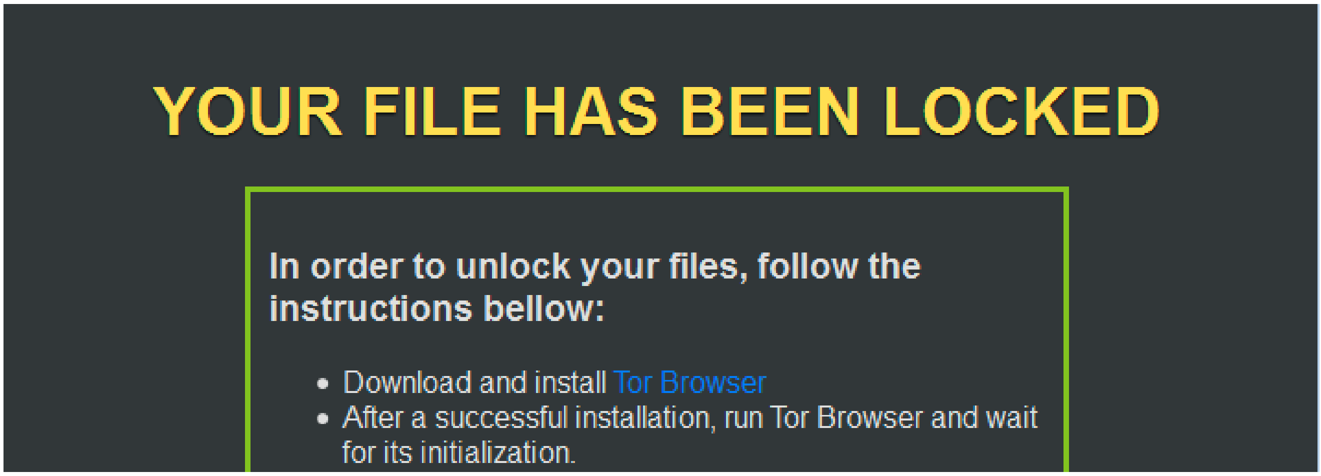
Rokku Ransomware shows possible link with Chimera

blog.malwarebytes.com/threat-analysis/2016/04/rokku-ransomware/

hasherezade

April 11, 2016



YOUR FILE HAS BEEN LOCKED

In order to unlock your files, follow the instructions bellow:

- Download and install Tor Browser
- After a successful installation, run Tor Browser and wait for its initialization.

Rokku is yet another ransomware, discovered in recent weeks. Currently, it's most common distribution method is spam where a malicious executable is dropped by a VB script belonging to the e-mail's attachment.

The building blocks of Rokku reminded us of the Chimera ransomware. That's why we decided to take a closer look, not only at the internal structure of this malware but also at the similarities and differences between these two products.

Analyzed samples

Malware:

Decryptor:

Special thanks to MalwareHunterTeam for sharing the sample.

Behavioral analysis

When we deploy the executable it runs silently – first dropping ransom notes (in two formats – HTML and TXT), and after that substituting files with their encrypted versions.

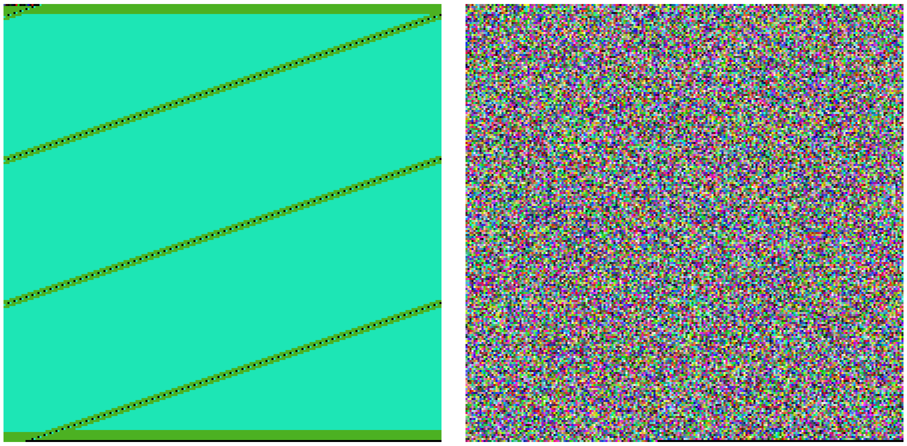
Rokku doesn't retrieve keys from the server, so the encryption process can be executed off-line as well.

Encryption process

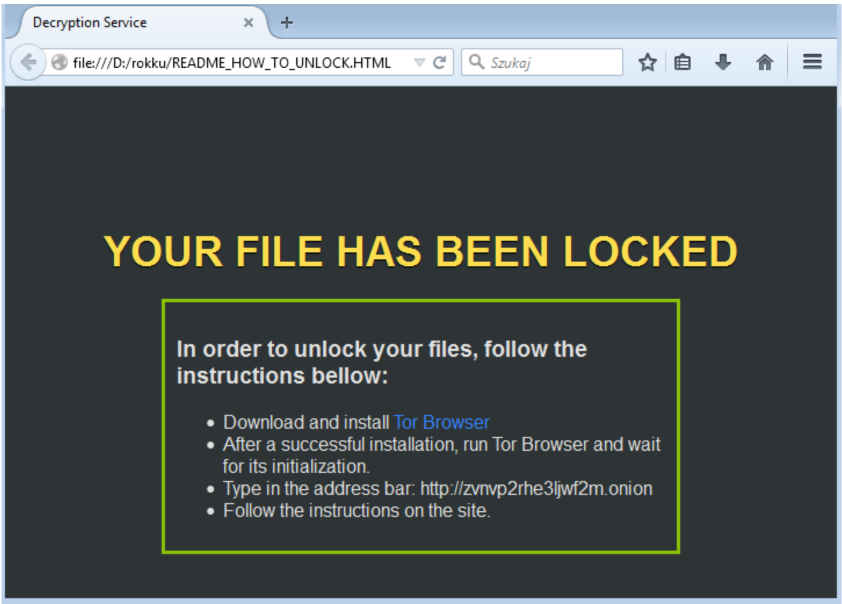
Files encrypted by this ransomware can be identified by the extension ***.rokku*** added to the original name.

The encrypted content has a high level of entropy and no patterns are visible. See below a visualization of bytes.

square.bmp : left – original, right encrypted with *Rokku*:



When the encryption finishes, the ransom note pops up:



YOUR FILE HAS BEEN LOCKED

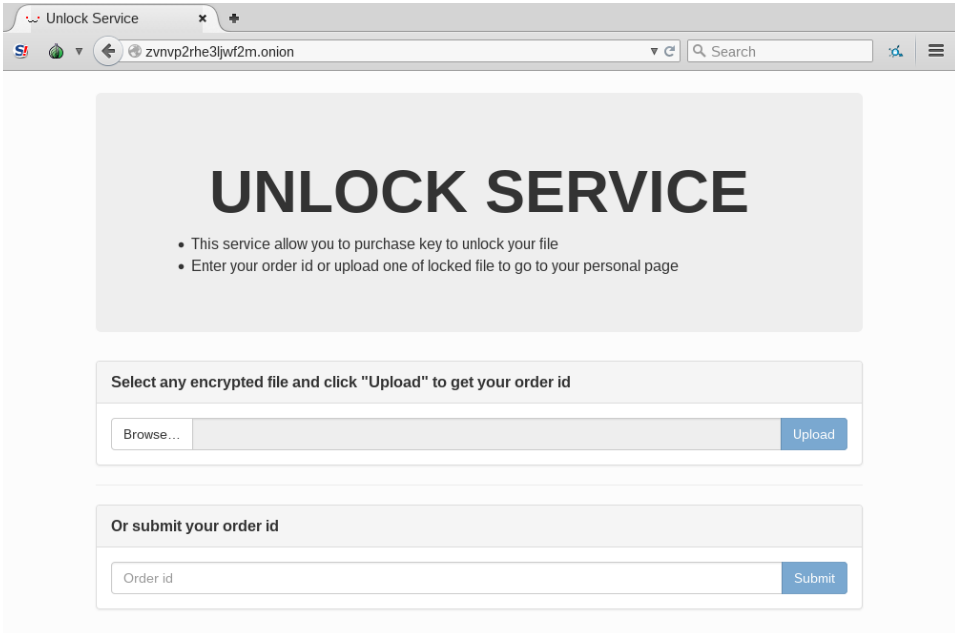
In order to unlock your files, follow the instructions bellow:

- Download and install Tor Browser
- After a successful installation, run Tor Browser and wait for its initialization.
- Type in the address bar: http://zvnvp2rhe3ljwf2m.onion
- Follow the instructions on the site.

Website for the victim

As many products of this type, Rokku has a web panel for victims, used to manage the payment and decrypt files. It is available via Tor.

The website have a neat design, however is very simple in comparison to other recent ransomware:



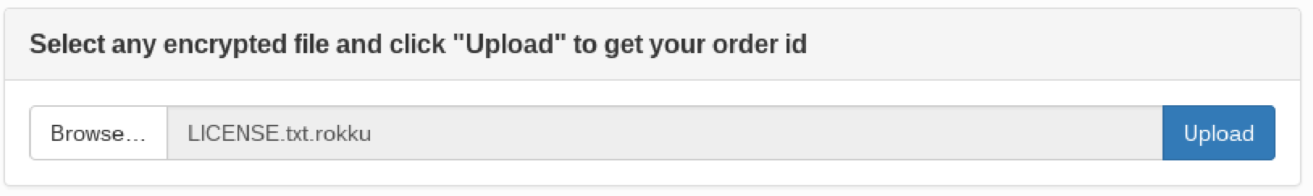
UNLOCK SERVICE

- This service allow you to purchase key to unlock your file
- Enter your order id or upload one of locked file to go to your personal page

Select any encrypted file and click "Upload" to get your order id

Or submit your order id

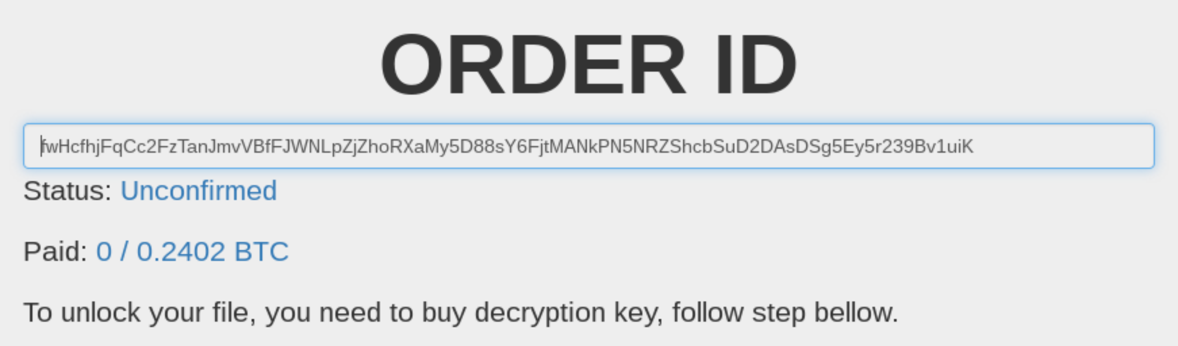
Rather than forcing a victim to type a unique ID it simply ask them to upload one file. All the necessary data are automatically fetched from the file.



Select any encrypted file and click "Upload" to get your order id

Browse… LICENSE.txt.rokku

Then it redirects to the personalized part of the panel and shows the order ID. This unique identifier can be used further to regain access to this page without the necessity to upload a file again:



ORDER ID

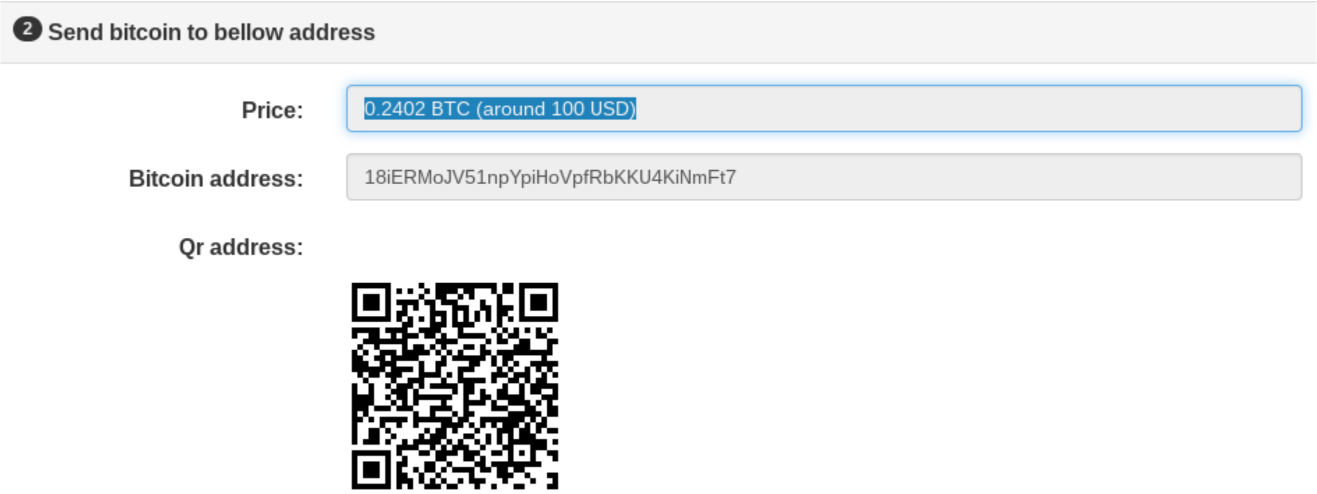
fwHcfhjFqCc2FzTanJmvVBfFJWNLpZjZhoRXaMy5D88sY6FjtMANkPN5NRZShcbSuD2DAsDSg5Ey5r239Bv1uiK

Status: Unconfirmed

Paid: 0 / 0.2402 BTC

To unlock your file, you need to buy decryption key, follow step bellow.

The required ransom amount is relatively low in comparison to other ransomware – 0.2402 BTC (around 100 USD). Currently we found no information suggesting that price is going to be incremented with time – so we can assume, that in this case distributors decided to use a fixed price.



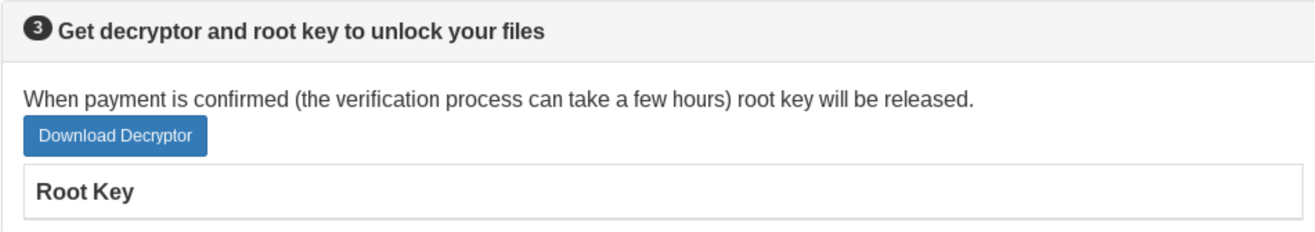
2 Send bitcoin to bellow address

Price: 0.2402 BTC (around 100 USD)

Bitcoin address: 18iERMoJV51npYpiHoVpfRbKKU4KiNmFt7

Qr address:

From the same site we can download the decrypting application. After the payment is processed, the root key, required to decrypt all the files is made available.

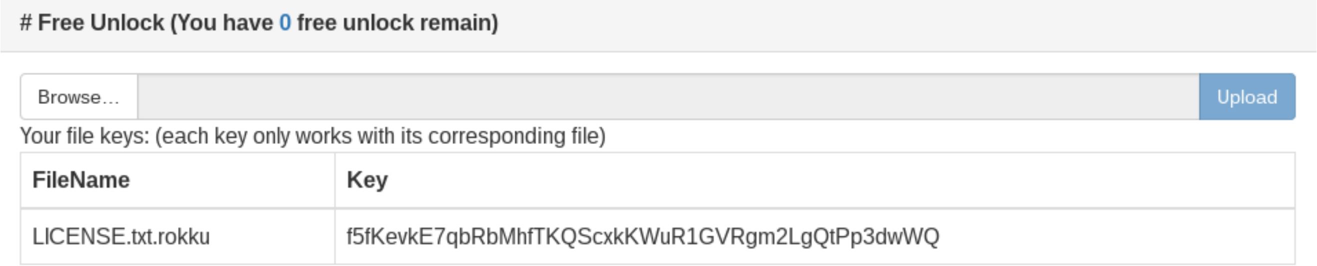


3 Get decryptor and root key to unlock your files

When payment is confirmed (the verification process can take a few hours) root key will be released.

Root Key

Even without a payment, one chosen file can be unlocked for demonstration purposes. Once an encrypted file is uploaded, it's individual file key is released. Then, it can be decrypted using this key and the decryptor available on the site.



Free Unlock (You have 0 free unlock remain)

Your file keys: (each key only works with its corresponding file)

FileName	Key
LICENSE.txt.rokku	f5fKevkE7qbRbMhfTKQScxkKWuR1GVRgm2LgQtPp3dwWQ

Findings

Looking at the features described above, we can deduce quite a lot of information about the internal logic of the encryption process. As usual, two types of cryptographic algorithms are used: asymmetric – for the root key, and symmetric – for the keys of individual files. Individual (random) key is used to encrypt the file content – then, itself is encrypted by the public root key and stored in the same file. Only an owner of the private root key can retrieve it – and with its help decrypt the original content.

The sample's individual key, displayed to a user is 45 characters long (it can also be interpreted as a Base64 encrypted, 36 byte long content).

Also, every file contains the Order ID. The displayed value is 86 characters long (may be interpreted as 66 byte long value Base64 encoded).

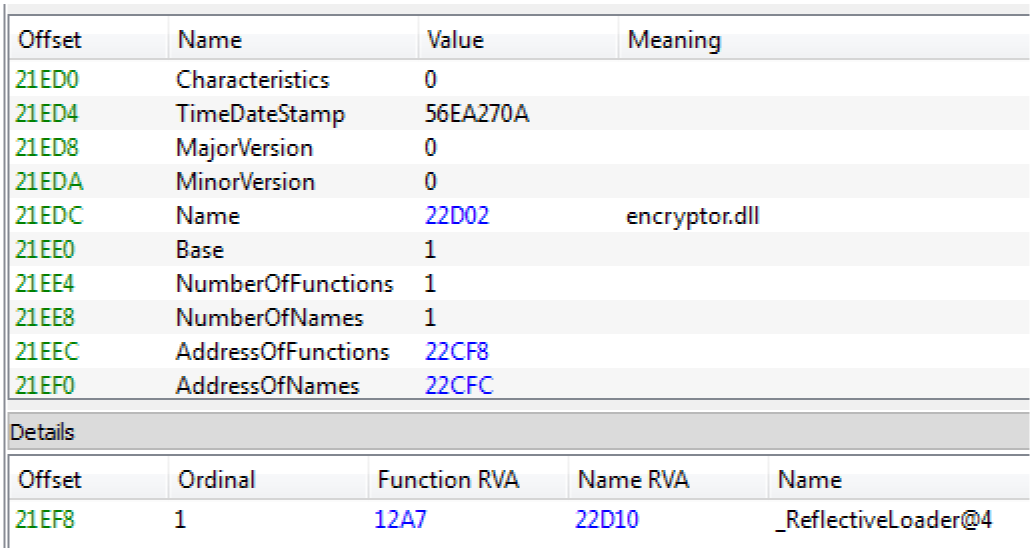
Inside the malware

Lets' have a look inside the malicious sample…

The original payload that is being distributed in a campaigns is UPX encrypted. This layer can be easily removed using typical UPX.

The next layer consists of some underground crypter/FUD.

After unpacking the crypter layer we can find the DLL with core malicious functionality – ***encryptor.dll*** (be6552aed5e7509b3b539cef8a965131)



Offset	Name	Value	Meaning
21ED0	Characteristics	0	
21ED4	TimeDateStamp	56EA270A	
21ED8	MajorVersion	0	
21EDA	MinorVersion	0	
21EDC	Name	22D02	encryptor.dll
21EE0	Base	1	
21EE4	NumberOfFunctions	1	
21EE8	NumberOfNames	1	
21EEC	AddressOfFunctions	22CF8	
21EF0	AddressOfNames	22CFC	

Details

Offset	Ordinal	Function RVA	Name RVA	Name
21EF8	1	12A7	22D10	_ReflectiveLoader@4

Similarly to the *Core.dll* of Chimera ransomware, it uses ReflectiveLoader.

ReflectiveLoader is a special stub belonging to the technique of ***Reflective DLL Injection***. This technique allows to produce a DLL that can be easily injected into another process. Similarly to a shellcode, such DLL is self-contained and automatically loads all it's dependencies.

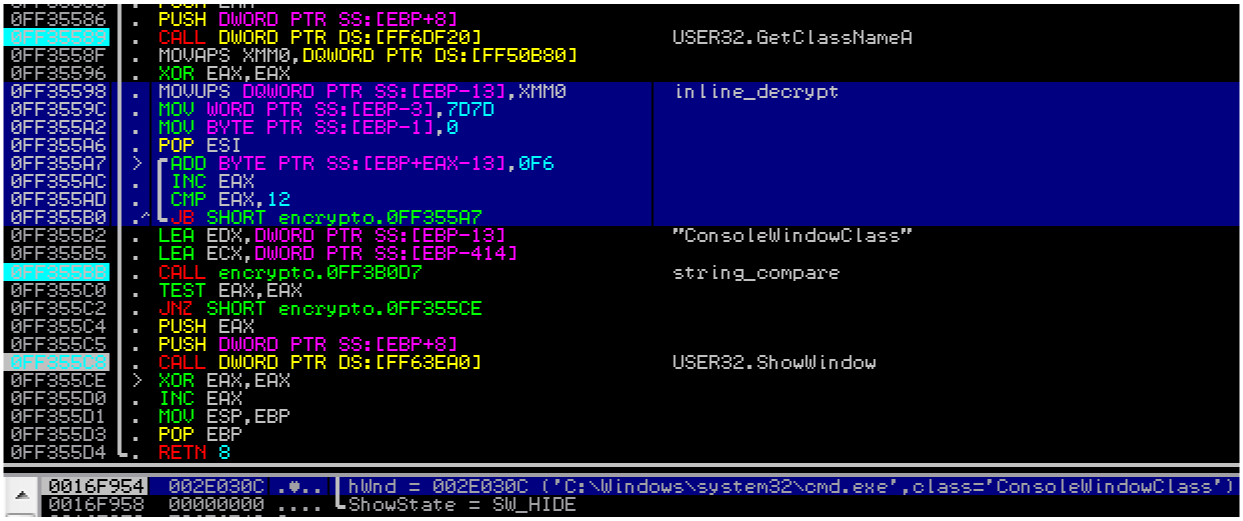
Execution flow

Execution of the malicious core follows several steps:

- Fetches information about the system.

- Removes local backups. It is very precise in this goal and it attacks several programs used for this purpose (used commands are listed below).
- Enumerates local disks, checking their existence by the alphabet (from Z to A) and makes a list of all their directories. Directories on network disks are also listed.
- Process the list of directories:
    - drops the ransom note in each of them
    - enumerates their files (using NtQueryDirectoryFile) and makes a list of paths.
- Encrypting routine takes the list of paths and encrypts them one by one. Information about the file, i.e size is retrieved using ZwQueryInformationFile.

In the initial phase, the malware makes a preparation to deploy its malicious features. It scans the environment and closes some programs. For example – searches if any console window is open, and if so, hides it:



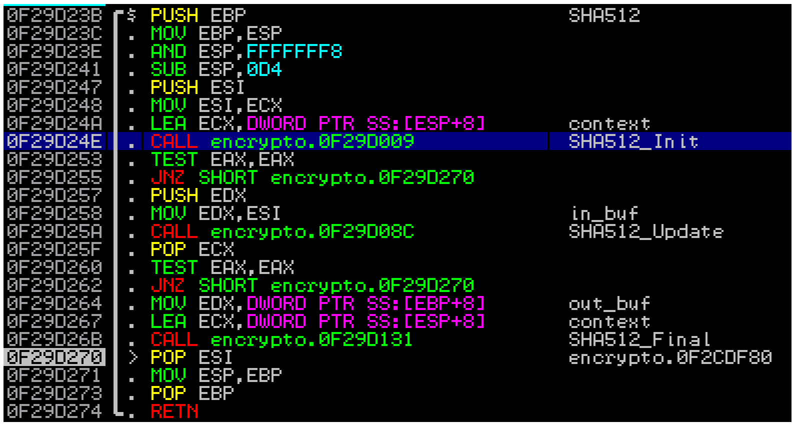
```
0FF35586  .  PUSH DWORD PTR SS:[EBP+8]
0FF35589  .  CALL DWORD PTR DS:[FF6DF20]                    USER32.GetClassNameA
0FF3558F  .  MOVAPS XMM0,DQWORD PTR DS:[FF50B80]
0FF35596  .  XOR EAX,EAX
0FF35598  .  MOVUPS DQWORD PTR SS:[EBP-13],XMM0             inline_decrypt
0FF3559C  .  MOV WORD PTR SS:[EBP-3],7D7D
0FF355A2  .  MOV BYTE PTR SS:[EBP-1],0
0FF355A6  .  POP ESI
0FF355A7  > ┌ADD BYTE PTR SS:[EBP+EAX-13],0F6
0FF355AC  . │INC EAX
0FF355AD  . │CMP EAX,12
0FF355B0  .^└JB SHORT encrypto.0FF355A7
0FF355B2  .  LEA EDX,DWORD PTR SS:[EBP-13]                  "ConsoleWindowClass"
0FF355B5  .  LEA ECX,DWORD PTR SS:[EBP-414]
0FF355BB  .  CALL encrypto.0FF3B0D7                         string_compare
0FF355C0  .  TEST EAX,EAX
0FF355C2  .  JNZ SHORT encrypto.0FF355CE
0FF355C4  .  PUSH EAX
0FF355C5  .  PUSH DWORD PTR SS:[EBP+8]
0FF355C8  .  CALL DWORD PTR DS:[FF63EA0]                    USER32.ShowWindow
0FF355CE  >  XOR EAX,EAX
0FF355D0  .  INC EAX
0FF355D1  .  MOV ESP,EBP
0FF355D3  .  POP EBP
0FF355D4  L. RETN 8

0016F954   002E030C  .☻..  ┌hWnd = 002E030C ('C:\Windows\system32\cmd.exe',class='ConsoleWindowClass')
0016F958   00000000  ....  └ShowState = SW_HIDE
```

In order to make analysis harder, this malware uses encrypted strings. They are decrypted on fly, just before they are needed. As you can see at the above screenshot – it is implemented with the help of small in-line routine using SSE (highlighted in the picture). Using an in-line routine have an advantage over a separate decrypting function – it is harder to locate all the calls to it and to decrypt strings just by tracing it's output.

Next, it reads a unique identifier of the current machine: retrieves GUID from the registry…

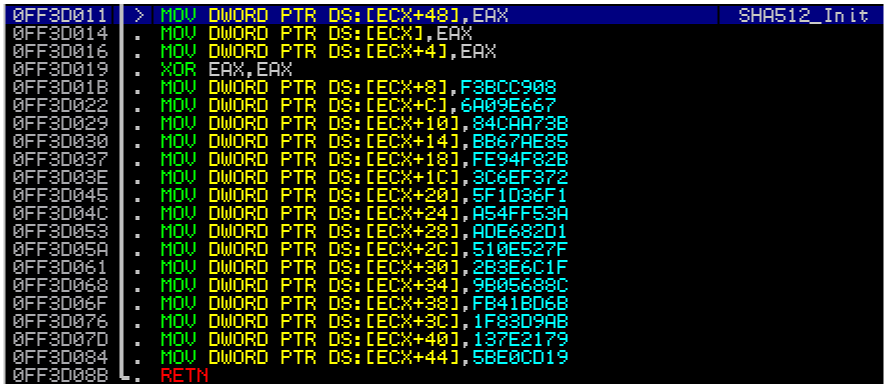
```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography -> "MachineGuid"
```

…and the volume serial number of the disk, where the Windows is installed (using GetVolumeInformation). Both parts are concatenated together (<*machine_guid*><*volume_serial*>) and hashed using local implementation of SHA512 (this implementation comes from OpenSSL)…



```
0F29D23B ┌$ PUSH EBP                                   SHA512
0F29D23C │. MOV EBP,ESP
0F29D23E │. AND ESP,FFFFFFF8
0F29D241 │. SUB ESP,0D4
0F29D247 │. PUSH ESI
0F29D248 │. MOV ESI,ECX
0F29D24A │. LEA ECX,DWORD PTR SS:[ESP+8]              context
0F29D24E │. CALL encrypto.0F29D009                    SHA512_Init
0F29D253 │. TEST EAX,EAX
0F29D255 │. JNZ SHORT encrypto.0F29D270
0F29D257 │. PUSH EDX
0F29D258 │. MOV EDX,ESI                                in_buf
0F29D25A │. CALL encrypto.0F29D08C                    SHA512_Update
0F29D25F │. POP ECX
0F29D260 │. TEST EAX,EAX
0F29D262 │. JNZ SHORT encrypto.0F29D270
0F29D264 │. MOV EDX,DWORD PTR SS:[EBP+8]              out_buf
0F29D267 │. LEA ECX,DWORD PTR SS:[ESP+8]              context
0F29D26B │. CALL encrypto.0F29D131                    SHA512_Final
0F29D270 │> POP ESI                                    encrypto.0F2CDF80
0F29D271 │. MOV ESP,EBP
0F29D273 │. POP EBP
0F29D274 └. RETN
```

We can see the typical SHA512 constants in the code:



```
0FF3D011 │> MOV DWORD PTR DS:[ECX+48],EAX                  SHA512_Init
0FF3D014 │. MOV DWORD PTR DS:[ECX],EAX
0FF3D016 │. MOV DWORD PTR DS:[ECX+4],EAX
0FF3D019 │. XOR EAX,EAX
0FF3D01B │. MOV DWORD PTR DS:[ECX+8],F3BCC908
0FF3D022 │. MOV DWORD PTR DS:[ECX+C],6A09E667
0FF3D029 │. MOV DWORD PTR DS:[ECX+10],84CAA73B
0FF3D030 │. MOV DWORD PTR DS:[ECX+14],BB67AE85
0FF3D037 │. MOV DWORD PTR DS:[ECX+18],FE94F82B
0FF3D03E │. MOV DWORD PTR DS:[ECX+1C],3C6EF372
0FF3D045 │. MOV DWORD PTR DS:[ECX+20],5F1D36F1
0FF3D04C │. MOV DWORD PTR DS:[ECX+24],A54FF53A
0FF3D053 │. MOV DWORD PTR DS:[ECX+28],ADE682D1
0FF3D05A │. MOV DWORD PTR DS:[ECX+2C],510E527F
0FF3D061 │. MOV DWORD PTR DS:[ECX+30],2B3E6C1F
0FF3D068 │. MOV DWORD PTR DS:[ECX+34],9B05688C
0FF3D06F │. MOV DWORD PTR DS:[ECX+38],FB41BD6B
0FF3D076 │. MOV DWORD PTR DS:[ECX+3C],1F83D9AB
0FF3D07D │. MOV DWORD PTR DS:[ECX+40],137E2179
0FF3D084 │. MOV DWORD PTR DS:[ECX+44],5BE0CD19
0FF3D08B └. RETN
```

First half of the SHA512 hash and the *<machine_guid><volume_serial>* are concatenated together and used as a mutex name (with the help of mutex malware prevent from being run more than once at the same time).

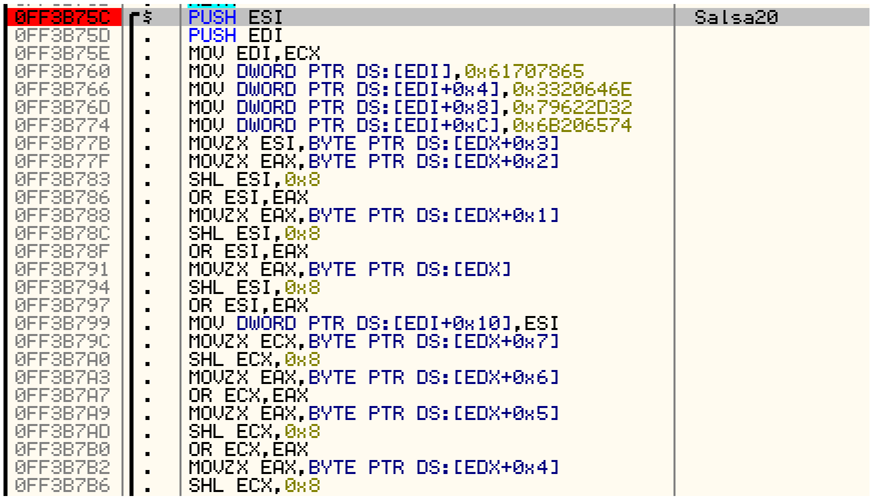
Finally, removing backups and stopping backup services is performed – by execution of the following commands:

```
wmic shadowcopy delete /nointeractive
vssadmin delete shadows /all /quiet
reg add "HKLM\SYSTEM\CurrentControlSet\services\VSS" /v Start /t REG_DWORD /d 4 /f
reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SystemRestore" /v
DisableSR /t REG_DWORD /d 1 /f
net stop vss
net stop swprv
net stop srservice
```

How does the encryption work?

From the behavioral analysis and experiments we concluded, that Rokku – like most of the ransomware – uses symmetric and asymmetric encryption.

As the main, symmetric encryption algorithm, authors decided to use Salsa20 (Salsa was also used by the Petya ransomware). Fragment of the implementation is shown below:



```
0FF3B75C  ┌$  PUSH ESI                                    Salsa20
0FF3B75D  │.  PUSH EDI
0FF3B75E  │.  MOV EDI,ECX
0FF3B760  │.  MOV DWORD PTR DS:[EDI],0x61707865
0FF3B766  │.  MOV DWORD PTR DS:[EDI+0x4],0x3320646E
0FF3B76D  │.  MOV DWORD PTR DS:[EDI+0x8],0x79622D32
0FF3B774  │.  MOV DWORD PTR DS:[EDI+0xC],0x6B206574
0FF3B77B  │.  MOVZX ESI,BYTE PTR DS:[EDX+0x3]
0FF3B77F  │.  MOVZX EAX,BYTE PTR DS:[EDX+0x2]
0FF3B783  │.  SHL ESI,0x8
0FF3B786  │.  OR ESI,EAX
0FF3B788  │.  MOVZX EAX,BYTE PTR DS:[EDX+0x1]
0FF3B78C  │.  SHL ESI,0x8
0FF3B78F  │.  OR ESI,EAX
0FF3B791  │.  MOVZX EAX,BYTE PTR DS:[EDX]
0FF3B794  │.  SHL ESI,0x8
0FF3B797  │.  OR ESI,EAX
0FF3B799  │.  MOV DWORD PTR DS:[EDI+0x10],ESI
0FF3B79C  │.  MOVZX ECX,BYTE PTR DS:[EDX+0x7]
0FF3B7A0  │.  SHL ECX,0x8
0FF3B7A3  │.  MOVZX EAX,BYTE PTR DS:[EDX+0x6]
0FF3B7A7  │.  OR ECX,EAX
0FF3B7A9  │.  MOVZX EAX,BYTE PTR DS:[EDX+0x5]
0FF3B7AD  │.  SHL ECX,0x8
0FF3B7B0  │.  OR ECX,EAX
0FF3B7B2  │.  MOVZX EAX,BYTE PTR DS:[EDX+0x4]
0FF3B7B6  │.  SHL ECX,0x8
```

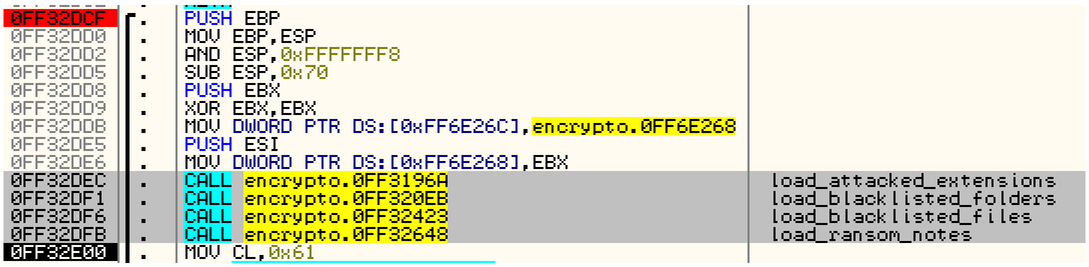
Every file is encrypted by Salsa20 with a new, random key. Random values are retrieved using *advapi32.SystemFunction036* – that is RtlGenRandom. Then, the random key is encrypted with a locally implemented RSA algorithm.

Research about the implementation details and possible flaws is in progress.

What is attacked?

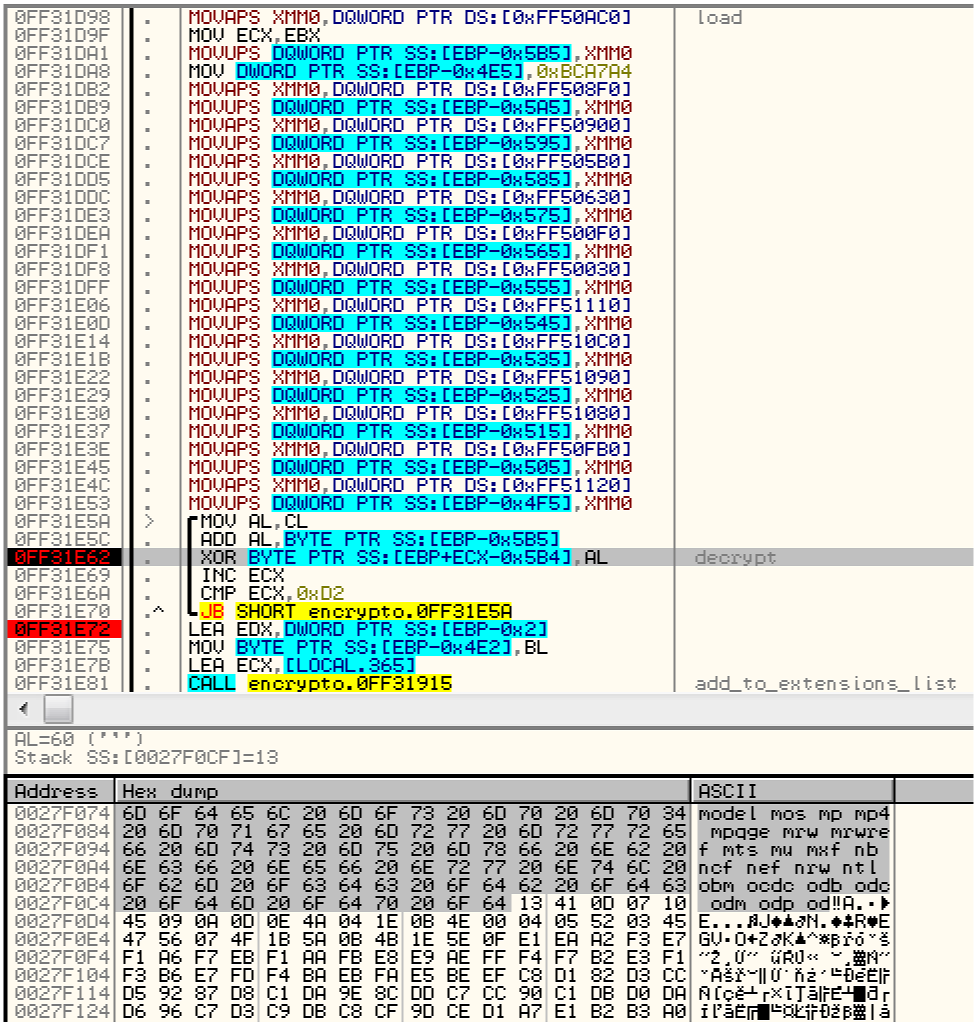
Rokku attacks local disks as well as network shares.

This malware doesn't have any external configuration – all the strings (including attacked file extensions and blacklisted paths) are hardcoded in obfuscated form and decrypted in-line. Loading the hardcoded settings is performed by dedicated functions (in the described sample it starts at RVA = 0x2dcf):



```
0FF32DCF  ┌.  PUSH EBP
0FF32DD0  │.  MOV EBP,ESP
0FF32DD2  │.  AND ESP,0xFFFFFFF8
0FF32DD5  │.  SUB ESP,0x70
0FF32DD8  │.  PUSH EBX
0FF32DD9  │.  XOR EBX,EBX
0FF32DDB  │.  MOV DWORD PTR DS:[0xFF6E26C],encrypto.0FF6E268
0FF32DE5  │.  PUSH ESI
0FF32DE6  │.  MOV DWORD PTR DS:[0xFF6E268],EBX
0FF32DEC  │.  CALL encrypto.0FF3196A                      load_attacked_extensions
0FF32DF1  │.  CALL encrypto.0FF320EB                      load_blacklisted_folders
0FF32DF6  │.  CALL encrypto.0FF32423                      load_blacklisted_files
0FF32DFB  │.  CALL encrypto.0FF32648                      load_ransom_notes
0FF32E00  │.  MOV CL,0x61
```

Attacked extensions are decrypted in chunks (each chunk contains several extensions) and then added to the list. Below you can see decrypting chunk of extensions:



```
0FF31D98  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF50AC0]        load
0FF31D9F  .  MOV ECX,EBX
0FF31DA1  .  MOVUPS DQWORD PTR SS:[EBP-0x5B5],XMM0
0FF31DA8  .  MOV DWORD PTR SS:[EBP-0x4E5],0xBCA7A4
0FF31DB2  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF508F0]
0FF31DB9  .  MOVUPS DQWORD PTR SS:[EBP-0x5A5],XMM0
0FF31DC0  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF50900]
0FF31DC7  .  MOVUPS DQWORD PTR SS:[EBP-0x595],XMM0
0FF31DCE  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF505B0]
0FF31DD5  .  MOVUPS DQWORD PTR SS:[EBP-0x585],XMM0
0FF31DDC  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF50630]
0FF31DE3  .  MOVUPS DQWORD PTR SS:[EBP-0x575],XMM0
0FF31DEA  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF500F0]
0FF31DF1  .  MOVUPS DQWORD PTR SS:[EBP-0x565],XMM0
0FF31DF8  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF50030]
0FF31DFF  .  MOVUPS DQWORD PTR SS:[EBP-0x555],XMM0
0FF31E06  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF51110]
0FF31E0D  .  MOVUPS DQWORD PTR SS:[EBP-0x545],XMM0
0FF31E14  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF510C0]
0FF31E1B  .  MOVUPS DQWORD PTR SS:[EBP-0x535],XMM0
0FF31E22  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF51090]
0FF31E29  .  MOVUPS DQWORD PTR SS:[EBP-0x525],XMM0
0FF31E30  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF51080]
0FF31E37  .  MOVUPS DQWORD PTR SS:[EBP-0x515],XMM0
0FF31E3E  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF50FB0]
0FF31E45  .  MOVUPS DQWORD PTR SS:[EBP-0x505],XMM0
0FF31E4C  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF51120]
0FF31E53  .  MOVUPS DQWORD PTR SS:[EBP-0x4F5],XMM0
0FF31E5A  >  ┌MOV AL,CL
0FF31E5C  .  │ADD AL,BYTE PTR SS:[EBP-0x5B5]
0FF31E62  .  │XOR BYTE PTR SS:[EBP+ECX-0x5B4],AL          decrypt
0FF31E69  .  │INC ECX
0FF31E6A  .  │CMP ECX,0xD2
0FF31E70  .^ └JB SHORT encrypto.0FF31E5A
0FF31E72  .  LEA EDX,DWORD PTR SS:[EBP-0x2]
0FF31E75  .  MOV BYTE PTR SS:[EBP-0x4E2],BL
0FF31E7B  .  LEA ECX,[LOCAL.365]
0FF31E81  .  CALL encrypto.0FF31915                     add_to_extensions_list

AL=60 ('`')
Stack SS:[0027F0CF]=13

Address   Hex dump                                         ASCII
0027F074  6D 6F 64 65 6C 20 6D 6F 73 20 6D 70 20 6D 70 34  model mos mp mp4
0027F084  20 6D 70 71 67 65 20 6D 72 77 20 6D 72 77 72 65   mpqge mrw mrwre
0027F094  66 20 6D 74 73 20 6D 75 20 6D 78 66 20 6E 62 20  f mts mu mxf nb
0027F0A4  6E 63 66 20 6E 65 66 20 6E 72 77 20 6E 74 6C 20  ncf nef nrw ntl
0027F0B4  6F 62 6D 20 6F 63 64 63 20 6F 64 62 20 6F 64 63  obm ocdc odb odc
0027F0C4  20 6F 64 6D 20 6F 64 70 20 6F 64 13 41 0D 07 10   odm odp od‼A.•►
0027F0D4  45 09 0A 0D 0E 4A 04 1E 0B 4E 00 04 05 52 03 45  E....J♦▲♂N.♦♣R♥E
0027F0E4  47 56 07 4F 1B 5A 0B 4B 1E 5E 0F E1 EA A2 F3 E7  GV•O←Z♂K▲^☼ßΩó≤τ
0027F0F4  F1 A6 F7 EB F1 AA FB E8 E9 AE FF F4 F7 B2 E3 F1  ±ª≈δ±¬√ΦΘ« ⌠≈▓π±
0027F104  F3 B6 E7 FD F4 BA EB FA E5 BE EF C8 D1 82 D3 CC  ≤╢τ²⌠║δ·σ╛∩╚╤é╙╠
0027F114  D5 92 87 D8 C1 DA 9E 8C DD C7 CC 90 C1 DB D0 DA  ╒Æç╪┴┌₧î▌╟╠É┴█╨┌
0027F124  D6 96 C7 D3 C9 DB C8 CF 9D CE D1 A7 E1 B2 B3 A0  ╓û╟╙╔█╚╧¥╬╤ºß▓│á
```

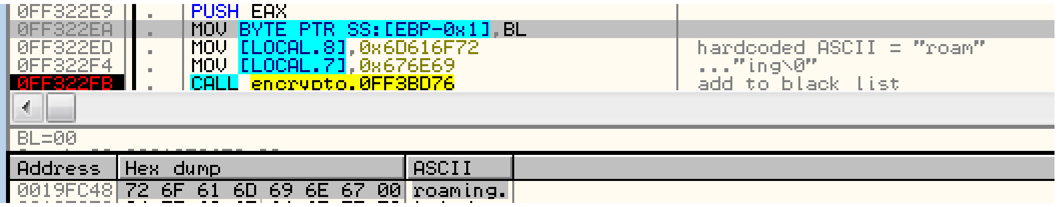
Summary of all the file extensions that are attacked:

```
001 1dc 3ds 3fr 7z a3s acb acbl accdb act ai ai3 ai4 ai5 ai6 ai7 ai8 aia aif aiff aip
ait anim apk arch00 ari art arw asc ase asef asp aspx asset avi bar bak bay bc6 bc7
bgeo big bik bkf bkp blob bmp bsa c c4d cap cas catpart catproduct cdr cef cer cfr
...
cgm cha chr cld clx cpp cr2 crt crw cs css csv cxx d3dbsp das dayzprofile dazip db
db0 dbf dbfv dcr dcs der desc dib dlc dle dlv dlv3 dlv4 dmp dng doc docm docx drf dvi
dvr dwf dwg dxf dxg eip emf emz epf epk eps eps2 eps3 epsf
...
epsp erf esm fbx ff fff fh10 fh11 fh7 fh8 fh9 fig flt flv fmod forge fos fpk fsh ft8
fxg gdb ge2 geo gho h hip hipnc hkdb hkx hplg hpp hvpl hxx iam ibank icb icxs idea
iff iiq indd ipt iros irs itdb itl itm iwd iwi
...
j2k java jp2 jpe jpeg jpf jpg jpx js k25 kdb kdc kf kys layout lbf lex litemod lrf
ltx lvl m m2 m2t m2ts m3u m4a m4v ma map mat mb mcfi mcfp mcgame mcmeta mdb mdbackup
mdc mddata mdf mdl mdlp mef mel menu mkv mll mlx mn
...
model mos mp mp4 mpqge mrw mrwref mts mu mxf nb ncf nef nrw ntl obm ocdc odb odc odm
odp ods odt omeg orf ott p12 p7b p7c pak pct pcx pdd pdf pef pem pfx php php4 php5
pic picnc pkpass png ppd ppt pptm pptx prj
...
prt prtl ps psb psd psf psid psk psq pst ptl ptx pwl pxn pxr py qdf qic r3d raa raf
rar raw rb re4 rgss3a rim rofl rtf rtg rvt rw2 rwl rwz sav sb sbx sc2save shp sid
sidd sidn sie sis skl skp sldasm sldprt slm
...
slx slxp snx soft sqlite sqlite3 sr2 srf srw step stl stp sum svg svgz swatch syncdb
t12 t13 tax tex tga tif tiff tor txt unity3d uof uos upk vda vdf vfl vfs0 vpk vpp_pc
vst vtf w3x wb2 wdx wma wmo wmv wallet ycbcra
...
wotreplay wpd wps x3f xf xl xlk xls xlsb xlsm xlsx xvc xvz xxx zdct zip ztmp py rb
tar gz sdf yuv max wav dat
```

In the same way, blacklisted paths are deobfuscated and loaded.

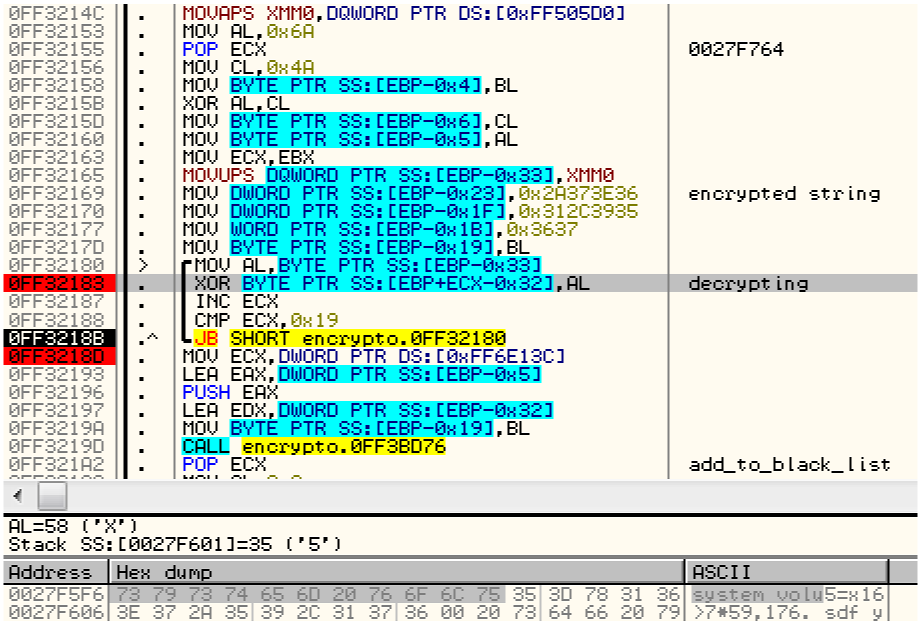
Here are some examples of in-line routines used to decrypt blacklisted paths:

Example 1 – adding hardcoded value “roaming”:



```
0FF322E9  .  PUSH EAX
0FF322EA  .  MOV BYTE PTR SS:[EBP-0x1],BL
0FF322ED  .  MOV [LOCAL.8],0x6D616F72          hardcoded ASCII = "roam"
0FF322F4  .  MOV [LOCAL.7],0x676E69            ..."ing\0"
0FF322FB  .  CALL encrypto.0FF3BD76            add to black list

BL=00

Address   Hex dump                 ASCII
0019FC48  72 6F 61 6D 69 6E 67 00  roaming.
```

Example 2 – decrypting “system volume information”



```
0FF3214C  .  MOVAPS XMM0,DQWORD PTR DS:[0xFF505D0]
0FF32153  .  MOV AL,0x6A
0FF32155  .  POP ECX                                          0027F764
0FF32156  .  MOV CL,0x4A
0FF32158  .  MOV BYTE PTR SS:[EBP-0x4],BL
0FF3215B  .  XOR AL,CL
0FF3215D  .  MOV BYTE PTR SS:[EBP-0x6],CL
0FF32160  .  MOV BYTE PTR SS:[EBP-0x5],AL
0FF32163  .  MOV ECX,EBX
0FF32165  .  MOVUPS DQWORD PTR SS:[EBP-0x33],XMM0
0FF32169  .  MOV DWORD PTR SS:[EBP-0x23],0x2A373E36          encrypted string
0FF32170  .  MOV DWORD PTR SS:[EBP-0x1F],0x312C3935
0FF32177  .  MOV WORD PTR SS:[EBP-0x1B],0x3637
0FF3217D  .  MOV BYTE PTR SS:[EBP-0x19],BL
0FF32180  >  MOV AL,BYTE PTR SS:[EBP-0x33]
0FF32183  .  XOR BYTE PTR SS:[EBP+ECX-0x32],AL                decrypting
0FF32187  .  INC ECX
0FF32188  .  CMP ECX,0x19
0FF3218B  .^ JB SHORT encrypto.0FF32180
0FF3218D  .  MOV ECX,DWORD PTR DS:[0xFF6E13C]
0FF32193  .  LEA EAX,DWORD PTR SS:[EBP-0x5]
0FF32196  .  PUSH EAX
0FF32197  .  LEA EDX,DWORD PTR SS:[EBP-0x32]
0FF3219A  .  MOV BYTE PTR SS:[EBP-0x19],BL
0FF3219D  .  CALL encrypto.0FF3BD76
0FF321A2  .  POP ECX                                          add_to_black_list

AL=58 ('X')
Stack SS:[0027F601]=35 ('5')

Address  Hex dump                                         ASCII
0027F5F6 73 79 73 74 65 6D 20 76 6F 6C 75 35 3D 78 31 36  system volu5=x16
0027F606 3E 37 2A 35 39 2C 31 37 36 00 20 73 64 66 20 79  >7*59,176. sdf y
```

Summary of folders excluded from encryption:

```
$recycle.bin
system volume information
windows.old
$windows.~bt
windows
windows
locallow
local
roaming
programdata
program files
program files (x86)
```

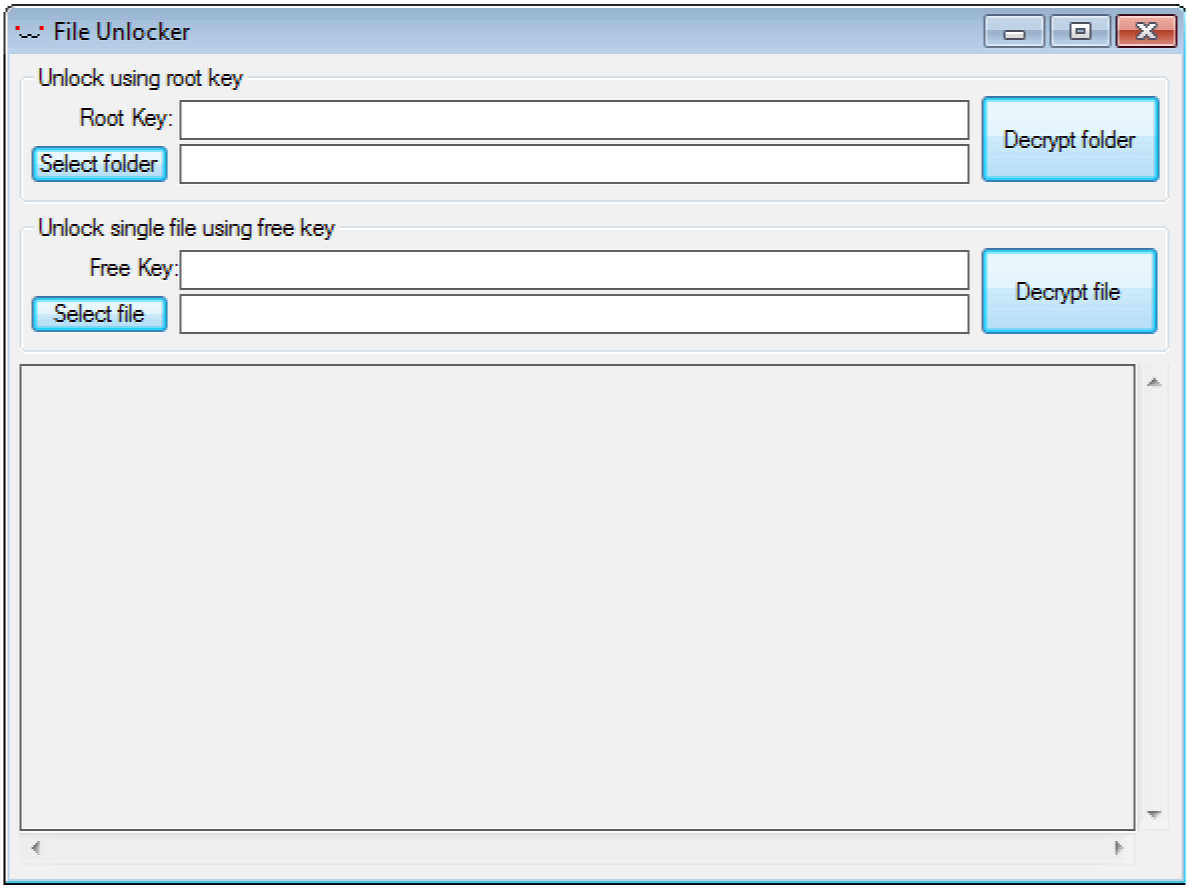
Some files – including ransom notes – are also excluded from encryption, i.e:

```
thumbs.db
iconcache.db
bootsec.bak
```

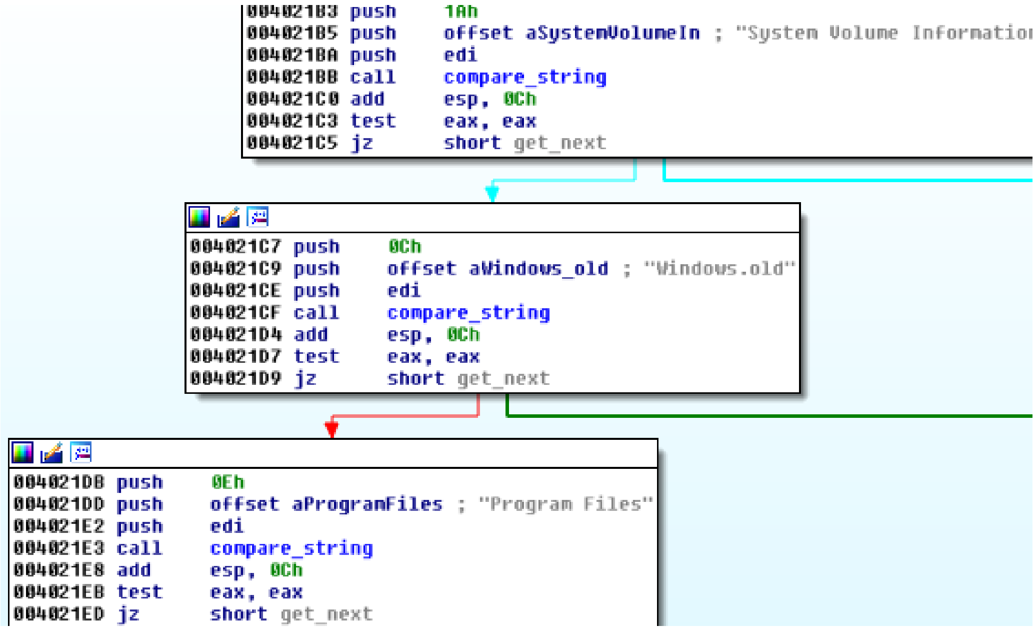
Inside the decryptor

Decryptor is an application that can be downloaded from the website for the victim and used to recover the files after purchasing the key.

It comes with a simple GUI, allowing two modes of decryption – for individual file of for full folder.

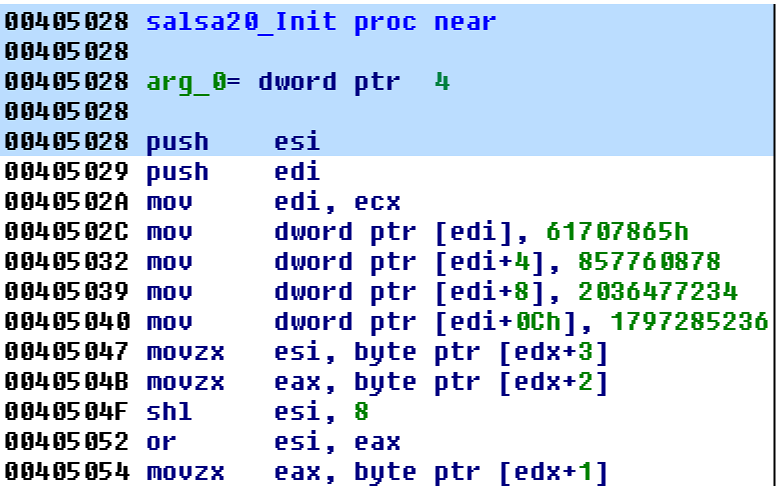


It is also UPX packed, but after removing this layer we can see valid strings. We can find there elements corresponding to the ***encryptor.dll*** – but with much less obfuscation added. For example – the same paths are skipped, but this time we can see them in clear text:



```
004021B3 push    1Ah
004021B5 push    offset aSystemVolumeIn ; "System Volume Informatio
004021BA push    edi
004021BB call    compare_string
004021C0 add     esp, 0Ch
004021C3 test    eax, eax
004021C5 jz      short get_next

004021C7 push    0Ch
004021C9 push    offset aWindows_old ; "Windows.old"
004021CE push    edi
004021CF call    compare_string
004021D4 add     esp, 0Ch
004021D7 test    eax, eax
004021D9 jz      short get_next

004021DB push    0Eh
004021DD push    offset aProgramFiles ; "Program Files"
004021E2 push    edi
004021E3 call    compare_string
004021E8 add     esp, 0Ch
004021EB test    eax, eax
004021ED jz      short get_next
```

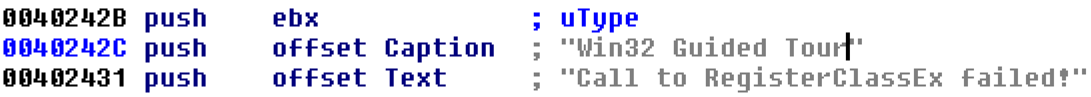
Below – fragment of Salsa20 implementation containing typical constants:



```
00405028 salsa20_Init proc near
00405028
00405028 arg_0= dword ptr  4
00405028
00405028 push    esi
00405029 push    edi
0040502A mov     edi, ecx
0040502C mov     dword ptr [edi], 61707865h
00405032 mov     dword ptr [edi+4], 857760878
00405039 mov     dword ptr [edi+8], 2036477234
00405040 mov     dword ptr [edi+0Ch], 1797285236
00405047 movzx   esi, byte ptr [edx+3]
0040504B movzx   eax, byte ptr [edx+2]
0040504F shl     esi, 8
00405052 or      esi, eax
00405054 movzx   eax, byte ptr [edx+1]
```

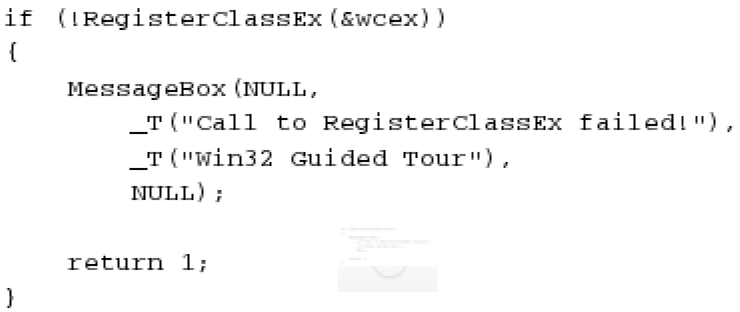
GUI programming in C++ is not the strong point of the authors. In the code of decryptor we can find fragments of a ready-made template. See below:

code fragment found in Rokku’s decryptor:



```
0040242B push    ebx                 ; uType
0040242C push    offset Caption      ; "Win32 Guided Tour"
00402431 push    offset Text         ; "Call to RegisterClassEx failed!"
```

corresponding code fragment – part of a skeleton application that have been demonstrated in a GUI programming course:



```
if (!RegisterClassEx(&wcex))
{
    MessageBox(NULL,
        _T("Call to RegisterClassEx failed!"),
        _T("Win32 Guided Tour"),
        NULL);

    return 1;
}
```

Authors of Chimera also didn’t felt confident in native GUI programming. Although they wrote most of the code in C++, the decryptor’s GUI was prepared in .NET framework (that makes GUI programming much easier). Decryptor’s core functions were called from a DLL written in C++.

Conclusion

In terms of architecture, Rokku shows several similarities with Chimera ransomware:

- the main part is a DLL, using ReflectiveLoader
- cryptography implemented locally (not via API calls)
- external decryptor that can be downloaded from the given location, before paying the ransom

Both products use, however, different ways to communicate with victims: Chimera uses bitmessage, while Rokku uses a Tor website (like most of the ransomware). Chimera requires an Internet connection in order to work – Rokku in contrary is fully independent from the CnC server.

The found similarities lead us to the conclusion, that Rokku may be a product of the same authors – prepared with a similar schema but with different needs in mind.

Rokku is detected by Malwarebytes Anti-Malware (MBAM) as well as by Malwarebytes Anti-Ransomware (MBARW).

Appendix

About Rokku by other vendors:

> http://www.bleepingcomputer.com/news/security/rokku-ransomware-encrypts-each-file-with-its-own-unique-key/ – Bleeping Computer

About Chimera:

> Inside Chimera Ransomware – the first 'doxingware' in wild

This video cannot be displayed because your *Functional Cookies* are currently disabled. To enable them, please visit our *privacy policy* and search for the Cookies section. Select *"Click Here"* to open the Privacy Preference Center and select *"Functional Cookies"* in the menu. You can switch the tab back to *"Active"* or disable by moving the tab to *"Inactive."* Click *"Save Settings."*