Icon

Description automatically generated

Practical Malware Analysis & Triage

Malware Analysis Report

Unknown\_RE1012018.zip

Feb 2024 | Amna Jasser | v1.0

# Table of Contents

[Table of Contents 2](#_Toc159787109)

[Executive Summary 3](#_Toc159787110)

[High-Level Technical Summary 4](#_Toc159787111)

[Malware Composition 6](#_Toc159787112)

[Stage2.exe 6](#_Toc159787113)

[Basic Static Analysis 7](#_Toc159787114)

[Basic Dynamic Analysis 10](#_Toc159787115)

[Advanced Static Analysis 12](#_Toc159787116)

[Advanced Dynamic Analysis 19](#_Toc159787117)

[Indicators of Compromise 22](#_Toc159787118)

[Network Indicators 22](#_Toc159787119)

[Host-based Indicators 23](#_Toc159787120)

[Appendices 24](#_Toc159787121)

[A. Yara Rules 24](#_Toc159787122)

[B. Callback URLs 24](#_Toc159787123)

# Executive Summary

|  |  |
| --- | --- |
| SHA256 hash | EC905CB2CB8E9F74790ADDE2C138807F3A6CDBECD5735FA5035B547280D7DB79 |

The malware `unknown\_re1012018` is a PE (Portable Executable) file that intricately conceals another PE within its code. This embedded PE, identified as "stage2.exe," is systematically extracted, uploaded into the system's memory, and unpacked in the Users' AppData Roaming directory. Following this process, the unpacked resource is executed, revealing a multifaceted approach to its malicious operations. The subsequent steps involve communication with the server definitely-not-evil.com, where a file is downloaded using a GET request from the specified path "/l00k\_wh4t\_y0u\_m4d3\_m3\_d0." The content of this downloaded file is then compared to the string "LMAO." Upon a successful match, a second message box is displayed, signifying the completion of the second stage and potentially providing a flag or pertinent information about the malware's activities. This layered execution strategy demonstrates the malware's sophistication in evading detection and executing malicious actions.

YARA signature rules are attached in Appendix A. Malware sample and hashes have been submitted to VirusTotal for further examination.

# High-Level Technical Summary

Unknown\_RE1012018.exe is a PE file that unpacked to another PE file which is the next stage.

Step 1: Search for a resource “stage2.exe” within the code.

Step 2: Upload the resource to the memory.

Step 3: Unpack the resource to the Users' AppData Roaming directory.

Step 4: Execute the unpacked resource.

Step 5: A message box appears.

Step 6: stage2.exe executes and connect to definitely-not-evil.com.

Step 7: Download a file from /l00k\_wh4t\_y0u\_m4d3\_m3\_d0 using a GET request.

Step 8: Compare the result of the download to the string "LMAO."

Step 9: After matching the word, display a second message box.

# Malware Composition

**Unknown\_RE1012018.exe** consists of the following components:

|  |  |
| --- | --- |
| File Name | SHA256 Hash |
| Unknown\_RE1012018 | 92730427321A1C4CCFC0D0580834DAEF98121EFA9BB  8963DA332BFD6CF1FDA8A |
| stage2.exe | 3EDA6E2DAE6FA86245A688EB24E0A29BF206242560C71C9D7726E5DE02D4538A |

Unknown\_RE1012018

The initial PE file that is packed with another PE stage2.exe, in the code it holds a flag and a message box will show.

Stage2.exe

This file is unpacked from the first file, then it connects to definitely-not-evil.com and request from it / l00k\_wh4t\_y0u\_m4d3\_m3\_d0 and compare it to “lmao” and if it’s the same it will show another message box with a flag.

# Basic Static Analysis

{Screenshots and description about basic static artifacts and methods}

Looking at the **strings** output we see:

|  |
| --- |
| !This program cannot be run in DOS mode. (twice) |
| hflagh{i\_shtay\_hout\_htoo\_hlateh\_goth\_nothhingh\_in\_hmy\_bhrainj} |
| 00000A70 Im totally malware  00000A84 totally not malware |
| Dimmaletyoufinishbut |
| definitely-not-evil.com |
| l00k\_wh4t\_y0u\_m4d3\_m3\_d0 |
| stage2.exe |
| 00000E07 Dimmaletyoufinishbut |
| 00001F10 l00k\_wh4t\_y0u\_m4d3\_m3\_d0 |

Since this !This program cannot be run in DOS mode. Was appeared twice then there is an indication that there is another PE file within the PE file.

**PEStudio:**

Notice the first 2 bytes are **MZ**meaning it's a PE Binary

A screenshot of a computer

Description automatically generated

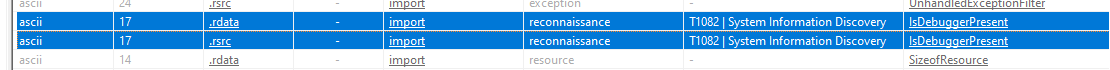
file-type: executable



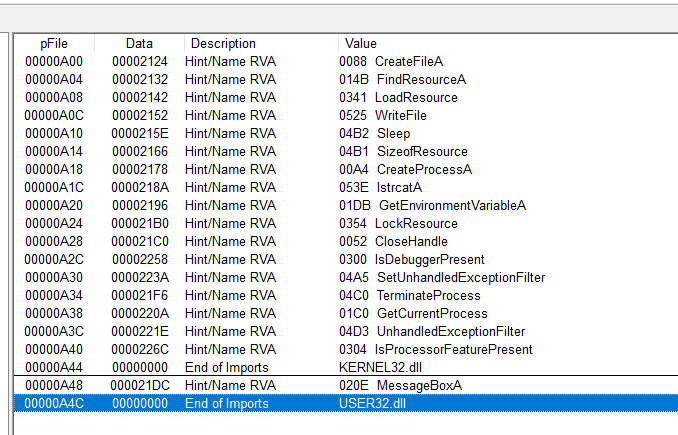


* Notice that is using [**FindResourceA**](https://docs.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-findresourcea) and [**CreateProcessA**](https://docs.microsoft.com/en-us/windows/desktop/api/processthreadsapi/nf-processthreadsapi-createprocessa). This means the executable might be extracting something out of the resource section and run it as a new process.

It is also checking if it is running in a debugger:



And this is the import table:



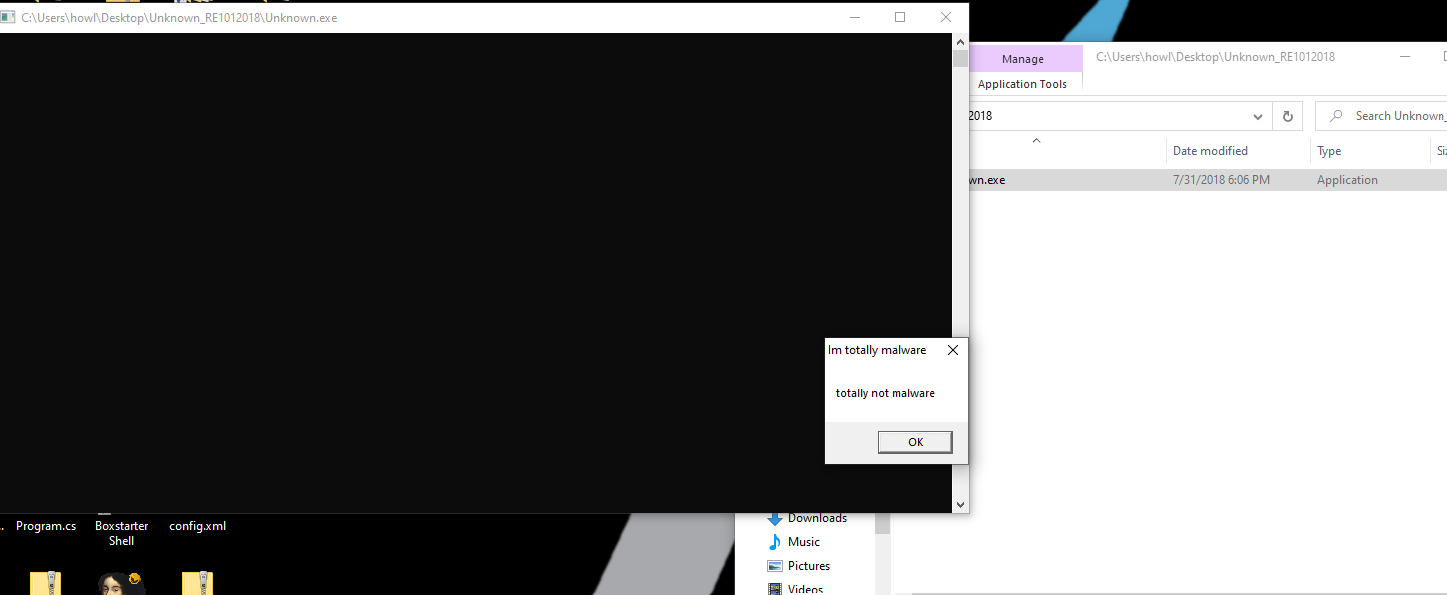


# Basic Dynamic Analysis

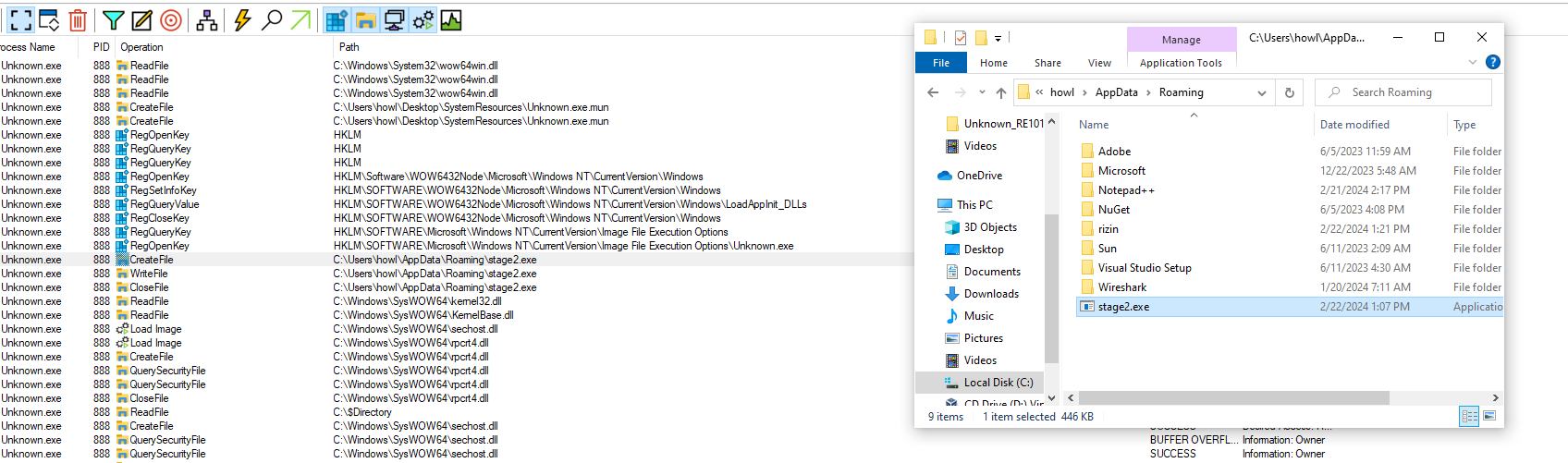
{Screenshots and description about basic dynamic artifacts and methods}

**When running the file without internet connection:**

1. It shows a black command line for a second and a message box with “totally not malware”:

.





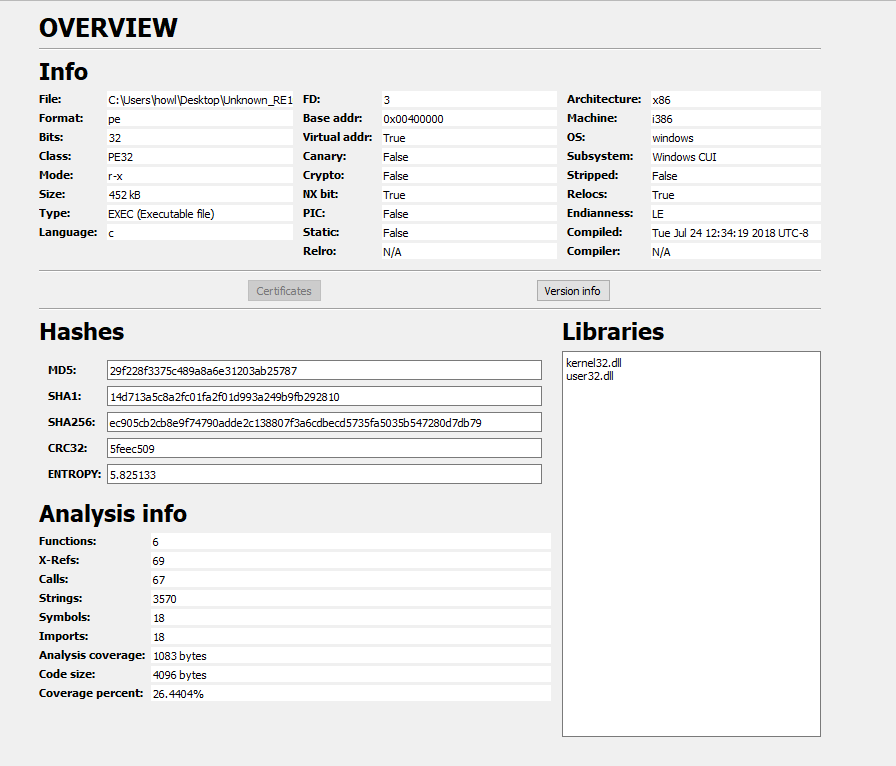
**When there is an internet connection:**

Even with internet it is the same result, So wee need to check with debugger.

# Advanced Static Analysis

{Screenshots and description about findings during advanced static analysis} Could not

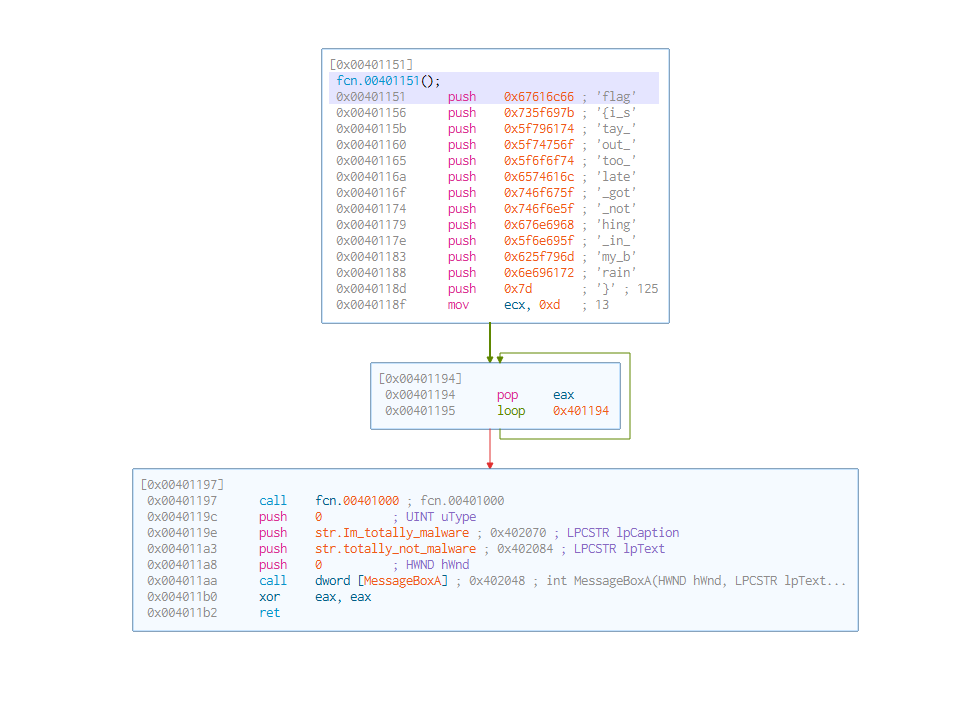
Looking at cutter info about this PE:



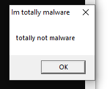
We now know it is built with C and x86 instruction.

Looking at the main function of Unknown\_RE1012018.exe:

Analysing Function **0x00401151:**



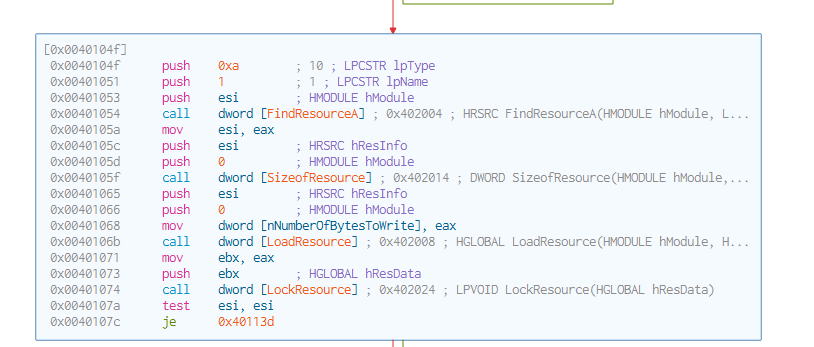
1. In the first we see the flagh{i\_stay\_out\_too\_late\_got\_nothing\_in\_my\_brain}
2. Then it will loop 13 times as loop decrement ecx value.
3. Then it will call function **0x00401000**
4. Call MessageBoxA function:

The result is this  A screen shot of a computer program

Description automatically generated

1. Xor eax, eax: This is a common idiom for zeroing out a register, set the eax register to 0 and then return from a subroutine.

Analyzing function **0x00401000**:



1. FindResourceA
2. SizeOfResource
3. LoadResource
4. LockResource

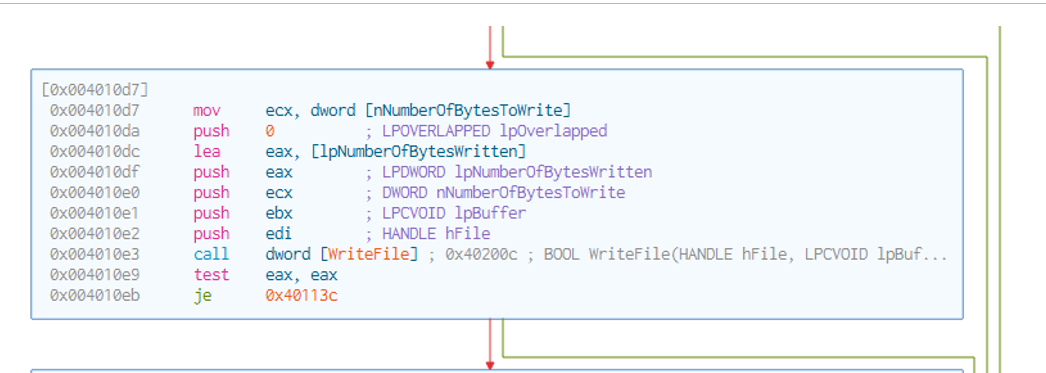
In this code: it Locate a resource within the executable and obtain a handle to the resource information. Then, determine its size and load it into memor(When you call LoadResource, it loads the specified resource (identified by the HRSRC handle) into memory, and it returns a handle (HGLOBAL) to the memory block where the resource is loaded. This handle allows you to access and manipulate the resource's data.).

Finally, obtain a pointer to the resource, allowing access to it and enabling its use.

A screen shot of a computer program

Description automatically generated







1. **GetEnviromentalVariableA**: First it get the environment variable in which the resource would be unpacked to which is in AppData, and we see that there is a buffer:

call edi is telling the processor to jump to the address stored in the edi register, treating it as the starting address of a subroutine. The return address (the address of the instruction following the call edi instruction) is pushed onto the stack, allowing the program to return to that address after the subroutine completes.

1. **WriteFile and CreateFile**:This code is creating a file which is stage2.exe and save it in as seeing from string in **C:/Users/Howl/AppData/Roaming.**

CreateFile is used for creating or opening files, obtaining a handle to them, and setting up the necessary parameters. Once you have a file handle obtained through CreateFile, you can use WriteFile to write data to that file. WriteFile is specifically focused on the act of writing data, while CreateFile is focused on obtaining a handle to a file or creating/opening a file.

Then the result of the creation file is stored in eax then to edi and tested on a jump function.

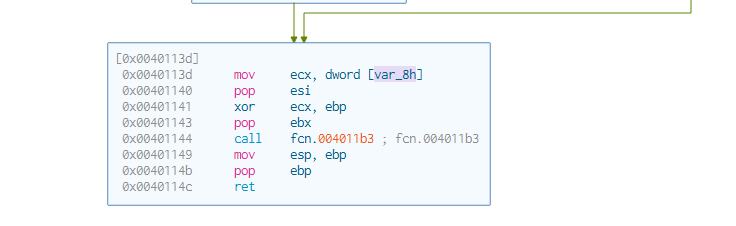
If the ZF is set Jump if equal is taken, and only ZF is set when the result of the writefunction is 0, then ZF set to 1 and then the program is exited by following the condition call and jumped to **0x40113c.**

**If the ZF is not set then the program continuous to execute:**

A screenshot of a computer

Description automatically generated





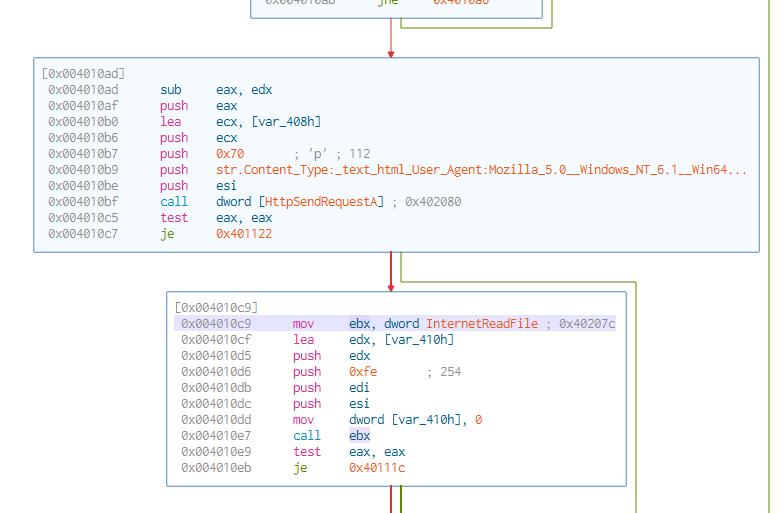


1. It call flirt.memset
2. Create a process
3. Call a function to terminate any running functions.

**Stage2.exe: Function 0x00401000:**

A screenshot of a computer

Description automatically generated



A screen shot of a computer program

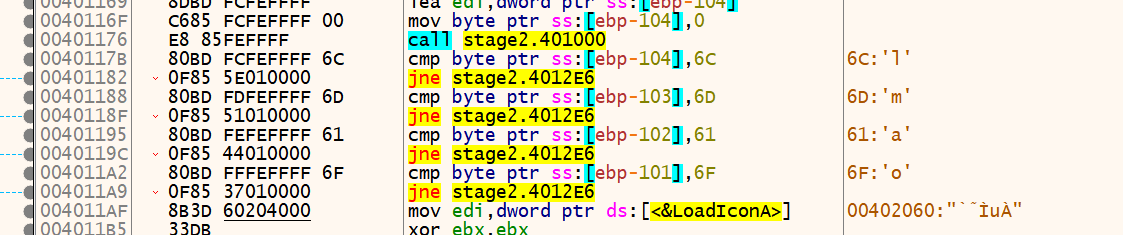
Description automatically generated

In this function its open a connection to definitely\_not\_evil.com and its requesting an html page and then it reads the file and store it in ebx register and at the end it closes the handle. the result of the Read file function is saved into EAX

# Advanced Dynamic Analysis

Running Stage2.exe in a debugger:

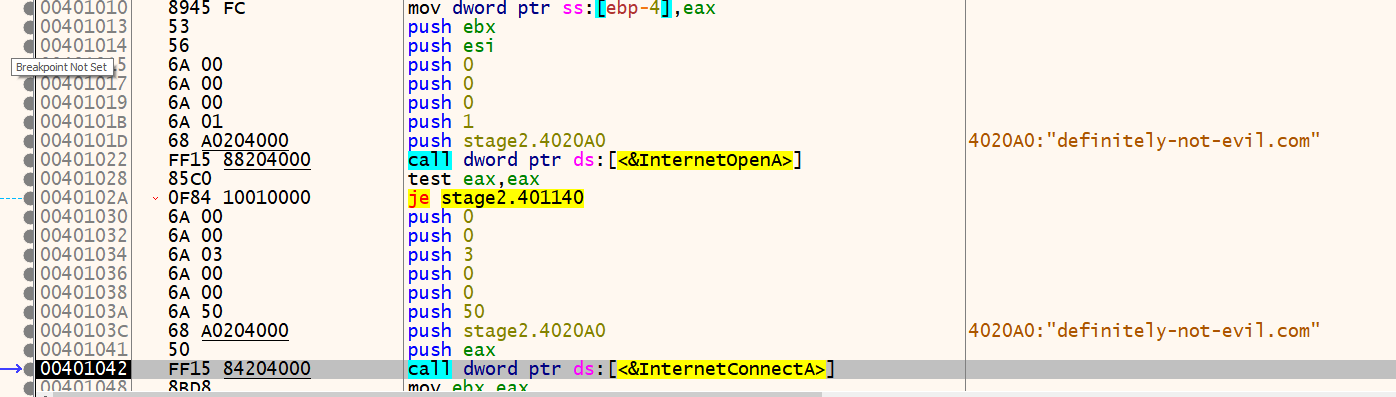
When reaching to the entry point, we saw a call to function 0x401000 and then we saw multiple comparison of the result of that function:



When entering 0x401000 function we saw it open connection and requestion definitely-not-evil.com/l00k\_wh4t\_y0u\_m4d3\_m3\_d0

A screenshot of a computer

Description automatically generated



A screenshot of a computer

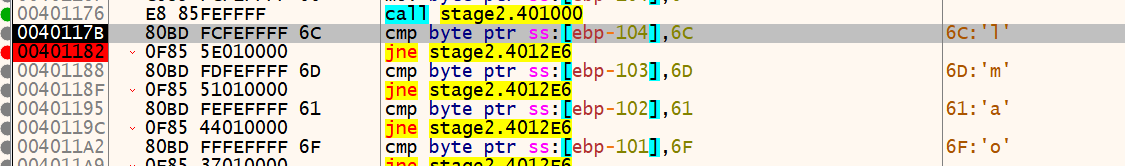
Description automatically generated



A screenshot of a computer

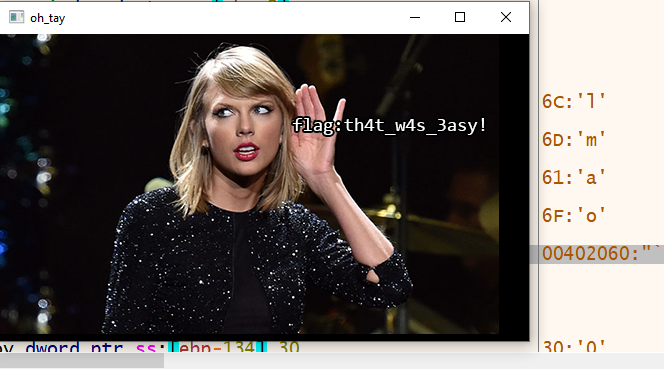
Description automatically generated

Open a connection to Definitely not evit.com then requesting /Look\_what\_you\_made\_me\_do



The **HttpSendRequest** function is typically used after **HttpOpenRequest** to actually send the HTTP request to the server.

Then it compares the result to “lmao”, and if it’s the same then the last message box is showed:

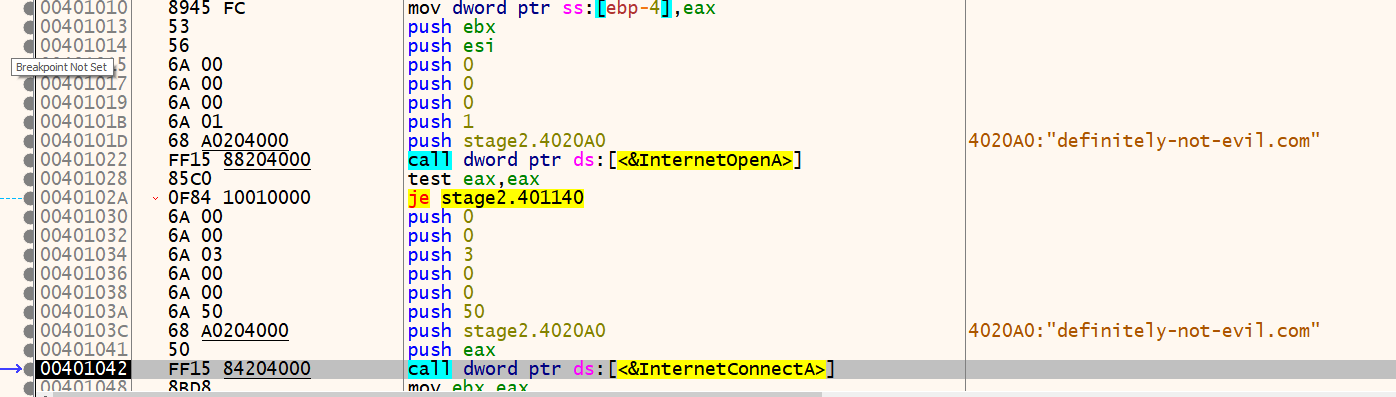


# Indicators of Compromise

The full list of IOCs can be found in the Appendices.

## Network Indicators

{Description of network indicators}



A screenshot of a computer

Description automatically generated



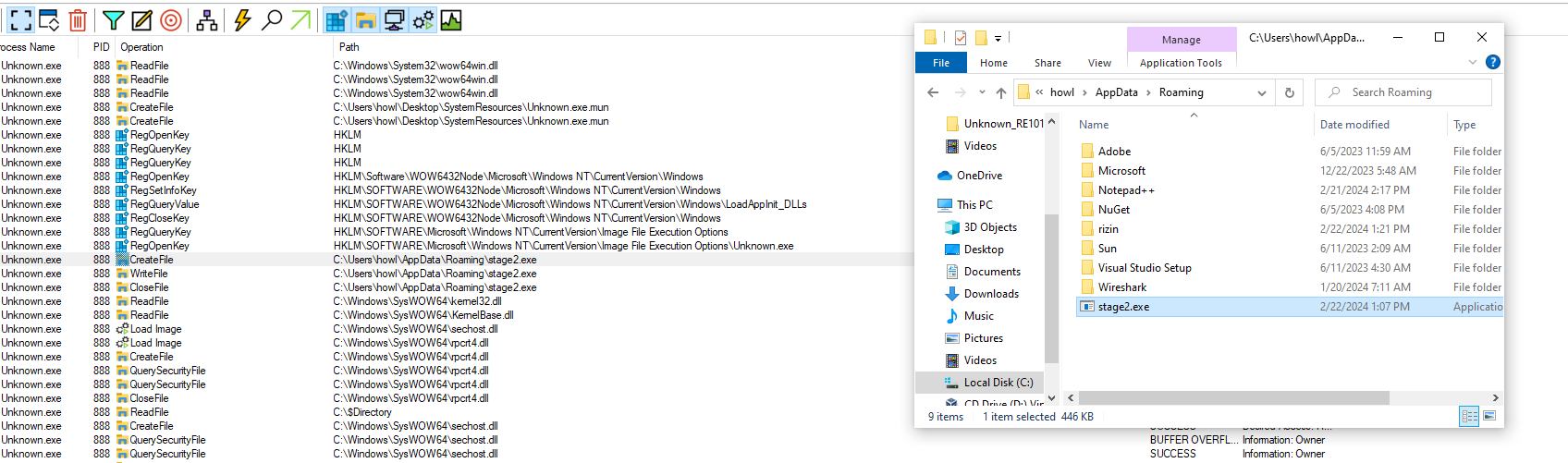
A screenshot of a computer

Description automatically generated

Using InternetConnectA, HttpOpenReqesrtA and HttpSendRequestA to definitely-not-evil.com

## Host-based Indicators

Stage2.exe, this file is executed as second stage:



# Appendices

## Yara Rules

rule YARA\_example {

    meta:

        description = "Unknown\_RE1012018"

        sha256 = "92730427321A1C4CCFC0D0580834DAEF98121EFA9BB8963DA332BFD6CF1FDA8A"

    strings:

        $string1="hflagh{i\_shtay\_hout\_htoo\_hlateh\_goth\_nothhingh\_in\_hmy\_bhrainj}" ascii

        $string2 = "Im totally malware" ascii

        $string3="Dimmaletyoufinishbut" ascii

        $string4="stage2.exe" ascii

        $URL1="definitely-not-evil.com" ascii

        $URL2="l00k\_wh4t\_y0u\_m4d3\_m3\_d0" ascii

        $IS\_PE\_FILE="MZ"ascii

    condition:

        $IS\_PE\_FILE at 0  and

        ($string1 and $string2 and $string3 and $string4 ) or ($URL1 or $URL2)

}

## Callback URLs

|  |  |
| --- | --- |
| Domain | Port |
| definitely-not-evil.com | 80 |