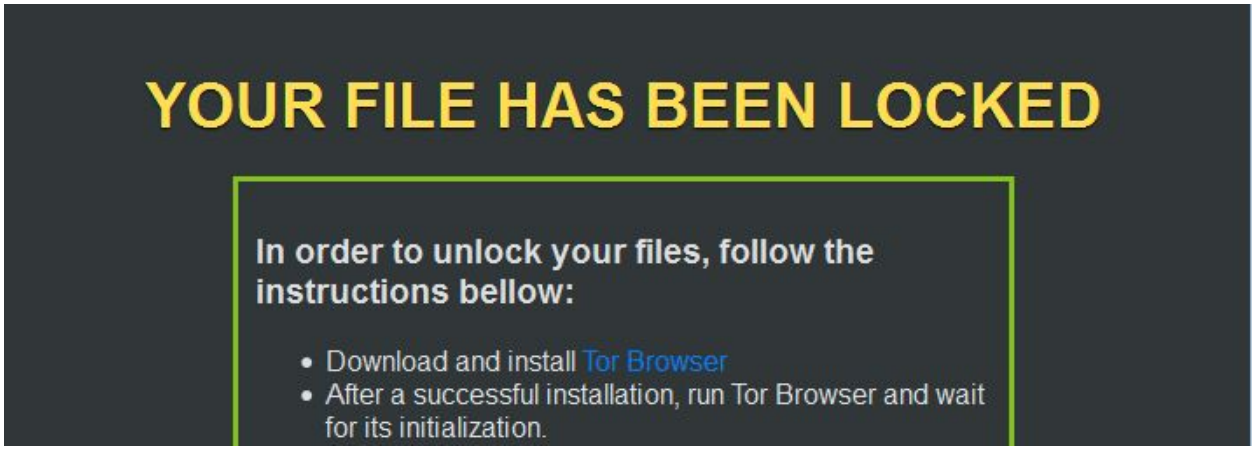


Rokku Ransomware shows possible link with Chimera

April 11, 2016 by hasherezade

Last updated: June 25, 2016

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:

- 97512f4617019c907cd0f88193039e7c – original executable
    - 5a0e3a6e3106e754381bd1cc3295c97f – UPX layer removed
        - be6552aed5e7509b3b539cef8a965131 – payload: ***encryptor.dll*** <-main focus of the analysis

Decryptor:

- 82fea20bb4c96050b4cf55f83de0f3e6 – original executable: ***decryptor.exe***
    - 1be4a0932a66ebdb9ede56214d8ccdf9 – UPX layer removed <-main focus of the analysis

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.

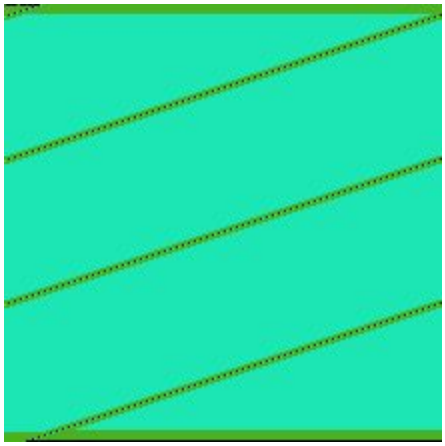
Rokku doesn't retrieve keys from the server, so the encryption process can be executed off-line as well.

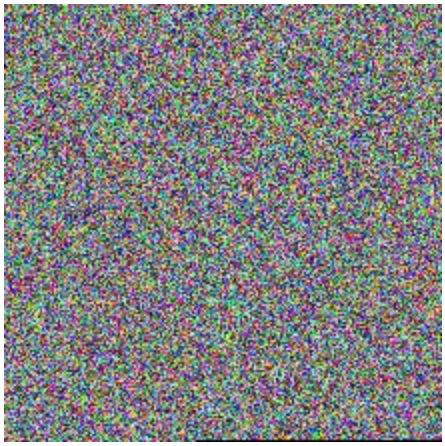
Encryption process

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

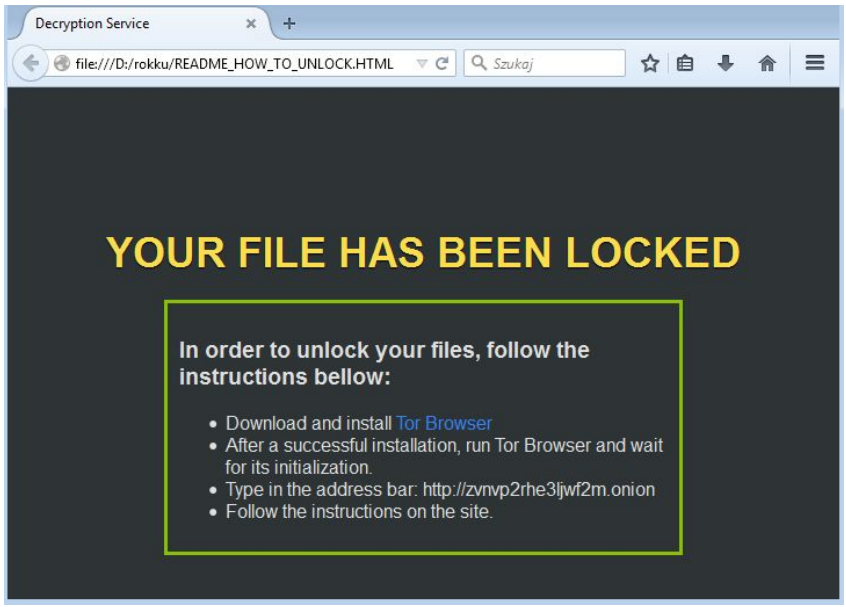
The encrypted content has a high level of entropy and no patterns are visible. See below a visualization of bytes.

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





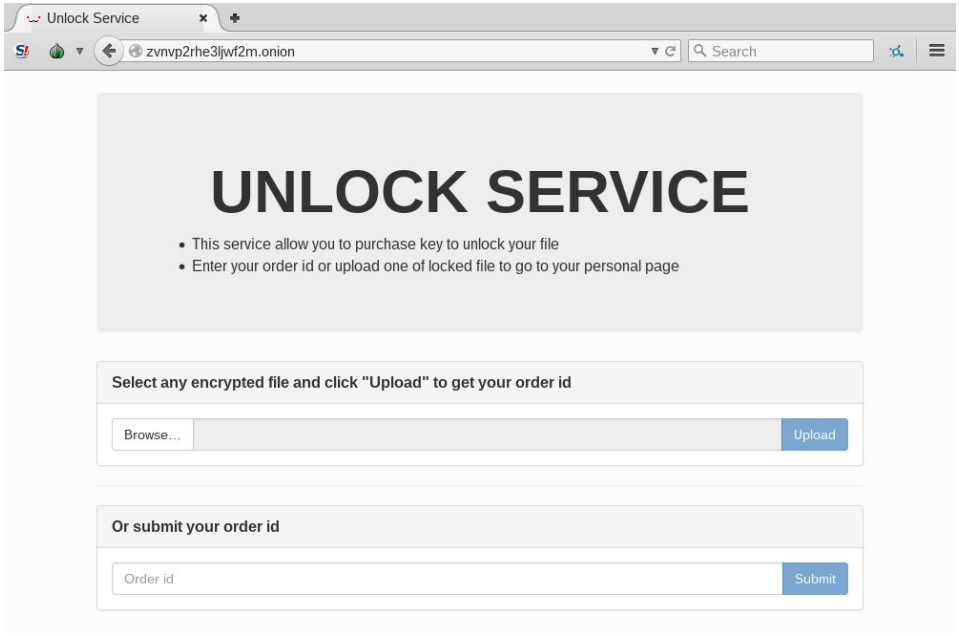
When the encryption finishes, the ransom note pops up:



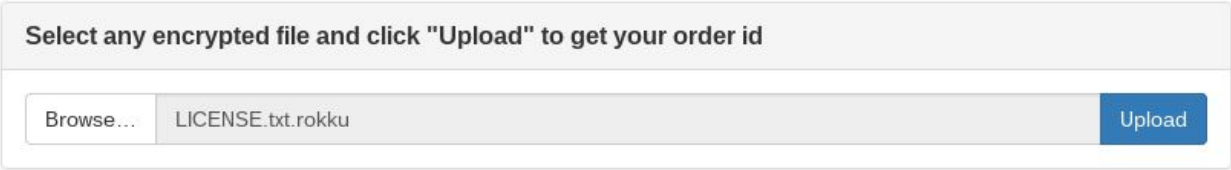
Website for the victim

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:



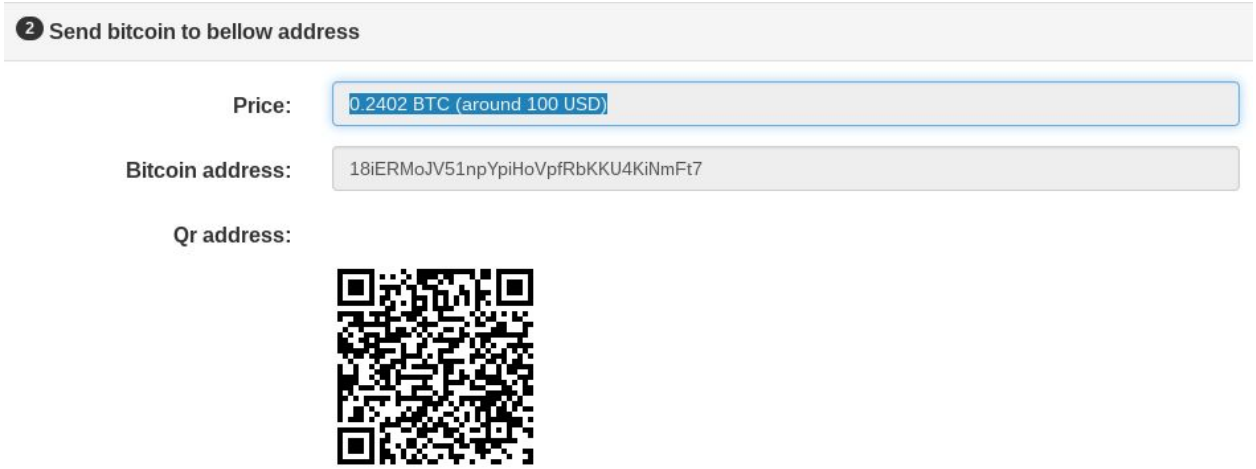
Rather than forcing a victim to type a unique ID it simply ask them to upload one file. All the necessary data are automatically fetched from the file.



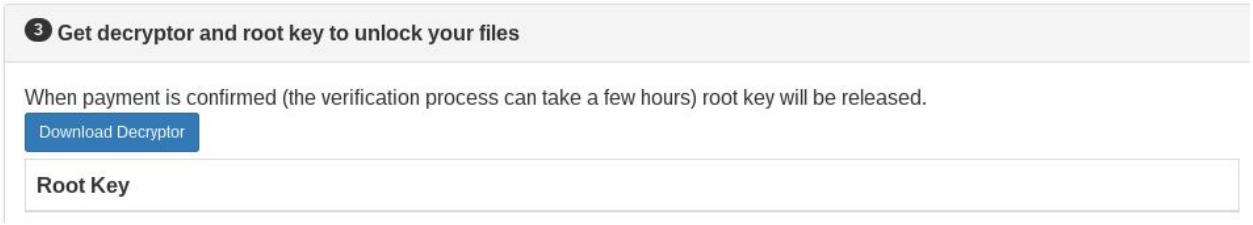
Then it redirects to the personalized part of the panel and shows the order ID. This unique identifier can be used further to regain access to this page without the necessity to upload a file again:

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

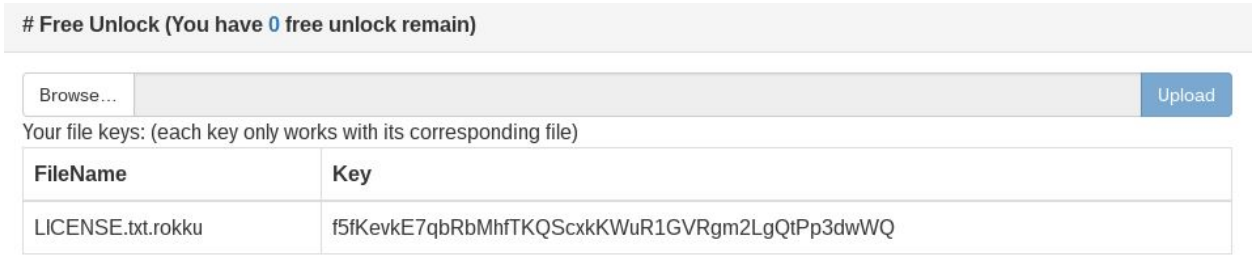
incremented with time – so we can assume, that in this case distributors decided to use a fixed price.



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.



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.



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.

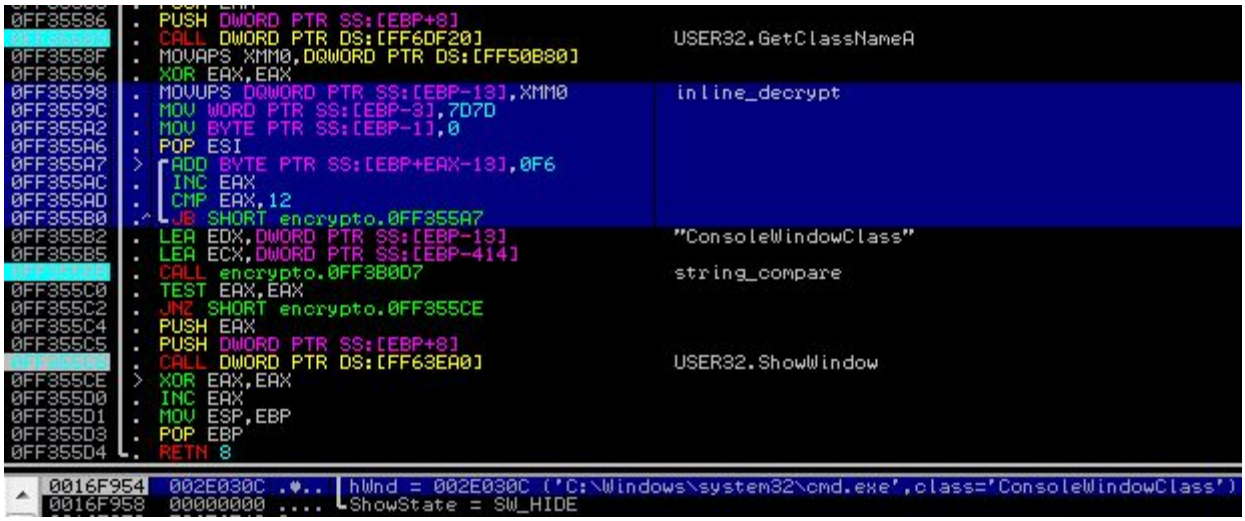
ReflectiveLoader is a special stub belonging to the technique of ***Reflective DLL Injection***. This technique allows to produce a DLL that can be easily injected into another process. Similarly to a shellcode, such DLL is self-contained and automatically loads all it's dependencies.

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:



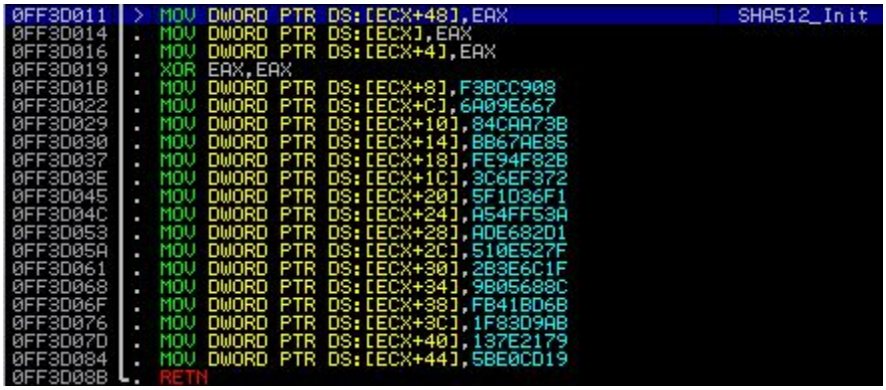
In order to make analysis harder, this malware uses encrypted strings. They are decrypted on fly, just before they are needed. As you can see at the above screenshot – it is implemented with the help of small in-line routine using SSE (highlighted in the picture). Using an in-line routine have an advantage 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)…

We can see the typical SHA512 constants in the code:



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:

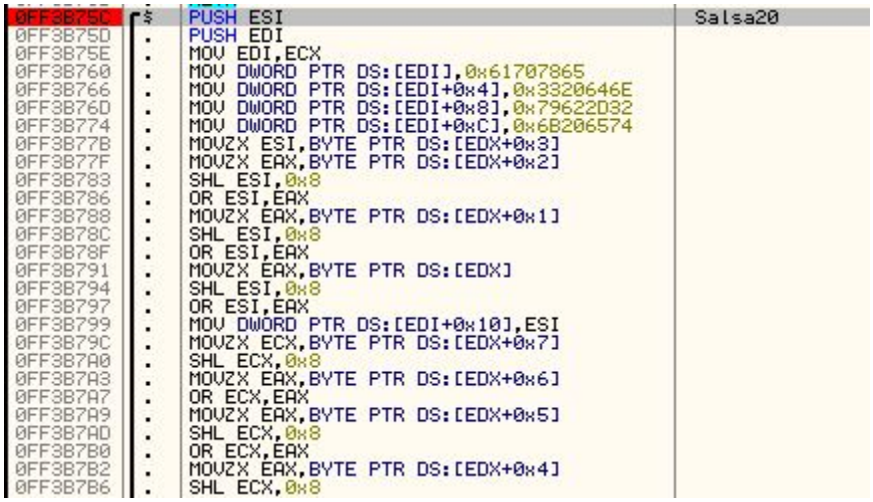
```
wmic shadowcopy delete /nointeractive
```

```
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:



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.

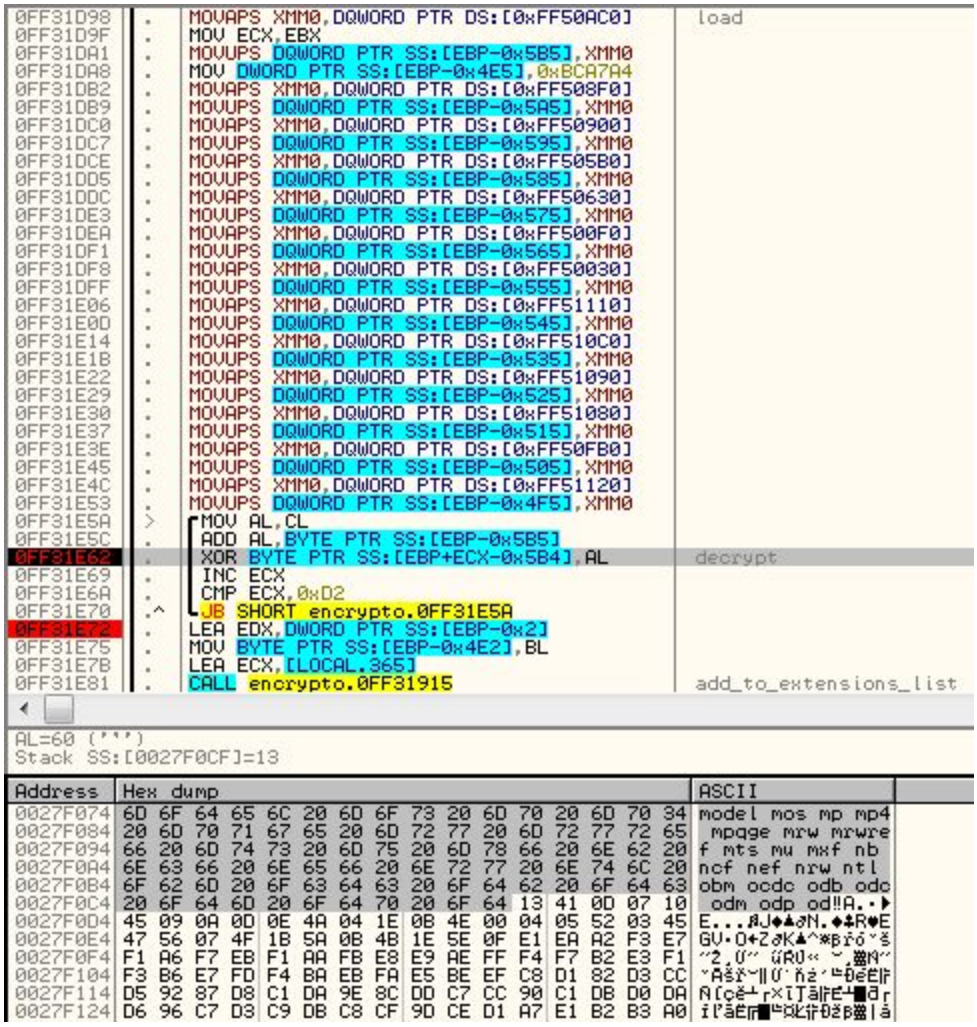
Research about the implementation details and possible flaws is in progress.

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):

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:



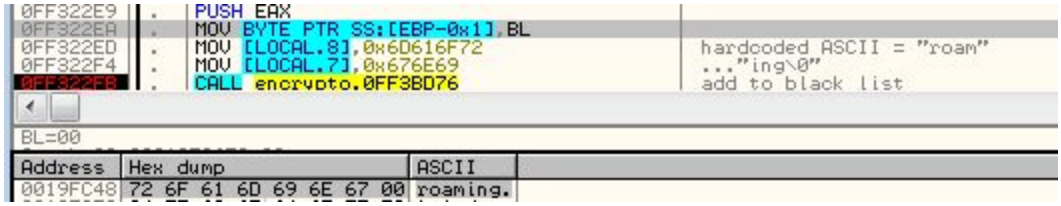
Summary of all the file extensions that are attacked:

```
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
```

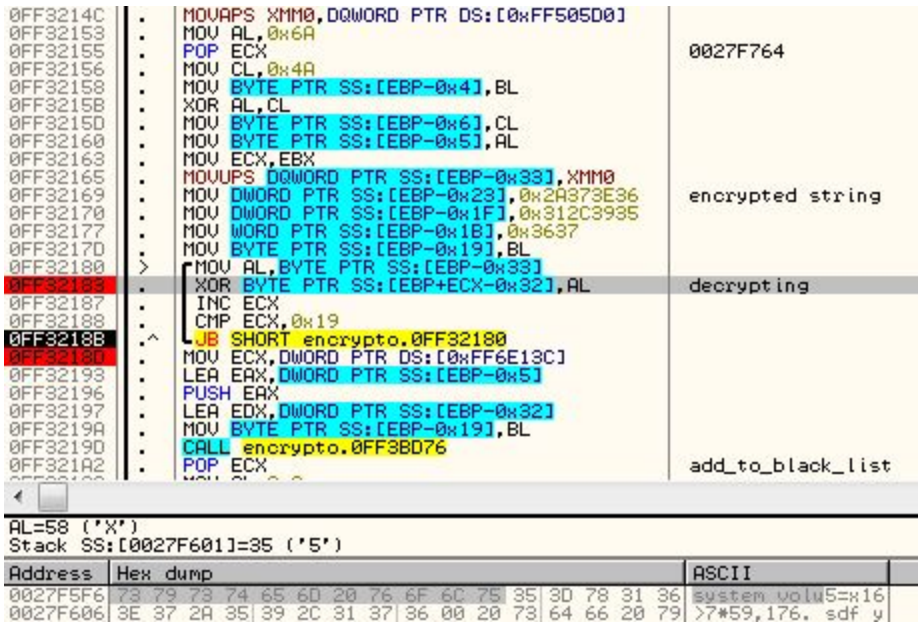
In the same way, blacklisted paths are deobfuscated and loaded.

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

Example 1 – adding hardcoded value “roaming”:



Example 2 – decrypting “system volume information”



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:

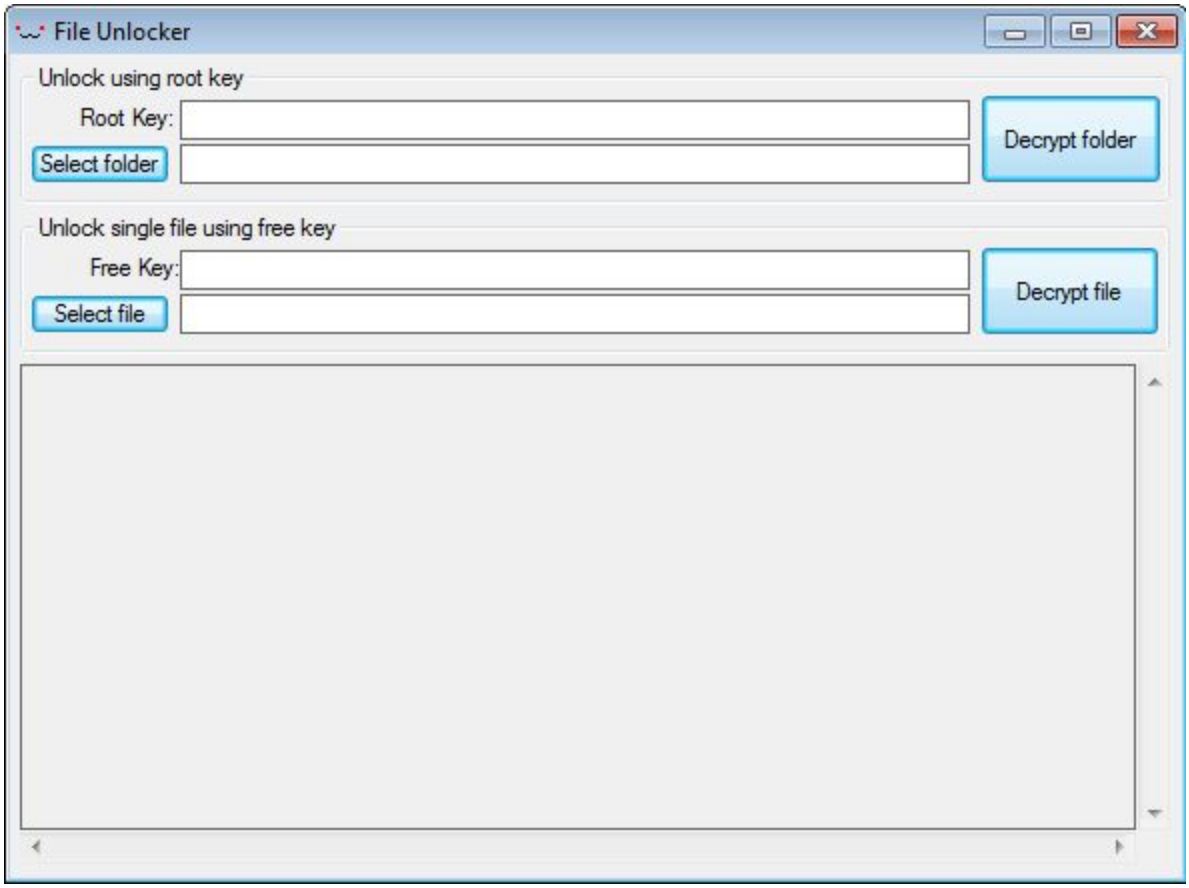
```
thumbs.db
```

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

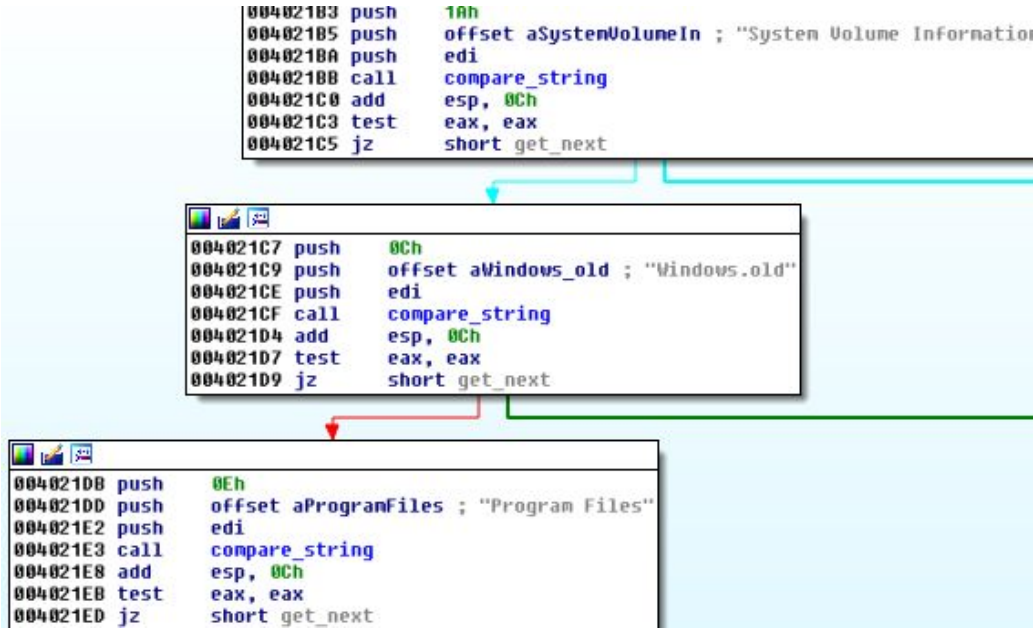
Inside the decryptor

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:



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