Practical Malware Analysis & Triage

Malware Analysis Report

WannaCry Malware

Dec 2021 | Tuhin Chowdhury | v1.0

d

Table of Contents

Executive Summary

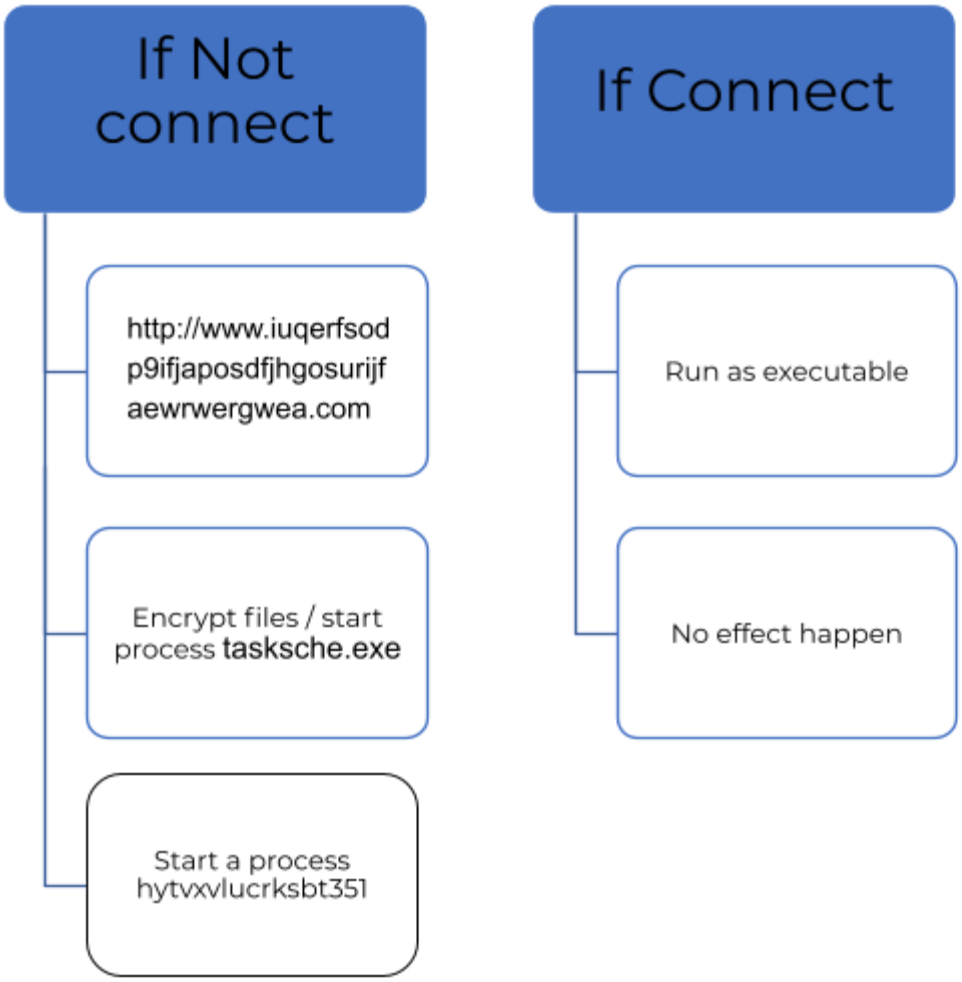
SHA256 hash	24D004A104D4D54034DBCFFC2A4B19A11F39008A575AA614EA04703480B1022C

WannaCry is a cryptor-dropper malware sample first identified on May 17th, 2017. It is a Microsoft Visual C++ 6.0 malware that runs on the x64 Windows operating system. It consists of two payloads that are executed in succession following a successful spear phishing attempt. Symptoms of infection include encrypting every file Appendix B, static warning screen popups on the endpoint, and an executable named “taskse.exe” appearing in the C:\ProgramData\hytvxvlucrksbt351 directory.

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

High-Level Technical Summary

WannaCry consists of two parts: an encrypted stage 0 dropper and an unpacked and decoded stage 2 command execution program. It first attempts to contact its callback URL (hXXtp://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com)



Malware Composition

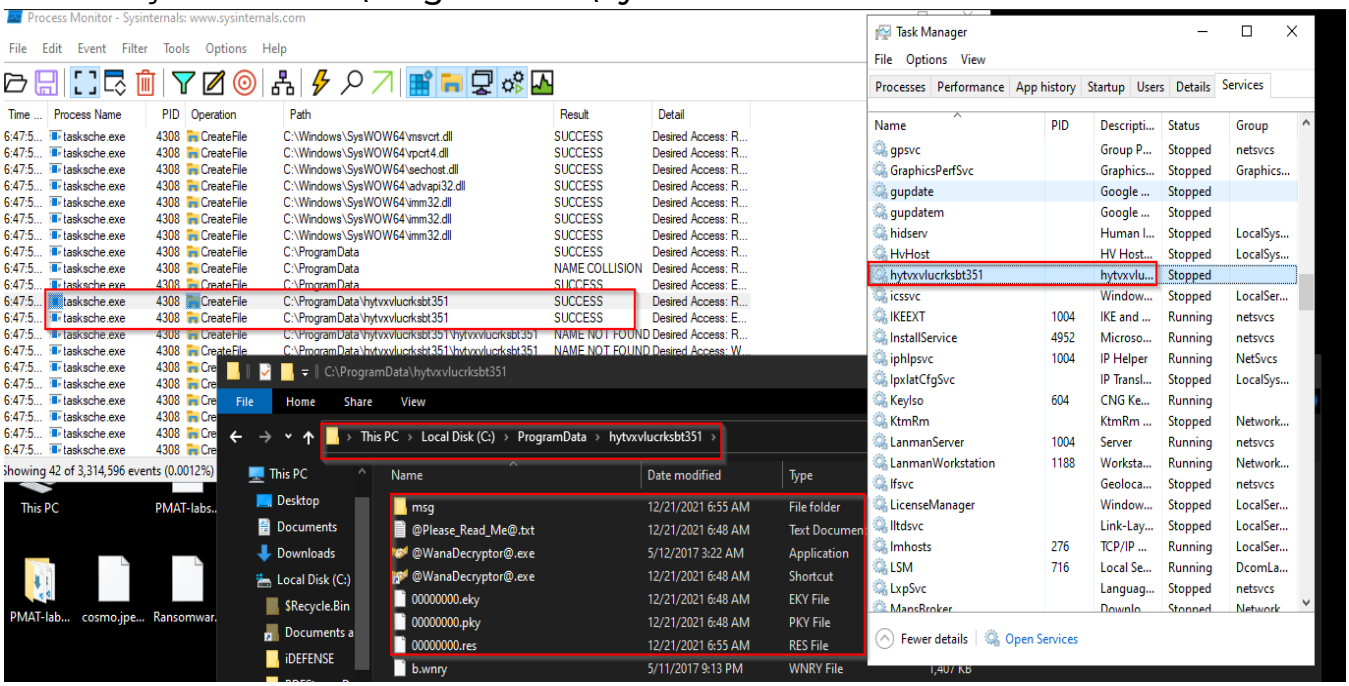
WannaCry consists of the following components:

File Name	SHA256 Hash
WannaCry	24D004A104D4D54034DBCFFC2A4B19A11F39008A575AA614EA04703480B1022C

WannaCry.exe

The initial executable that runs if the program don't make the connection with the callback url

A directory created in C:\ProgramData\hytvxvlucrksbt351



Basic Static Analysis

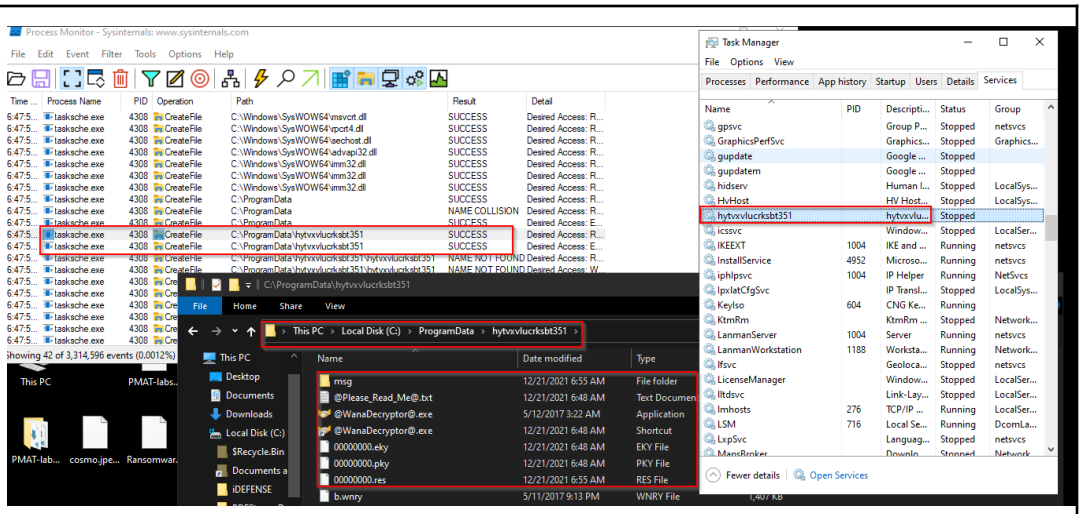
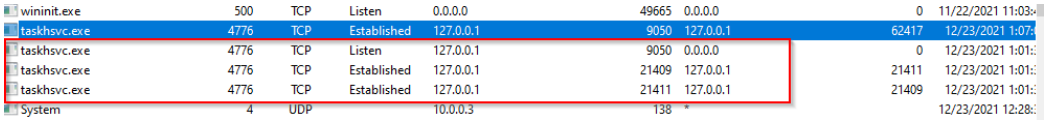
{Screenshots and description about basic static artifacts and methods}

String / Floss Output	http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com KERNEL32.dll mscoree.dl MM/dd/yy CryptEncrypt CryptDecrypt CryptDestroyKey CryptGenRandom

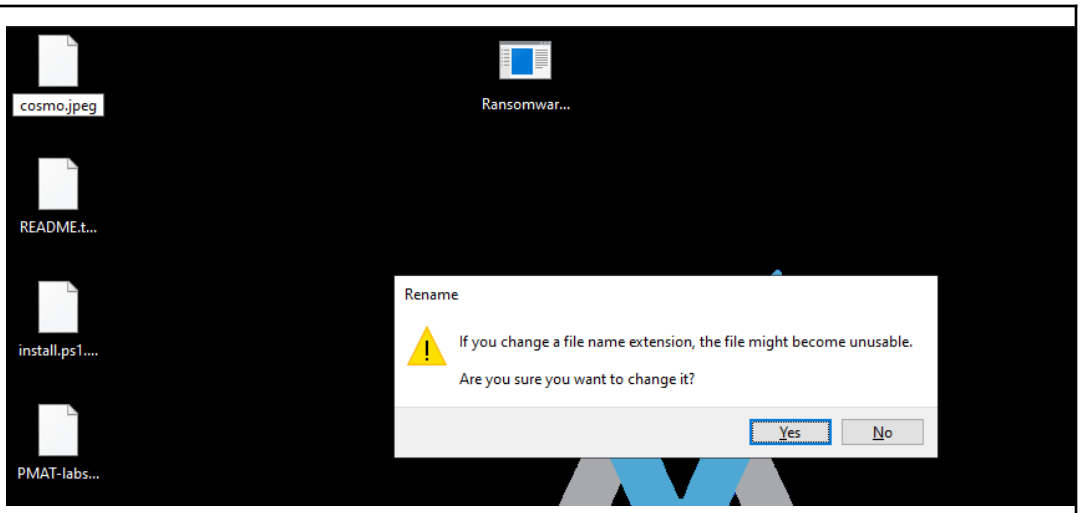
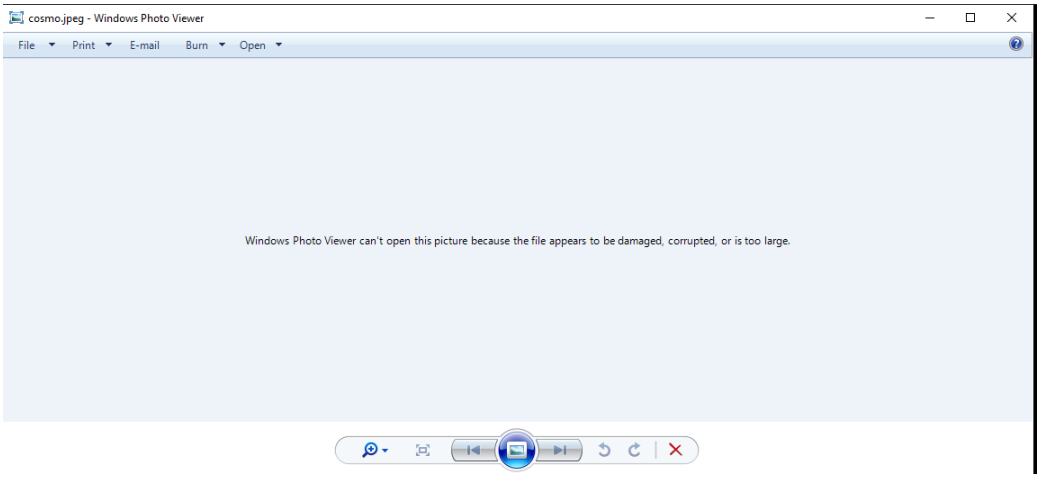
We can find the url and some windows API related to cryptography, So as far our basic analysis we conclude that some function of the malware work with crypto. So it can be ransomware.

Basic Dynamic Analysis

{Screenshots and description about basic dynamic artifacts and methods}

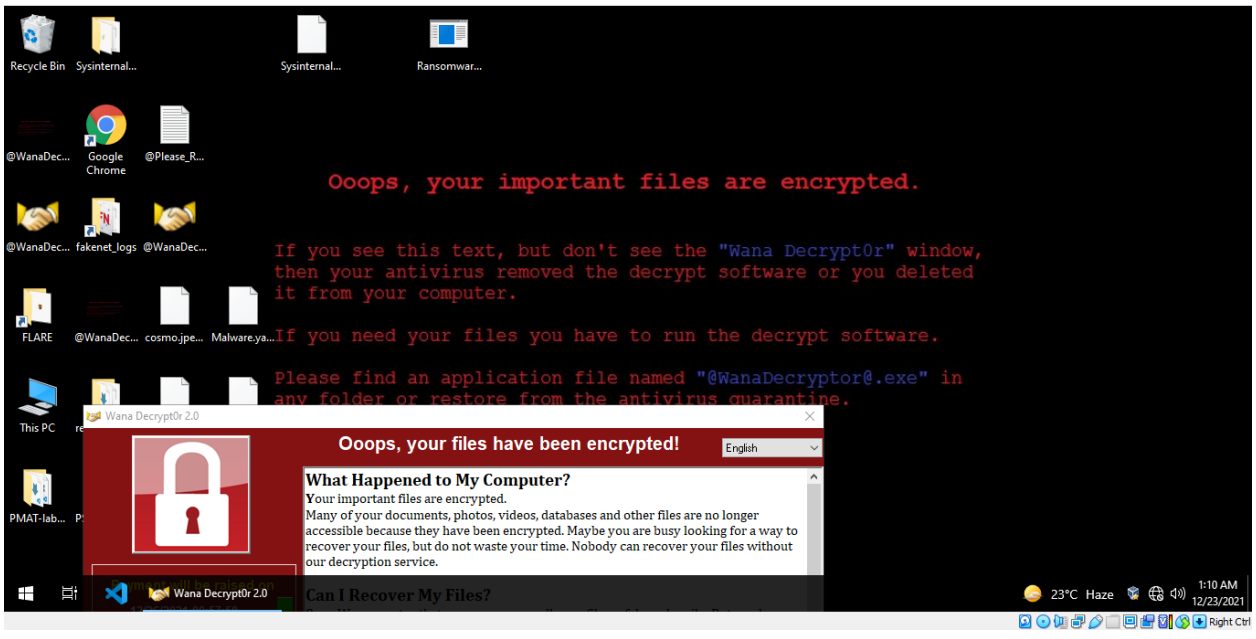
Create file (It create a directory in %ProgramData% name hytvxvlucrksbt351 and file taskse.exe)	 BD-Figure-1
taskhsvc.exe connection status	 BD-Figure-2

d

Try to open after deleting extension	 BD-Figure-3  BD-Figure-4

In the BD-Figure-1 we can see a directory and a file created in %ProgramData% folder and start a process. A Program name taskhsvc.exe starts listening on 9050 port (BD-Figure-2). When we try to open any file after removing the extension (BD-Figure-3), We got an output like BD-Figure-4

d

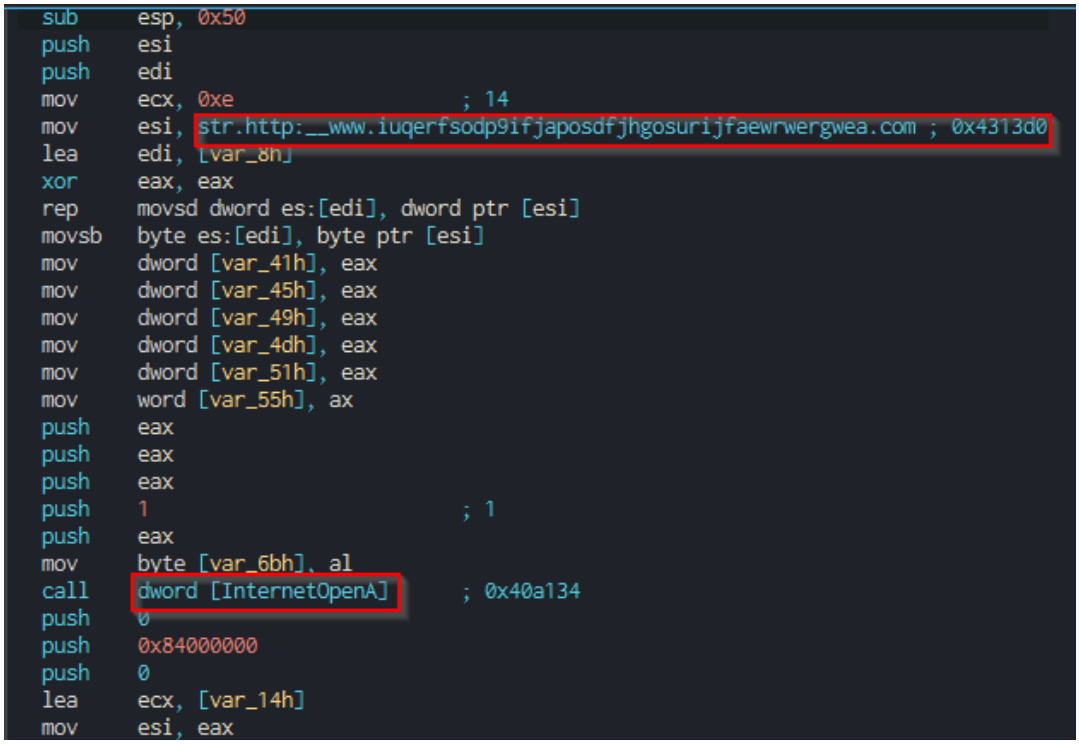


After execution the program the wallpaper changed and a decrypter program start with a message and also files got encrypted

WannaCry Malware
Dec 23
v1.0

Advanced Static Analysis

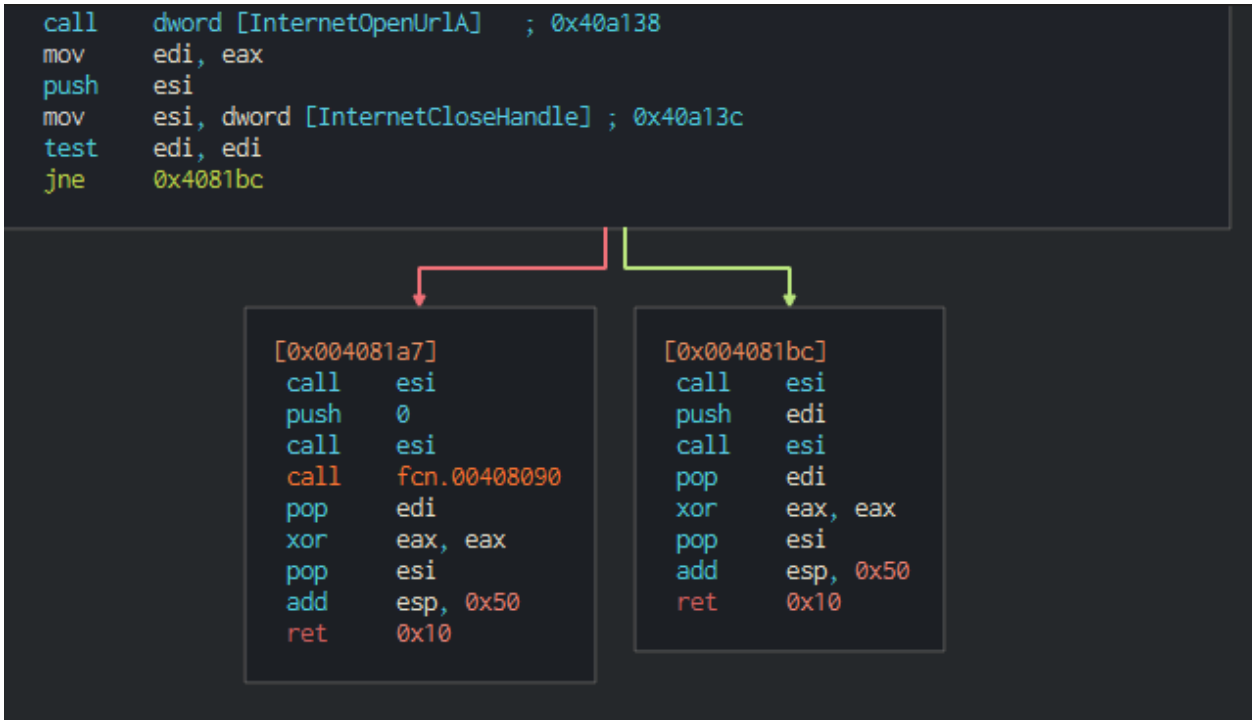
{Screenshots and description about findings during advanced static analysis}



AS-Figure-1

In the AS-Figure-1 Cutter output we can see the call back url and first InternetOpenA windows API call.

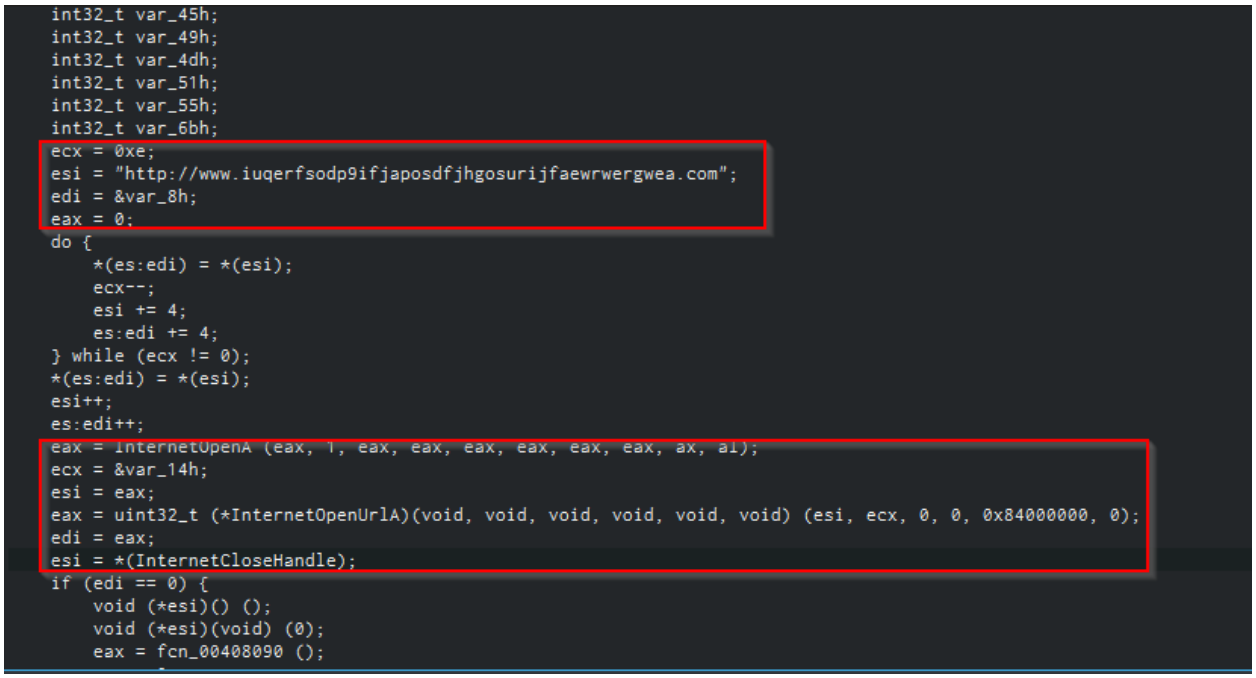
d



AS-Figure-2

In the AS-figure-2 Cutter output we can see the conditional logic if code execution as per the screenshot in both condition the code approximately
run same but in left hand indicator a special function call in 00408090 location. So we can determine this the separation part of the logic flow.

d



```
int32_t var_45h;
int32_t var_49h;
int32_t var_4dh;
int32_t var_51h;
int32_t var_55h;
int32_t var_6bh;
ecx = 0xe;
esi = "http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com";
edi = &var_8h;
eax = 0;
do {
    *(es:edi) = *(esi);
    ecx--;
    esi += 4;
    es:edi += 4;
} while (ecx != 0);
*(es:edi) = *(esi);
esi++;
es:edi++;
eax = InternetOpenA (eax, 1, eax, eax, eax, eax, eax, eax, ax, al);
ecx = &var_14h;
esi = eax;
eax = uint32_t (*InternetOpenUrlA)(void, void, void, void, void, void) (esi, ecx, 0, 0, 0x84000000, 0);
edi = eax;
esi = *(InternetCloseHandle);
if (edi == 0) {
    void (*esi)() ();
    void (*esi)(void) (0);
    eax = fcn_00408090 ();
```

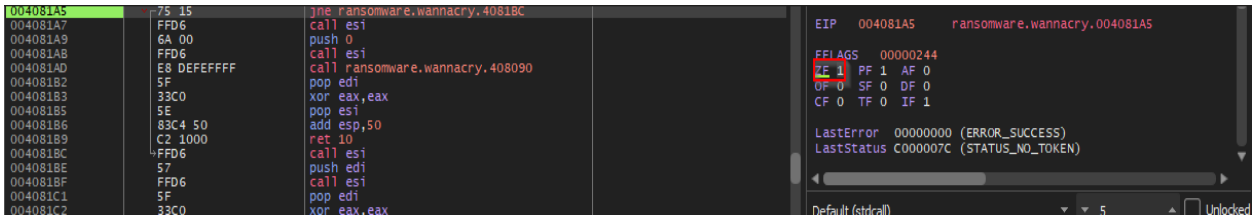
AS-Figure-3

This is the decompile main function code. Here we can actually where the url store and when the API call happens.

d

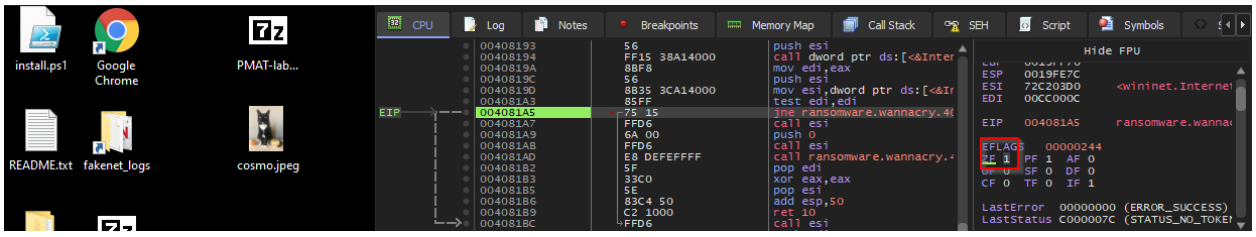
Advanced Dynamic Analysis

{Screenshots and description about advanced dynamic artifacts and methods}



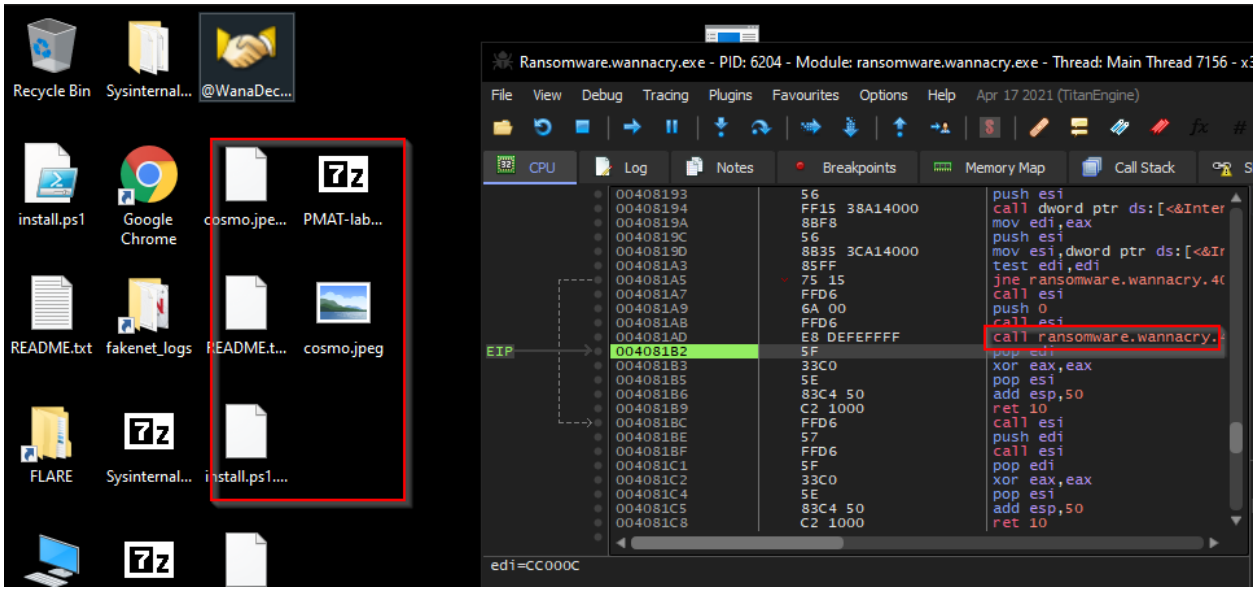
AD-Figure-1

In the x32 debugger we change the flag 0 to 1 manually to see the result. In that condition the program connect with the callback url in 1 flag but as we changed this into 0 so the program code manipulated and don't get the connection.



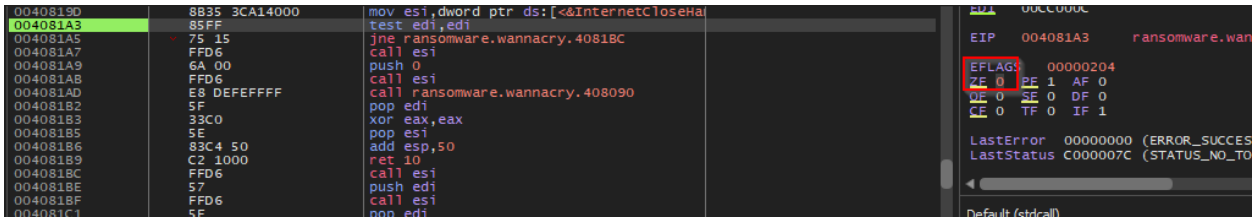
Before taking the call (Files are not encrypted)

d



After taking the call (Files got encrypted)

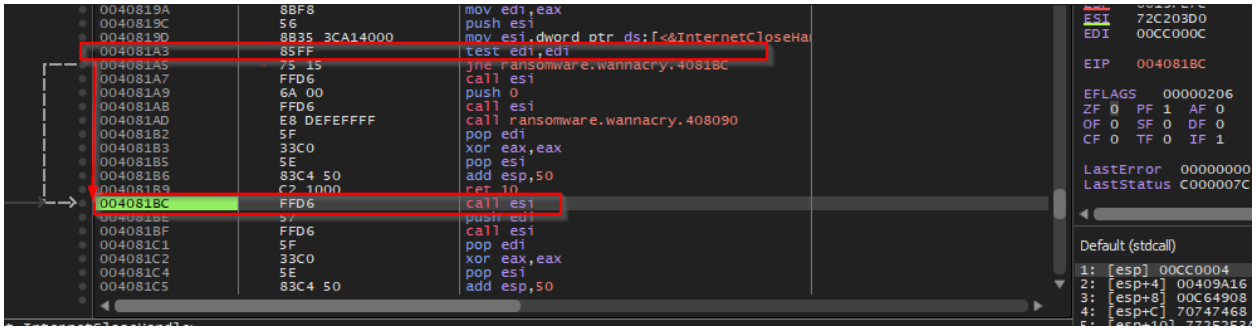
Before taking the call:



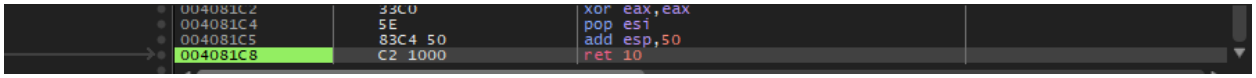
In this condition we let the flag 0

After Taking the call:

d



Return:



In the above scenario as you can see if the program successfully connects with the callback url then it jumps from 004081A3 location to 004081BC. But if it doesn't connect it go into 004081A3 to and continues to 004081A5 to 004081B9. This is the main execution portion.

d

Indicators of Compromise

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

Network Indicators

{Description of network indicators}

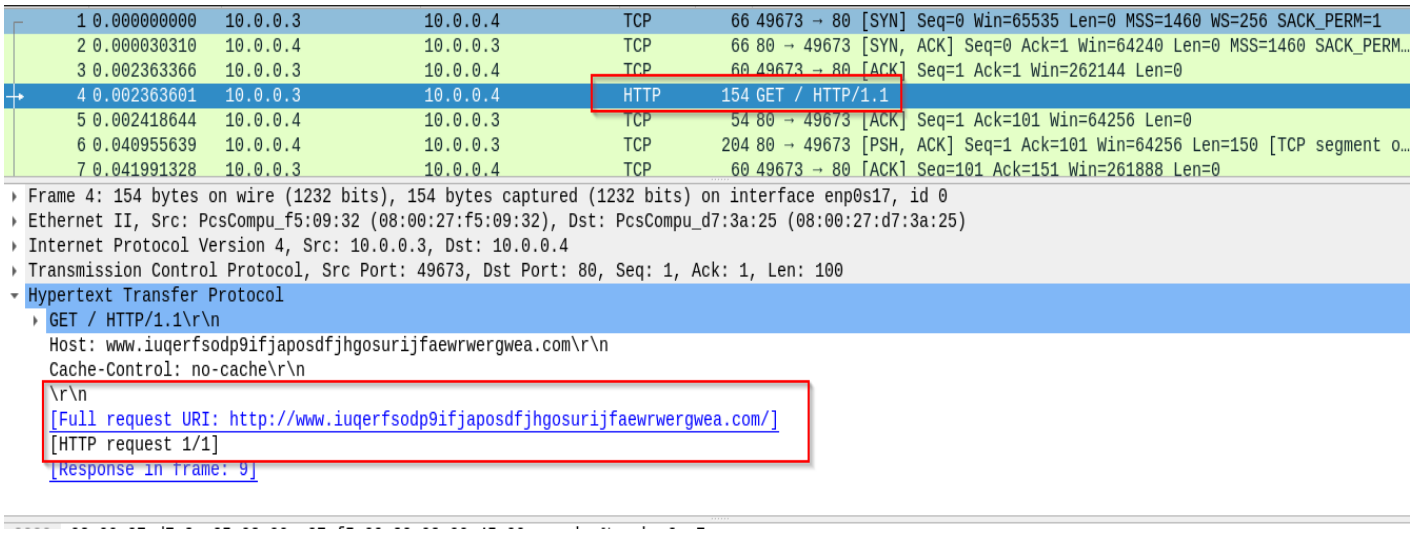


Fig 3: WireShark Packet Capture of initial beacon check-in

Rules & Signatures

A full set of YARA rules is included in Appendix A.

{Information on specific signatures, i.e. strings, URLs, etc}

d

Appendices

A. Yara Rules

```
rule Yara_Example {

    meta:
        last_updated = "2021-12-23"
        author = "Tuhin"
        description = "rule for wanna cry"

    strings:
        // simple rules
        $string1 = "Ooops, your files have been encrypted!” wide ascii nocase
ascii
        $string2 = "Wanna Decryptor" wide ascii nocase
        $PE_magic_byte = "MZ"
        $sus_hex_string = { FF E4 ?? 00 FF }

    condition:
        // Basic condition
        $PE_magic_byte at 0 and
        ($string1 and $string2) or

        $sus_hex_string
}
```

B. Callback URLs

Domain	Port
http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com	80

d

C. Decompiled Code Snippets

```
int32_t var_45h;
int32_t var_49h;
int32_t var_4dh;
int32_t var_51h;
int32_t var_55h;
int32_t var_6bh;
ecx = 0xe;
esi = "http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com";
edi = &var_8h;
eax = 0;
do {
    *(es:edi) = *(esi);
    ecx--;
    esi += 4;
    es:edi += 4;
} while (ecx != 0);
*(es:edi) = *(esi);
esi++;
es:edi++;
eax = InternetOpenA (eax, 1, eax, eax, eax, eax, eax, eax, ax, al);
ecx = &var_14h;
esi = eax;
eax = uint32_t (*InternetOpenUrlA)(void, void, void, void, void, void) (esi, ecx, 0, 0, 0x84000000, 0);
edi = eax;
esi = *(InternetCloseHandle);
if (edi == 0) {
    void (*esi)() ();
    void (*esi)(void) (0);
    eax = fcn_00408090 ();
```

Fig 5: Process Injection Routine in Cutter

Program Execution flow

- If url connected successfully
    - Malware not executed
    - The host seems ok with no errors.
- If url cannot connect
    - Program execute
    - Encrypt every file
    - Start a process called tasksche.exe
    - Create a file in Program Data
    - Change the background