"LegionLoader" exposed ! - TEHTRIS

By Pierre-Henri PEZIER

Published: 2025-02-03 · Archived: 2026-04-05 19:46:44 UTC

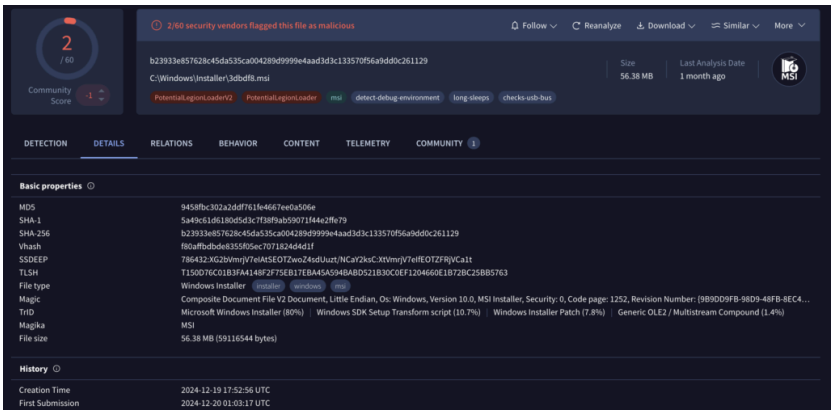
The hidden threat no one is talking about… Yet!

Lefebvre Fabien (CTI), Pezier Pierre-Henri (CTI)

LegionLoader, also known as *Satacom*, *CurlyGate*, and *RobotDropper*, is an active downloader that has been operating in the shadows, gained significant traction in recent months, quietly amassing over 2,000 samples in just a matter of weeks. VirusTotal (VT) retro-hunting and live-hunting have allowed us to uncovered an ongoing campaign using *LegionLoader* that appears to have **kicked off on December 19, 2024.**

This is TEHTRIS Threat Intelligence team analysis. Wee'll break down everything we've uncovered so far (including: list of IoCs, phishing url, IDAPython script etc.).

#Stayinformed, #staysecured



LegionLoader oldest found sample

VT Submission rate over time

The malware seems to target entities globally, with Brazil emerging as the most affected country, accounting for approximately 10% of all submissions.

VT Submission per originating country

In this article, we will delve into the reverse engineering and analysis of the MSI file and the first two stages of this malware campaign, uncovering its techniques and behaviors.

The following list consists of the samples analyzed in the current article. This list is non-exhaustive.

Samples

Hashes of “setup.msi”:

Type	Value
File Type	Composite Document File V2 Document, Little Endian, Os: Windows, Version 10.0, MSI Installer, Code page: 1252, Name c Rotq App, Create Time/Date: Sun Jan 19 16:03:15 2025, Last Printed: Sun Jan 19 16:03:15 2025, Title: Installation Database, Author: Viqwo Stars Ci, Keywords: Installer, MSI, Database, Comments: This installer database contains the logic and data r App., Template: x64;2057, Revision Number: {8D95B8E0-79E2-422A-8620-2E2986EF3D60}, Last Saved Time/Date: Tue J Number of Pages: 450, Number of Words: 10, Security: 0
DateTimestamp	N/A
Size	59.4 MB
MD5	70a9a5c89b0bb7b8a61515131e3d49f0
SHA256	41c1006feead9af3e9a563e2814acc8550d36b991e0998015cee00ebb0ac4e85
SHA512	abb057c218327b9f82d6fdc9e4a2c4d910356e70c651164b7ba58ef44fa1e470b039a20c1cd046fe15c847cfefa554102e4fe91f05

Type	Value
File Type	Composite Document File V2 Document, Little Endian, Os: Windows, Version 10.0, MSI Installer, Security: 0, Code page: 1 {0E8F0152-F75A-41DE-82DC-13392C51F4A9}, Number of Words: 10, Subject: Joas App, Author: Barsoc Quite Sols, Nam Joas App, Template: x64;2057, Comments: This installer database contains the logic and data required to install Joas App., Ti Keywords: Installer, MSI, Database, Create Time/Date: Sat Jan 18 12:53:02 2025, Last Saved Time/Date: Sat Jan 18 12:53:0 18 12:53:02 2025, Number of Pages: 450

DateTimestamp	N/A
Size	59.4 MB
MD5	cc041f6ca77fbb37f083e557ed051055
SHA256	cd72eaba97bb94947529a1e652e2d1cc7197b6224e00bf39e55ad634b7e82047
SHA512	f10c541123b79745c6c4870433cfdba40cc8784bb55cd12a0c7a90356279789395b06048e18b8bca06718be2bc4e10095c46a72

Type	Value
File Type	Composite Document File V2 Document, Little Endian, Os: Windows, Version 10.0, MSI Installer, Security: 0, Code page: 1 {38EAC3F4-3CAF-4CEE-9FFA-36F53F42006E}, Number of Words: 10, Subject: Rotq App, Author: Viqwo Stars Ci, Name Rotq App, Template: x64;2057, Comments: This installer database contains the logic and data required to install Rotq App., T Keywords: Installer, MSI, Database, Create Time/Date: Mon Jan 20 11:06:39 2025, Last Saved Time/Date: Mon Jan 20 11:06 Mon Jan 20 11:06:39 2025, Number of Pages: 450
DateTimestamp	N/A
Size	59.4 MB
MD5	3f86649d211a7faea0cf75296e3ed3c8
SHA256	e88cb0e892537a1dfd7d7d7a4802caeee43d25f871602466a735df0eb5096eb3
SHA512	e71e8e39f01b1a5b5033c1ff3634ff57ed9a5a25678573aa0d6d40b8ff11dcd89d3e8fcbc9138d331391c402e7bc750507d623732

Hashes of “obs.dll” (Dropper payload):

Type	Value
File Type	PE32+ executable (DLL) (GUI) x86-64, for MS Windows
DateTimestamp	2025-01-18 12:59:37
Size	1.2 MB (pesize = 1.2 MB)
MD5	f7e61f06fc606f68b1f8a6270752b832
SHA256	23f064df01ee9eedf9e1341185505b86148873ccc0a922c64bb085ceb5b091fc
SHA512	fe067a0c121678f9f20248dafd56d7567c531a309dbdba2e58c80242453ede5ae486caab22749dd5b3076fef9b1fe7fdba99946ed

Type	Value
File Type	PE32+ executable (DLL) (GUI) x86-64, for MS Windows
DateTimestamp	2025-01-19 16:52:44
Size	1.1 MB (pesize = 1.1 MB)
MD5	0b5d9b80c9bbee71482202720d1bbc3a
SHA256	4c3772e12e710645341f18015c05f67e8f320dd13a4259eff05dacca4c664244
SHA512	3ed21f93293d4eb1af2d902f1d2dd678e23c9700b5a43312aec09b60f15d188210b4d7d890bac2d02a62f554b62c4d1f19009b90

Type	Value
File Type	PE32+ executable (DLL) (GUI) x86-64, for MS Windows
DateTimestamp	2025-01-20 11:31:08
Size	1.1 MB (pesize = 1.1 MB)
MD5	908431381d588caea53a651679dacee8
SHA256	4df98a4f9ecacf1f1676814ad5980dd94d7d33ce4b7d9aec9d96f3c3ea602363
SHA512	e2eb3874ac7c03aa7fd7a2449d7b948962b48552a825c34248902566923f2cca9531adcd9e3a807457e8f7e7fa3856849cea1caf8

Hashes of “stage2.exe” (The payload dropped by the previous samples of legion dropper):

Type	Value
File Type	PE32 executable (GUI) Intel 80386, for MS Windows
DateTimestamp	2025-01-19 13:28:07
Size	151.6 kB (pesize = 151.6 kB)
MD5	97a42de72ada85aaa4198559779b58b0
SHA256	76cbe366ea370235dfea2d72378f9d946e49370b4c7bac58e99073e117062e1f
SHA512	77a5f20dcfa8b61196fe98c8c024fdf3bc8f39962d73ee774f308c0091c90b687cbf1f162b6ea8e374ab2b9b2645ec905a5e6eb50e

Type	Value
File Type	PE32 executable (GUI) Intel 80386, for MS Windows
DateTimestamp	2025-01-17 15:47:34
Size	151.6 kB (pesize = 151.6 kB)
MD5	4756fa2af7d98078f29911d5ffc90ec7
SHA256	b1cff28f26270779d53e14797430d77d9e44911976c916966e4ab2049aa5232e
SHA512	b686b6daa4f37cef3590f3994ef3def41de7022156358f4edd9915102b7c0d084aaaae0eb2d93cda0a8eababc400bc7314d3f619b

Code details

The MSI files were created using Advanced Installer, the stage 1 is a 64 bit DLL compiled with Visual C/C++ 2022 as identified by DIE and the stage 2 is a 32 bit executable.

Techniques

The following techniques have been identified by our team:

Analysis

Initial access

The malware is delivered by “drive by download” technique. Pcrisk has identified insecure websites such as illegal download platforms that redirect use traffic into unverified web pages.

Credit: pcrisk.com

The malicious pages are usually deleted in few hours. It almost every time inciting the user to redirects himself to a mega share with a single zip file in it. The “.monster” TLD is massively used to host the malicious redirection page.

Malicious zip hosted by mega.io

This archive contains 2 files: A 7zip password protected archive, the sole purpose of this protection is to bypass malware analysis; and a picture file reminding the password of the next archive to the victim.

Malicious archive content

The campaign has high activity with 25 unique URL detected in 12H on the 31/01/2025.

MSI

The sample requires user interaction to execute, triggered by the execution of the setup.msi file.

The MSI had a VT detection ranging between 3 and 9 out of 60 at the time or analysis.

Within the MSI, two anti-sandbox mechanisms have been identified.

The first anti-sandbox measure presents a button with the label “Please verify that you are not a robot.”.

Anti sandbox: Are you a robot ?

In one of the samples, it was observed that a virtual environment is detected through a feature of Advanced Installer. This detection can be bypassed by modifying the MSI file using Orca.

Disable the antisandbox using Orca

The MSI extracts multiple files into a predefined directory within **%APPDATA%**. These include several clean DLLs and executables, a password-protected archive named **iwhgjds.rar**, and the extraction utility **UnRar.exe**. The **UnRar.exe** utility

is used to decompress the archive using a hard coded password embedded in the MSI, revealing the Stage 1 payload, **obs.dll**.

The command used for extraction is as follows:

```
"C:\Users\???\AppData\Roaming\Viqwo Stars Ci\Rotq App\UnRar.exe" x -p3809610121t -o+ "C:\Users\???\AppData\Roaming\Viqwo
```

Following extraction, the MSI executes **obsffmpegmux.exe**, which sideloads the malicious **obs.dll**.

A script has been developed to directly extract the data from MSI to dll. It extracts the payload and displays the password like this:

```
 python extract_stage1.py setup.msi obs.dll
File exists. Overwrite? [y/N]: y
Found password: 156427613t
Found rarfile: iwhgjds.rar
found file: obs.dll
Stage 2 extracted to: obs.dll
```

This script is attached in appendice.

Stage 1 (obs.dll)

Static analysis

The exports of this DLL are largely empty. Using scripting in IDA, we identified four exports containing code; however, these appear to be largely nonsensical and intended primarily to waste an analyst's time. The IDA script is available at the bottom of the article.

We used BinDiff to compare multiple malicious **obs.dll** and found out that the similarity score on all functions is 1.00, which indicate exact replicates. The difference between the samples are the stage 2 payload and compilation artifacts.

Dynamic analysis

Using x64dbg, we set a breakpoint on **VirtualAlloc** to find buffer creation that could indicate unpacking.

High entropy buffer

This buffer is then decoded, revealing the shellcode.

Shellcode decoding

Shellcode entrypoint

Another buffer in then created and data is copied to it using the **rep movsb** instruction. This buffer is then decoded, revealing the stage 2 executable. We can extract this executable by dumping the memory section of the buffer for further analysis.

Stage 2 decoding

The api calls are retrieved by hashing the function names with a custom hash. In the following captures from left to right: the call to the custom **getprocaddress**, the hashing function, and the relocation section parsing. A yara is available at the end of the article to detect this specific method.

Shellcode API calls retrieving

The shellcode then proceed to start **explorer.exe** and use process hollowing to load the malicious stage 2 using **CreateProcessInternalA, ZwQueryInformationProcess, ReadProcessMemory, ZtUnmapViewOfSection, VirtualAllocEx, WriteProcessMemory** and **NtResumeThread** API calls.

Stage2

The stage2 is not part of LegionDropper, because the malware family is a dropper, any payload can be embedded inside. The analysis has been performed on the previously given samples.

Stage 2 is responsible for communicating with the command and control (C2) server and appears to be designed to download the final payload. However, all extracted C2 servers were inactive at the time of analysis, preventing further investigation.

The domain name is hard coded within the executable.

The user-agent string **Mozilla/5.0 (Windows NT 6.3; Trident/7.0; Touch; rv:11.0) like Gecko** is decoded by XORing two values:

User agent decoding

Additionally, the parameter ‘a’, which is transmitted to the C2 later on, is randomly generated, and always consists of 10 characters:

Parameter ‘a’ generation

Once these values are set, the malware attempts to establish a connection to the C2 using **WinHttpSendRequest**.

If the connection to the C2 was successful, the malware would perform the following actions.

Disables file system redirection by creating a thread that calls **Wow64DisableWow64FsRedirection**.

Downloads a shellcode, saves it as a .dat file, decrypts it using XTEA, and executes it in memory.

Shellcode with Xtea decryption

The malware downloads an additional file from the C2 using **URLDownloadToFileA**, then creates a directory in **%TMP%** with a randomly generated 15-character name. It copies the downloaded file into this directory under the name svchost, retaining the original extension, and finally executes it using **ShellExecuteA** with the “open” option.

Since the malware later attempts to execute the payload using **rundll32**, it is highly likely that the final payload is a DLL, which is launched via **ShellExecuteA** with the “open” option.

ShellExecuteA

IOC

SHA256

Similar SHA256 samples have been encountered. This is a short list,hundreds of sample have been found.

- 41c1006feead9af3e9a563e2814acc8550d36b991e0998015cee00ebb0ac4e85
- cd72eaba97bb94947529a1e652e2d1cc7197b6224e00bf39e55ad634b7e82047
- e88cb0e892537a1dfd7d7d7a4802caeee43d25f871602466a735df0eb5096eb3
- 21d325a59140755b3cf6b075d5e157f37c2771deb29ae7756092fa8978209f77
- 7e9d148d6ebcf927292bba0948ab4d006cb0667084a7f43c04ab7d7efcb9074b
- 23f064df01ee9eedf9e1341185505b86148873ccc0a922c64bb085ceb5b091fc
- 4df98a4f9ecacf1f1676814ad5980dd94d7d33ce4b7d9aec9d96f3c3ea602363
- 76cbe366ea370235dfea2d72378f9d946e49370b4c7bac58e99073e117062e1f
- 8134948177ca6fc350b4c651f27137eaef8dabbb2daf9a1d0447bf1102cfd7d9

HTTP requests

- flash3hit.com/front.php
- flash-hit.com/front.php
- fatal-hit.com/front.php
- vikincdesigns.com/front.php
- lamotionpicture.com/front.php

Files and directories

- Appdata\Roaming\Viqwo Stars Ci\Rotq App\iwhgjds.rar
- Appdata\Roaming\Viqwo Stars Ci\Rotq App\obs.dll
- Appdata\Roaming\Viqwo Stars Ci\Rotq App\UnRar.exe
- AppData\Roaming\Barsoc Quite Sols\Joas App\iwhgjds.rar
- AppData\Roaming\Barsoc Quite Sols\Joas App\obs.dll
- AppData\Roaming\Barsoc Quite Sols\Joas App\UnRar.exe

Registry

- HKU\S-1-5-21-178964467-512603846-2268572703-1002\SOFTWARE\Barsoc Quite Sols\Joas App\Version
- HKU\S-1-5-21-178964467-512603846-2268572703-1002\SOFTWARE\Barsoc Quite Sols\Joas App\Path
- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\Folders\C:\Users\???\AppData\Roaming\Barsoc Quite Sols\Joas App\una_front

Phishing URL

Hundreds of malicious URL have been found (up to 30 in 12 hours for monster TLD), this is just a shortlist.

Phishing URL	Samples related	malicious file URL
https://linefreeapp[.]monster	d43590b090ac1ece44ded29b03301323958e344394e94c439999f6a2d0648c53, a6b5759a273fd6df4dcb0f5c82935b4b60a6f28bfb4d69b6c7c503c8614c39d0, 17be6c8a4cf914056e5cb5d6a1d087069bd4c8d5a3ed104fefeace42c4fc6083	https://mega.nz/file/SN8j(https://dipsos-troak.com/s https://mega.nz/file/GRhg
https://dipsos-troak[.]com	e69a7a881daca7637220d0407454e678ef3a9cf373406b363179f002acd8144d, 038cbe87c4ddb39e7c7accc95d221950d96f2adb0649acaaea60258255c203a6	https://dipsos-troak.com/s https://dipsos-troak.com/s
https://webrecentapp[.]monster/	74ed663ad5369aed6f784d601c1755bbb12ab5df4c5111599332b1bf057d8fe9, d2bcc865d00890a3ba675dc1952c3470205dc9811d4fb354a0b44630879df7c7	https://mega.nz/file/SQ9E
https://runstarapp[.]monster/	5b790d2d085d2498aa63822812562acc256a26febae6cc78563ba656eb9d0c1f, 2eae05e829f353c9a8d01683187eb759dbf73f90ccd435f03d46761b03247fbd	https://mega.nz/file/nRcB
https://topstarapp[.]monster/	4c2c0de6474c17486e5abe2323da0abe4af395a89d0cc46994265ca7719e4ccc, eaaec1cc3ee9a3d590d17c73ab7b174354c1c7be13d26026891424289d0c57fe	https://mega.nz/file/wjUV
https://saveactiveapps[.]monster/	23d0db70ba7848789fa117d25f2e94936cf06e58a03fc36647defdd91bf6f1ca, cd0a77c945f9eb2a8e0cc7b16f00b8426b737618da06df7e65c1913eefbcc18b	https://mega.nz/file/6nozv

https://cleanactiveapp[.]monster/	66241b0c08194263eeb62bae9c4e8ef7e38bb447e671638c9c340d305e23af16, 1a43da62d09a56f50e2797cffb77001027461a6b5ef0713c63d96c60bf8ecadd	https://mega.nz/file/uFwV
https://webnewapp[.]monster/	49c74021ab818ff7a07c184c920585b96000e9079d5beaed3a3dc0ed2fd4834b	https://mega.nz/file/XBQI
https://elitenewapp[.]monster/	d8f2f667708a14734a20d7731ab659fa1ab23ddd25ee96ba4ca33fedf4b7c613, 082a0596b474806cc0ea58c4f7067a4f1166dbb4aa1800bc58af6f99f1209a4a	https://mega.nz/file/vUtj2
https://eliteleaderapp[.]monster/	3938e304ddb11dc02b514e10daa2810bc91fd963e007f5bfba789846e08c6b8e, b59e172cda955322b0cbdc152f723b82eef222014a631dc3b1d8fe4144480374	https://mega.nz/file/xq1H
https://freeleaderapp[.]monster/	5f01f481065fefdf0c34c7f1e0a5dd527857962dae46bcbddb4a2b941bf5a3dc, 1f8ec7a76f4486fdff94743275b2d65e1e4c871f7f933ed5c65c1dfca22909be	https://mega.nz/file/Utpzl
https://webabilityapp[.]monster/	75cdf91e7f10807b81e9cc9754dc37d447d46912537f585e6f6b3e2a84fdb7df	https://mega.nz/file/87sgE
https://topgrandapp[.]monster/	b974015e21e86ca6c89545e86e69732d4dd6e41d588aeb31e4e112a6cd0e237f	https://mega.nz/file/3Vcy
https://safegrandapp[.]monster/	77bbf883dc365ca72fa4e5cd203055a2e14787fc363fbf3409ca266c0607185e	https://mega.nz/file/9MJx
https://extragrandapp[.]monster/	f1064a9546766a69b2df901a0d9df31d31b01c6507cf614ef3ab73f5869af524	https://mega.nz/file/WV5
https://safepowerapp[.]monster/	82eda9820fc42229b2f75d075ef34d11d1b4feb598983640226770c5e2cf8475	https://mega.nz/file/AVpg
https://freepowerapp[.]monster/	9cd58f52226fc376f837447d0c4ebed7b0473cc4166f9e8ad0265bbfd7ac4462	https://mega.nz/file/toZU
https://getglobal[.]monster	f4f4dd8a1fca44d6d7c78da7dc5741b91250eabf8faae79604c786672ea2efb8	https://drive.google.com/f
https://sendspeed[.]monster/	d1a0115f4afe30d9a973cb18bf95d34b67b2d548b4d49989fd0e36399dc562d0	https://drive.google.com/f

Detection

yara

```
 rule crypto_alg_obs_dll {
    meta:
        author = "PEZIER Pierre-Henri. Copyright TEHTRIS 2025"
        description = "The cryptographic diffuser of OBS.dll"
    strings:
        $obj_180005250 = { // 4707b17284e0bdbb92d915e66a8fe4dff18441c958a5230c786d5af6fa05b4bd "C:\Users\user\Desktop\Legi
            49 8d 41 01           // lea     rax, [r9+1]
            44 0f b6 c8           // movzx   r9d, al
            48 8d 52 01           // lea     rdx, [rdx+1]
            43 0f b6 0c 11        // movzx   ecx, byte ptr [r9+r10]
            4a 8d 04 19           // lea     rax, [rcx+r11]
            44 0f b6 d8           // movzx   r11d, al
            43 0f b6 04 13        // movzx   eax, byte ptr [r11+r10]
            43 88 04 11           // mov     [r9+r10], al
            43 88 0c 13           // mov     [r11+r10], cl
            43 0f b6 04 11        // movzx   eax, byte ptr [r9+r10]
            48 03 c1              // add     rax, rcx
            0f b6 c0              // movzx   eax, al
            42 0f b6 0c 10        // movzx   ecx, byte ptr [rax+r10]
            30 4a ff              // xor     [rdx-1], cl
            49 83 e8 01           // sub     r8, 1
            75 c1                 // jnz     short loc_180005250
        }
    condition:
         all of them
}


rule shellcode_library_resolution {
    meta:
        author = "PEZIER Pierre-Henri. Copyright TEHTRIS 2025"
        description = "Legion Loader implementation of GetProcAddress"
    strings:
        $hashed_libs = { // 27e48b5e7925fdc17bef8b7efb8576ee336dbfba31b5f3296bfa9d33c906e385 "C:\Users\user\Desktop\Legior
            ba (4A 0D CE 09|DD F5 53 CD|A0 F7 BF 08|C5 B1 66 2D|33 13 E2 81|4D 82 2E E6|FE 90 CB 49|42 AE C7 F7|2E 97 58 4
            48 [4]                // mov     rcx, [rsp+0A8h+var_78]
            e8                    // call    RE_GETPROCADDRESS
        }
```

```
        $hash_algorithm = { // 27e48b5e7925fdc17bef8b7efb8576ee336dbfba31b5f3296bfa9d33c906e385 "C:\Users\user\Desktop\Leg
            d1 e8                // shr     eax, 1
            8b [3]               // mov     ecx, [rsp+28h+RE_FUNCNAME]
            81 e1 20 83 b8 ed    // and     ecx, 0EDB88320h
            33 c1                // xor     eax, ecx
            89 ?? ??             // mov     [rsp+28h+var_28], eax
            eb                   // jmp     short loc_153D97E0662
        }
        $section_parser = { // 27e48b5e7925fdc17bef8b7efb8576ee336dbfba31b5f3296bfa9d33c906e385 "C:\Users\user\Desktop\Leg
            c7 44 ?? ?? 20 00 00 00   // mov     [rsp+18h+var_10], 20h ; ' '
            c7 44 ?? ?? b9 79 37 9e   // mov     [rsp+18h+var_C], 9E3779B9h
            8b 44 ?? ??               // mov     eax, [rsp+18h+var_C]
            0f af 44                  // imul    eax, [rsp+18h+var_10]
        }
        $path_loading = { // 27e48b5e7925fdc17bef8b7efb8576ee336dbfba31b5f3296bfa9d33c906e385 "C:\Users\user\Desktop\Legio
            48 ?? 5c 53 79 73 57 4f 57 36   // mov     rcx, 36574F577379535Ch; \SysWOW6
            [8-16]
            48 ?? 34 5c 65 78 70 6c 6f 72   // mov     rcx, 726F6C7078655C34h; 4\explor
            [8-16]
            48 ?? 65 72 2e 65 78 65 00 00   // mov     rcx, 6578652E7265h;  er.exe
        }
    condition:
         #hashed_libs > 5
         or $hash_algorithm
         or $section_parser
         or $path_loading
}
```

Appendice

Stage 2 extractor

```
#include <windows.h>
#include <stdio.h>


char suffix[] = "_stage2.exe";
BYTE writeprocessmemory_trampoline[] = {
        0x48, 0xB8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     // mov rax, &MyWriteProcessMemory
        0xFF, 0xE0                                                      // jmp rax
};


BOOL MyWriteProcessMemory(HANDLE hProcess, LPVOID lpBaseAddress, LPCVOID lpBuffer, SIZE_T nSize, SIZE_T *lpNumberOfBytesWrit
{
        char file_name[MAX_PATH + 1] = {0};
        if(!GetModuleFileName(NULL, file_name, MAX_PATH) || strlen(file_name) >= MAX_PATH - sizeof(suffix)) {
                fprintf(stderr, "Cannot get file name\n");
                exit(0);
        }
        strncat(file_name, suffix, MAX_PATH);
        printf("Stage 2 saved to %s\n", file_name);

        FILE *out = fopen(file_name, "wb");
        if(!out) {
                fprintf(stderr, "Cannot open file for writing: %s\n", file_name);
                exit(0);
        }
        fwrite(lpBuffer, 1, nSize, out);
        fclose(out);
        exit(0);  // DO NOT EXECUTE THE PAYLOAD
        return TRUE;
}


int main(int argc, char ** argv)
{
        if(argc != 2) {
```

```
                fprintf(stderr, "Usage: %s <file_to_extract>\n", argv[0]);
                return 1;
        }
        DWORD oldprotect;
        if(!VirtualProtect(WriteProcessMemory, sizeof(writeprocessmemory_trampoline), PAGE_EXECUTE_READWRITE, &oldprotect))
                fprintf(stderr, "Failed to patch WriteProcessMemory\n");
                return 1;
        }
        SIZE_T MyWriteProcessMemory_addr = (SIZE_T)&MyWriteProcessMemory;
        memcpy(&writeprocessmemory_trampoline[2], &MyWriteProcessMemory_addr, sizeof(MyWriteProcessMemory_addr));

        memcpy(WriteProcessMemory, writeprocessmemory_trampoline, sizeof(writeprocessmemory_trampoline));

        if(!LoadLibrary(argv[1])) {
                fprintf(stderr, "Failed to load the DLL (%i).\n", GetLastError());
                return 2;
        }
        system("pause");
}
```

IDAPython script to identify exports

```
import idc
import pefile
import ida_nalt
import idaapi

file_path = idc.get_input_file_path()
mype = pefile.PE(file_path)

for imp in mype.DIRECTORY_ENTRY_EXPORT.symbols:
        if idaapi.get_byte(ida_nalt.get_imagebase() + imp.address) != 194:
                print(impt.name, hex(ida_nalt.get_imagebase() + imp.address))
```

Stage1 extractor

```
import magic
import re
import shutil
import subprocess
import binascii
import tempfile
from pathlib import Path
from cabarchive import CabArchive
from rarfile import RarFile
"""
pip install git+https://github.com/hughsie/python-cabarchive.git
"""

CABINET_FILE_HEADER = binascii.unhexlify("4D 53 43 46 00 00 00 00".replace(" ", ""))


def extract_legionstealer_stage1(file_path: Path, output_file: Path) -> None:
    assert "MSI Installer" in magic.from_file(file_path), "The given file is not a MSI executable"
    assert file_path.stat().st_size < 1024 * 1024 * 100, "File too big"
    data = file_path.read_bytes()
    assert (key_re := re.search(rb"QuiteSes(.{,20}?)(?=#)", data)), "Unable to find the rar password. Is it a LegionLoader s
    try:
        password = key_re.group(1).decode("utf-8")
    except UnicodeError:
        raise AssertionError("Unable to find the rar password. Is it a LegionLoader sample?") from UnicodeError
    print("Found password:", password)
    assert (offset := data.find(CABINET_FILE_HEADER)), "Cannot find cabinet file. Is MSI corrupted ?"
    archive = CabArchive(data[offset:])
    assert (rarfile := archive.find_file("*.rar")), "No rar file in the MSI. Is it a LegionLoader sample?"
```

```
    with tempfile.TemporaryDirectory() as _tmp:
        tmp = Path(_tmp)
        tmp_rar = tmp / "tmp.rar"
        print("Found rarfile:", rarfile.filename)
        tmp_rar.write_bytes(rarfile.buf)
        try:
            res = subprocess.run(["/usr/bin/unar", "-p", password, "-o", tmp, tmp_rar], capture_output=True)
        except FileNotFoundError:
            raise AssertionError("Unable to find unar. Please install unar (apt install unar).")
        if res.returncode:
            raise AssertionError("Unable to extract rarfile. Is it a LegionLoader sample?")
        tmp_rar.unlink()
        assert len(res := list(tmp.glob("*.dll"))) == 1, "More than one file found. Is it a LegionLoader sample?"
        print("found file:", res[0].name)
        shutil.copy(res[0], output_file)


if __name__ == "__main__":
    import sys
    if len(sys.argv) != 3:
        print("Usage:", sys.argv[0], "<msi> <output_stage_2>", file=sys.stderr)
        sys.exit(1)
    output_file = Path(sys.argv[2])
    if output_file.exists() and input("File exists. Overwrite? [y/N]: ") != "y":
        print("exitting")
    else:
        extract_legionstealer_stage1(Path(sys.argv[1]), output_file)
        print("Stage 2 extracted to:", output_file)
```

Source: https://tehtris.com/en/blog/legionloader-exposed/