LIFARS
your digital world, **secured**

DEARCRY RANSOMWARE MALWARE ANALYSIS AND REVERSE ENGINEERING

Prepared by: LIFARS, LLC
Date: 03/25/2021

TABLE OF CONTENTS

244 Fifth Avenue, Suite 2035, New York, NY 10001
LIFARS.com (212) 222-7061 info@lifars.com

The goal of this paper is to provide deep analysis of DearCry ransomware and demonstrate some techniques of malware analysis, and especially reverse engineering of malicious sample for educational purposes.

INTRODUCTION

The DearCry ransomware has been used in current attacks related to the exploitation of Microsoft Exchange Servers. Unlike other ransomwares, DearCry is special in terms of its complexity. It is very simple malware, and it could be reverse engineered in couple of minutes as we demonstrate in this paper. The main objective of this document is to provide not only the analysis of DearCry ransomware, but also to provide educational tips and tricks, which could be useful in the cybersecurity community and students of computer science.

STATIC ANALYSIS

Static analysis is usually the initial stage of malware analysis. Commonly the samples are scanned with antivirus software and IOC scanners. This phase also includes the analysis of sample metadata, embedded strings, resources, imports and exports (in case of Portable executable files, .EXE), presence of macros and auto-open or auto-close actions (in case of Office Documents).

DearCry Sample

In this paper we analyze DearCry ransomware sample (often classified also as DoejoCrypt) obtained from Malware Bazaar. It is a portable executable file, and it is approximately 1.2 MB in size. This means that it is relatively large malware sample.

SHA256 hash:	e044d9f2d0f1260c3f4a543a1e67f33fcac265be114a1b135fd575b860d2b8c6
SHA3-384 hash:	6c75fb488c64166576e2fa4da6f2625c6ba8553a088b024329eb666744c940d35efe08b0824d30c4bffdc8307cdf20ee
SHA1 hash:	56eec7392297e7301159094d7e461a696fe5b90f
MD5 hash:	cdda3913408c4c46a6c575421485fa5b
humanhash:	mars-pennsylvania-magnesium-minnesota
File name:	e044d9f2d0f1260c3f4a543a1e67f33fcac265be114a1b135fd575b860d2b8c6.bin
Download:	download sample
Signature	DoejoCrypt Alert
File size:	1'322'496 bytes
First seen:	2021-03-11 22:55:38 UTC
Last seen:	2021-03-12 06:50:49 UTC
File type:	exe
MIME type:	application/x-dosexec
imphash	f8b8e20e844ccd50a8eb73c2fca3626d
ssdeep	24576:C5Nv2SkWFP/529IC8u2bAs0NIzkQS+KpPbEasBY2iKDl1fpxkLVZgMCS+:oB70s9yjE62iIl1fpxkLVZgMC3
TLSH	3855CFC3F6C248F2D886097951B3973B5F3ADA11832AC5C3CFA15DA59C21AD1663E3C9
Reporter	@ArkbirdDevil
Tags:	DearCry DoejoCrypt Ransomware

Figure 1: DearCry Metadata from Malware Bazaar repository

LIFARS your digital world, secured
244 Fifth Avenue, Suite 2035, New York, NY 10001
LIFARS.com (212) 222-7061 info@lifars.com

Strings

DearCry is very simple ransomware, as we can see even by extraction of the embedded strings. We use Sysinternals tool called strings.exe.

```
1#SNAN
%02x
DEARCRY!
.CRYPT
rb+
*.*
%s%s
%s\%s
readme.txt
DESKTOP
/readme.txt
msupdate
Your file has been encrypted!
 If you want to decrypt, please contact us.
 %s
 And please send me the following hash!
 %s
%c:\
%c:\%s
create rsa error
pYS
RSDS.
C:\Users\john\Documents\Visual Studio 2008\Projects\EncryptFile -svcV2\Release\EncryptFile.exe.pdb
a"N
/4N
5>N
```

Figure 2: Extracted strings with ransom note template and name of the ransomware.

There is no obfuscation, all strings are clearly visible. For example, the ransom note. The sample leaks some debug information about its origin, too. From the PDB filepath we can determine the username, used development tools and original name of the project.

```
,%I
-64OS
NKeb
-----BEGIN RSA PUBLIC KEY-----
MIIBCAKCAQEA5+mVBe75OvCzCW4oZHl7vqPwV2O4kgzgfp9odcL9LZc8Gy2+NJPD
wrHbttKI3z4Yt3G04lX7bEp1RZjxUYfzX8qvaPC2EBduOjSN1WMSbJJrINs1Izkq
XRrggJhSbp881Jr6NmpE6pns0Vfv//Hk1idHhxsXg6QKtfXlzAnRbgA1WepSDJq5
H08WGFBZrgUVM0zBYI3JJH3b9jIRMVQMJUQ57w3jZpOnpFXSZoUy1YD7Y3Cu+n/Q
6cEft6t29/FQgacXmeA2ajb7ssSbSntBpTpoyGc/kKoaihYPrHtNRhkMcZQayy5a
XTgYtEjhzJAC+esXiTYqklWMXJS1EmUpoQIBAw==
-----END RSA PUBLIC KEY-----
dear!!!
.TIF .TIFF .PDF .XLS .XLSX .XLTM .PS .PPS .PPT .PPTX .DOC .DOCX .LOG .MSG .RTF .TEX
JSP .PHP .KEYCHAIN .PEM .SQL .APK .APP .BAT .CGI .ASPX .CER .CFM .C .CPP .GO .CONFIG
B .BIN .DB .MDB .MDF .BAK .LOG .EDB .STM .DBF .ORA
WINDIR
TEMP
APPDATA
PROGRAMFILES
konedieyp@airmail.cc or uenwonken@memail.com
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
```

Figure 3: Extracted strings with RSA public key and file extensions to be encrypted

RSA Public key is visible here, and also the list of file extensions. DearCry ransomware will probably encrypt files with these extensions, as we will see later.

Capabilities

As a next step, we can quickly identify capabilities in the analyzed sample with the capa tool. There is lot of cryptography, ciphers, hashes. And it is linked against OpenSSL cryptography library.

```
C:\Work\DearCry>capa e044d9f2d0f1260c3f4a543a1e67f33fcac265be114a1b135fd575b860d2b8c6.bin
loading : 100%|                                                                                | 469/469 [00:00<0
matching: 100%|                                                                                | 3626/3626 [03:44<
+-----------------------+-------------------------------------------------------------------------------------------+
| md5                   | cdda3913408c4c46a6c575421485fa5b                                                          |
| sha1                  | 56eec7392297e7301159094d7e461a696fe5b90f                                                  |
| sha256                | e044d9f2d0f1260c3f4a543a1e67f33fcac265be114a1b135fd575b860d2b8c6                          |
| path                  | e044d9f2d0f1260c3f4a543a1e67f33fcac265be114a1b135fd575b860d2b8c6.bin                       |
+-----------------------+-------------------------------------------------------------------------------------------+

+-----------------------+-------------------------------------------------------------------------------------------+
| ATT&CK Tactic         | ATT&CK Technique                                                                          |
|-----------------------+-------------------------------------------------------------------------------------------|
| DEFENSE EVASION       | Deobfuscate/Decode Files or Information [T1140]                                           |
|                       | Obfuscated Files or Information [T1027]                                                   |
| DISCOVERY             | File and Directory Discovery [T1083]                                                      |
|                       | System Information Discovery [T1082]                                                     |
| EXECUTION             | Command and Scripting Interpreter [T1059]                                                 |
|                       | Shared Modules [T1129]                                                                    |
| PERSISTENCE           | Create or Modify System Process::Windows Service [T1543.003]                              |
+-----------------------+-------------------------------------------------------------------------------------------+

+---------------------------+---------------------------------------------------------------------------------------+
| MBC Objective             | MBC Behavior                                                                          |
|---------------------------+---------------------------------------------------------------------------------------|
| ANTI-BEHAVIORAL ANALYSIS  | Debugger Detection::Software Breakpoints [B0001.025]                                  |
|                           | Debugger Detection::Timing/Delay Check GetTickCount [B0001.032]                       |
|                           | Execution Guardrails::Runs as Service [E1480.m07]                                     |
| COMMAND AND CONTROL       | C2 Communication::Receive Data [B0030.002]                                            |
|                           | C2 Communication::Send Data [B0030.001]                                               |
| COMMUNICATION             | Socket Communication::Receive Data [C0001.006]                                        |
|                           | Socket Communication::Send Data [C0001.007]                                           |
| CRYPTOGRAPHY              | Crypto Library [C0059]                                                                |
|                           | Cryptographic Hash::SHA1 [C0029.002]                                                  |
|                           | Cryptographic Hash::SHA224 [C0029.004]                                                |
|                           | Cryptographic Hash::SHA256 [C0029.003]                                                |
```

Figure 4: Sample overview reported by capa tool

```
| get common file path                                  | host-interaction/file-system                      |
| delete file                                           | host-interaction/file-system/delete               |
| check if file exists (2 matches)                      | host-interaction/file-system/exists               |
| enumerate files recursively                           | host-interaction/file-system/files/list           |
| get file attributes                                   | host-interaction/file-system/meta                 |
| read file (2 matches)                                 | host-interaction/file-system/read                 |
| write file (3 matches)                                | host-interaction/file-system/write                |
| get memory capacity                                   | host-interaction/hardware/memory                  |
| get disk information (3 matches)                      | host-interaction/hardware/storage                 |
| access the Windows event log                          | host-interaction/log/winevt/access                |
| get thread local storage value (4 matches)            | host-interaction/process                          |
| set thread local storage value (3 matches)            | host-interaction/process                          |
| terminate process (5 matches)                         | host-interaction/process/terminate                |
| run as service (2 matches)                            | host-interaction/service                          |
| delete service                                        | host-interaction/service/delete                   |
| link function at runtime (9 matches)                  | linking/runtime-linking                           |
| link many functions at runtime                        | linking/runtime-linking                           |
| linked against OpenSSL                                | linking/static/openssl                            |
| parse PE header (74 matches)                          | load-code/pe                                      |
+-------------------------------------------------------+---------------------------------------------------+
```

Figure 5: Sample capabilities - file operations; OpenSSL crypto library used.

BEHAVIORAL ANALYSIS

During behavioral analysis, the sample is executed in sandbox. This protected environment is monitored for any activities performed by the sample, such as spawning new processes, network communication, dropping files or overwriting the existent files. By reviewing of sample's behavior, we can often say if the sample is malicious and if yes, what kind of malware it is (e.g., ransomware).

With behavioral analysis we can also quickly collect lot of indicators of compromise (IOC) which could be used by rest of the team for effective incident response, forensic analysis, threat hunting or for monitoring and prevent threats in the customers' infrastructure.

For now, in this paper we skip this step right now, because we already know that this is a DearCry ransomware sample which encrypts files. We will rather deep dive into the DearCry internals and code. We will demonstrate the process of reverse engineering the malware. However, we will later do a crosscheck of our findings with output from the sandbox, in this case, just for educational purposes.

REVERSE ENGINEERING

Reverse engineering is the phase in which we decompile or disassemble the machine instructions of program into more readable form. In this case, analyzed sample is a Portable Executable file produced by Microsoft Visual Studio compiler. We use IDA, Interactive Disassembler, for reverse engineering of this DearCry sample.

IDA Flirt Signatures

When IDA finished its automatic analysis, we can see disassembled program with lot of functions. By default, almost all functions are assumed to be regular (blue color in program navigation bar), without known library functions (light cyan in navigation bar). We can fix this by applying IDA's FLIRT signatures, for example, Microsoft Visual C runtime signatures identified more than 600 functions. But there is still very small portion of all functions.

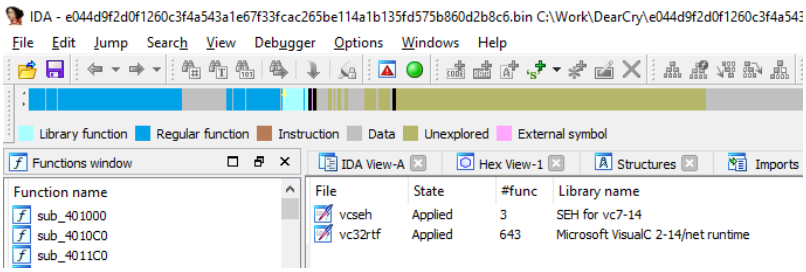

Figure 6: Applied IDA FLIRT Signatures and Program overview in Navigation Bar

Recall that capa tool identified that this sample is linked against OpenSSL, and there are many strings containing the OpenSSL term. It seems that DearCry is statically linked against OpenSSL version 1.1.

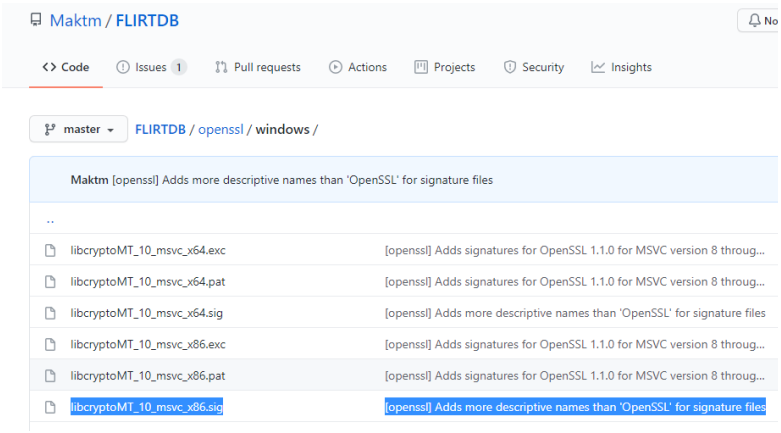

Figure 7: GitHub repository with FLIRT signatures for OpenSSL

We can obtain the signatures for OpenSSL from the [community driven collection of IDA FLIRT signature files]. They are available for couple of common libraries. We will download and use two which fits most to our case - OpenSSL 1.1 compiled by Visual Studio 2008, as we saw in the extracted strings. With these two FLIRT signatures applied, we have identified more than 3000 of library functions. Now it seems that only small portion of DearCry functions is custom, developed by authors of the ransomware.

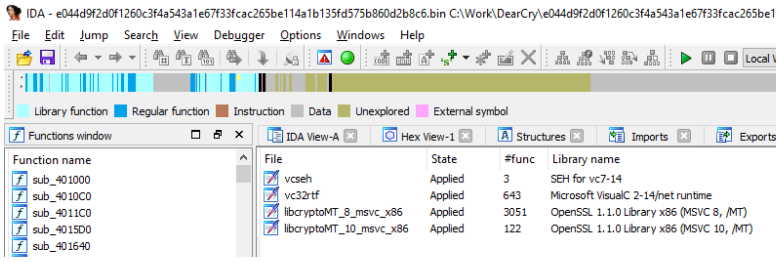

Figure 8: Applied IDA FLIRT Signatures for OpenSSL library

When we examine imports, they are mostly related to cryptography, because of dependencies of embedded OpenSSL library. On the other hand, there is only one exported symbol called start, which is the usual entry point of portable executable files.

Entry Point

Now we are ready to begin with analysis of disassembled code. Our objective is to understand what the analyzed program does and how it works.

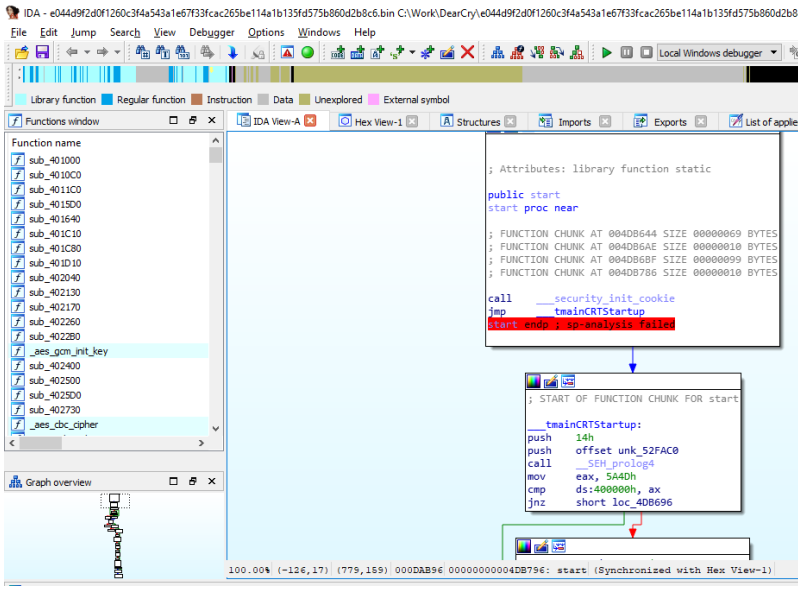

Figure 9: Start routine of the analyzed sample

This is more less standard start routine, with checking for "MZ" (5A4Dh) and "PE" executable headers, then parsing command line arguments and set environment variables. After that, near to the end of start routine, there is a call with three arguments. This is the main function of the programs developed in C or C++.

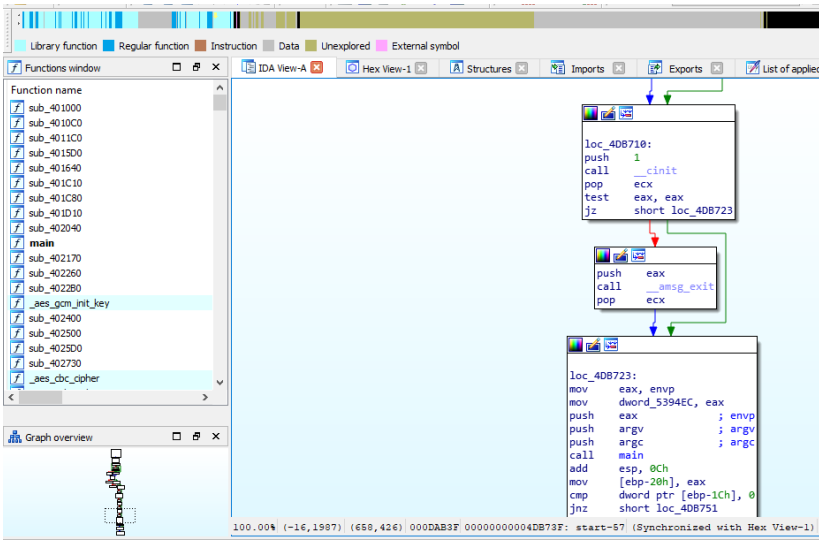

Figure 10: End of the start routine with call to main function

Main Function

The main function is simple. It starts service control dispatcher, which connects the main service thread to the service control manager. The service related to this ransomware is called "msupdate".

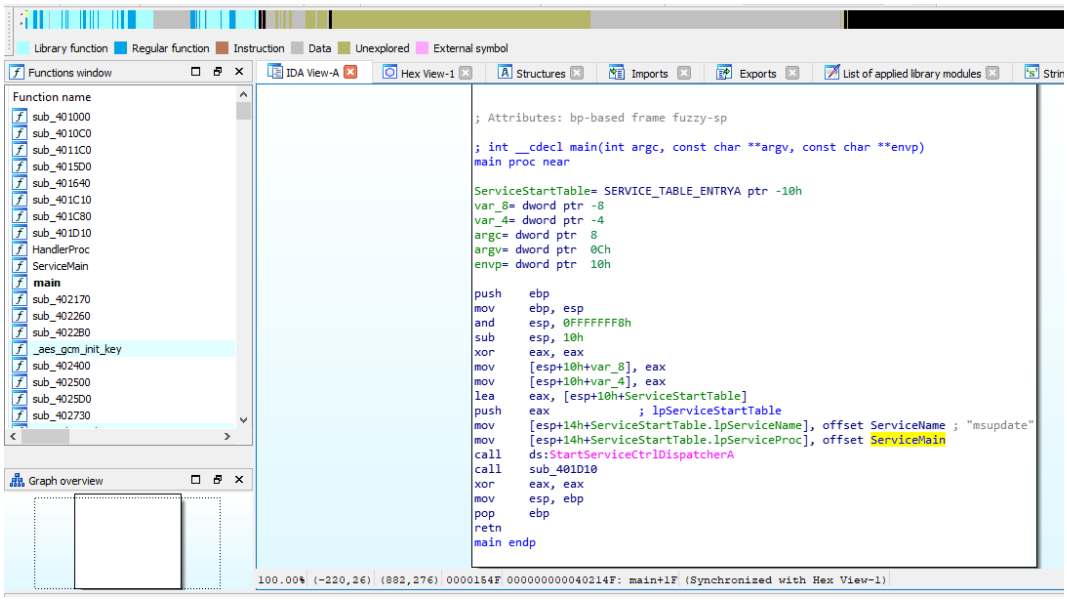

Figure 11: Disassembled main function of the ransomware sample

ServiceMain Function

ServiceMain function is also simple, it registers service control handler for this "msupdate" service. And then, it calls yet unknown function sub_401D10.

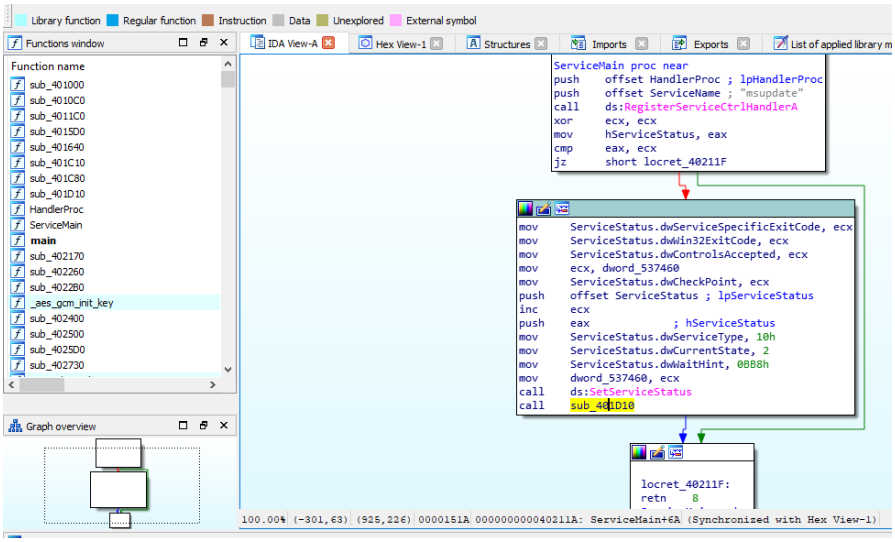

Figure 12:Disassembled ServiceMain function

Back in main function, we can see that the same sub_401D10 function is called right after service dispatcher. It seems that this function is responsible for all malicious things performed by this ransomware sample. Hence, it will probably do some ransomware stuff.

"Do-ransomware-stuff" Function

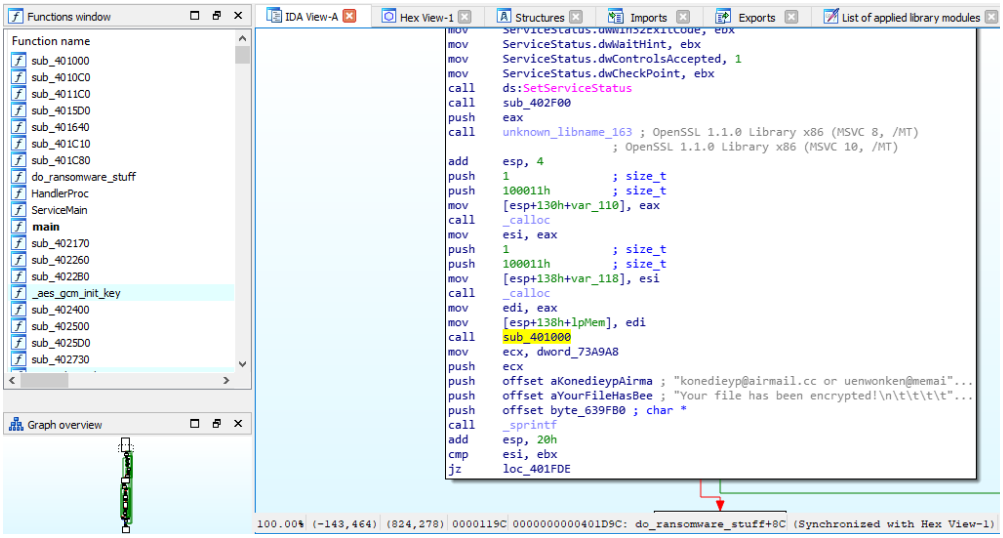

Figure 13: "do_ransomware_stuff" function called from main and ServiceMain functions

Let's look into the ransomware stuff function. First interesting function is sub_401000. It references the embedded RSA Public Key and creates string with hexadecimal representation of some values in loop. It actually creates a formatted string with hash value of RSA key.

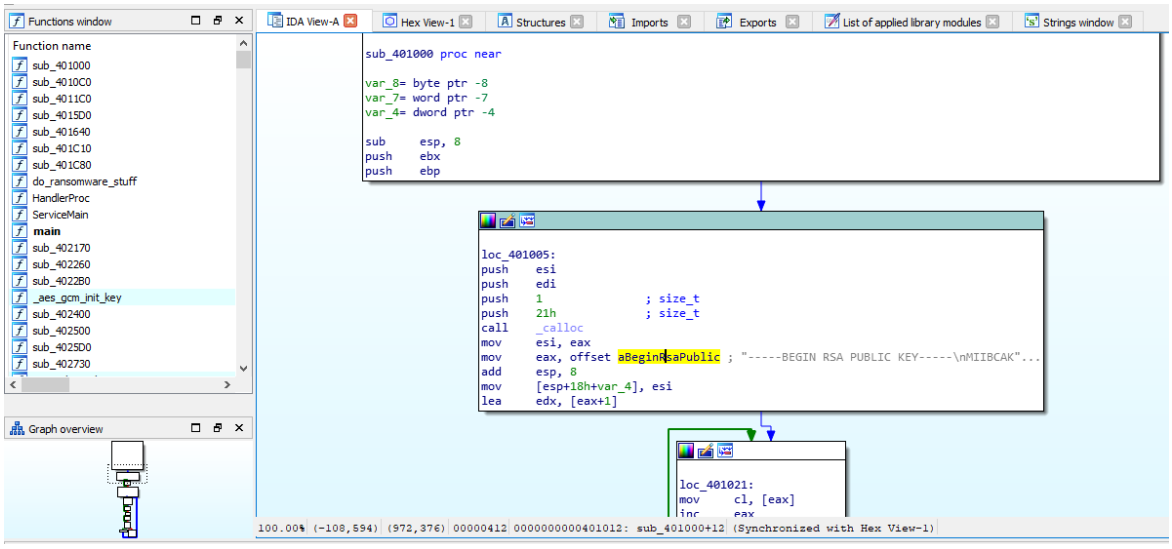

Figure 14: Obtaining the hexadecimal representation of the embedded RSA Public Key's hash value

Next, the ransomware stuff function then prepares a formatted ransom note message and get list of logical drives of the infected machine. It searches for drives with letters between C and Z included, and all types of drive except CD-ROM drive.

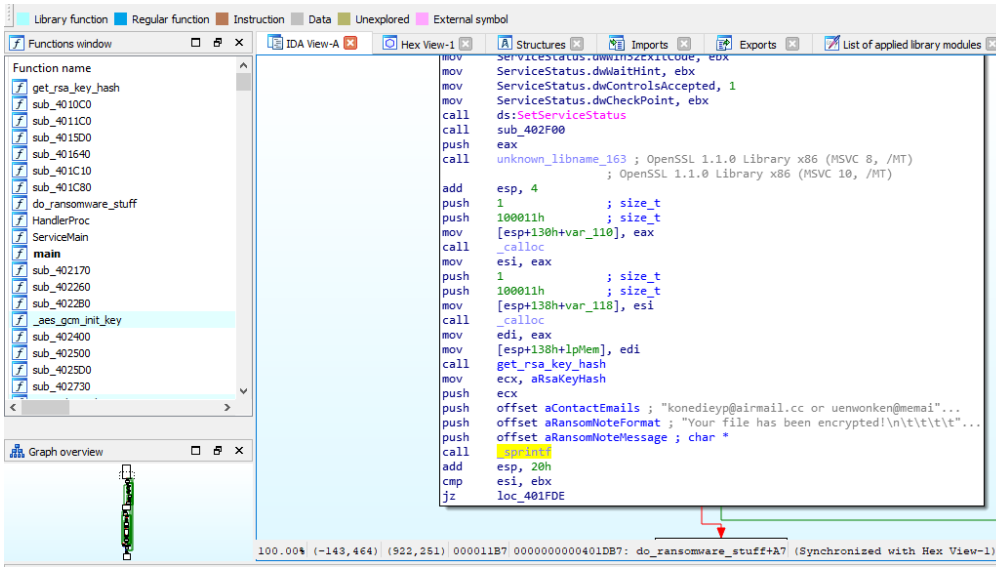

Figure 15: Preparation of formatted ransom note message in the ransomware stuff function

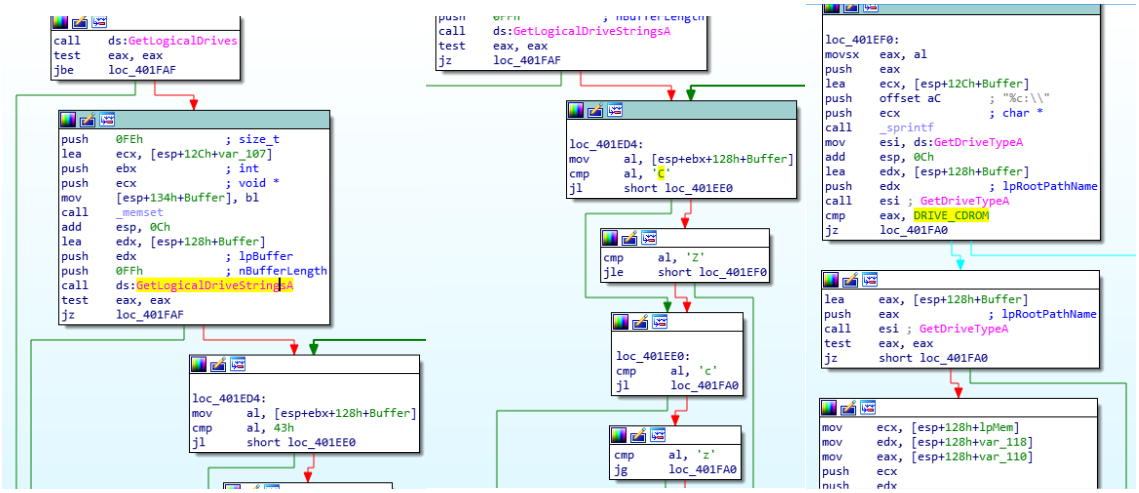

Figure 16: Enumeration of logical drives of the infected machine

After that, it passes each drive to the function sub_401640, which will probably be responsible for encrypting drive or folder.

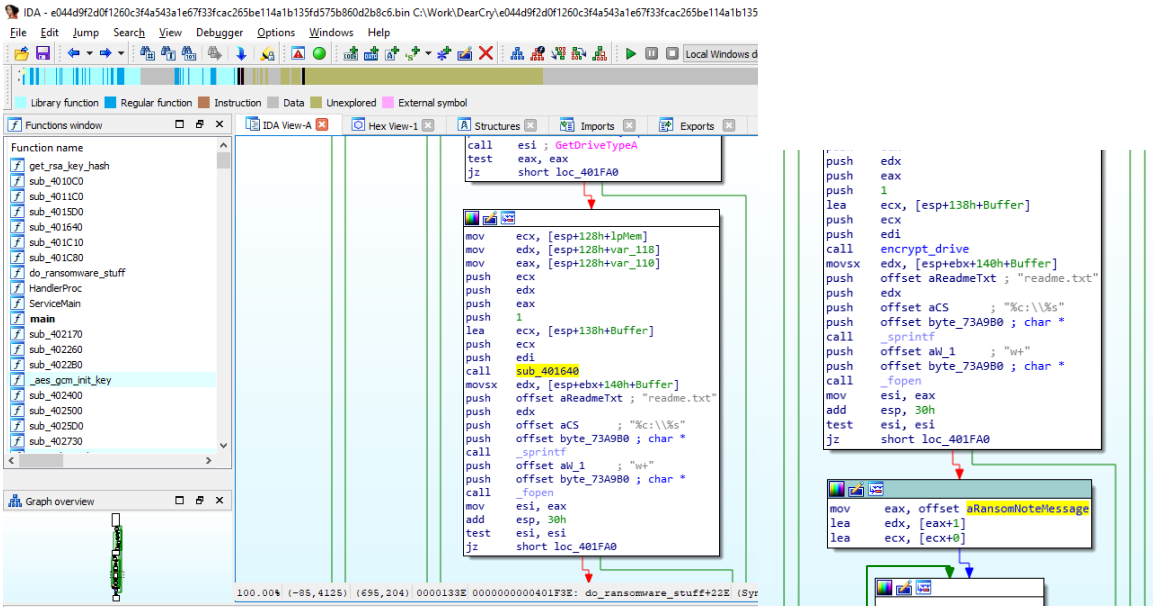

Figure 17: Hypothesis: function sub_401640 is responsible for encrypting drive or folder and creation of readme.txt file with ransom note

Then, this ransomware stuff function drops readme.txt file with the ransom note. And finally, the last call will delete service "msupdate", created by this ransomware previously.

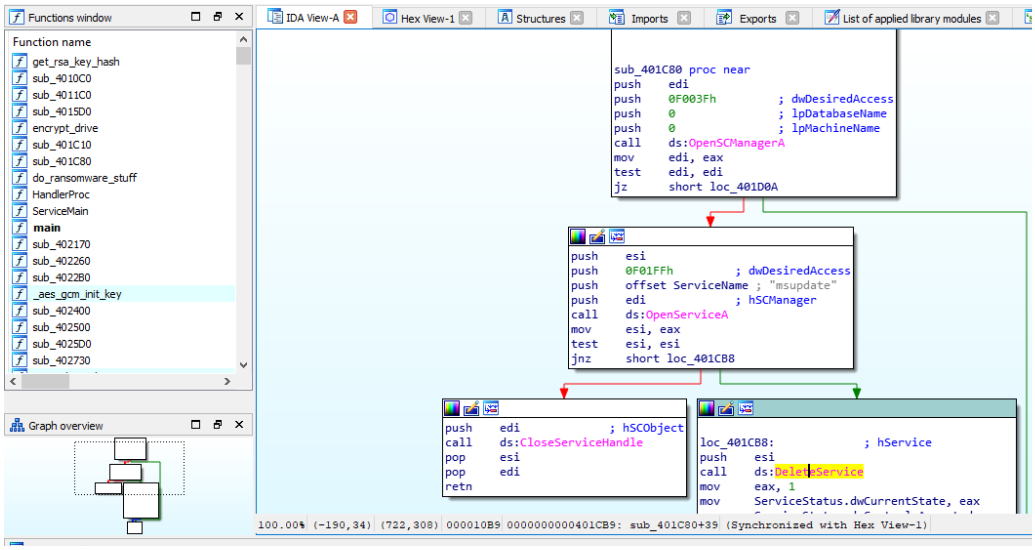

Figure 18: Removing msupdate service created by this ransomware previously

File Encryption

Until now, we used top-down methodology for analysis of ransomware logic. Now we can change our approach and use bottom-up methodology instead of top-down.

Encrypt-file Function

During static analysis we saw string ".CRYPT", which looks like an extension of the files encrypted by this ransomware. Let's examine the cross references to this string in IDA. It is referenced only in one function; thus this function should be responsible for writing an encrypted file to disk.

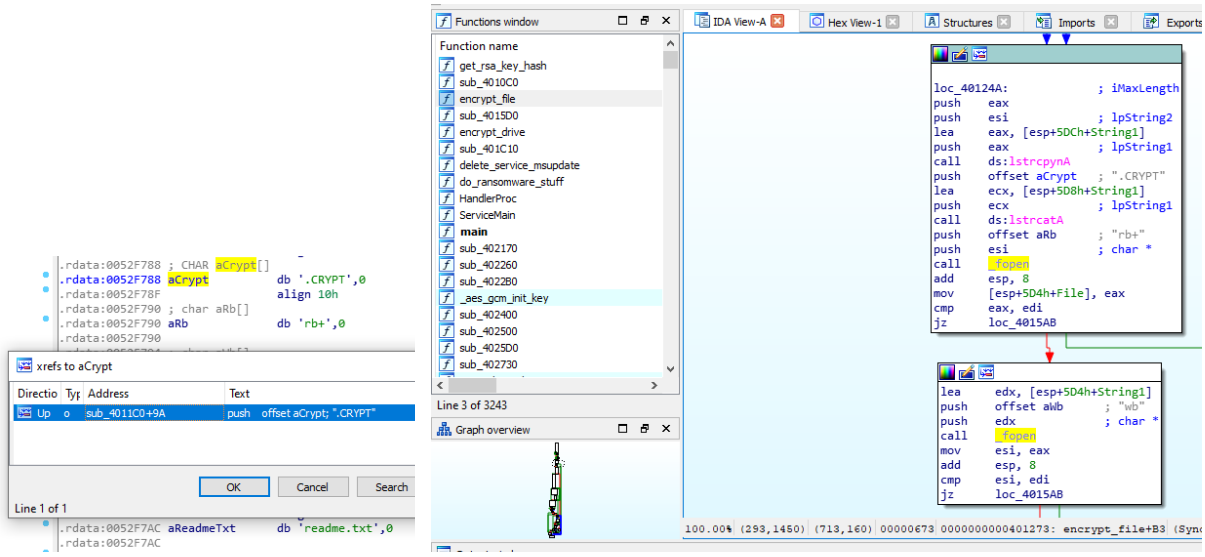

Figure 19:Cross references to ".CRYPT" and creating the encrypted file by ransomware

Let us examine this encrypt file function. Mode "rb+" means that the original file is opened for updating. To be more specific, for reading and writing. The "wb" mode means, that file with ".CRYPT" extension is opened for writing. Hence, DearCry uses copy encryption instead of in-place encryption of files, and it is similar to the infamous WannaCry ransomware.

In the picture below we can see that DearCry ransomware prepends a "DEARCRY!" marker to the beginning of the encrypted .CRYPT files.

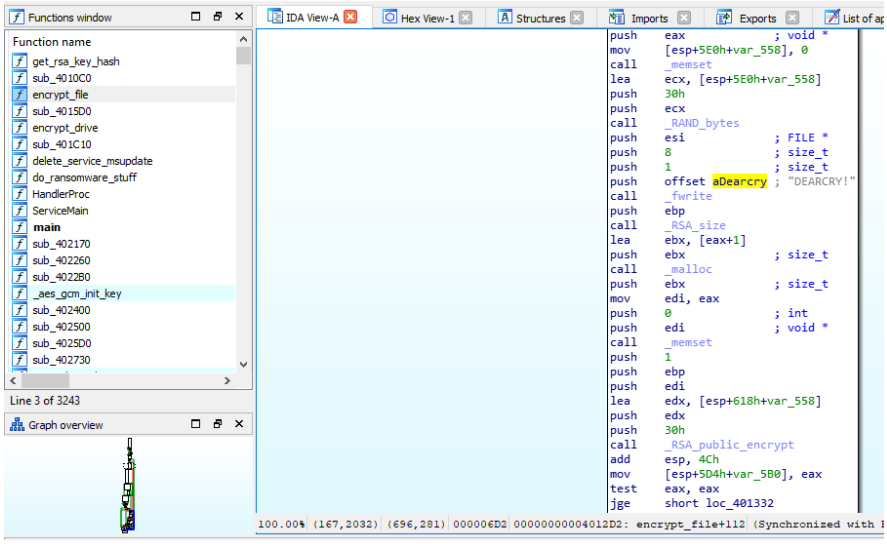

Figure 20: DEARCRY! file marker and encryption of AES key with RSA

OpenSSL Encryption: RSA+AES

The ransomware uses OpenSSL for generating a random key for symmetric encryption (AES-256-CBC) and encrypts this symmetric key with RSA using the embedded public key (2048-bit length):

```
-----BEGIN RSA PUBLIC KEY-----
MIIBCAKCAQEA5+mVBe75OvCzCW4oZHl7vqPwV2O4kgzgfp9odcL9LZc8Gy2+NJPD
wrHbttKI3z4Yt3G04lX7bEp1RZjxUYfzX8qvaPC2EBduOjSN1WMSbJJrINs1Izkq
XRrggJhSbp881Jr6NmpE6pns0Vfv//Hk1idHhxsXg6QKtfXlzAnRbgA1WepSDJq5
H08WGFBZrgUVM0zBYI3JJH3b9jIRMVQMJUQ57w3jZpOnpFXSZoUy1YD7Y3Cu+n/Q
6cEft6t29/FQgacXmeA2ajb7ssSbSntBpTpoyGc/kKoaihYPrHtNRhkMcZQayy5a
XTgYtEjhzJAC+esXiTYqklWMXJS1EmUpoQIBAw==
-----END RSA PUBLIC KEY-----
```

```
$ openssl asn1parse -in rsa.pub
    0:d=0  hl=4 l= 264 cons: SEQUENCE
    4:d=1  hl=4 l= 257 prim: INTEGER           :E7E99505EEF93AF0B3096E2864797BBEA3F05763B8920CE07E9F6875C2
5187F35FCAAF68F0B610176E3A348DD563126C926B20DB3523392A5D1AE08098526E9F3CD49AFA366A44EA99ECD157EFFFF1E4D627
334CC1608DC9247DDBF6321131540C254439EF0DE36693A7A455D2668532D580FB6370AEFA7FD0E9C11FB7AB76F7F15081A71799E0
CB2E5A5D3818B448E1CC9002F9EB1789362A92558C5C94B5126529A1
  265:d=1  hl=2 l=   1 prim: INTEGER           :03
```

Figure 21:Prime factors of 2048-bit RSA public key

Then, the encrypted symmetric key is written as a part of header of the encrypted file after the "DEARCRY!" marker.

```
call    _EVP_CIPHER_CTX_new
push    1
push    0
mov     edi, eax
push    0
push    0
mov     [esp+5E4h+lpMem], edi
call    sub_402F00
push    eax
push    edi
call    _EVP_CipherInit_ex
```

Figure 22: OpenSSL functions for encryption. sub_402F00 has not been identified by used FLIRT signature

In the Figure 22 there are calls to OpenSSL's functions _EVP_CIPHER_CTX_new, _EVP_CipherInit_ex and sub_402F00, which has not been recognized by used FLIRT signature, but this function should return the type of encryption to be used. Let's identify this function manually by quick review of OpenSSL library and its usage in DearCry ransomware.

From the OpenSSL documentation, the first two parameters of EVP_CipherInit_ex are context (EVP_CIPHER_CTX) and type (EVP_CIPHER):

```
int EVP_CipherInit_ex(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *type,
 ENGINE *impl, const unsigned char *key, const unsigned char *iv, int enc);
```

Example usage of this function could look like this:

```
EVP_CIPHER_CTX ctx;
EVP_CIPHER_CTX_init(&ctx);
EVP_CipherInit_ex(&ctx, EVP_rc4(), NULL, &key, &iv, 1);
```

The EVP_rc4() function is the example of candidate for the unknown function sub_402F00. Actually, functions such as EVP_rc4() are very simple, they contain only couple of instructions which return the object describing the type of the cipher, as is depicted in the Figure 23.

```
.rdata:004F0890 dword_4F0890    dd 427                  ; DATA XREF: sub_402F00+11↑o
.rdata:004F0894                 dd 10h
.rdata:004F0898                 dd 20h
.rdata:004F089C                 dd 10h
.rdata:004F08A0                 dd 1002h
.rdata:004F08A4                 dd offset sub_402730
.rdata:004F08A8                 dd offset _aes_cbc_cipher
.rdata:004F08AC                 db    0
```

Figure 23: EVP_Cipher type object returned by sub_402F00

The first value (427) is something called NID, numbered value of ASN.1 object identifier. The NID value of 427 is associated with the AES-256-CBC cipher.

```
$ grep 427 -A 1 -B 2 openssl-master/include/openssl/obj_mac.h
#define SN_aes_256_cbc          "AES-256-CBC"
#define LN_aes_256_cbc          "aes-256-cbc"
#define NID_aes_256_cbc         427
#define OBJ_aes_256_cbc         OBJ_aes,42L
```

Figure 24: Identification of AES-256-CBC encryption used by DearCry ransomware

244 Fifth Avenue, Suite 2035, New York, NY 10001
LIFARS.com (212) 222-7061 info@lifars.com

Put it all together

So, what have we analyzed? It seems that the chain between start or main function and encrypt_file function is almost completely analyzed, except one function, sub_4015D0; see Figure 25.

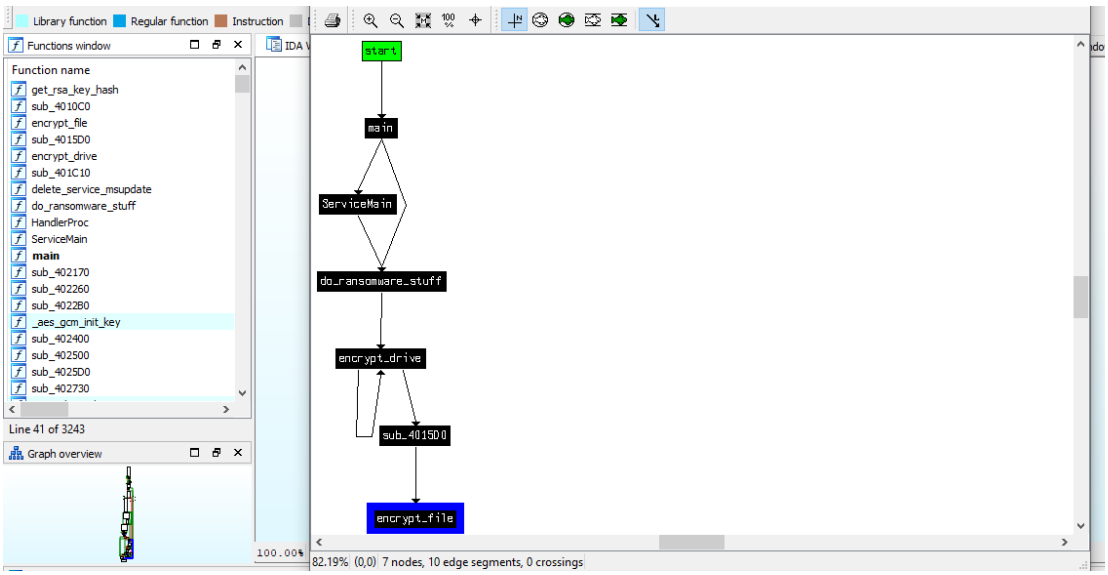

Figure 25: Function graph with already analyzed functions

Check-marker Function

Let's focus on function sub_4015D0. This time, a file is opened in read mode, and handle to this file is passed to another function, sub_4010C0. It reads first 8 bytes and compare them with the string DearCry. After that, it performs additional checks. Therefore, it checks header and marker and verifies if file is already encrypted by the ransomware. After these checks by check_marker function (originally sub_4010C0), the actual encrypt file function is executed depending on the results of checks.

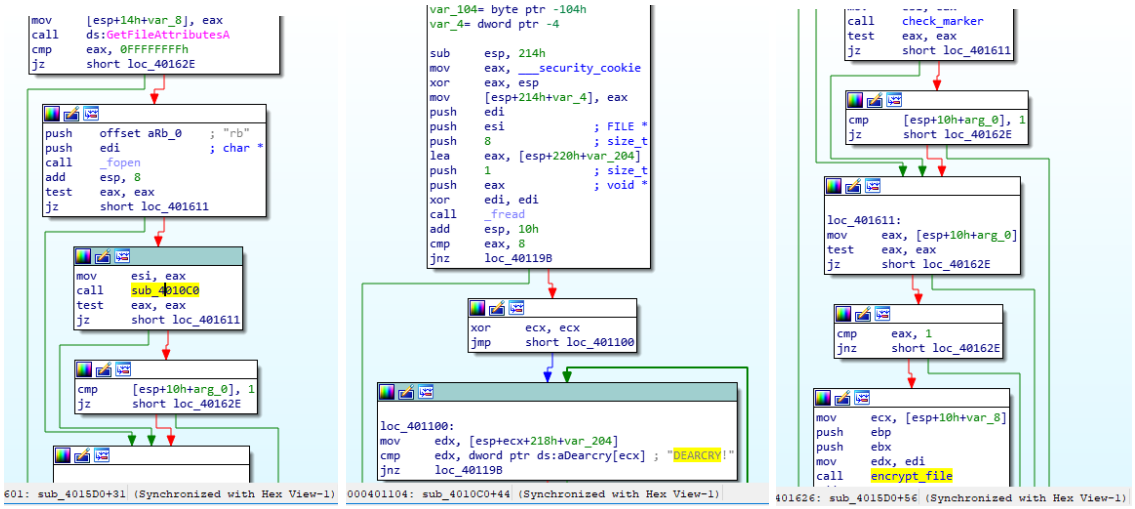

Figure 26: Checking the "DEARCRY!" file marker in sub_4010C0, followed by encrypt_file in sub_4015D0

So, we just analyzed another two functions, for checking files and "DEARCRY!" markers before encryption itself. But we also see now, that the encrypt drive/folder function calls itself recursively, and it seems that it will be rather function for encrypting folders instead of drives only.

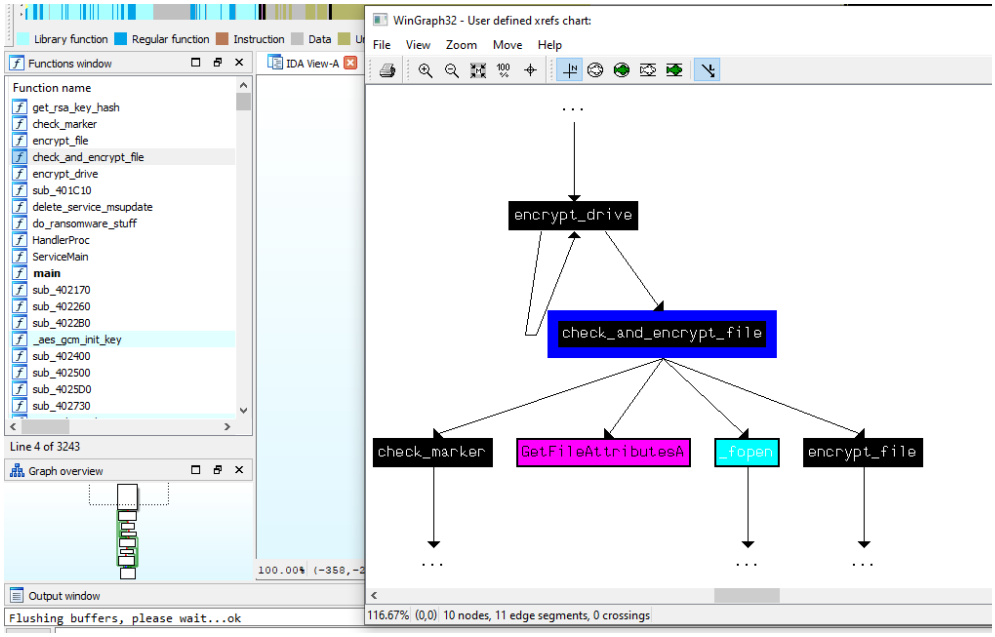

Figure 27: Function graph related to the files encryption and recursive function encrypt_drive/folder

Encrypt-folder Function

Let's dive into the encrypt_folder function. It uses Find first file and find next file API calls for searching files in current directory. For files with extension from the aforementioned list of extensions, it calls already analyzed encryption function.

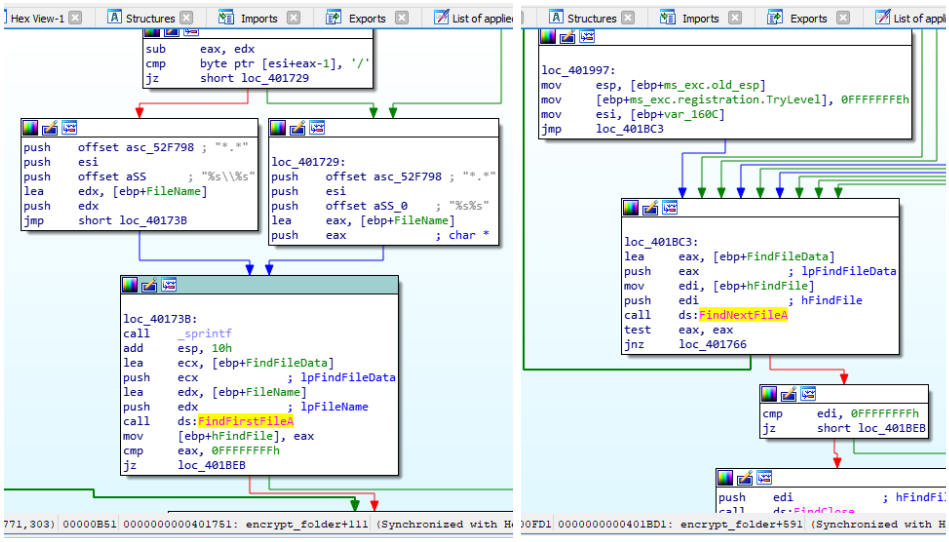

Figure 28: Encrypt_folder function uses Win32 API calls FindFirstFileA and FindNextFileA

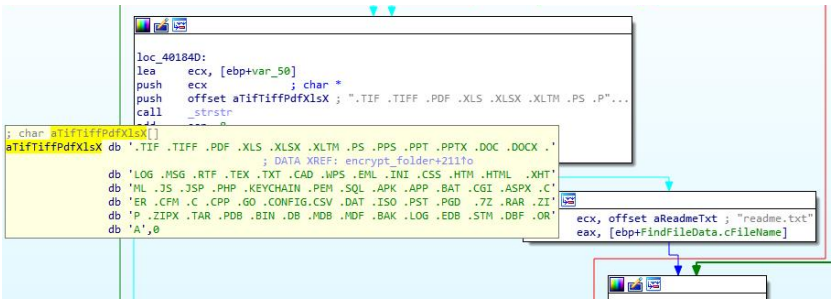

Figure 29: Checking of file extensions to encrypt

ReportServiceStatus Function

Now there is only one not yet analyzed function, sub_401C10. Quick look into it reveals that it is kind of report service status for indicating the service state.

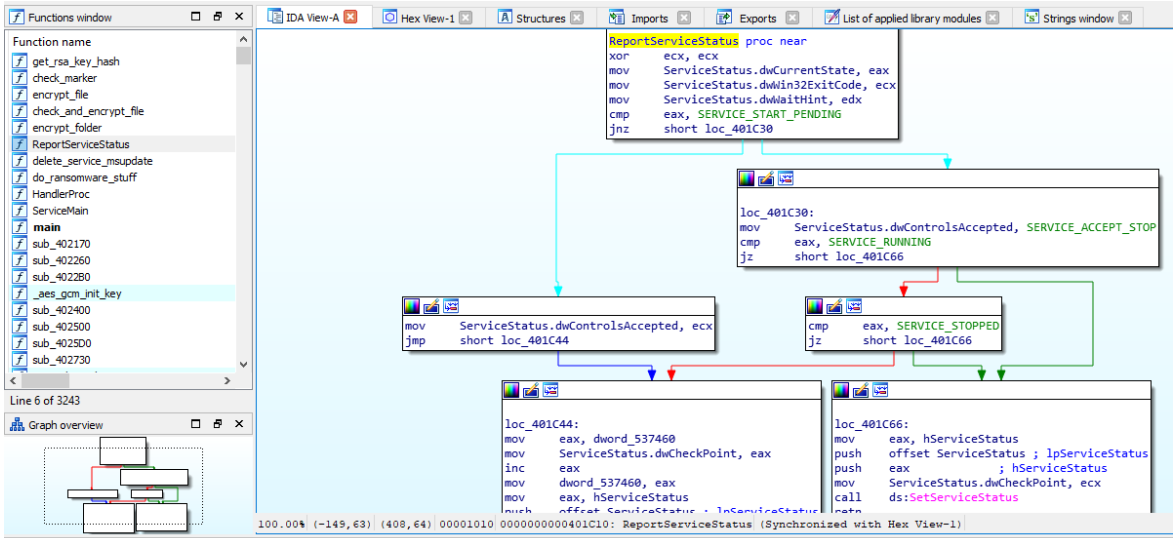

Figure 30: Last custom function - ReportServiceStatus

Now we have analyzed every regular function written by authors of the ransomware and we have rather good understanding what this ransomware does and how it works.

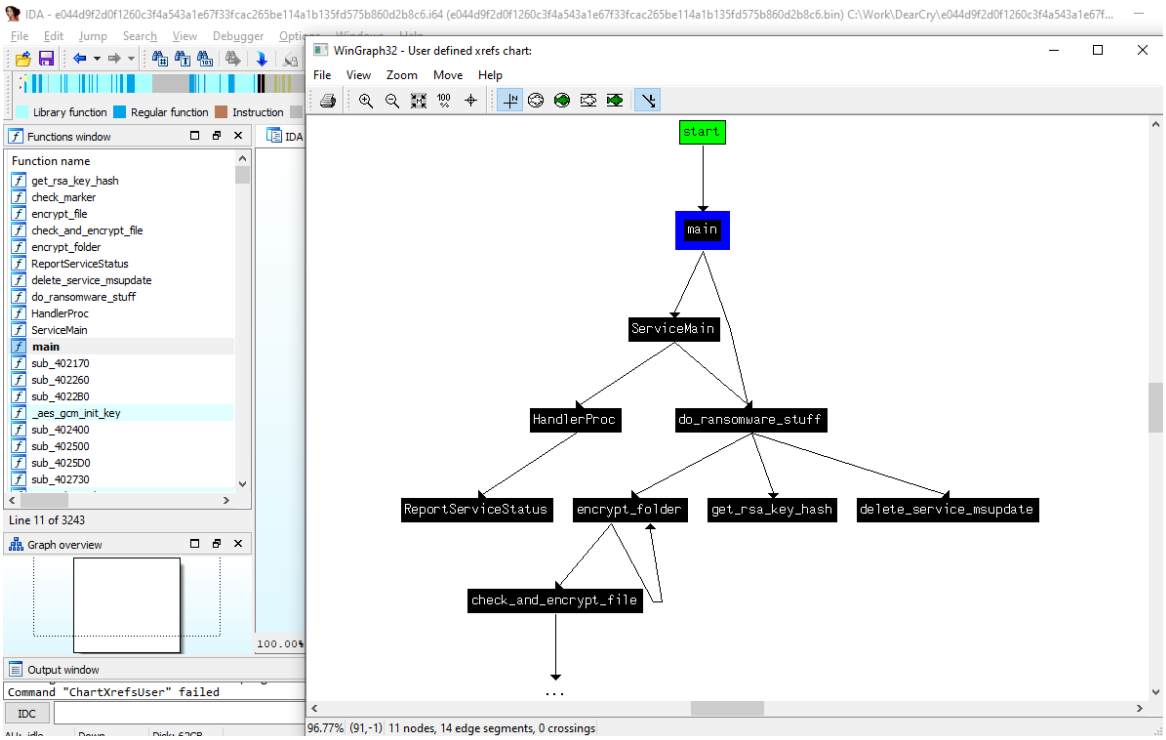

Figure 31: Function graph of analyzed functions reveals the program logic, too

Cross-check with behavioral analysis

We can cross-check our results with the results from the behavioral analysis previously performed in sandbox. For example, the encrypted files with the CRYPT extension and DearCry marker in its beginning are clearly visible in the results.

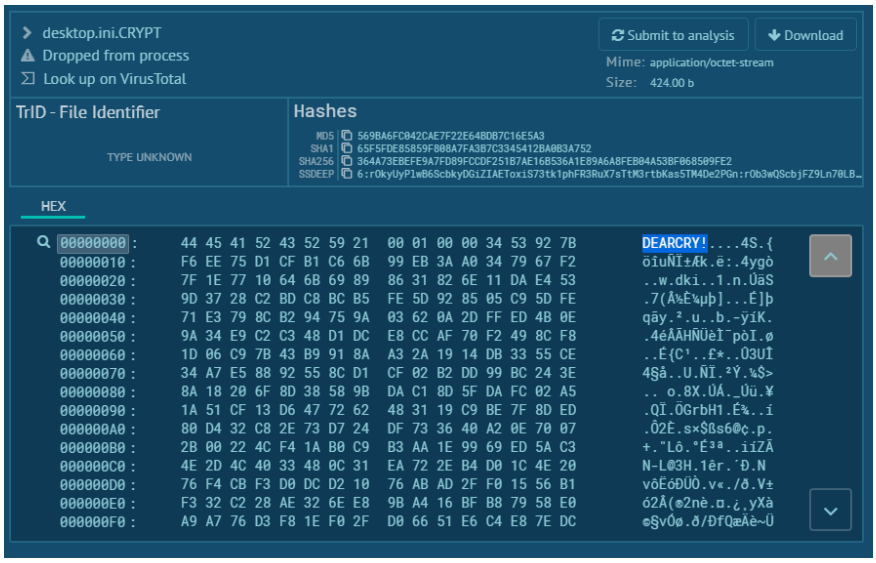

Figure 32: File with .CRYPT extension and "DEARCRY!" file marker

Also, file readme.txt contains the formatted ransom note message including the contact emails and hash of the RSA key.

Figure 33: Formatted ransom note with emails and hash of the RSA key

CONCLUSION

We introduced several principles of malware analysis and demonstrated them during the analysis of DearCry ransomware sample, which has been used in connection with the recent attacks on vulnerable Microsoft Exchange servers.

During this analysis, we spent most of the time with reverse engineering, including top-down and bottom-up methodologies for analysis of unknown programs. As a result, we provided overview of DearCry ransomware's logic and in-depth analysis of files encryption. We also covered all of the functions written by authors of this ransomware.

Last, but not least, if somebody is interested and wants to do his own analyze of DearCry ransomware, the analyzed sample is available on Malware Bazaar website and this paper can be used as a walkthrough for educational purposes.

RESOURCES

- DearCry Sample at Malware Bazaar
- Analysis in Any.Run sandbox
- LIFARS Video from the analysis and reverse engineering of DearCry ransomware sample
- https://lifars.com/wp-content/uploads/2021/03/Microsoft-Exchange-case-study.pdf
- Sysinternals strings tool
- FireEye capa tool
- Hex-Rays IDA Freeware Disassembler
- GitHub repository with community FLIRT signatures for IDA
- OpenSSL EVP_CipherInit documentation

244 Fifth Avenue, Suite 2035, New York, NY 10001
LIFARS.com (212) 222-7061 info@lifars.com

APPENDIX

Sample Information

File Name	e044d9f2d0f1260c3f4a543a1e67f33fcac265be114a1b135fd575b860d2b8c6.bin
File Size	1,322,496 bytes
Mime Type	application/x-dosexec
File Type	PE32 executable (console) Intel 80386, for MS Windows
MD5 hash	cdda3913408c4c46a6c575421485fa5b
SHA1 hash	56eec7392297e7301159094d7e461a696fe5b90f
SHA256 hash	e044d9f2d0f1260c3f4a543a1e67f33fcac265be114a1b135fd575b860d2b8c6
SSDeep hash	24576:C5Nv2SkWFP/529IC8u2bAs0NIzkQS+KpPbEasBY2iKDl1fpxkLVZgMCS+:oB70s9yjE62iIl1fpxkLVZgMC3
Imphash	f8b8e20e844ccd50a8eb73c2fca3626d

List of File Extensions

DearCry ransomware encrypts the files with the following extensions:

```
.TIF .TIFF .PDF .XLS .XLSX .XLTM .PS .PPS .PPT .PPTX .DOC .DOCX .LOG .MSG
.RTF .TEX .TXT .CAD .WPS .EML .INI .CSS .HTM .HTML  .XHTML .JS .JSP .PHP
.KEYCHAIN .PEM .SQL .APK .APP .BAT .CGI .ASPX .CER .CFM .C .CPP .GO
.CONFIG.CSV .DAT .ISO .PST .PGD  .7Z .RAR .ZIP .ZIPX .TAR .PDB .BIN .DB
.MDB .MDF .BAK .LOG .EDB .STM .DBF .ORA
```

RSA Public Key

```
-----BEGIN RSA PUBLIC KEY-----
MIIBCAKCAQEA5+mVBe75OvCzCW4oZHl7vqPwV2O4kgzgfp9odcL9LZc8Gy2+NJPD
wrHbttKI3z4Yt3G04lX7bEp1RZjxUYfzX8qvaPC2EBduOjSN1WMSbJJrINs1Izkq
XRrggJhSbp881Jr6NmpE6pns0Vfv//Hk1idHhxsXg6QKtfXlzAnRbgA1WepSDJq5
H08WGFBZrgUVM0zBYI3JJH3b9jIRMVQMJUQ57w3jZpOnpFXSZoUy1YD7Y3Cu+n/Q
6cEft6t29/FQgacXmeA2ajb7ssSbSntBpTpoyGc/kKoaihYPrHtNRhkMcZQayy5a
XTgYtEjhzJAC+esXiTYqklWMXJS1EmUpoQIBAw==
-----END RSA PUBLIC KEY-----
```

Ransomnote

```
Your file has been encrypted!
                                        If you want to decrypt, please contact us.
                                        konedieyp@airmail.cc or uenwonken@memail.com
                                        And please send me the following hash!
                                        d37fc1eabc6783a418d23a8d2ba5db5a
```