SPRING 2023 CS390R Final Project - Checkpoint #1

Table of Contents

- General Overview
- Set-Up Debug Environment
- Map Out Target Code-Base
- Plans for Rest of Project
- Resources and Research

General Overview

Instruction: *Provide a short general overview of your target. What are its intended use-cases, what type of features does it support, etc.*

We want to understand how malware works by reversing it. Our plan involves using the Practical Malware Analysis textbook and examining a sample of malware. Our initial target is WannaCry, a well-documented ransomware cryptoworm that targets computers, encrypts their files, and asks for Bitcoin as remittence for decryption. We aim to understand the structure of malware by analyzing the decompiled binary in Ghidra. Although the source code of WannaCry is available online, we want to gain a comprehensive understanding of it by going through the entire process of reversing and examining the binary.

We know that WannaCry is based off asymmetric and symmetric encryption. The symmetric encryption employs a single key for encryption and decryption (AES encryption system). The asymmetric encryption portion employs two keys: one public, one private. The public key encrypts while the private key decrypts (RSA public-key encryption system). This is how our initial target works:

1. makes symmetric and asymmetric keys (attacker side)
2. encrypts files with asymmetric (attacker side)
3. encrypts asymmetric key with symmetric key (victim side)
4. send private key to attacker then delete (victim side)
5. delete private key (victim side)

Since WannaCry has a built-in kill switch that is (supposedly) easy to switch and requires the Internet to check for the domain name iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com, we want to look at harder samples of malware after this target to look at and reverse.

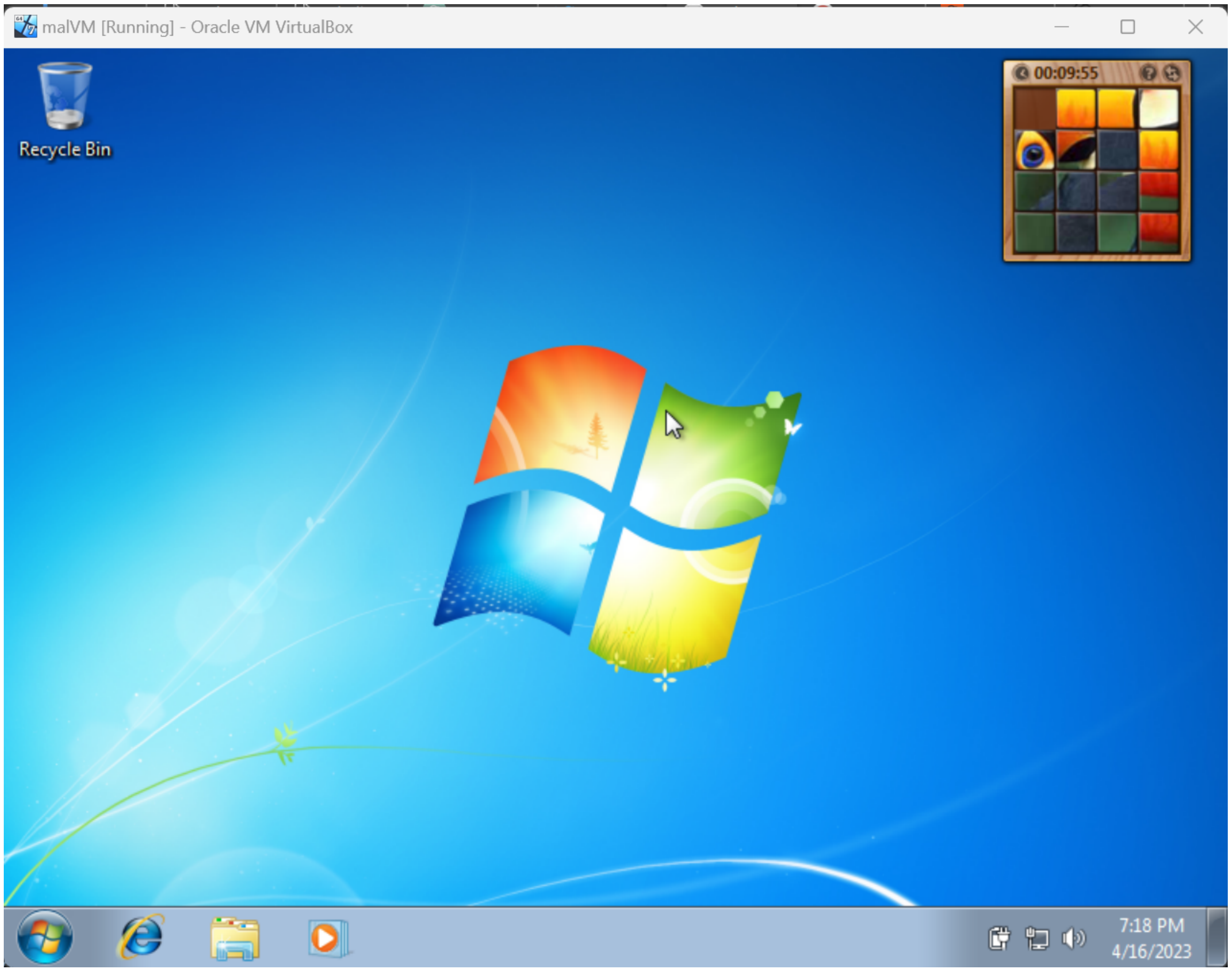
Back to Top

Set-Up Debug Environment

Instruction: *Set up a debug environment for your target. If you are targeting an embedded device this might include setting up emulation, for windows targets you might need a windows debugger, and in some cases you may want to play around with compile flags to set up source code debugging if possible. Do what you can given your target, describe what you did, and upload required files for this to your team's github repository.*

We plan on setting up a virtual machine for everyone to work off of. Since WannaCry depends on a vulnerability found in Windows' Server Message Block (SMB) protocol, we would need to set up a Windows environment to debug. WannaCry also does not work on Windows Vista, 8, 10, or any other modern Windows release, so we found a Windows 7 iso to work off of. We also found FlareVM as a potential tool for setting up the reversing environment on a VM.

After getting the VM set up with Windows 7, we first had to install Firefox since Internet Explorer was too outdated to function. We got Ghidra to statically analyze the virus, then downloaded the WannaCry binary from the Github repository linked later on. After downloading, I disconnected the network connection between the VM and the host while also diabling the auto-connect setting VMware has enabled by default.



Back to Top

Map Out Target Code-Base

Instruction: *Map out the targets code-base. Your target probably has many source directories with various different files. In this part we want you to describe the general layout and purpose of different files, major functions, etc. From this we should geta good idea of where exactly in the target-code we should look to find specific features, code-sections, etc.*

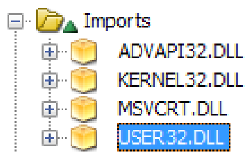
We have a sample of the WannaCry binary here from theZoo Github repository. There are many samples of this live malware on the Internet as well as source code. We are not going to explicitly look at the source code so not to be spoiled by it itself, so we are going to be doing manual reversing on this sample. The password for the zip file is `infected`, as given by the `.pass` file in the directory.

Unpacking the binary reveals a single `.exe` file. We can use Ghidra to look at the decompiled binary.

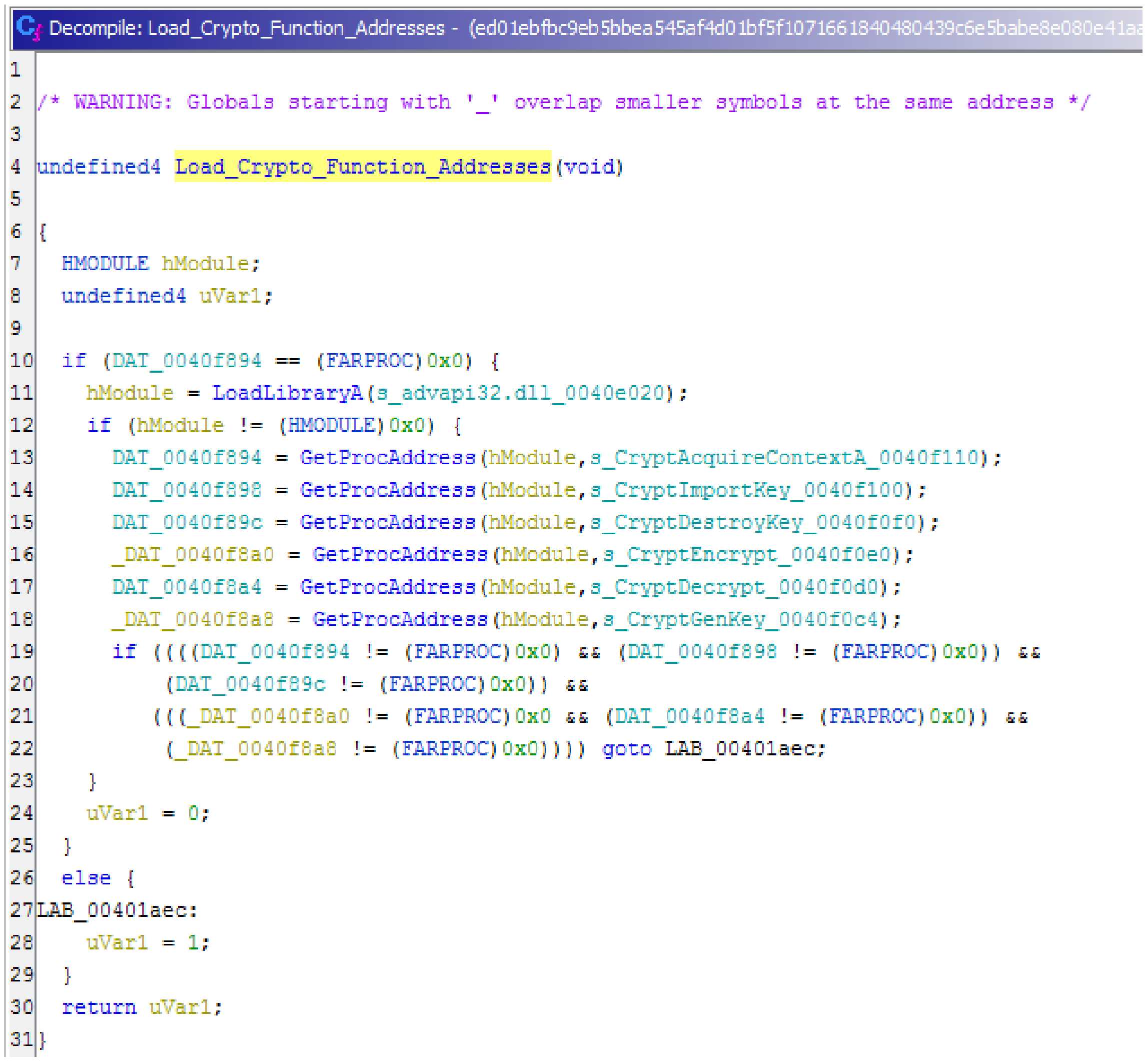
The first thing I noticed was the `.dll` 's included. These help contextualize what the program is capable of doing. It has 4 `.dll` libraries:

- `ADVAPI32.DLL`, which deals with windows registry and security. This includes encryption, which makes sense for a ransomware

- `KERNAL32.DLL` , which handles a variety of things in regards to memory. Reading, writing, copying files. Getting file attributes, getting absolute pathnames, allocating and freeing heap memory, pointers, etc.
- `MSVCRT.DLL` , the C standard library which allows the c code to be compiled and run properly.
- `USER32.DLL` , a user interface library that presumbly is used to create the popup that the victims



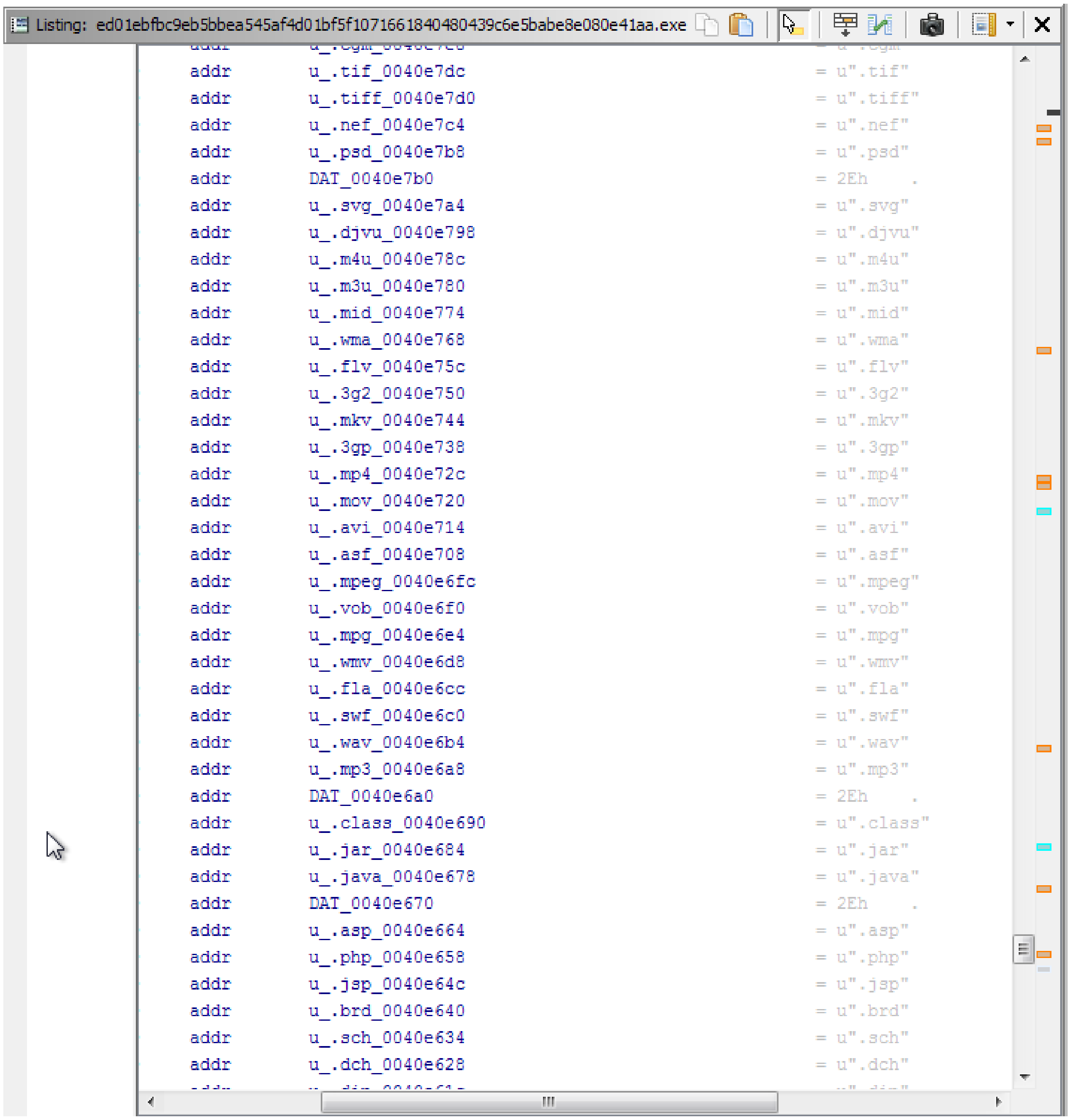
Interestingly, there are not a lot of crypto functions in `ADVAPI` listed in Ghidra. However, when going to the data section, string for the windows crypto functions can be found. Following the listed call location to the function all of them are in, you can see that the program gets their address for use.


Decompile: Load_Crypto_Function_Addresses - (ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa

```

/* WARNING: Globals starting with '_' overlap smaller symbols at the same address */

undefined4 Load_Crypto_Function_Addresses(void)

{
  HMODULE hModule;
  undefined4 uVar1;

  if (DAT_0040f894 == (FARPROC)0x0) {
    hModule = LoadLibraryA(s_advapi32.dll_0040e020);
    if (hModule != (HMODULE)0x0) {
      DAT_0040f894 = GetProcAddress(hModule,s_CryptAcquireContextA_0040f110);
      DAT_0040f898 = GetProcAddress(hModule,s_CryptImportKey_0040f100);
      DAT_0040f89c = GetProcAddress(hModule,s_CryptDestroyKey_0040f0f0);
      _DAT_0040f8a0 = GetProcAddress(hModule,s_CryptEncrypt_0040f0e0);
      DAT_0040f8a4 = GetProcAddress(hModule,s_CryptDecrypt_0040f0d0);
      _DAT_0040f8a8 = GetProcAddress(hModule,s_CryptGenKey_0040f0c4);
      if ((((DAT_0040f894 != (FARPROC)0x0) && (DAT_0040f898 != (FARPROC)0x0)) &&
          (DAT_0040f89c != (FARPROC)0x0)) &&
         (((_DAT_0040f8a0 != (FARPROC)0x0 && (DAT_0040f8a4 != (FARPROC)0x0)) &&
          (_DAT_0040f8a8 != (FARPROC)0x0)))) goto LAB_00401aec;
    }
    uVar1 = 0;
  }
  else {
LAB_00401aec:
    uVar1 = 1;
  }
  return uVar1;
}
```

There is a variety of other interesting strings to be found in the `data` section. Some of particular note:

All the file extensions that our target looks to encrypt (not all fit in screenshot):



Listing: ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe

```
addr     u_.tif_0040e7dc          = u".tif"
addr     u_.tiff_0040e7d0         = u".tiff"
addr     u_.nef_0040e7c4          = u".nef"
addr     u_.psd_0040e7b8          = u".psd"
addr     DAT_0040e7b0             = 2Eh    .
addr     u_.svg_0040e7a4          = u".svg"
addr     u_.djvu_0040e798         = u".djvu"
addr     u_.m4u_0040e78c          = u".m4u"
addr     u_.m3u_0040e780          = u".m3u"
addr     u_.mid_0040e774          = u".mid"
addr     u_.wma_0040e768          = u".wma"
addr     u_.flv_0040e75c          = u".flv"
addr     u_.3g2_0040e750          = u".3g2"
addr     u_.mkv_0040e744          = u".mkv"
addr     u_.3gp_0040e738          = u".3gp"
addr     u_.mp4_0040e72c          = u".mp4"
addr     u_.mov_0040e720          = u".mov"
addr     u_.avi_0040e714          = u".avi"
addr     u_.asf_0040e708          = u".asf"
addr     u_.mpeg_0040e6fc         = u".mpeg"
addr     u_.vob_0040e6f0          = u".vob"
addr     u_.mpg_0040e6e4          = u".mpg"
addr     u_.wmv_0040e6d8          = u".wmv"
addr     u_.fla_0040e6cc          = u".fla"
addr     u_.swf_0040e6c0          = u".swf"
addr     u_.wav_0040e6b4          = u".wav"
addr     u_.mp3_0040e6a8          = u".mp3"
addr     DAT_0040e6a0             = 2Eh    .
addr     u_.class_0040e690        = u".class"
addr     u_.jar_0040e684          = u".jar"
addr     u_.java_0040e678         = u".java"
addr     DAT_0040e670             = 2Eh    .
addr     u_.asp_0040e664          = u".asp"
addr     u_.php_0040e658          = u".php"
addr     u_.jsp_0040e64c          = u".jsp"
addr     u_.brd_0040e640          = u".brd"
addr     u_.sch_0040e634          = u".sch"
addr     u_.dch_0040e628          = u".dch"
```

The name of the process it runs itself as: `tasksche.exe`

```
            s_tasksche.exe_0040f4d8                    XREF[4]:     FUN_00401f5d:00401f92(*),
                                                                    FUN_00401fe7:00402061(*),
                                                                    FUN_00401fe7:0040206d(*),
                                                                    FUN_00401fe7:00402075(*)
 74 61 73        ds          "tasksche.exe"
 6b 73 63
 68 65 2e ...
 00              ??          00h
 00              ??          00h
 00              ??          00h
```

A couple of interesting looking commands and possible injections:

```
            s_icacls_._/grant_Everyone:F_/T_/C_0040f4fc     XREF[1]:     FUN_00401fe7:004020e8(*)
69 63 61        ds          "icacls . /grant Everyone:F /T /C /Q"
63 6c 73
20 2e 20 ...
```

```
            s_cmd.exe_/c_"%s"_0040f42c                      XREF[1]:     FUN_00401ce8:00401d4e(*)
63 6d 64        ds          "cmd.exe /c \"%s\""
2e 65 78
65 20 2f ...
```

```
            s_Global\MsWinZonesCacheCounterMut_0040f4b4     XREF[1]:     FUN_00401eff:00401f08(*)
47 6c 6f        ds          "Global\\MsWinZonesCacheCounterMutexA"
62 61 6c
5c 4d 73 ...
```

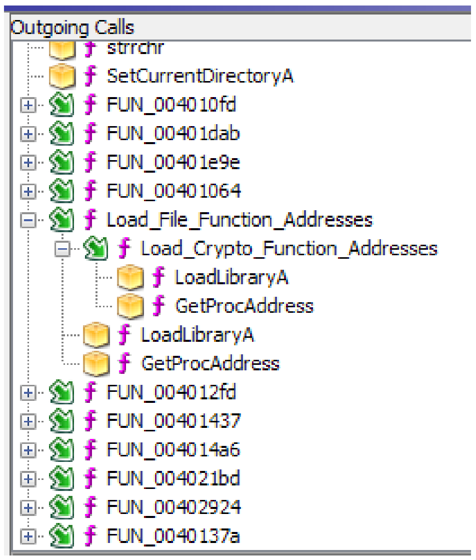
A strange looking file extension worth investigating:

```
            s_t.wnry_0040f4f4                               XREF[1]:     FUN_00401fe7:00402125(*)
74 2e 77        ds          "t.wnry"
6e 72 79 00
00              ??          00h
```

Addresses to pay bitcoin to:

```
                    s_115p7UMMngoj1pMvkpHijcRdfJNXj6Lr_0040f440     XREF[1]:     FUN_00401e9e:00401ebe(*)
40f440 31 31 35         ds          "115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn"
       70 37 55
       4d 4d 6e ...
40f463 00               ??          00h

                    s_12t9YDPgwueZ9NyMgw519p7AA8isjr6S_0040f464     XREF[1]:     FUN_00401e9e:00401eb7(*)
40f464 31 32 74         ds          "12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw"
       39 59 44
       50 67 77 ...
40f487 00               ??          00h

                    s_13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb_0040f488     XREF[1]:     FUN_00401e9e:00401eb0(*)
40f488 31 33 41         ds          "13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94"
       4d 34 56
       57 32 64 ...
```

By starting here, we get a general sense of what the program is capable of as well as some very good clues as to what some of the functions are doing. This allows us to identify the functions involved with the string, and use them as a stepping stone to understand the functions that call those functions. Along with looking at what library functions are being called in each function, even though the number of functions called by the program is quite a large number this helps narrow down the areas of interest.



```
strrchr
SetCurrentDirectoryA
FUN_004010fd
FUN_00401dab
FUN_00401e9e
FUN_00401064
Load_File_Function_Addresses
    Load_Crypto_Function_Addresses
        LoadLibraryA
        GetProcAddress
    LoadLibraryA
    GetProcAddress
FUN_004012fd
FUN_00401437
FUN_004014a6
FUN_004021bd
FUN_00402924
FUN_0040137a
```

Back to Top

Plans for Rest of Project

Instruction: *What are your plans for the rest of the project?*

As mentioned previously in the general overview section, we plan to use WannaCry as an initial stepping stone to understand how to reverse malware. We plan on finding other samples to reverse engineer to hopefully be able to identify common patterns that could be identified in antivirus software as well as see what can be done to mitigate the effects of them aside from patching exploits.

Back to Top

Resources and Research

- [Textbook] Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software by Michael Sikorski and Andrew Honig
- FlareVM for setting up a reverse engineering environment on a VM w/ helpful scripts
- WannaCry binary from theZoo GitHub repository
- Windows 7 iso

Back to Top

Give feedback