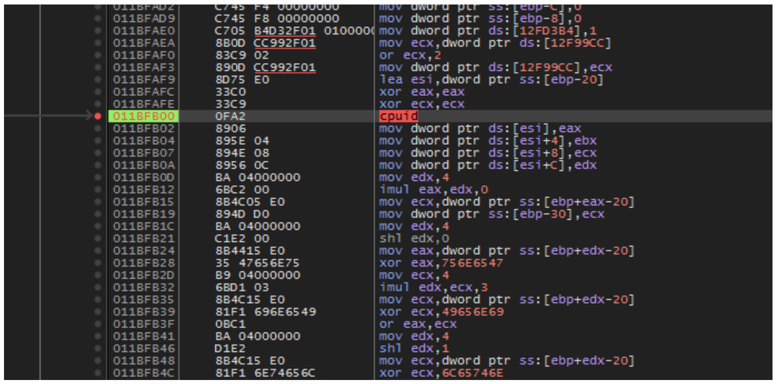
If it takes longer than it should, then there's an assumption that the program is being debugged and raises an exception. This is done by calling *QueryPerformanceCounter*, which is a high resolution API call that measures the time interval between calls in microseconds. An example of this call is shown below.



A further technique this binary uses in measuring timestamps is its use in RDTSC. The machines timestamp is read multiple times to look for any irregularities.



An anti-vm technique that the binary uses is with CPUID which, when executed on real hardware, will return 0. When executed on a VM, CPUID will return 1.



Indicators of Compromise

An obvious indicator would be a binary attempting to contact the above mentioned IP address, 195.140.214.82 on port 6703. This is especially the case considering the port number is unusual.

Two Yara rules I developed were the blacklisted strings from PEStudio. In order for the rule to be satisfied, all of the strings need to be present. Requiring the AND logic is preferred as to limit false positive rates.

Yara Rule

```
rule P2_strings
{
meta:
        description = "Blacklisted strings found in P2"
strings:
        $a = "InterlockedPushEntrySList"
        $b = "RegisterTypeLibForUser"
        $c = "RegDeleteKey"
        $d = "VirtualProtect"
        $e = "HeapQueryInformation"
        $f = "FindFirstFileEx"
        $g = "FindNextFile"
        $h = "SetThreadStackGuarantee"
        $i = "GetCurrentThreadId"
        $j = "PostThreadMessage"
        $k = "GetCurrentProcessId"
        $l = "TerminateProcess"
        $m = "GetCurrentThread"
        $n = "GetEnvironmentStrings"
        $o = "SetEnvironmentVariable"
        $p = "RaiseException"
```

```
        $q = "GetModuleFileName"
        $r = "GetModuleFileName"
        $s = "GetModuleHandleEx"
        $t = "OutputDebugString"
        $u = "GetProcessWindowStation"
        $v = "GetUserObjectInformation"
        $w = "SystemFunction036"
        $x = "SetConsoleCtrlHandler"
        $y = "ReadConsole"
        $z = "IsDebuggerPresent"
condition:
        all of them
}
```

The other Yara rule presented are two sections from the PE file, .tls and .00cfg. The latter was blacklisted in PEStudio while the former is typically only present in malicious software. Either can be present for the rule to take hold.

Yara Rule

```
rule P2_sections
{
meta:
        description = "Blacklisted and suspicious sections"
strings:
        $a = ".00cfg"
        $b = ".tls"
condition:
        $a or $b
}
```

The last Yara rule includes a screenshot of Florian Roth's rule that was detected in Joe's Sandbox. The rule can be found here and includes both Codoso_Gh0st_1 and Codoso_Gh0st_2.

```
rule Codoso_Gh0st_1
{

    meta:
        description = "Detects Codoso APT Gh0st Malware"
        author = "Florian Roth"
        reference = "https://www.proofpoint.com/us/exploring-bergard-old-malware-new-tricks"
        date = "2016-01-30"
        super_rule = 1
        hash1 = "5402c785037614d09ad41e41e11093635455b53afd55aa054a09a84274725841"
        hash2 = "7dc7cec2c3f7e56499175691f64060ebd955813002d4db780e68a8f6e7d0a8f8"
        hash3 = "d7004910a87c90ade7e5ff6169f2b866ece667d2feebed6f0ec856fb838d2297"

    strings:
        $x1 = "cmd.exe /c ping 127.0.0.1 && ping 127.0.0.1 && sc start %s && ping 127.0.0.1 && sc start %s" fullword ascii
        $x2 = "rundll32.exe \"%s\", RunMeByDLL32" fullword ascii
        $x3 = "Elevation:Administrator!new:{3ad05575-8857-4850-9277-11b85bdb8e09}" fullword wide
        $x4 = "\\\\.\\keymmdrv1" fullword ascii
        $s1 = "spideragent.exe" fullword ascii
        $s2 = "AVGIDSAgent.exe" fullword ascii
        $s3 = "kavsvc.exe" fullword ascii
        $s4 = "mspaint.exe" fullword ascii
        $s5 = "kav.exe" fullword ascii
        $s6 = "avp.exe" fullword ascii
        $s7 = "NAV.exe" fullword ascii
        $c1 = "Elevation:Administrator!new:" wide
        $c2 = "Global\\RUNDLL32EXITEVENT_NAME{12845-8654-543}" fullword ascii
        $c3 = "\\sysprep\\sysprep.exe" fullword wide
        $c4 = "\\sysprep\\CRYPTBASE.dll" fullword wide
        $c5 = "Global\\TERMINATEEVENT_NAME{12845-8654-542}" fullword ascii
        $c6 = "ConsentPromptBehaviorAdmin" fullword ascii
        $c7 = "\\sysprep" fullword wide
        $c8 = "Global\\UN{5FFC0C8B-8BE5-49d5-B9F2-BCDC8976EE10}" fullword ascii

    condition:
        uint16(0) == 0x5a4d and filesize < 1000KB and ( 4 of ($s*) or 4 of ($c*) ) or 1 of ($x*) or 6 of ($c*)
}
```

Roth's rule uses various parameters when deciding if it passes, matching on four strings between S1-S7, or four between C1-C8, etc.

Lastly, using the bytes that were mentioned earlier from the shell code injected into the parent running process would be a good indicator of compromise to use. To create a Yara rule for this, one would need to detect the shell code in memory during execution.

Sources

Hybrid Analysis
VirusTotal
Intezer Analyze
ZeroBox
Joe Sandbox
Yara Repository