Recover your files with StrongPity

anchorednarratives.substack.com/p/recover-your-files-with-strongpity

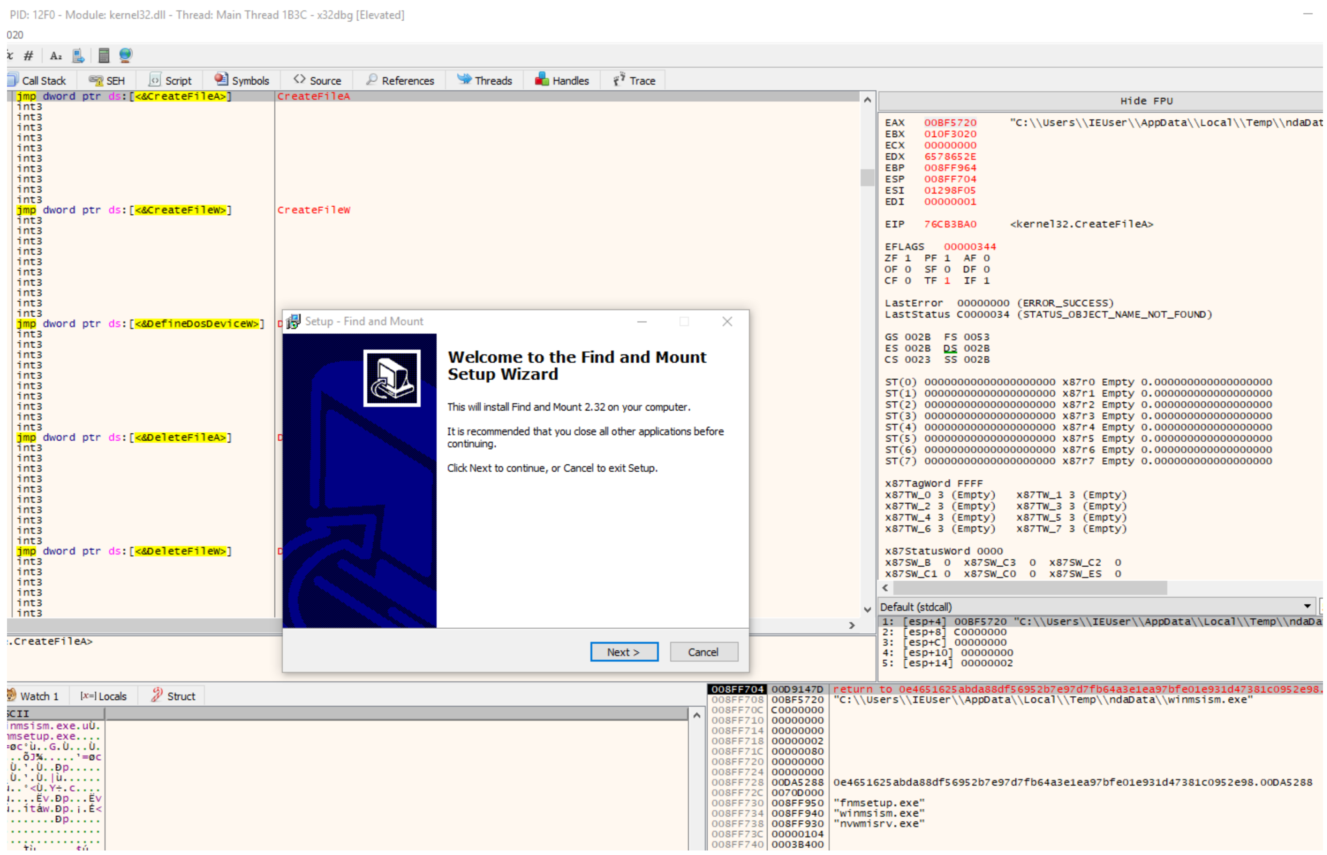
RJM

A case study of a multi-year nation-state cyber surveillance campaign

RJM

Apr 18, 2021

Disclaimer: The views, methods, and opinions expressed at Anchored Narratives are those of the author and do not necessarily reflect the official policy or position of my employer.



```
jmp dword ptr ds:[<&CreateFileA>]        CreateFileA
int3
int3
int3
int3
int3
int3
int3
int3
int3
int3
jmp dword ptr ds:[<&CreateFileW>]        CreateFileW
int3
int3
int3
int3
int3
int3
int3
int3
int3
int3
jmp dword ptr ds:[<&DefineDosDeviceW>]
int3
int3
int3
int3
int3
int3
int3
int3
int3
int3
jmp dword ptr ds:[<&DeleteFileA>]
int3
int3
int3
int3
int3
int3
int3
int3
int3
int3
jmp dword ptr ds:[<&DeleteFileW>]
int3
int3
int3
int3
int3
int3
```

```
.CreateFileA>
```

Setup - Find and Mount

Welcome to the Find and Mount Setup Wizard

This will install Find and Mount 2.32 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

```
EAX   00BF5720     "C:\\Users\\IEUser\\AppData\\Local\\Temp\\ndaDat
EBX   010F3020
ECX   00000000
EDX   6578652E
EBP   008FF964
ESP   008FF704
ESI   01298F05
EDI   00000001

EIP   76CB3BA0     <kernel32.CreateFileA>

EFLAGS   00000344
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 1  IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) 00000000000000000000 x87r0 Empty 0.000000000000000000
ST(1) 00000000000000000000 x87r1 Empty 0.000000000000000000
ST(2) 00000000000000000000 x87r2 Empty 0.000000000000000000
ST(3) 00000000000000000000 x87r3 Empty 0.000000000000000000
ST(4) 00000000000000000000 x87r4 Empty 0.000000000000000000
ST(5) 00000000000000000000 x87r5 Empty 0.000000000000000000
ST(6) 00000000000000000000 x87r6 Empty 0.000000000000000000
ST(7) 00000000000000000000 x87r7 Empty 0.000000000000000000

x87TagWord FFFF
x87TW_0 3 (Empty)   x87TW_1 3 (Empty)
x87TW_2 3 (Empty)   x87TW_3 3 (Empty)
x87TW_4 3 (Empty)   x87TW_5 3 (Empty)
x87TW_6 3 (Empty)   x87TW_7 3 (Empty)

x87StatusWord 0000
x87SW_B  0  x87SW_C3  0  x87SW_C2  0
x87SW_C1 0  x87SW_C0  0  x87SW_ES  0
```

```
1: [esp+4] 00BF5720 "C:\\Users\\IEUser\\AppData\\Local\\Temp\\ndaDa
2: [esp+8] C0000000
3: [esp+C] 00000000
4: [esp+10] 00000000
5: [esp+14] 00000002
```

```
SCII
inmsism.exe.uÙ.
nmsetup.exe....
=øc°ù..G.Ù...Ù.
..õJ¾.....'=øc
Ù.'.Ù..Ðp.....
Ù.'.Ù.|ù......
1..°<Ù.Y÷.c....
1....Ëv.Ðp...Ëv
1..ítàw.Ðp.¡.Ë<
.......Ðp.....
..............
```

```
008FF704 00D9147D return to 0e4651625abda88df56952b7e97d7fb64a3e1ea97bfe01e931d47381c0952e98.
008FF708 00BF5720 "C:\\Users\\IEUser\\AppData\\Local\\Temp\\ndaData\\winmsism.exe"
008FF70C C0000000
008FF710 00000000
008FF714 00000000
008FF718 00000002
008FF71C 00000080
008FF720 00000000
008FF724 00000000
008FF728 00DA5288 0e4651625abda88df56952b7e97d7fb64a3e1ea97bfe01e931d47381c0952e98.00DA5288
008FF72C 0070D000
008FF730 008FF950 "fnmsetup.exe"
008FF734 008FF940 "winmsism.exe"
008FF738 008FF930 "nvwmisrv.exe"
008FF73C 00000104
008FF740 0003B400
```

Cover: StrongPity APT Actor hides the backdoor in legitimate “recovery” software.

Introduction

First, a warm welcome to the new subscribers of the Anchored Narratives mailing list. For the ones who are new to the list, I regularly pick an exciting tweet that matched my intelligence requirements and generated *anchored* stories on geopolitical (cyber) threats, digital forensics, and crime from that. Usually, I pick a story that I have no real in-depth or prior knowledge about. The goal is to understand a particular topic better, improve my investigation or writing skills, and generate a reliable story anchored with evidence. This time the story will start with a tweet that matched my intelligence requirements on 15 March 2021:

```
"#apt #strongpity new sample hunted md5:95ff679f525c44e4abac8e61f8052ca5 c2 :
transferprotocolpolicy.com"
```

The information in the tweet tells people with interest in this field that someone found a malicious malware sample with a unique value “*95ff679f525c44e4abac8e61f8052ca5*”from an Advanced Persistent Threat actor group called StrongPity. APT is an industry name for referring to states involved in cyber operations. The referred malware sample communicates to its command and control server “*transferprotocolpolicy.com*” (c2) for further instructions. This tweet triggered some personal interest to start a deep dive into this nation-state actor group. They have been around for many years, deploy interesting tactics at scale, and are observed in geopolitical disputes. This article will outline the background of this alleged Turkish nation-state actor or nation-state-sponsored group. Furthermore, the malicious backdoor will be reversed briefly and based on that intelligence to hunt for additional indicators, and finally, the article will end with some observations and a conclusion. Let’s go.

Background StrongPity

The StrongPity actor group has been around since *2012* and employs the same tactics, namely adding backdoors to legitimate software used by specific users. Some call this technique water holing. The group is also referred to as APT-C-41 and PROMETHIUM. In *2016* StrongPity was detected by Kaspersky in a campaign that targeted specific users in Belgium and Italy who were interested in Truecrypt and Winrar software. The software packages are used by niche user groups interested in solid encryption. The actor group set up a domain name that mimicked the official WinRAR distribution site and placed links to the trojanized WinRAR installer on a certified distributor website. In the same year, Microsoft observed a campaign by the same group targeting specific users with a zero-day vulnerability in Adobe Flash. The zero-day exploit was tracked as CVE-2016-4117. In *2017* ESET published research where they detected StrongPity while tracking the FinFisher group and an Internet Service Provider's involvement. Their analysis revealed that users were redirected to trojanized software packages. Some of the targeted software were the following software packages.

- `CCleaner v 5.34`
- `Driver Booster`
- `The Opera Browser`
- `Skype`
- `The VLC Media Player v2.2.6 (32bit)`

- `WinRAR 5.50`

In their research, *ESET* states that an exfiltration component in the StrongPity backdoor collects files with the following extensions:

> .ppt,.pptx,.xls,.xlsx,.txt,.doc,.docx,.pdf,.rtf

The stolen files are sent to a central server operated by the StrongPity actor, and the backdoor waits for further instructions.

By *2018* *Citizenlab* found several so-called deep packet inspection devices in Türk Telekom's network where users were redirected to download trojanized installers Avast Antivirus, CCleaner, Opera, and 7-Zip. The surveillance was set up so that users who searched for official downloads on the authorized vendor websites were silently redirected to the trojanized versions of Avast, CCleaner etcetera. Citizenlab referred to the malware as StrongPity, which was used after they stopped using FinFisher spyware. FinFisher was sold to governments as a lawful interception capability. Citizenlab also described that these injection techniques were also observed by other nation-states, China (Great Cannon) and the US (NSA's QUANTUM).

In June *2020,* *Bitdefender* published research where StrongPity employed similar tactics to infect victims in Turkey and Syria selectively. According to Bitdefender, the group was specifically interested in the Kurdish community giving it a geopolitical angle.

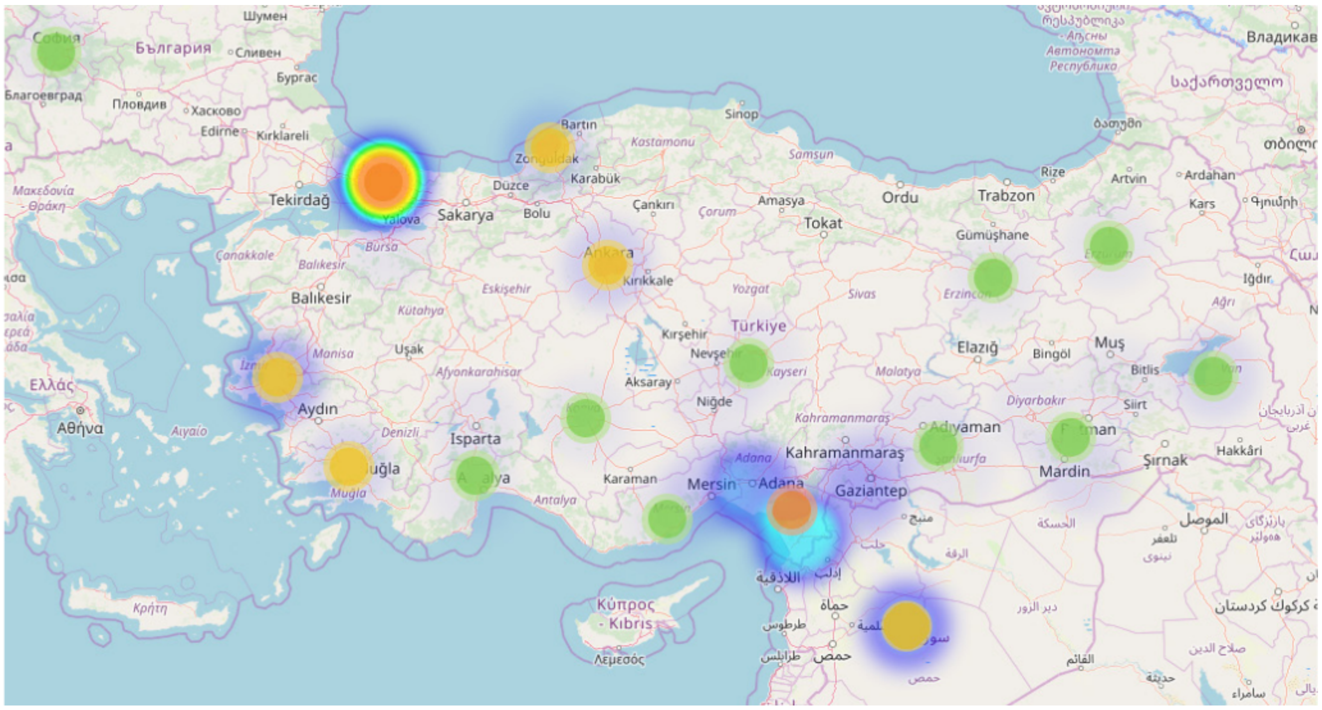


Figure 1: Figure adopted from the BitDefender report. Victims were concentrated in the area of Turkey and Syria.
In their investigation, *Bitdefender* found trojanized versions of the following software:

- `7-ZIP`
- `WinRAR`
- `McAfee Security Scan Plus`
- `File recovery application - Recuva`
- `TeamViewer`
- `WhatsApp`
- `CCleaner`
- `CleverFiles Disk Drill`
- `DAEMON Tools Lite`

They also found a particular *tag* used as authentication and is influenced by the file's compilation time. These tags could look like something like “v11_kt26“ for example. To me, these tags resemble campaign identifiers used by actors to distinguish between different targets. The researchers from Bitdefender added a tremendous amount of StrongPity samples in their report indicating an extensive campaign.

Researchers from *Cyble* released a report in December 2020 that the StrongPity actors expanded their global reach and included mass phishing e-mail campaigns. According to their research, victims were now widespread across Europe, Nothern Africa, Canada, and Asia. *Cyble* discovered that the victim was targeted through a trojanized version of the Partition Find and Mount software utility. Their analysis refers to a screenshot (figure 6) that should demonstrate the decryption routines and decrypted payloads in the process memory. Especially those screenshots are blurred and not readable. After that, they report that the malware creates a mutex1 with a particular name (figure 7) and then how the malware connects to a specific domain in a debugger (figure 8). It remains unclear how the mutex's name is generated and where and how the command and control information is stored from their research. The claim of mass phishing attacks is not substantiated by e-mail samples in their report. Their released StrongPity indicators already contain the “transferprotocolpolicy[.]com” as a command and control server that matched the starting tweet, which matched my intelligence requirements in March 2021.

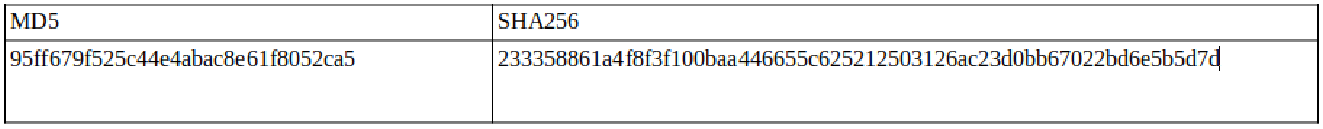
In *2021 LMNTRIX* released research into what they call “*the Turkish APT group APT-C-41 (aka StrongPity and Promethium)*”. They claim that the group targets Financial organizations, Industrial plants, and Educational institutes after installing a backdoor on its victims. Their research provides some screenshots of a disassembly tool in which they state that the malware has so-called anti-debugging functionality enabled (IsDebuggerPresent check).

They further state: “*After bypassing these functions, we found the command and control domain embedded into the code. The snapshot shows the communication happens to the malicious domain, which we highlighted below [mailtransfersagents(dot)com]:*”

The malware samples referenced in their research are indeed StrongPity samples. Based on the screenshots *LMNTRIX* provided, I could not observe the command and control domain embedded in the provided snapshots.

Both reports of *Cyble* and *LMNTRIX* triggered me to dive into some reversing of the backdoor functionality to determine how the StrongPity backdoor stores its configuration, as this was not clear to me from their analysis. Let’s start with the sample that triggered my intelligence requirements in the first place.

Reversing the StrongPity backdoor



MD5	SHA256
95ff679f525c44e4abac8e61f8052ca5	233358861a4f8f3f100baa446655c625212503126ac23d0bb67022bd6e5b5d7d

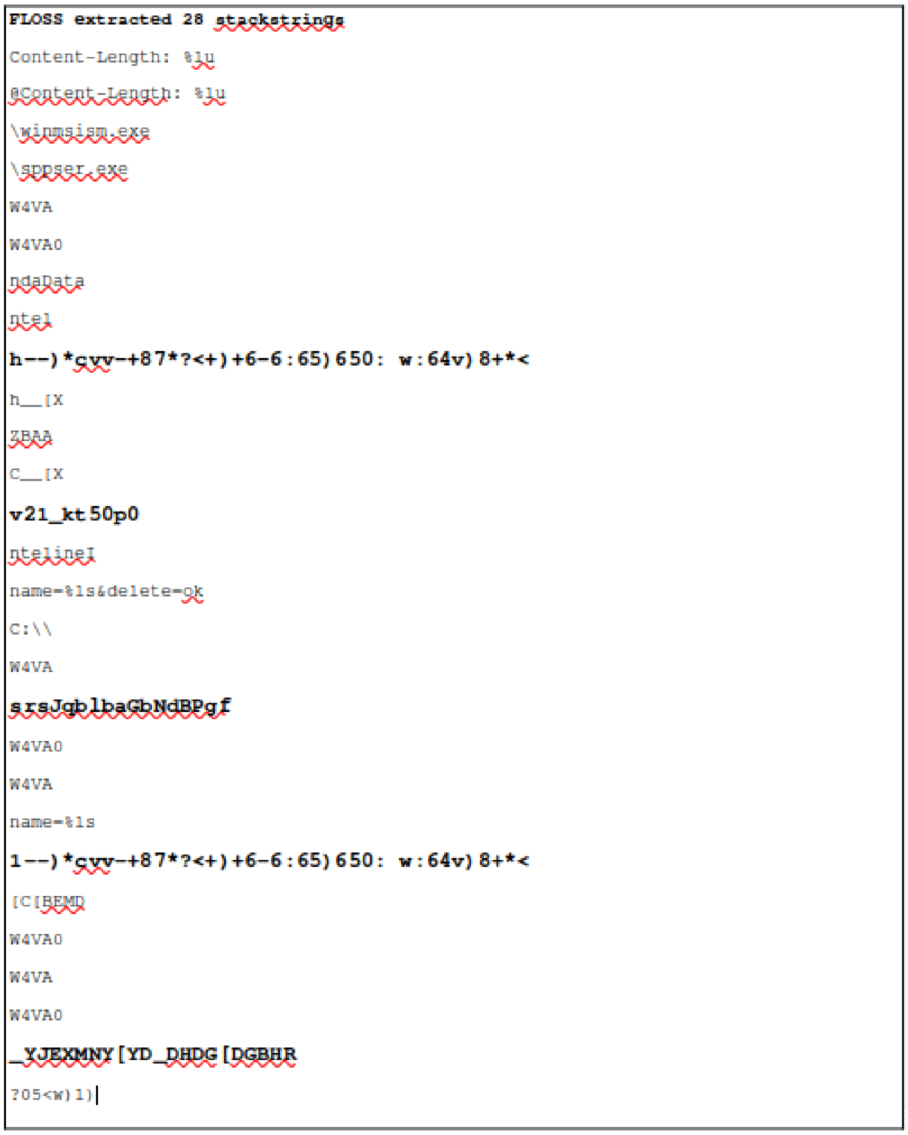
Table 1: Checksums of the StrongPity backdoor that will be investigated.
The StrongPity backdoor is installed via trojanized installations of legitimate and popular software products. The extensive research of *Citizenlab* indicates that a Telecom provider in Turkey was involved in the redirection to the trojanized downloads to its victims.

What is not clear to me is how the configuration data is stored in the malware. To understand how that data is stored, I will follow the regular malware reversing process. You start with static analysis. What information can you get out of the malware sample without executing it? If things are not evident by then, you can also execute the malware in a sandbox or run it in a so-called debugger. For readability, I will only focus on the main findings.

Static analysis

Usually, you’ll start looking at which ‘*strings*’ (text) are present in the malware sample. Analyzing strings in binary files is an essential aspect of malware analysis. This technique provides valuable information about the program’s use and its functionality. Usually, string output is used to develop *Yara* signatures. *Yara* is a tool to identify and classify malware families. Unique strings, constants, or byte patterns are used in the so-called *Yara* signatures to find more samples. Usually, these signatures hold indicators of compromise, like filenames or specific user agents observed in the malware samples. Malware authors generally leverage obfuscation or encryption techniques to hide their secrets that they need to store in the binary. They will also employ anti-debugging tricks to hinder automated analysis. To leverage the Windows operating system's functionalities, malware authors often rely on standard Windows Application Programming Interfaces (API) for their backdoors to interact with the system. Usually, these APIs are seen in the ‘*strings*’ output, but malware authors can also hide this. In the StrongPity sample, many of these APIs were observed, like *CreateMutexW*, *CreateProcessW*, *WinHttpConnect*, and *IsDebuggerPresent*. The regular ‘*strings*‘ command on Linux revealed no domain information, however.

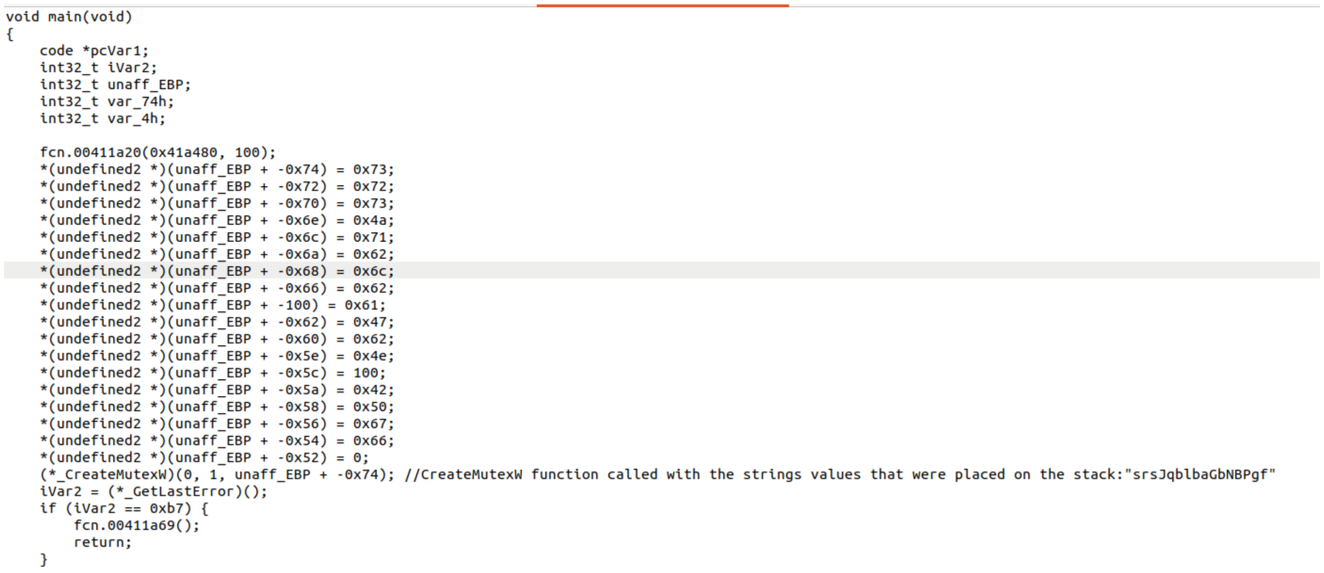
To determine if the StrongPity malware authors employed an obfuscation technique called stack strings, the ‘*floss*’ program was used. That revealed the following information:



```
FLOSS extracted 28 stackstrings
Content-Length: %lu
@Content-Length: %lu
\winmsism.exe
\sppser.exe
W4VA
W4VA0
ndaData
ntel
h--)*cvv-+87*?<+)+6-6:65)650:  w:64v)8+*<
h__[X
ZBAA
C__[X
v21_kt50p0
ntelineI
name=%ls&delete=ok
C:\\
W4VA
srsJgblbaGbNdBPgf
W4VA0
W4VA
name=%ls
1--)*cvv-+87*?<+)+6-6:65)650:  w:64v)8+*<
[C[BEMD
W4VA0
W4VA
W4VA0
_YJEXMNY[YD_DHDG[DGBHR
?05<w)1)
```

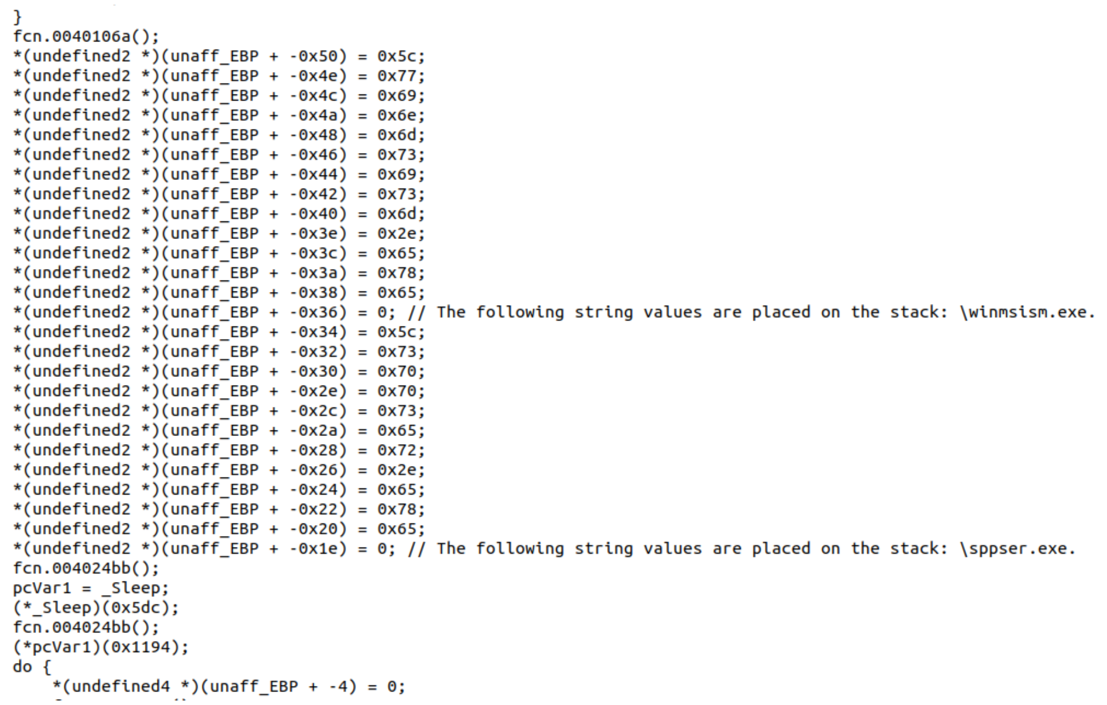
Screenshot 1: Floss detected stack strings in the StrongPity backdoor

The extracted information reveals file names (winmsism.exe. sppser.exe), but also “ndaData“ the directory where the malware collects its information before sending it to the operators, according to the reports. Other than those indicators, I have highlighted some suspicious string patterns. By briefly assessing this output, it looks like this is the config information stored in the StrongPity backdoor. But we need to a bit more digging, and I will use a free open source disassembly tool called *Cutter* for that. A decompiler is a program that analyzes executable programs and tries to create a high-level representation of the machine code from it. *Cutter* has a feature to decompile an executable program to reconstruct the source code. This feature helps to understand the analyst's flow and how the malware program calls certain functions or routines. By decompiling the main function of the StrongPity malware, it becomes immediately apparent how the file names and the mutex observed in the *floss* stack strings output are being passed to the relevant functions.



```
void main(void)
{
    code *pcVar1;
    int32_t iVar2;
    int32_t unaff_EBP;
    int32_t var_74h;
    int32_t var_4h;

    fcn.00411a20(0x41a480, 100);
    *(undefined2 *)(unaff_EBP + -0x74) = 0x73;
    *(undefined2 *)(unaff_EBP + -0x72) = 0x72;
    *(undefined2 *)(unaff_EBP + -0x70) = 0x73;
    *(undefined2 *)(unaff_EBP + -0x6e) = 0x4a;
    *(undefined2 *)(unaff_EBP + -0x6c) = 0x71;
    *(undefined2 *)(unaff_EBP + -0x6a) = 0x62;
    *(undefined2 *)(unaff_EBP + -0x68) = 0x6c;
    *(undefined2 *)(unaff_EBP + -0x66) = 0x62;
    *(undefined2 *)(unaff_EBP + -100) = 0x61;
    *(undefined2 *)(unaff_EBP + -0x62) = 0x47;
    *(undefined2 *)(unaff_EBP + -0x60) = 0x62;
    *(undefined2 *)(unaff_EBP + -0x5e) = 0x4e;
    *(undefined2 *)(unaff_EBP + -0x5c) = 100;
    *(undefined2 *)(unaff_EBP + -0x5a) = 0x42;
    *(undefined2 *)(unaff_EBP + -0x58) = 0x50;
    *(undefined2 *)(unaff_EBP + -0x56) = 0x67;
    *(undefined2 *)(unaff_EBP + -0x54) = 0x66;
    *(undefined2 *)(unaff_EBP + -0x52) = 0;
    (*_CreateMutexW)(0, 1, unaff_EBP + -0x74); //CreateMutexW function called with the strings values that were placed on the stack:"srsJqblbaGbNBPgf"
    iVar2 = (*_GetLastError)();
    if (iVar2 == 0xb7) {
        fcn.00411a69();
        return;
    }
```

Screenshot 2: String put on the stack passed onto the *CreateMutexW* function
So the mutex created from the stack strings in the StrongPity backdoor can be seen in screenshot 2.



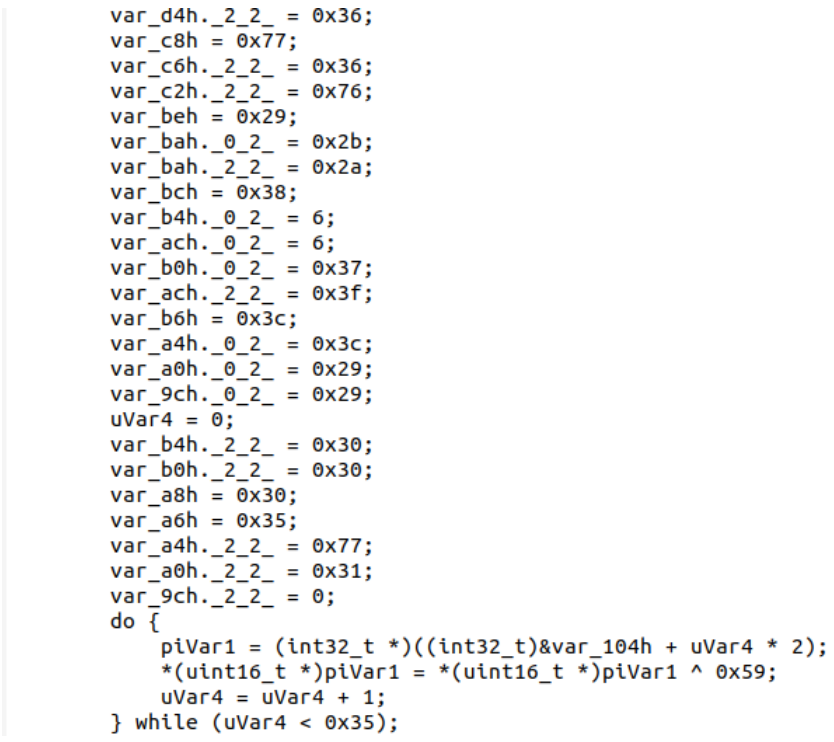
```
}
fcn.0040106a();
*(undefined2 *)(unaff_EBP + -0x50) = 0x5c;
*(undefined2 *)(unaff_EBP + -0x4e) = 0x77;
*(undefined2 *)(unaff_EBP + -0x4c) = 0x69;
*(undefined2 *)(unaff_EBP + -0x4a) = 0x6e;
*(undefined2 *)(unaff_EBP + -0x48) = 0x6d;
*(undefined2 *)(unaff_EBP + -0x46) = 0x73;
*(undefined2 *)(unaff_EBP + -0x44) = 0x69;
*(undefined2 *)(unaff_EBP + -0x42) = 0x73;
*(undefined2 *)(unaff_EBP + -0x40) = 0x6d;
*(undefined2 *)(unaff_EBP + -0x3e) = 0x2e;
*(undefined2 *)(unaff_EBP + -0x3c) = 0x65;
*(undefined2 *)(unaff_EBP + -0x3a) = 0x78;
*(undefined2 *)(unaff_EBP + -0x38) = 0x65;
*(undefined2 *)(unaff_EBP + -0x36) = 0; // The following string values are placed on the stack: \winmsism.exe.
*(undefined2 *)(unaff_EBP + -0x34) = 0x5c;
*(undefined2 *)(unaff_EBP + -0x32) = 0x73;
*(undefined2 *)(unaff_EBP + -0x30) = 0x70;
*(undefined2 *)(unaff_EBP + -0x2e) = 0x70;
*(undefined2 *)(unaff_EBP + -0x2c) = 0x73;
*(undefined2 *)(unaff_EBP + -0x2a) = 0x65;
*(undefined2 *)(unaff_EBP + -0x28) = 0x72;
*(undefined2 *)(unaff_EBP + -0x26) = 0x2e;
*(undefined2 *)(unaff_EBP + -0x24) = 0x65;
*(undefined2 *)(unaff_EBP + -0x22) = 0x78;
*(undefined2 *)(unaff_EBP + -0x20) = 0x65;
*(undefined2 *)(unaff_EBP + -0x1e) = 0; // The following string values are placed on the stack: \sppser.exe.
fcn.004024bb();
pcVar1 = _Sleep;
(*_Sleep)(0x5dc);
fcn.004024bb();
(*pcVar1)(0x1194);
do {
    *(undefined4 *)(unaff_EBP + -4) = 0;
```

Screenshot 3: File names that were put on the stack that was detected by *floss*
The file names are also created from the stack-based strings values.

In screenshot 3 the function with the name *fcn.0040106a();* is executed. That function leads to two so-called byte encoding algorithms by leveraging a single-byte XOR operation with 0x59 and 0x2b. Malware authors often use XOR as this algorithm obfuscates data easily. XOR is a bitwise operation. If you XOR something twice with the same key, this will result in the original value. In the example below, the capital character “A” will be XOR’ed with the XOR Key “B”. The output data of the XOR operation is a non-printable character (NP). If we then XOR that value with the original XOR key “B” we have the original value back.

```
Input data  = 'A' = 01000001
 XOR Key    = 'B' = 01000010
Output data (NP)  = 00000011
 XOR Key    = 'B' = 01000010
Output data = 'A' = 01000001
```

As shown in the example above, these operations can result in unreadable information as non-printable characters are not printed or detected by the ‘strings’ or ‘*floss*’ utilities. Let’s continue with the analysis.



```
        var_d4h._2_2_ = 0x36;
        var_c8h = 0x77;
        var_c6h._2_2_ = 0x36;
        var_c2h._2_2_ = 0x76;
        var_beh = 0x29;
        var_bah._0_2_ = 0x2b;
        var_bah._2_2_ = 0x2a;
        var_bch = 0x38;
        var_b4h._0_2_ = 6;
        var_ach._0_2_ = 6;
        var_b0h._0_2_ = 0x37;
        var_ach._2_2_ = 0x3f;
        var_b6h = 0x3c;
        var_a4h._0_2_ = 0x3c;
        var_a0h._0_2_ = 0x29;
        var_9ch._0_2_ = 0x29;
        uVar4 = 0;
        var_b4h._2_2_ = 0x30;
        var_b0h._2_2_ = 0x30;
        var_a8h = 0x30;
        var_a6h = 0x35;
        var_a4h._2_2_ = 0x77;
        var_a0h._2_2_ = 0x31;
        var_9ch._2_2_ = 0;
        do {
            piVar1 = (int32_t *)((int32_t)&var_104h + uVar4 * 2);
            *(uint16_t *)piVar1 = *(uint16_t *)piVar1 ^ 0x59;
            uVar4 = uVar4 + 1;
        } while (uVar4 < 0x35);
```

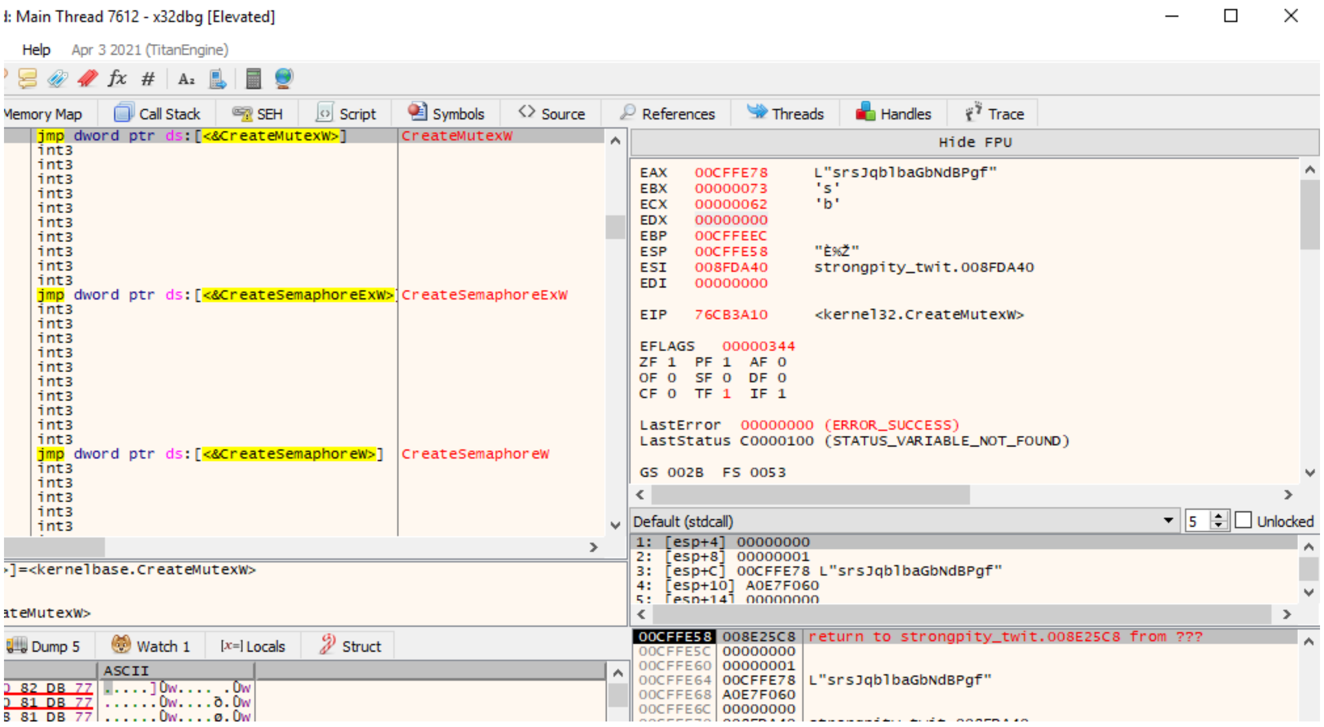
Screenshot 4:

Strings put on the stack are xor’ed with 0x59

The XOR operation with the 0x59 byte values will eventually decode the encoded stack strings to the first domain and URL, namely *“hxxps://transferprotocolpolicy.com/parse_ini_file.php"* The XOR operation with the 0x2b byte values finally results in the following domain and URL after decoding *“hxxps://transferprotocolpolicy.com/phpinfo.php"*

Dynamic analysis

So after the static analysis, the StrongPity sample was executed in x64dbg on my isolated virtual machine for some dynamic confirmation of the initial findings. By setting a breakpoint on the *CreateMutexW* and *GetTempPathW* API functions, the StrongPity backdoor reveals the creation of the same mutex and later on deobfuscation of the domains and URL used by the StrongPity actors. I will briefly describe the findings with some screenshots below.



```
jmp dword ptr ds:[<&CreateMutexW>]            CreateMutexW
int3
int3
int3
int3
int3
int3
int3
int3
int3
int3
jmp dword ptr ds:[<&CreateSemaphoreExW>]      CreateSemaphoreExW
int3
int3
int3
int3
int3
int3
int3
int3
int3
int3
jmp dword ptr ds:[<&CreateSemaphoreW>]        CreateSemaphoreW
int3
int3
int3
int3
int3

]=<kernelbase.CreateMutexW>

ateMutexW>
```

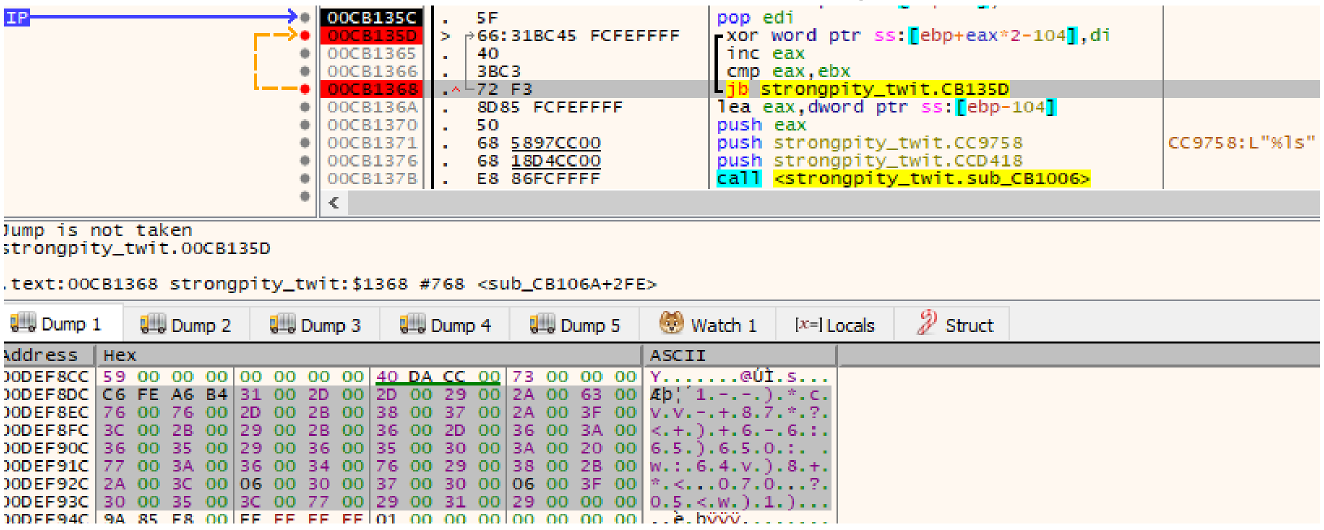
```
EAX    00CFFE78     L"srsJqblbaGbNdBPgf"
EBX    00000073     's'
ECX    00000062     'b'
EDX    00000000
EBP    00CFFEEC
ESP    00CFFE58     "È%Ž"
ESI    008FDA40     strongpity_twit.008FDA40
EDI    00000000

EIP    76CB3A10     <kernel32.CreateMutexW>

EFLAGS   00000344
ZF 1   PF 1   AF 0
OF 0   SF 0   DF 0
CF 0   TF 1   IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000100 (STATUS_VARIABLE_NOT_FOUND)

GS 002B   FS 0053
```

```
1: [esp+4]  00000000
2: [esp+8]  00000001
3: [esp+C]  00CFFE78 L"srsJqblbaGbNdBPgf"
4: [esp+10] A0E7F060
5: [esp+14] 00000000
```

```
00CFFE58 | 008E25C8 | return to strongpity_twit.008E25C8 from ???
00CFFE5C | 00000000 |
00CFFE60 | 00000001 |
00CFFE64 | 00CFFE78 | L"srsJqblbaGbNdBPgf"
00CFFE68 | A0E7F060 |
00CFFE6C | 00000000 |
```

```
ASCII
 82 DB 77 | .....]Ow.... .Ow
 81 DB 77 | ......Ow....ð.Ow
 81 DB 77 | ......Ow....ø.Ow
```

Screenshot 5:CreateMutex creation based on the stack strings *floss* output

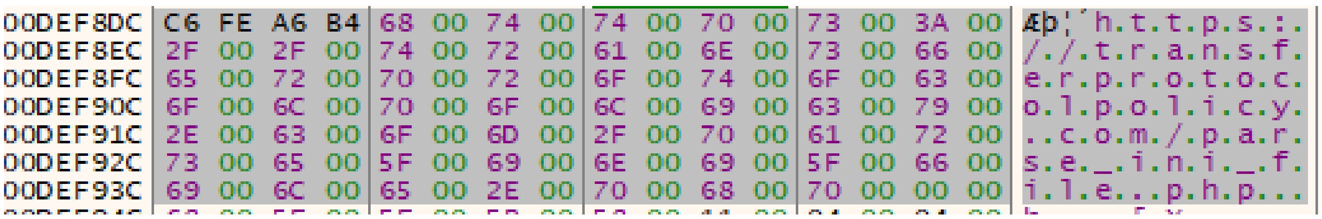


```
00CB135C  .    5F                 pop edi
00CB135D  >   ↱66:31BC45 FCFEFFFF  xor word ptr ss:[ebp+eax*2-104],di
00CB1365  .    40                 inc eax
00CB1366  .    3BC3               cmp eax,ebx
00CB1368  .^  └72 F3              jb strongpity_twit.CB135D
00CB136A  .    8D85 FCFEFFFF      lea eax,dword ptr ss:[ebp-104]
00CB1370  .    50                 push eax
00CB1371  .    68 5897CC00        push strongpity_twit.CC9758         CC9758:L"%ls"
00CB1376  .    68 18D4CC00        push strongpity_twit.CCD418
00CB137B  .    E8 86FCFFFF        call <strongpity_twit.sub_CB1006>

Jump is not taken
strongpity_twit.00CB135D

.text:00CB1368 strongpity_twit:$1368 #768 <sub_CB106A+2FE>
```

```
Address  | Hex                                             | ASCII
00DEF8CC | 59 00 00 00 00 00 00 00 40 DA CC 00 73 00 00 00 | Y.......@ÚÌ.s...
00DEF8DC | C6 FE A6 B4 31 00 2D 00 2D 00 29 00 2A 00 63 00 | Æþ¦´1.-.-.).*.c.
00DEF8EC | 76 00 76 00 2D 00 2B 00 38 00 37 00 2A 00 3F 00 | v.v.-.+.8.7.*.?.
00DEF8FC | 3C 00 2B 00 29 00 2B 00 36 00 2D 00 36 00 3A 00 | <.+.).+.6.-.6.:.
00DEF90C | 36 00 35 00 29 00 36 00 35 00 30 00 3A 00 20 00 | 6.5.).6.5.0.:. .
00DEF91C | 77 00 3A 00 36 00 34 00 76 00 29 00 38 00 2B 00 | w.:.6.4.v.).8.+.
00DEF92C | 2A 00 3C 00 06 00 30 00 37 00 30 00 06 00 3F 00 | *.<...0.7.0...?.
00DEF93C | 30 00 35 00 3C 00 77 00 29 00 31 00 29 00 00 00 | 0.5.<.w.).1.)...
00DEF94C | 9A 85 E8 00 FF FF FF FF 01 00 00 00 00 00 00 00 | ..è.hÿÿÿ........
```

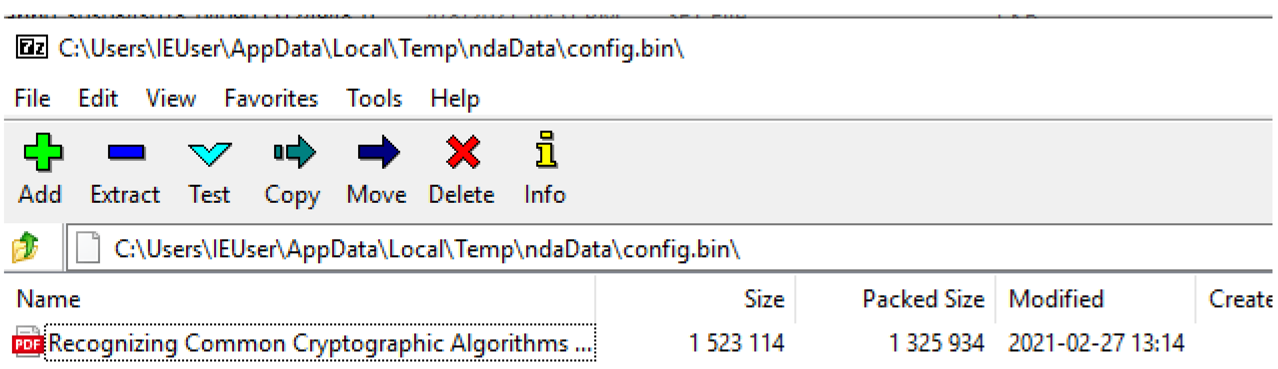
Screenshot 6: Encoded stack strings stored in memory (dump) before first XOR (0x59) routine

Screenshot 6 displays the stack strings (partially) found by *floss,* encoding the domain name.



```
00DEF8DC | C6 FE A6 B4 68 00 74 00 74 00 70 00 73 00 3A 00 | Æþ¦´h.t.t.p.s.:.
00DEF8EC | 2F 00 2F 00 74 00 72 00 61 00 6E 00 73 00 66 00 | /./.t.r.a.n.s.f.
00DEF8FC | 65 00 72 00 70 00 72 00 6F 00 74 00 6F 00 63 00 | e.r.p.r.o.t.o.c.
00DEF90C | 6F 00 6C 00 70 00 6F 00 6C 00 69 00 63 00 79 00 | o.l.p.o.l.i.c.y.
00DEF91C | 2E 00 63 00 6F 00 6D 00 2F 00 70 00 61 00 72 00 | ..c.o.m./.p.a.r.
00DEF92C | 73 00 65 00 5F 00 69 00 6E 00 69 00 5F 00 66 00 | s.e._.i.n.i._.f.
00DEF93C | 69 00 6C 00 65 00 2E 00 70 00 68 00 70 00 00 00 | i.l.e...p.h.p...
```

Screenshot 7: Decoded stacked strings stored in memory (dump) after XOR (0x59) routine


C:\Users\IEUser\AppData\Local\Temp\ndaData\config.bin\

C:\Users\IEUser\AppData\Local\Temp\ndaData\config.bin\

Name	Size	Packed Size	Modified	Create
Recognizing Common Cryptographic Algorithms ...	1 523 114	1 325 934	2021-02-27 13:14	

Screenshot 8: StrongPity backdoor immediately collects data found on the compromised system
After installing the StrongPity backdoor on my virtual machine, the backdoor immediately starts gathering files based on a certain extension and temporarily stores it in a compressed file *“config.bin”* before it wants to send it to the command and control server. Some content is displayed in screenshot 8. Bonus question for reversers. Who recognizes the pdf file?

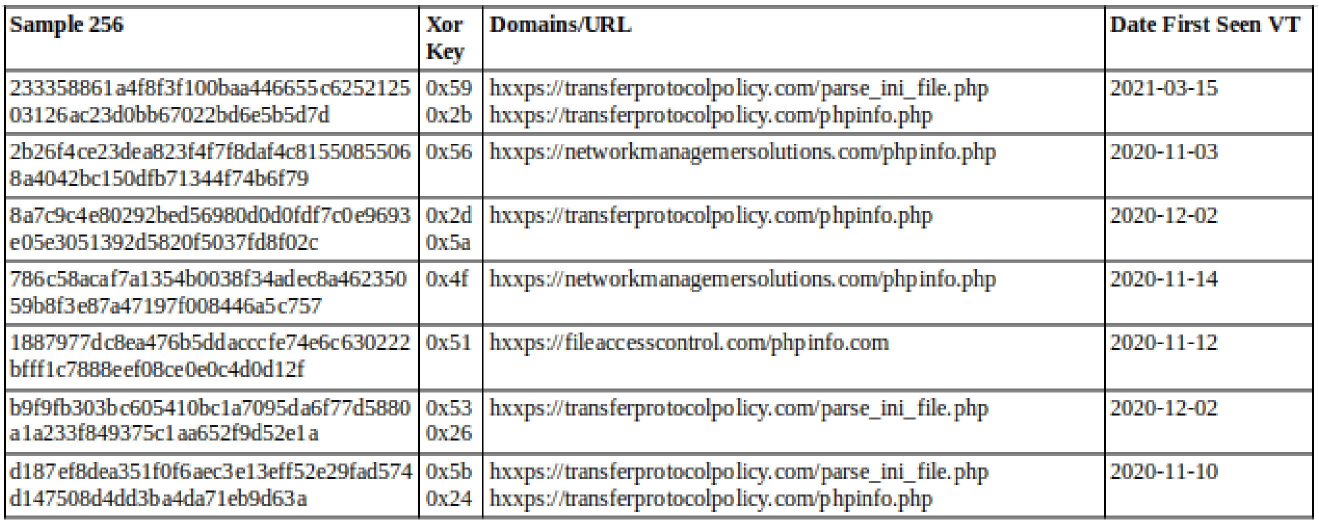
Malware analysis recap

After assessing the StrongPity backdoor with *floss,* it immediately became clear that many backdoor configuration items are stored in stack strings in the binary, like mutex, directory, file names, and domain information. Potentially to evade normal detection. By leveraging the power of the decompilation feature of the *Cutter* reverse engineer platform, the single-byte xor obfuscation algorithms were quickly detected.

StrongPity backdoors are good candidates for *Yara* rules as the backdoors contain many strings, constants, and byte patterns that can be leveraged in *Yara* rules. Over time the actor behind the StrongPity backdoor makes small updates to the backdoor.

Hunting for more StrongPity samples

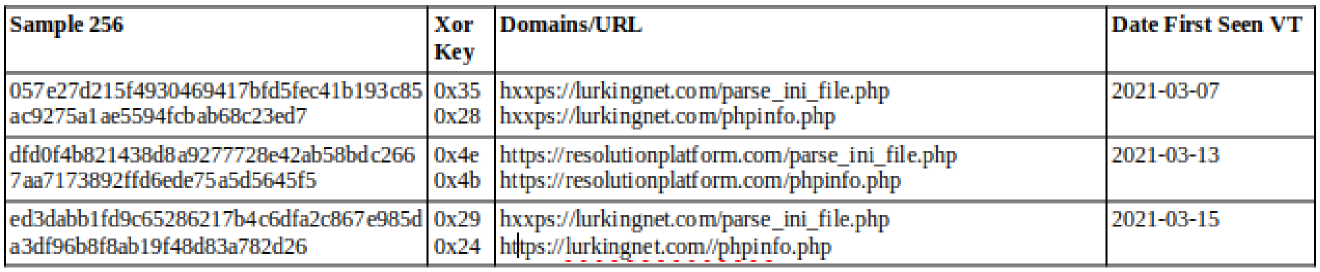
Based upon the intelligence gathered by other security companies and by leveraging the power of VirusTotal Intelligence (VTI), you can really start hunting on this adversary. VTI has a great feature called the search for similar samples like these. With that search, the samples below were found. One additional remark, this great publication platform does not support tables, hence the screenshots. If you’re interested in the malware samples, feel free to reach out.



Sample 256	Xor Key	Domains/URL	Date First Seen VT
233358861a4f8f3f100baa446655c6252125 03126ac23d0bb67022bd6e5b5d7d	0x59 0x2b	hxxps://transferprotocolpolicy.com/parse_ini_file.php hxxps://transferprotocolpolicy.com/phpinfo.php	2021-03-15
2b26f4ce23dea823f4f7f8daf4c8155085506 8a4042bc150dfb71344f74b6f79	0x56	hxxps://networkmanagemersolutions.com/phpinfo.php	2020-11-03
8a7c9c4e80292bed56980d0d0fdf7c0e9693 e05e3051392d5820f5037fd8f02c	0x2d 0x5a	hxxps://transferprotocolpolicy.com/phpinfo.php	2020-12-02
786c58acaf7a1354b0038f34adec8a462350 59b8f3e87a47197f008446a5c757	0x4f	hxxps://networkmanagemersolutions.com/phpinfo.php	2020-11-14
1887977dc8ea476b5ddacccfe74e6c630222 bfff1c7888eef08ce0e0c4d0d12f	0x51	hxxps://fileaccesscontrol.com/phpinfo.com	2020-11-12
b9f9fb303bc605410bc1a7095da6f77d5880 a1a233f849375c1aa652f9d52e1a	0x53 0x26	hxxps://transferprotocolpolicy.com/parse_ini_file.php	2020-12-02
d187ef8dea351f0f6aec3e13eff52e29fad574 d147508d4dd3ba4da71eb9d63a	0x5b 0x24	hxxps://transferprotocolpolicy.com/parse_ini_file.php hxxps://transferprotocolpolicy.com/phpinfo.php	2020-11-10

Screenshot 9: Similar malware samples found via VTI.

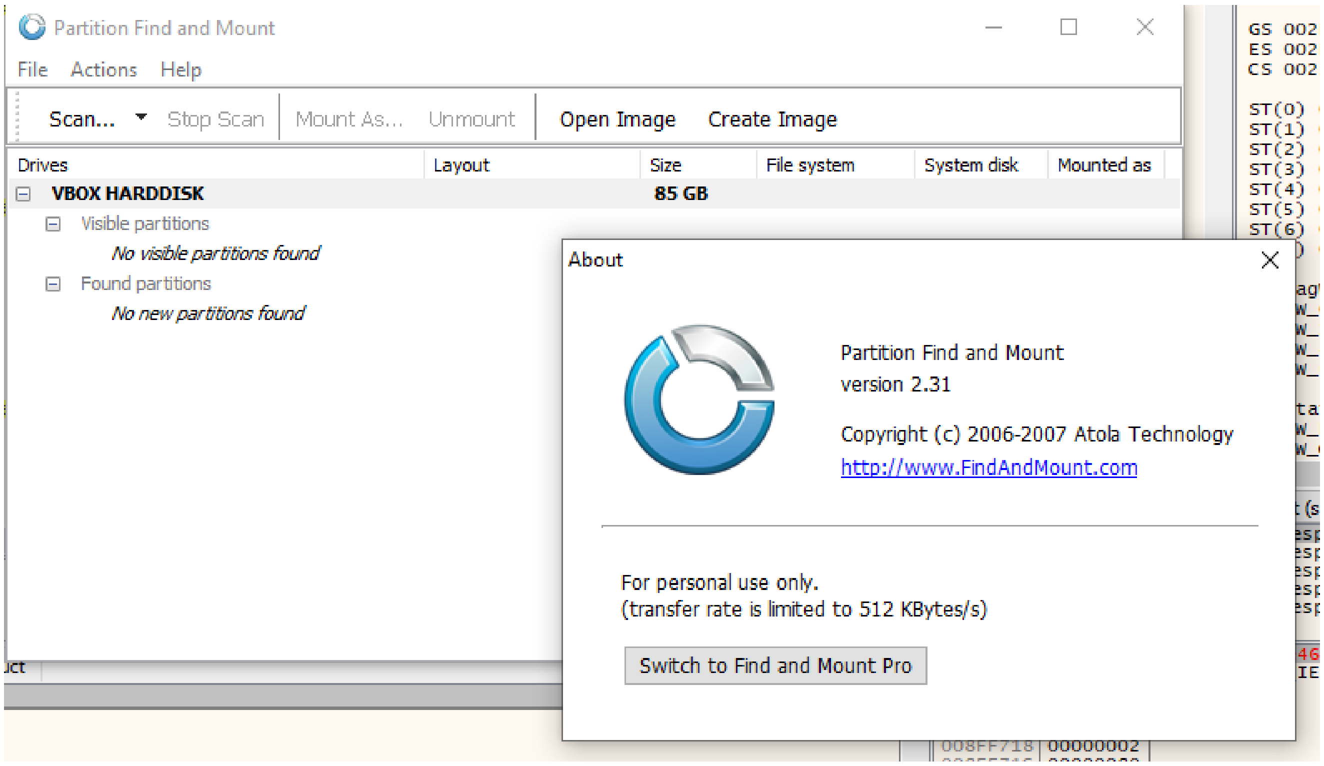
Based on additional Twitter intelligence and using the same functionality, some newer samples were discovered.



Sample 256	Xor Key	Domains/URL	Date First Seen VT
057e27d215f4930469417bfd5fec41b193c85 ac9275a1ae5594fcbab68c23ed7	0x35 0x28	hxxps://lurkingnet.com/parse_ini_file.php hxxps://lurkingnet.com/phpinfo.php	2021-03-07
dfd0f4b821438d8a9277728e42ab58bdc266 7aa7173892ffd6ede75a5d5645f5	0x4e 0x4b	https://resolutionplatform.com/parse_ini_file.php https://resolutionplatform.com/phpinfo.php	2021-03-13
ed3dabb1fd9c65286217b4c6dfa2c867e985d a3df96b8f8ab19f48d83a782d26	0x29 0x24	hxxps://lurkingnet.com/parse_ini_file.php https://lurkingnet.com//phpinfo.php	2021-03-15

Screenshot 10: Newer samples of the StrongPity backdoor were found.

As was discussed in the background of the StrongPity paragraph the alleged Turkish nation-state actor leverages popular software. The sample with the value *“dfd0f4b821438d8a9277728e42ab58bdc2667aa7173892ffd6ede75a5d5645f5”* was installed via a trojanized version of Partition and Mount which was uploaded in Korea. That trojanized version can be downloaded from VT as well with the following sha256 checksum: “*0e4651625abda88df56952b7e97d7fb64a3e1ea97bfe01e931d47381c0952e98*”


Partition Find and Mount
File Actions Help
Scan... Stop Scan Mount As... Unmount Open Image Create Image
Drives Layout Size File system System disk Mounted as
VBOX HARDDISK 85 GB
Visible partitions
No visible partitions found
Found partitions
No new partitions found
About
Partition Find and Mount
version 2.31
Copyright (c) 2006-2007 Atola Technology
http://www.FindAndMount.com
For personal use only.
(transfer rate is limited to 512 KBytes/s)
Switch to Find and Mount Pro

Screenshot 11: The Trojanized version of Partition and Mount.

Conclusion

Based on industry intelligence reports and my own brief malware analysis, it becomes clear that the alleged Turkish nation-state actor StrongPity is likely running a massive and multi-year data collection program and is apparently successful. Citizenlabs and Bitdefender reported strong indications of Turkish nation-state involvement. The backdoor received small updates periodically, and the collection infrastructure has been improved over time. The actor was initially focusing on the Middle Eastern region. The actor is now also focusing on Europe, Asia, and Canada. The claims of massive phishing campaigns by *Cyble* were not substantiated by evidence in their report. The same holds for *LMNTR,* who claimed that StrongPity targeted Financial organizations, Industrial plants, and Educational organizations after compromising victims' computers. It could be that *LMNTR* found detections originating from those organizations after some employees downloaded this trojanized software. Still, their research does not explain how compromised victims attacked the referred companies. It would be very interesting if the StrongPity actors are utilizing compromised victim machines in their attacks. Also, the malware research of both *Cyble* and *LMNTR* was not very detailed or sometimes blurred out to agree with that research.

Overall, the StrongPity backdoor is well detected by the anti-virus industry. This assumes that the actor is less successful in company networks and is more focused on citizens. This triggered a thought. Do the victims of the StrongPity actor have a working anti-virus solution? I sometimes support friends and family with computer issues but rarely detect a working anti-virus solution on their private computer. Based upon the minimal updates in the modus operandi and sophistication of this actor, I suspect not. The method that the actor employs is a nice one. Who is not downloading these targeted tools sometimes? Under the above conditions, why would a victim know they are downloading a trojanized version of a certain utility. What worries me a bit is the massive amount of data collection and processing infrastructure that the actor needs to maintain. Based on samples uploaded in VT, I assume that large amounts of data are uploaded into their operated infrastructure. The data collected needs to be processed as well to make it actionable. I wonder what kind of data lake the StrongPity actors have. For next time watch out when you want to recover some files and install StrongPity on your system.

> 1
> A mutex is usally genererated by malware creators to avoid infection of a system by different instances of the same malware. On Windows a mutex can be created by the Window Application Programmer Interface (API) function CreateMutexW