Challenges in Cyber Security: Ransomware Phenomenon

Vlad-Raul Paşca and Emil Simion

Abstract Ransomware has become one of the major threats nowadays due to its huge impact and increased rate of infections around the world. According to https://www.adaware.com/blog/cryptowall-ransomware-cost-users-325-million-in-2015, just one family, CryptoWall 3, was responsible for damages of over 325 millions of dollars, since its discovery in 2015. Recently, another family of ransomware appeared in the cyberspace which is called WannaCry, and according to https://www.cnet.com/news/wannacry-wannacrypt-uiwix-ransomware-everything-you-need-to-know, over 230,000 computers around the world, in over 150 countries, were infected. This type of ransomware exploited a vulnerability which is present in the Microsoft Windows operating systems called EternalBlue, an exploit which was developed by the US National Security Agency (NSA) and released by The Shadow Brokers on April 14, 2017.

Spora ransomware is a major player in the field of ransomware families and is prepared by professionals. It has the ability to encrypt files offline like other families of ransomware, DMA Locker 3.0, Cerber, or some editions of Locky. Currently, there is no decryptor available in the market for the Spora ransomware.

Spora is distributed using phishing e-mails and infected websites which drops malicious payloads. There are some distribution methods which are presented in http://malware-traffic-analysis.net/2017/02/14/index2.html (the campaign from February 14, 2017) and http://malware-traffic-analysis.net/2017/03/06/index.html (the campaign from March 6, 2017).

Once the infection has begun, Spora runs silently and encrypts files with a specific extension, not all extensions are encrypted. This type of ransomware is interested in office documents, PDF documents, Corel Draw documents, database files, images, and archives and is important to present the entire list of extension in order to warn people about this type of attack: xls, doc, xlsx, docx, rtf, odt, pdf, psd, dwg, cdr, cd, mdb, 1cd, dbf, sqlite, accdb, jpg, jpeg, tiff, zip, rar, 7z, backup, sql,

V.-R. Paşca (✉) · E. Simion
Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Bucharest, Romania
e-mail: emil.simion@upb.ro

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_15

and bak. One crucial point here is that everybody can rename the files in order to avoid such infections, but the mandatory requirement is to back up the data.

Spora doesn't add extensions to the encrypted files, which is really unusual in the case of ransomware, for example, Locky adds .locky extension, TeslaCrypt adds .aaa extension, and WannaCry appends .WNCRY extension. In this case, each file is encrypted with a separate key, and it is a nondeterministic encryption (files with an identical content are encrypted in different ciphertexts); the content which was encrypted has a high entropy and visualization of an encrypted file, which suggests that a stream cipher or chained block was used (AES in CBC mode is suggested, because of the popularity of this mode of operation in ransomware's encryption schemes).

There are some methods which are used frequently to assure that a single copy of a malware is running, for example, the creation of a mutex, which means that the encrypted data is not encrypted again; therefore, we have a single step of encryption. Of course, there are some folders which are excluded from encryption, because the system must remain in a working state in order to make a payment, so Spora doesn't encrypt the files which are located in the following directories: windows, program files, program files (x86), and games.

Spora uses Windows Crypto API for the whole encryption process. Firstly the malware comes with a hardcoded AES 256 key, which is being imported using CryptImportKey (the parameters which are passed to this function reveal that an AES 256 key is present). The AES key is further used to decrypt another key, which is a RSA public key, using a CryptDecrypt function (a ransom note is also decrypted using the AES key, as well as a hardcoded ID of the sample).

For every computer, Spora creates a new pair of RSA keys. This process uses the function CryptGenKey with some parameters which are specific for RSA keys, after that the private key from the pair is exported using the function CryptExportKey and Base64 encoded using the function CryptBinaryToString. A new AES 256 key is generated using CryptGenKey, is exported using CryptExportKey, and is used to encrypt the generated private RSA key (finally, the key is encrypted using the hardcoded RSA public key and stored in the ransom note). For every file a new AES key is generated which is used to encrypt the file, is encrypted using the generated public RSA key, and is stored at the end of every encrypted file.

Spora is a professional product created by skilled attackers, but the code is not obfuscated or packed, which makes the analysis a little bit easier. The implementation of cryptographic algorithms uses the Windows Crypto API and seems to be consistent; nonetheless the decryption of files is not really possible without paying the ransom. The ability to handle a complex process of encryption offline makes Spora ransomware a real danger for unprepared clients.

Ransomware usually uses the RSA algorithm to protect the encryption key and AES for encrypting the files. If these algorithms are correctly implemented, then it is impossible to recover the encrypted information.

Some attacks, nonetheless, work against the implementation of RSA. These attacks are not against the basic algorithm, but against the protocol. Examples of such attacks on RSA are chosen-ciphertext attack, common modulus attack, low encryption exponent attack, low decryption exponent attack, attack on encryption and signing with the same pair of keys, and attack in case of small difference between prime numbers p and q.

The attacks on AES implementation include ECB attack, CBC implementation without HMAC verification and oracle padding attack.

In the following sections, we present the fully analysis on three representative ransomware: Spora, DMA Locker, and WannaCry.

1 Spora Ransomware

Name: 9ae49d4a4202b14efe.exe
md5: 116d339b412cd1baf48bcc8e4124a20b
Type: encrypting ransomware

In Fig. 1 a detection report by VirusTotal scanner mechanism is presented, which shows that the malware is known and most vendors already offer a protection mechanism for it. Figure 2 shows us that the malware itself is not packed; nonetheless later results will show that the malware is obfuscated and hence the complexity of the analysis grows.

Figure 3 shows a string which is pushed on the stack 699 times; this trick is used to obfuscate the code.

In Fig. 4 it is shown that a function is called 700 times (the function calls **OpenMutexA**, which tries to open an existing mutex), which doesn't make sense

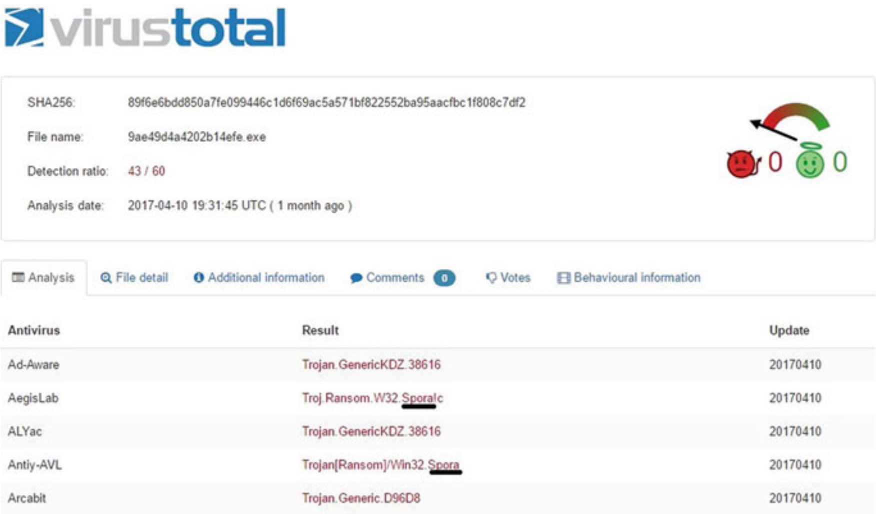

Fig. 1 VirusTotal report

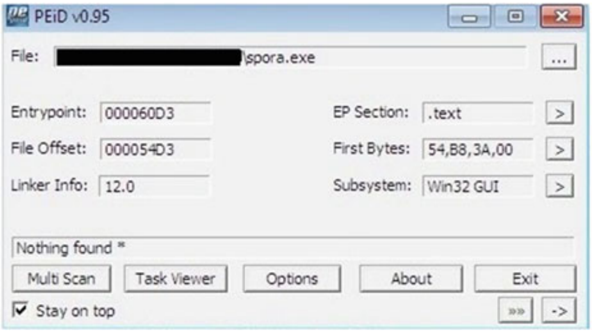

Fig. 2 PEiD report

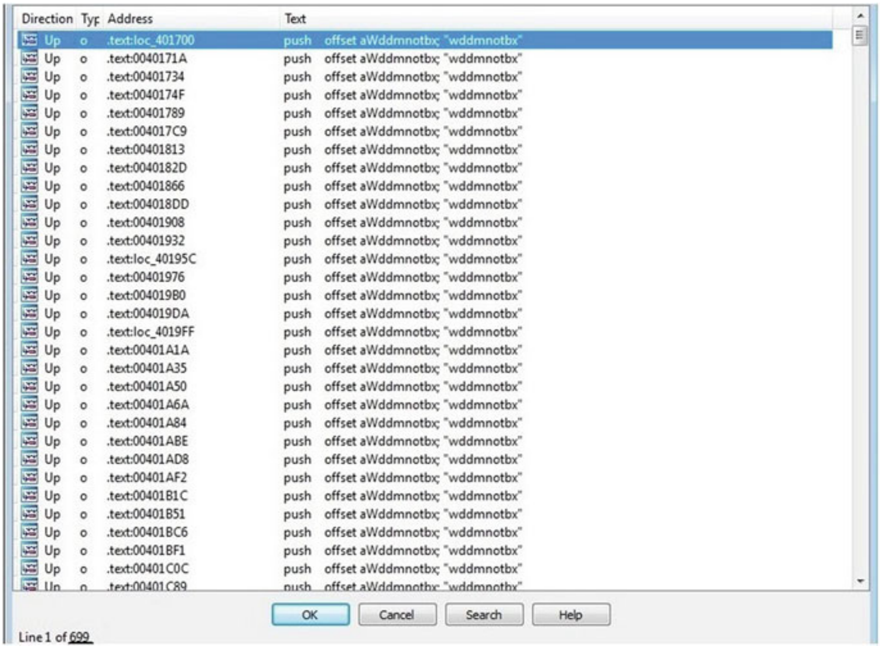

Fig. 3 IDA Pro 1

in this case, because the malware doesn't call **CreateMutexA**; this is another trick used to complicate the analysis. The malware uses the function **VirtualAlloc** to allocate space in the process address space, and then it writes the actual payload in that space. The initial conclusion is that the initial executable is just a packer and the actual malicious code is contained in the newly executable, which has the md5 97e84cc8afca475d15d8c3e1f38d deba.

The malware calls **GetVolumeInformationW** to get information about the file system and volume associated with the root directory, as shown in Fig. 5.

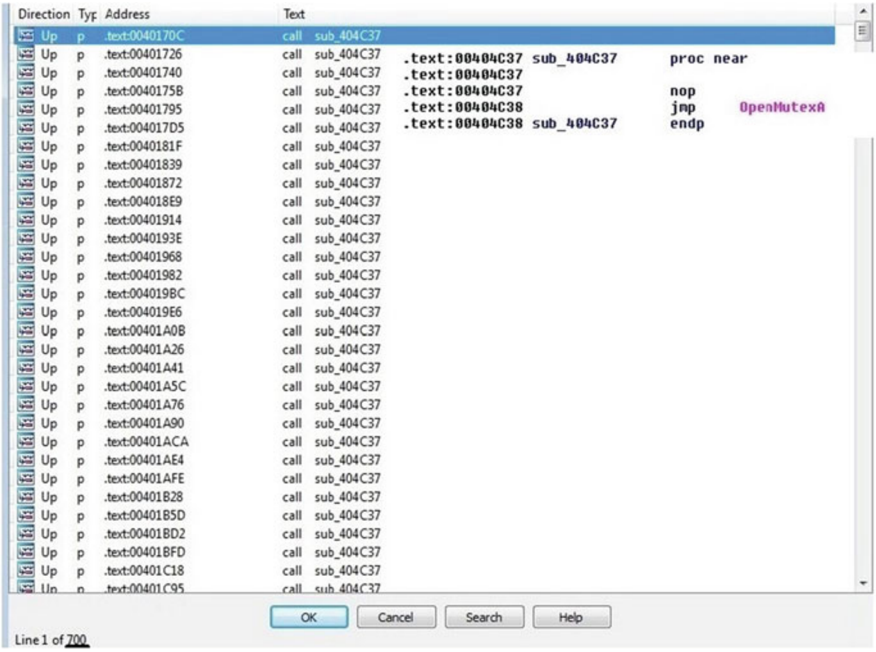

Fig. 4 IDA Pro 2

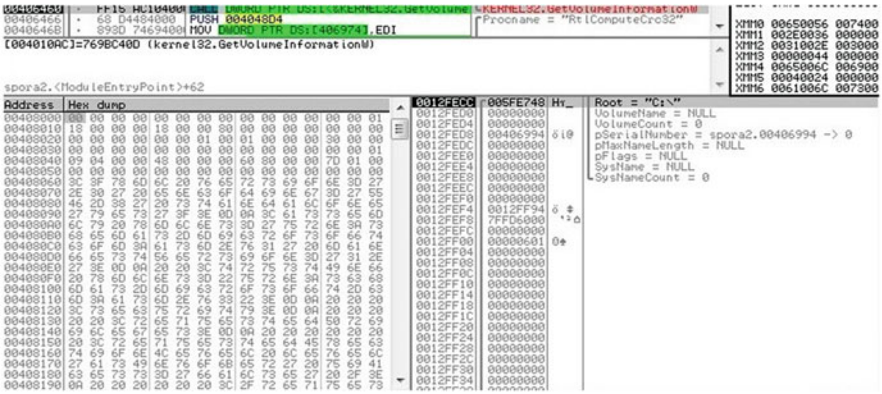

Fig. 5 GetVolumeInformationW call

A mutex is created and it has the following format: m⟨GetVolumeInformationResult⟩ (in decimal), to ensure that the malware runs only once. The sample creates a file which has the following name: C:\Users\⟨$user$⟩\AppData\Roaming\⟨$Mutex$⟩. The malware comes with a hardcoded key, which is being imported using the function **CryptImportKey**, as shown in Fig. 6. It represents an AES256 key, stored in a form of a blob. The

Fig. 6 CryptImportKey call

explanation of the fields is: 08 represents PLAINTEXTKEYBLOB and means that the key is a session key; 02 CUR_BLOB_VERSION, 0x00006610 which represents Alg_ID: CALG_AES_256, 0x20=32 represents key length.

The AES key is used to decrypt another key, which is a RSA key embedded in the binary, as shown in Fig. 7. The AES key is also used to decrypt the ransom note and the binary's hardcoded ID. The malware uses **GetLogicalDrives** to obtain the currently available disk drives and then a loop, which selects the files that have a specified extension which is attacked by this ransomware, is created. The malware also uses **WNetOpenEnum** and **WNetEnumResource** APIs to enumerate the network resources, and the created file is used to store temporary data, like the files which will be encrypted.

Fig. 7 CryptDecrypt calls

The attacked extensions are presented in the table below:

.xls	.doc	.xlsx	.docx	.rtf	.odt	.pdf	.ppt	.pptx
.psd	.dwg	.cdr	.cd	.mdb	.1cd	.dbf	.sqlite	.accdb
.jpg	.jpeg	.tiff	.zip	.rar	.7z	.backup	.sql	.bak

The next folders are excluded from the attack:

windows	programfiles	programfiles(x86)	games

For every victim, the malware creates a pair of RSA keys. The fragment which generates the RSA key pair (1024 bits) is shown in Fig. 8.

The relevant parameters for **CryptGenKey** are 0xA400 which represents AlgId: CALG_RSA_KEYX and 0x04000001 which represents RSA1024BIT_KEY | CRYPT_EXPORTABLE. The private RSA key is exported and Base64 encoded, as shown in Fig. 9. The encryption of the private RSA key is stored into a buffer alongside the data regarding the machine and the infection, like date, username, country code, malware ID, and statistics of encrypted file types. An example is shown in Fig. 10. The malware uses a MD5 algorithm to hash the buffer which contains the private RSA key (the hash is used to create the user ID) as shown in Fig. 11. Another AES key is generated; then it's exported and encrypted using public RSA key that was hardcoded. In Fig. 12 this process is shown. The generated AES key is used to encrypt the data (including the RSA private key), as shown in Fig. 13. Finally, all encrypted data is Base64 encoded and stored in the ransom note.

Fig. 8 CryptGenKey call

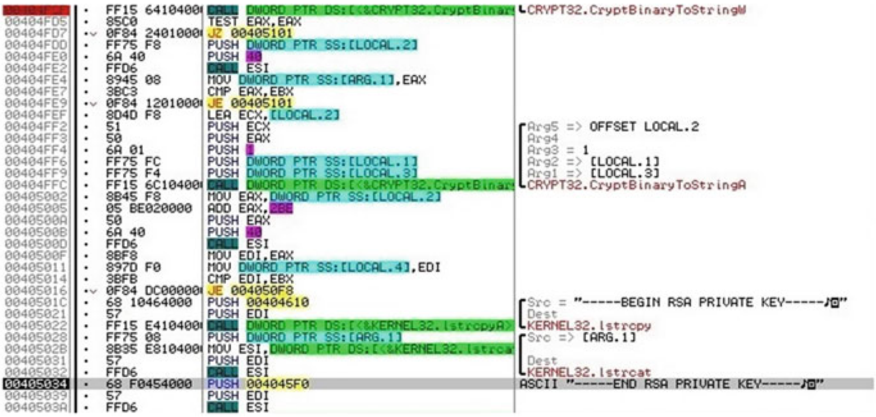

Fig. 9 RSA key is Base64 encoded

For every file a new AES256 key is generated, as shown in Fig. 14. The AES key is encrypted using the generated public RSA key, and it is appended to the encrypted file; also the CRC32 is being computed and stored in the file (Fig. 15). Each file is encrypted using the AES key, as shown in Fig. 16.

In order to decrypt a file, a ransom note is uploaded to the server giving the attacker access to all information needed. He uses the private RSA key corresponding to the hardcoded public RSA key to decrypt the first AES key, and then the key is used to decrypt the generated private RSA key. Because of the fact that each AES256 key is encrypted using the corresponding public RSA key and stored at the end of each file, it is possible to decrypt each key and then decrypt each file individually.

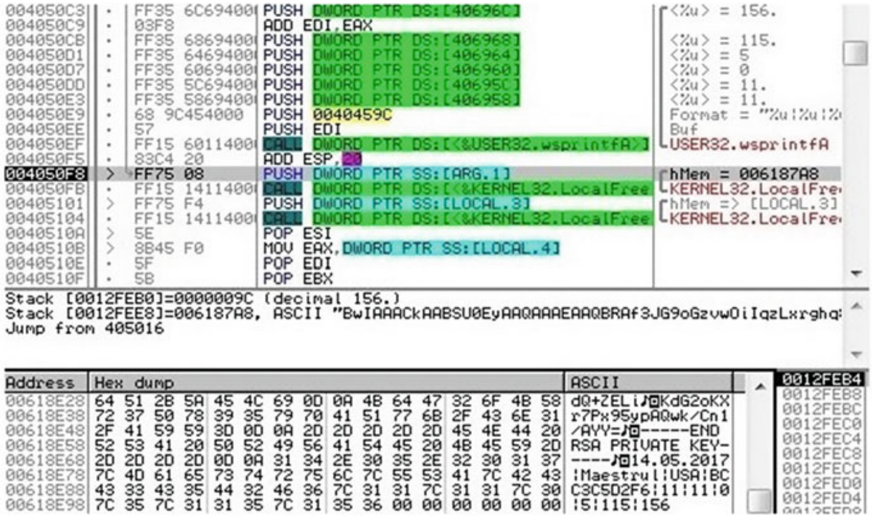

Fig. 10 Buffer contains information about the system

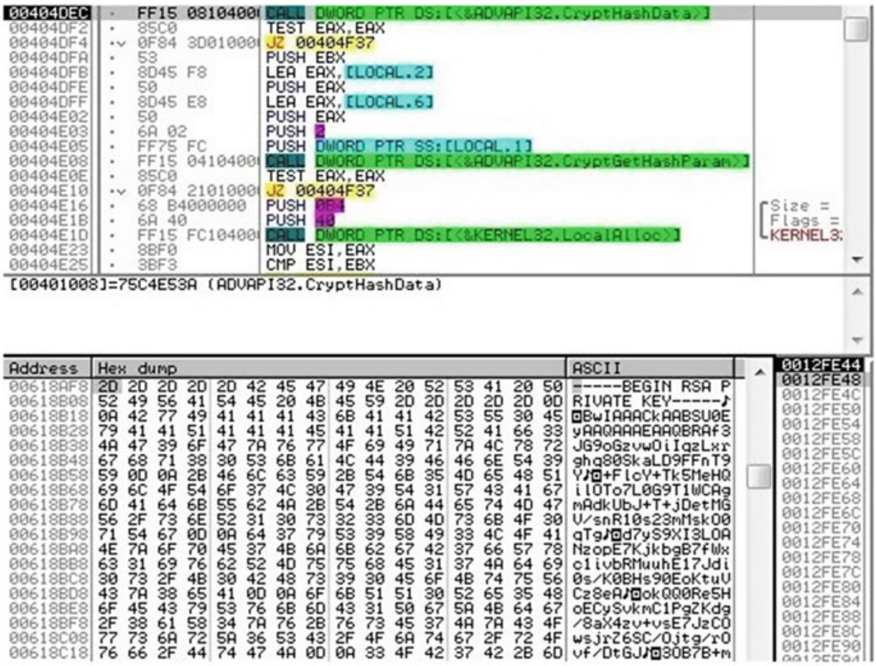

Fig. 11 MD5 Algorithm is used to hash the buffer

Fig. 12 Another AES key is generated, exported, and encrypted using the embedded RSA key

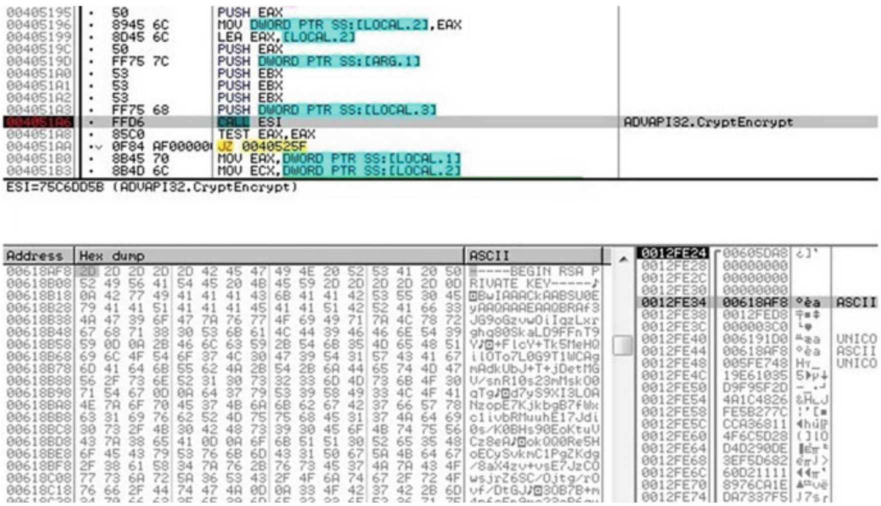

Fig. 13 The AES key, which was generated, is used to encrypt a private RSA key

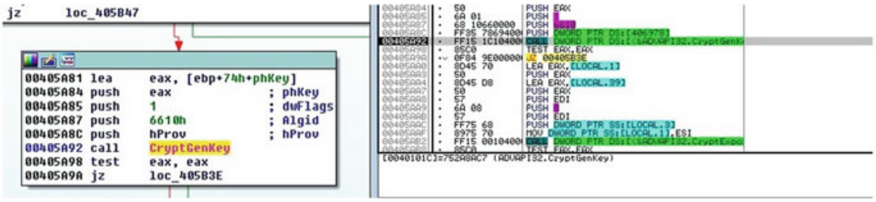

Fig. 14 Another AES256 key is generated

Fig. 15 The AES key is encrypted using RSA key

Fig. 16 The file is encrypted using the AES key

2 DMA Locker Ransomware

Name: dma.exe
md5: FDECD41824E51F79DE6A25CDF62A04B5
Type: encrypting ransomware

In Fig. 17 a report by VirusTotal, which shows that the malware is known to most vendors, is presented.

According to Fig. 18, the ransomware isn't packed; if this is obfuscated, it is then necessary to reveal it. As shown in Fig. 19, the malware moves the original file to C:\ProgramData and renames the file svchosd.exe (the author of ransomware is trying to hide the malicious purposes, in order to look like the Service Host Process svchost.exe). Once the file is copied, the malware

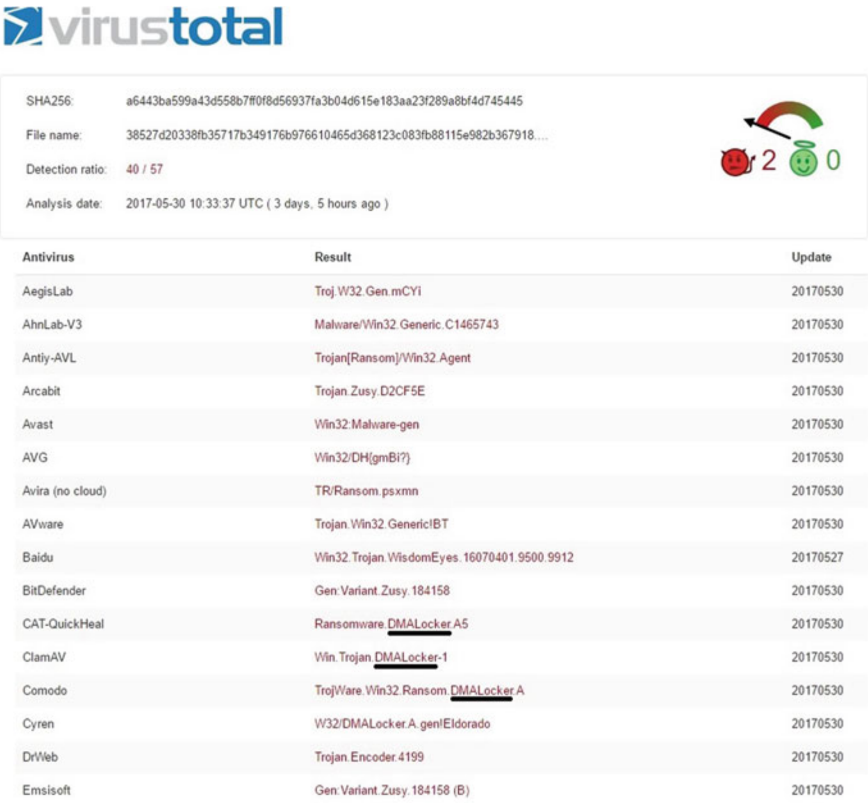

Fig. 17 VirusTotal report DMA Locker

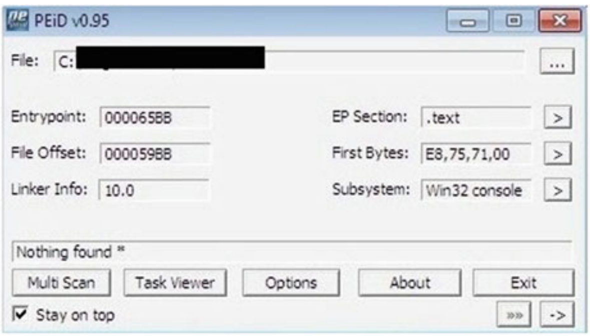

Fig. 18 PEiD report DMA Locker

Fig. 19 The malware moves the original file to another location

starts svchosd.exe process (which obviously is a copy of the original process) and then exits. As shown in Fig. 20, the function **CreateProcessW** is used. The original process creates two keys in registry for persistence:**HKLM\Software\Microsoft\Windows\CurrentVersion\Run\Windows Firewall** which has the value C:\ProgramData\svchosd.exe and **HKLM\Software\Microsoft\Windows\CurrentVersion\ Run\Windows Update**, which has the value notepad C:\ProgramData\cryptinfo.txt (at every reboot the ransom note is shown). The DMA Locker deletes backups and shadow copies, using the native Windows utility VSSAdmin, as shown in Fig. 21. A start.text file is created to show that the encryption has begun (and there is no need to restart it again). Logical disks and network shares are attacked, and checks against the Floppy and CD using **QueryDosDeviceA**(Floppy and CD are skipped) are made, as shown in Fig. 22. The sample uses a hardcoded public RSA key, stored in a form of BLOB, as shown in Fig. 23. Some directories are excluded from the encryption process; this entire list is in Fig. 24. A list of skipped extensions is presented in Fig. 25. A unique AES256 key is generated for every file using the API **CryptGenRandom**, as shown in Fig. 26. The AES key is used to encrypt 16-byte-long data with AES ECB mode, as shown in Fig. 27. Once used, the AES key is encrypted using the hardcoded RSA key (Fig. 28). The structure of the encrypted file is the prefix which is added, the encrypted AES key, and the encrypted original content (Fig. 29). Once the encryption process is complete, a message alert is presented (Fig. 30). The malware may be fooled in order to avoid the encryption through the creation of the files start.txt and cryptinfo.txt in ProgramData directory. If these two files are present, the encryption cannot start and only the ransom message is displayed. However, if the algorithms, which are used in the encryption process, are consistent, the decryption without the RSA private key which is kept secret will not be possible.

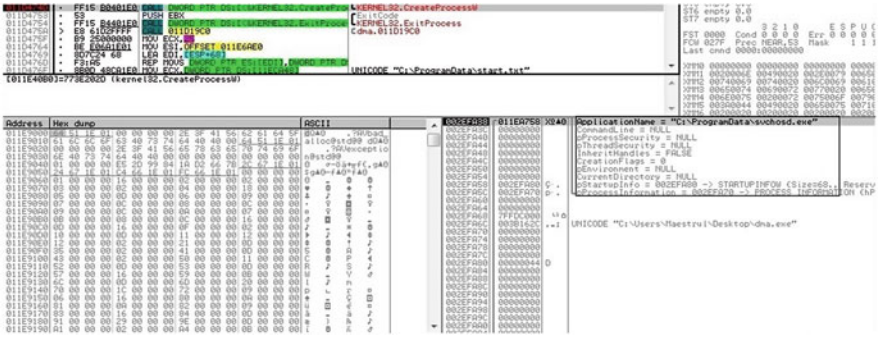

Fig. 20 The malware starts a copy of the original process

Fig. 21 DMA Locker deletes backups and shadow copies

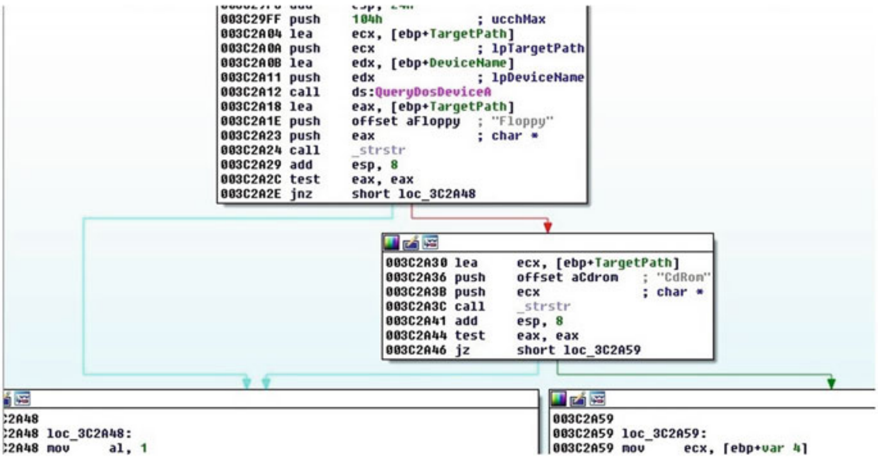

Fig. 22 Floppy and CD are skipped

```
Address  Hex dump
0018F9F8 06 02 00 00 00 A4 00 00 52 53 41 31 00 04 00 00
0018FA08 01 00 01 00 A3 4A 8F 85 FA 88 7F FD A1 4D 32 FF
0018FA18 EE BB 5D 14 7E 5C 1F 92 3D F9 16 B6 3D 61 9D 63
0018FA28 41 E8 4A 61 F2 79 48 DA A2 96 31 39 E9 BE BE 80
0018FA38 7C 23 A3 20 FD 73 64 7E 83 00 7C C1 A2 58 91 C6
0018FA48 52 6A 86 FE EB B3 9B 6C 4B FB 8A 5E F2 60 E0 92
0018FA58 DB 52 02 A0 2C 18 8D 6C 6B 4C 28 FA 74 28 D9 08
0018FA68 0A AF C4 29 BD 1F 11 AA A4 73 0C 1F 08 58 3A AF
0018FA78 9B 24 51 2C C8 5D AB A5 9A 58 84 B4 AF 80 AE 4D
0018FA88 C5 ED 0D B6 00 D7 3C 00 A4 FA 18 00 4F 03 74 6A
0018FA98 0F D7 3C 00 E4 FA 18 00 65 65 3C 00 01 00 00 00
0018FAA8 78 18 55 00 B8 18 55 00 37 00 74 6A 00 00 00 00
0018FAB8 00 00 00 00 00 80 FD 7F D0 FA 18 00 00 00 00 00
0018FAC8 00 00 00 00 B0 FA 18 00 69 DF 66 B7 20 FB 18 00
0018FAD8 20 88 3C 00 63 94 51 6A 00 00 00 00 F0 FA 18 00
0018FAE8 74 11 43 77 00 80 FD 7F 30 FB 18 00 F5 B3 51 77
0018FAF8 00 80 FD 7F 54 54 69 77 00 00 00 00 00 00 00 00
0018FB08 00 80 FD 7F 00 00 00 00 00 00 00 00 00 00 00 00
0018FB18 FC FA 18 00 00 00 00 00 FF FF FF FF 4D D7 4D 77
0018FB28 FC AA 20 00 00 00 00 00 48 FB 18 00 C8 B3 51 77
0018FB38 BB 65 3C 00 00 80 FD 7F 00 00 00 00 00 00 00 00
0018FB48 00 00 00 00 00 00 00 00 BB 65 3C 00 00 80 FD 7F
0018FB58 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0018FB68 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0018FB78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0018FB88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Fig. 23 Hardcoded RSA key

```
003C2880 push    ebp
003C2881 mov     ebp, esp
003C2883 sub     esp, 2Ch
003C2886 push    esi
003C2887 mov     [ebp+var_2C], offset aWindows ; "\\Windows\\"
003C288E mov     [ebp+var_28], offset aWindows_0 ; "\\WINDOWS\\"
003C2895 mov     [ebp+var_24], offset aProgramFiles ; "\\Program Files\\"
003C289C mov     [ebp+var_20], offset aProgramFilesX8 ; "\\Program Files (x86)\\"
003C28A3 mov     [ebp+var_1C], offset aGames ; "Games"
003C28AA mov     [ebp+var_18], offset aTemp ; "\\Temp"
003C28B1 mov     [ebp+var_14], offset aSamplePictures ; "\\Sample Pictures"
003C28B8 mov     [ebp+var_10], offset aSampleMusic ; "\\Sample Music"
003C28BF mov     [ebp+var_C], offset aCache ; "\\cache"
003C28C6 mov     [ebp+var_8], offset aCache_0 ; "\\Cache"
003C28CD xor     esi, esi
003C28CF nop
```

Fig. 24 The directories which are excluded from the encryption

```
003C2907 mov     [ebp+var_30], offset a_exe ; ".exe"
003C290E mov     [ebp+var_2C], offset a_msi ; ".msi"
003C2915 mov     [ebp+var_28], offset a_dll ; ".dll"
003C291C mov     [ebp+var_24], offset a_pif ; ".pif"
003C2923 mov     [ebp+var_20], offset a_scr ; ".scr"
003C292A mov     [ebp+var_1C], offset a_sys ; ".sys"
003C2931 mov     [ebp+var_18], offset a_msp ; ".msp"
003C2938 mov     [ebp+var_14], offset a_com ; ".com"
003C293F mov     [ebp+var_10], offset a_lnk ; ".lnk"
003C2946 mov     [ebp+var_C], offset a_hta ; ".hta"
003C294D mov     [ebp+var_8], offset a_cpl ; ".cpl"
003C2954 mov     [ebp+var_4], offset a_msc ; ".msc"
```

Fig. 25 Skipped extensions

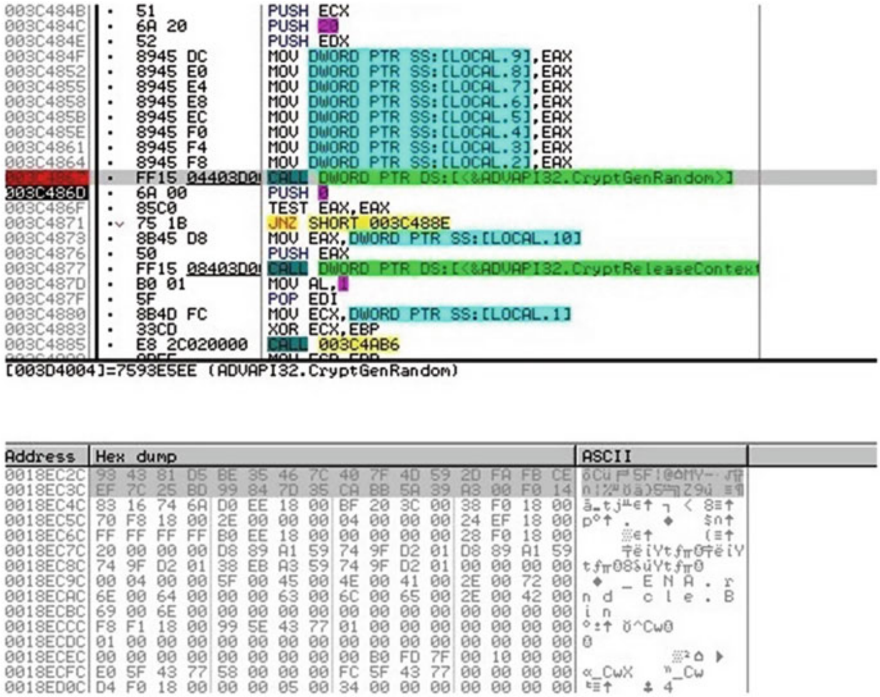

Fig. 26 A unique AES key is generated

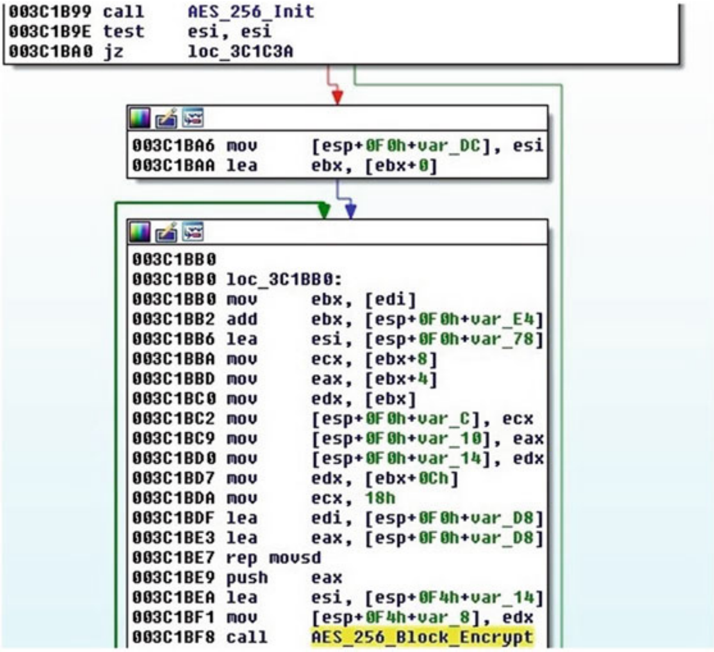

Fig. 27 The data is split in chunks of 16 bytes and encrypted

Fig. 28 The AES key is encrypted using the hardcoded RSA key

Fig. 29 A prefix is added to each file

Fig. 30 DMA Locker Message

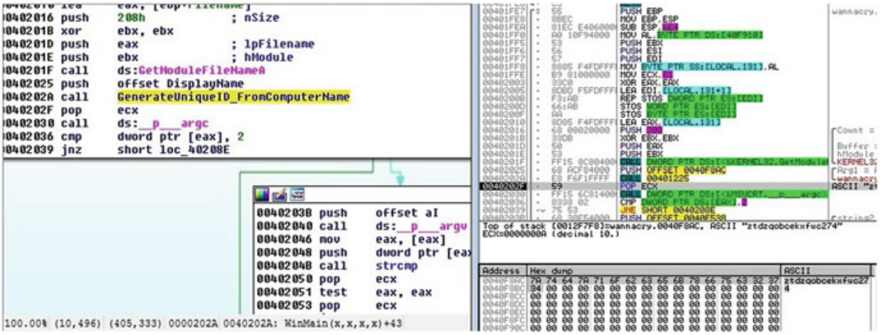

Fig. 31 A unique identifier is generated for every victim

Fig. 32 Ransom notes

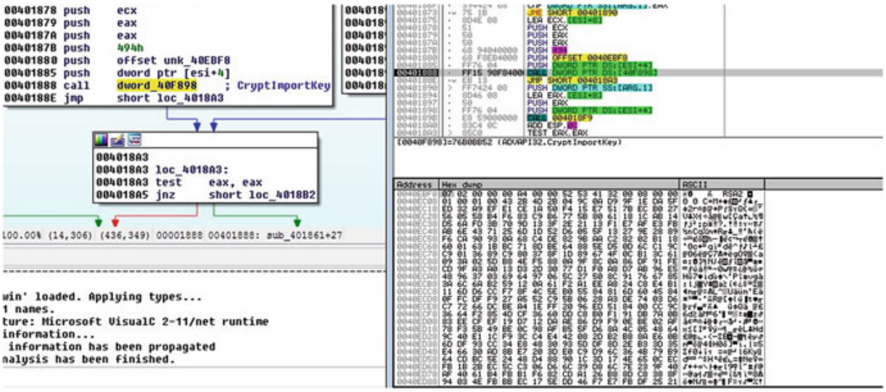

Fig. 33 Private RSA key is being imported

Fig. 34 The encrypted key is decrypted using private RSA key

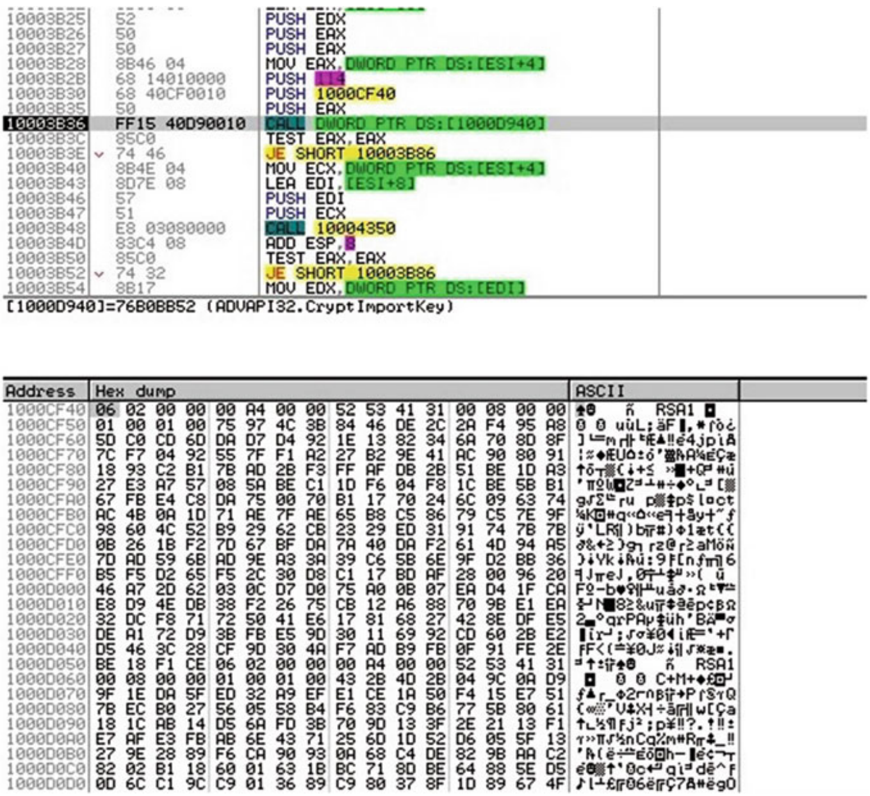

Fig. 35 Public RSA key is being imported

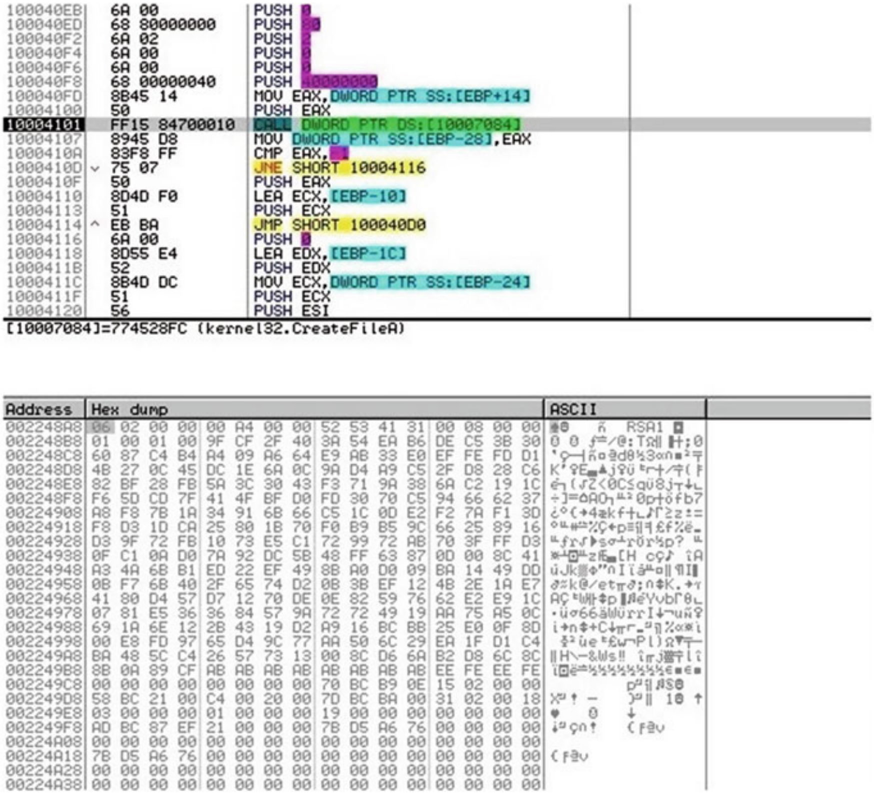

Fig. 36 The public key is exported and saved to 00000000.pky

Fig. 37 The private key is encrypted using hardcoded RSA key

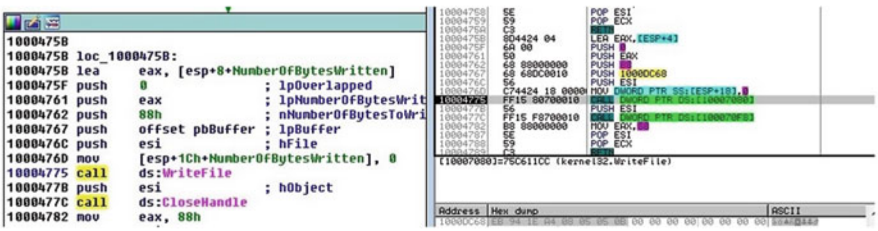

Fig. 38 Firstly, the 8 generated bytes and 128 zero bytes are written to the file

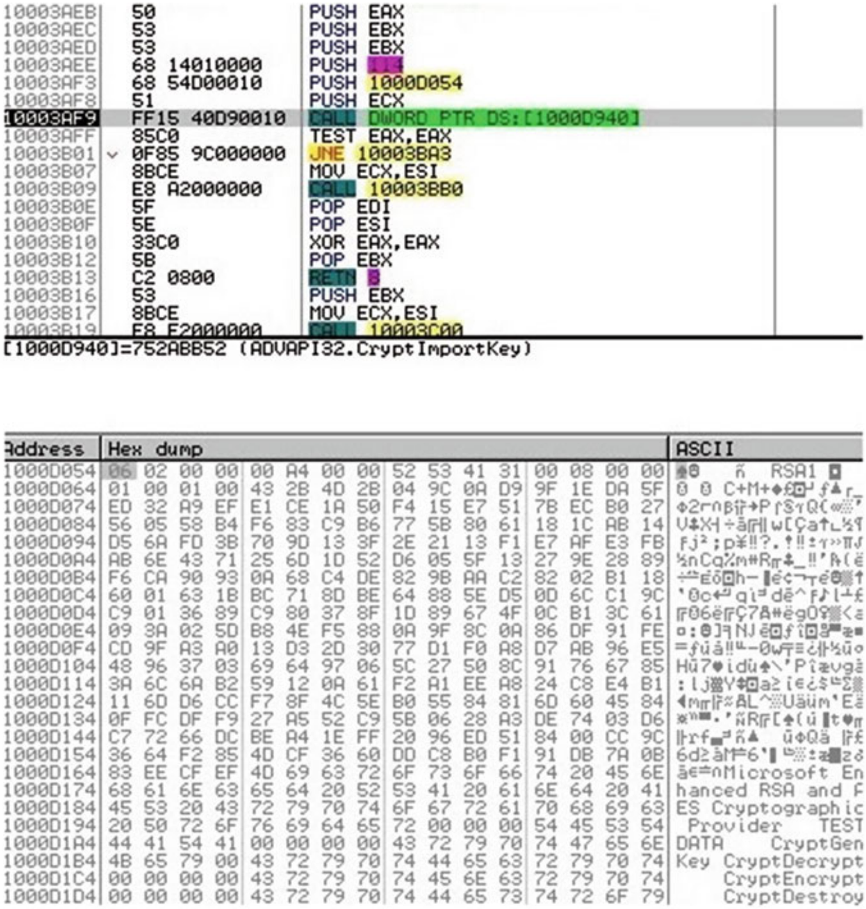

Fig. 39 Another RSA key is being imported

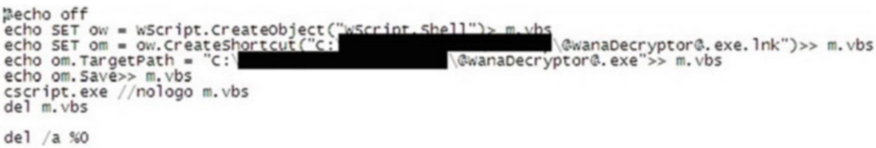

Fig. 40 The malware creates a LNK which points to @WanaDecryptor@.exe

```
.der.pfx.key.crt.csr.p12.pem.odt.ott.sxw.stw.uot.3ds.max.3dm.ods.ots.sxc
.stc.dif.slk.wb2.odp.otp.sxd.std.uop.odg.otg.sxm.mml.lay.lay6.asc.sqlite3
.sqlitedb.sql.accdb.mdb.db.dbf.odb.frm.myd.myi.ibd.mdf.ldf.sln.suo.cs
.c.cpp.pas.h.asm.js.cmd.bat.ps1.vbs.vb.pl.dip.dch.sch.brd.jsp.php.asp.rb
.java.jar.class.sh.mp3.wav.swf.fla.wmv.mpg.vob.mpeg.asf.avi.mov.mp4.3gp.mkv
.3g2.flv.wma.mid.m3u.m4u.djvu.svg.ai.psd.nef.tiff.tif.cgm.raw.gif.png.bmp
.vcd.iso.backup.zip.rar.7z.gz.tgz.tar.bak.tbk.bz2.PAQ.ARC.aes.gpg.vmx.vmdk
.vdi.sldm.sldx.sti.sxi.602.hwp.edb.potm.potx.ppam.ppsx.ppsm.pps.pot.pptm.xltm
.xltx.xlc.xlm.xlt.xlw.xlsb.xlsm.dotx.dotm.dot.docm.docb.jpg.jpeg.snt.onetoc2
.dwg.pdf.wk1.wks.123.rtf.csv.txt.vsdx.vsd.eml.msg.ost.pst.pptx.ppt.xlsx.xls.docx
.doc
```

Fig. 41 Targeted extensions by malware

Fig. 42 A new AES key is generated for every file

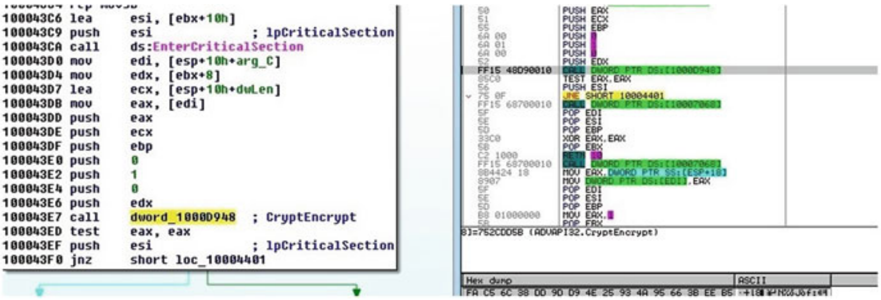

Fig. 43 The AES key is encrypted using RSA key

```
100058D3 push    0                  ; lpExitCode
100058D5 push    0                  ; dwMilliseconds
100058D7 push    offset aTaskkill_exeFI ; "taskkill.exe /f /im Microsoft.Exchange.".
100058DC call    sub_10001080
100058E1 push    0                  ; lpExitCode
100058E3 push    0                  ; dwMilliseconds
100058E5 push    offset aTaskkill_exe_0 ; "taskkill.exe /f /im MSExchange*"
100058EA call    sub_10001080
100058EF push    0                  ; lpExitCode
100058F1 push    0                  ; dwMilliseconds
100058F3 push    offset aTaskkill_exe_1 ; "taskkill.exe /f /im sqlserver.exe"
100058F8 call    sub_10001080
100058FD push    0                  ; lpExitCode
100058FF push    0                  ; dwMilliseconds
10005901 push    offset aTaskkill_exe_2 ; "taskkill.exe /f /im sqlwriter.exe"
10005906 call    sub_10001080
1000590B push    0                  ; lpExitCode
1000590D push    0                  ; dwMilliseconds
1000590F push    offset aTaskkill_exe_3 ; "taskkill.exe /f /im mysqld.exe"
10005914 call    sub_10001080
10005919 add     esp, 3Ch
```

Fig. 44 Executed commands after the encryption is over

3 WannaCry Ransomware

Name: diskpart.exe
md5: 84c82835a5d21bbcf75a61706d8ab549
Type: encrypting ransomware

The malware generates a unique identifier based on the computer name, as shown in Fig. 31. A check is made to see if the malware was started with /i argument.

Run with /i Argument

The malware copies the binary to C:\ProgramData\ $\langle GeneratedID \rangle$ \ tasksche.exe if the directory exists; otherwise it is copied to C:\Intel \ $\langle GeneratedID \rangle$\tasksche.exe and updates the current directory to the new directory. The binary tries to open the service named $\langle GeneratedID \rangle$. If it doesn't exist, the malware creates one with DisplayName $\langle GeneratedID \rangle$, the BinaryPath of cmd \c $\langle PathOftasksche.exe \rangle$, and starts the service. It attempts to open the mutex Global\MsWinZonesCacheCounterMutexA0; if it isn't created within 60 s, the malware starts itself with no arguments.

Run Without /i Argument

The binary updates the current directory to the path of the module and creates a new registry key HKLM\Software\WanaCrypt0r\wd which is set to the CD. The malware then loads the XIA resource and extracts multiple files to the current directory; the complete list is shown below:

Filename	MD5 hash
b.wnry	c17170262312f3be7027bc2ca825bf0c
c.wnry	ae08f79a0d800b82fcbe1b43cdbdbefc
r.wnry	3e0020fc529b1c2a061016dd2469ba96
s.wnry	ad4c9de7c8c40813f200ba1c2fa33083
t.wnry	5dcaac857e695a65f5c3ef1441a73a8f
u.wnry	7bf2b57f2a205768755c07f238fb32cc
taskdl.exe	4fef5e34143e646dbf9907c4374276f5
taskse.exe	8495400f199ac77853c53b5a3f278f3e

The msg directory is created with different ransom notes in multiple languages (Fig. 32). The ransomware opens c.wnry (configuration data) and loads it into memory. The malware chooses between three bitcoin addresses, 13AM4VW2dhxYgXeQepoHkH SQuy6NgaE b94, 12t9YDPgwueZ9NyMgw519-p7AA8isjr6SMw, and 115p7UMMngoj1 pMvkpHijcRdfJNXj 6LrLn, writes it to offset 0xB2 in the config data, and writes the updates back to c.wnry. The binary sets a hidden attribute to the current directory using CreateProcessA API with **attrib +h** and executes the command **icacls ./grant Everyone:F /T /C /Q** in order to grant all users permissions to the current directory.

The malware uses **CryptImportKey** to import the hardcoded private RSA key (Fig. 33). The file t.wnry is then opened and the first 8 bytes are compared with the magic value "WANACRY!"; the next 4 bytes need to be 0x100; then the next 256 bytes are written in memory. The encrypted key decrypts to the AES key BEE19B98D2E5B12 211CE211EECB13DE6, as shown in Fig. 34. The AES key is used to decrypt the encrypted data, which was read from t.wnry and saves the result as a DLL. The TaskStart export function of the DLL is called, and it deals with the encryption of the files. It creates a mutex which is called **MsWinZonesCacheCounterMutexA** and reads the configuration file c.wnry. A new mutex is then created by the ransomware, **Global\MsWinZonesCacheCounterMutexA0**.

The binary will try to open a file 00000000.dky file, which at this point doesn't exist, and it will then try to load a 00000000.pky file. If this one doesn't exist, the ransomware will then import a public RSA key, as shown in Fig. 35. A new pair of RSA2048 keys is generated and the public key is saved to 00000000.pky, as shown in Fig. 36. The malware uses the hardcoded RSA key to encrypt the generated private key and saves the result to 00000000.eky (Fig. 37). A thread that writes 136 bytes to 00000000.key is created every 25 s (if it exists, otherwise it is created). Initially, as shown Fig. 38, 8 generated bytes and 128 zero bytes are written to the file, and after that it is written to a buffer, which contains the current time of the system. A thread that launched taskdl.exe, which is used to delete encrypted files, is created (which has that specific extension). Another thread is created that scans for new drives every 3 s; if it finds a new drive and this isn't a CDROM drive, it

encrypts the drive. The sample imports another RSA key, as shown in Fig. 39. The process @WanaDecryptor@.exe with the “fi” argument is created, and this one can communicate with the server in order to obtain an updated bitcoin address. The file u.wrny is copied and saved as @WanaDecryptor@.exe; a script file is created and executed with the content shown below. The ransomware reads the content of r.wnry, updates the content with a ransom amount and bitcoin address, and writes the content to @Please_Read_Me@.txt (Fig. 40). The process starts scanning a directory, creates a hidden file with the prefix “∼SD,” and then deletes it. The files which have the .exe, .dll, and .WNCRY extensions as well as the files which were created by the malware are not encrypted. The list of attacked extensions is presented in Fig. 41. Each file is encrypted using AES-128 algorithm in CBC mode with NULL IV. For every file a unique AES key is generated, as is shown in Fig. 42. The structure of an encrypted file is WANACRY!, length of RSA encrypted data, RSA encrypted AES key, file type, original file size, and AES encrypted content. The AES key is encrypted using the embedded RSA key or generated RSA key depending on a number which is generated (if it is a multiple of 100, the AES key is encrypted using the embedded RSA key; otherwise it is encrypted using the generated RSA public key), as shown in Fig. 43. The ransomware executes the following commands after the encryption is finished (Fig. 44). The process is trying to encrypt the logical drives that aren’t of DRIVE_CD
ROM type; it executes the commands **@WanaDecryptor@.exe co** and **cmd.exe /c start /b @WanaDecryptor@.exe vs** and copies the b.wnry to every folder on the desktop (it is saved as @WanaDecryptor@.bmp). The encryption algorithms are consistent and it is not possible to restore the files without paying the ransom; however there are some decryptors that work for Windows XP, Windows 7, Windows Vista, and Windows Servers 2003 and 2008.

Acknowledgment The authors would like to thank University Politehnica of Bucharest for the financial support.