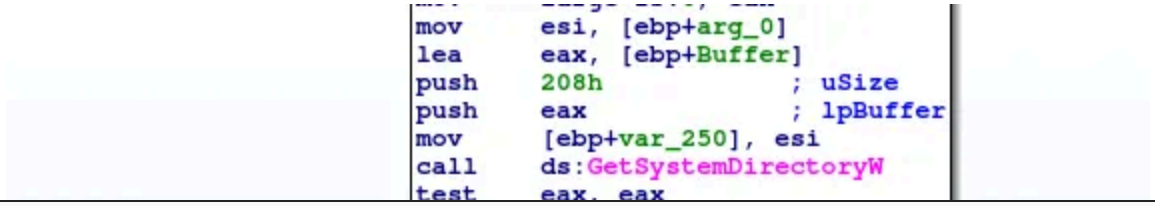
LockerGoga — input arguments, IPC communication and others

Initially, I thought that tgytutrc8597.exe file was unpacked out of original LockerGoga.exe binary, but then I saw this command in action, so it became clear for me why original file — in this case LockerGoga.exe — disappeared. It was just moved to some other place. Below you can also have a look at the function that is responsible for preparing the full path to cmd.exe.

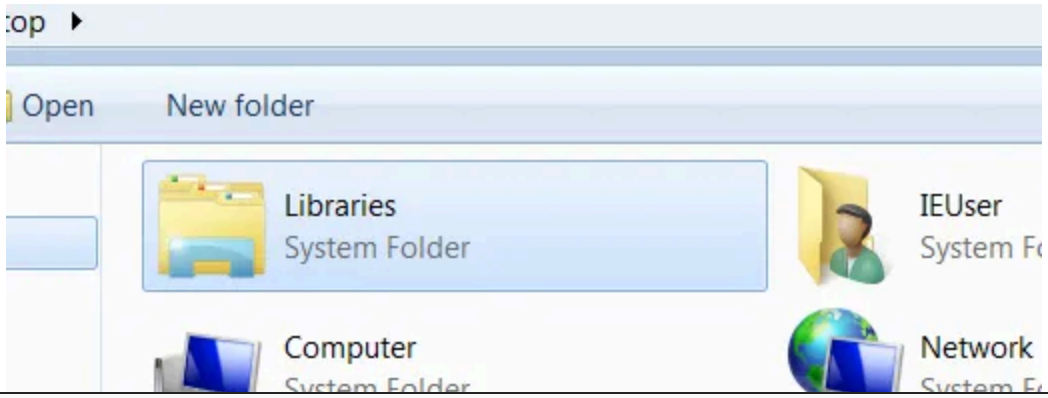


At first GetSystemDirectoryW() which in this case will return C:\Windows\system32 and then path is appended with “cmd.exe” string.

LockerGoga input arguments

Since I was experimenting with the sample I have found out one interesting thing while debugging it. If you start the dropper without any parameters then it will hide in %Temp% directory and start to encrypt your precious files. But if you will start the dropper with “-m” parameter it won’t hide anywhere. It will start its actions within the same place that it is currently

At first let's called it slave process, creates README_LOCKED.txt file on the Desktop.



Debugging the sample in a master mode lead me to the point that one of the call that eventually creates child processes was not simply call of the static address from process address space — it would be easy to follow the instruction flow in Ida. It was the address within the scope of process address space but to call it processor needed to jump to the address within ESI register. This is probably one of the obfuscation technique used to hide from reversing tools.

Another thing is that debugging lead me to IMO options parser. Below you can see screens from my OllyDBG window

Going back to the log option. While looking for some clues I got through strings in Ida. Guess what was there?

subroutines. At the end of this there were pretty interesting code block. Let's have a look at it.

```
scanning...
[1/0/330]>C:\a9f8e999f736c8dc9221\1032\LocalizedData.xml
[2/0/329]>C:\a9f8e999f736c8dc9221\1053\LocalizedData.xml
[1/1/329]+C:\a9f8e999f736c8dc9221\1032\LocalizedData.xml
[2/1/328]>C:\a9f8e999f736c8dc9221\1055\LocalizedData.xml
[1/2/328]+C:\a9f8e999f736c8dc9221\1053\LocalizedData.xml
[2/2/327]>C:\a9f8e999f736c8dc9221\Strings.xml
[1/3/327]+C:\a9f8e999f736c8dc9221\1055\LocalizedData.xml
...
[1/382/2163]>C:\ProgramData\Microsoft\Windows\Caches\{6AF0698E-
D558-4F6E-9B3C-3716689AF493}.2.ver0x0000000000000005.db
scan finised
[0/382/2164]-C:\ProgramData\Microsoft\Windows\Caches\{6AF0698E-
D558-4F6E-9B3C-3716689AF493}.2.ver0x0000000000000005.db
```

While I wanted to get into details of the function that seems to be responsible for IPC communication(I jumped into the binary segment where string ‘MX-tgytutrc’ was located) which I found very interesting to look into, I have found out list of files extensions — probably the ones that should be encrypted.

Since I am not Windows kind of guy, I would need some help from MSDN. There I have found more information about this particular function. Let's see.

eventually zeroed esi register), desiredAccess=0x1F0001(MUTEX_ALL_ACCESS). Then mutex handle(its address of course) is read from eax register and goes to hHandle local variable.

I have bolded the most important part. Thread will wait at most 10s(0x2710 = 10000ms) until mutex object will signal its state. In my own words it would be mutex that is locked and have not been freed within 10s. Then depending of the return value of the WaitForSingleObject() the code will go to the end of the function if 0x80(**WAIT_ABANDONED**) will be the result of the function all. Next block checks if edi register(where function result is stored) is equal to zero — signal came from mutex object.

As you can see those are arguments that CreateFileMappingA function has been called with. Especially MapName argument is really interesting. It seems like “SM-tgytutrc” comes from SharedMemory? Let’s wait for OpenFileMapping call and its arguments. Okay. There are so many call to

```
int _tmain()
{
   HANDLE hMapFile;
   LPCTSTR pBuf;
   hMapFile = OpenFileMapping(
                  FILE_MAP_ALL_ACCESS,   // read/write access
                  FALSE,                 // do not inherit the
name
                  szName);               // name of mapping
object
   if (hMapFile == NULL)
   {
      _tprintf(TEXT("Could not open file mapping object (%d).\n"),
            GetLastError());
      return 1;
   }
```

```
>>> base64.b64decode('!FzpcQ29uZmlnLk1zaVw1NWZmZi5yYmY=')
b'\x17:\\Config.Msi\\55fff.rbf'
>>> base64.b64decode('dHVwRW5naW5lLmRsbA==')
b'tupEngine.dll'
```

So it seems like this is were file paths are exchanged between process. Other decrypted base64 strings shows that these are some files paths, but not only those.

```
b'C:\\Config.Msi\\5667c.rbf'
>>>
base64.b64decode('XEFzc2lzdGFuY2VcQ2xpZW50XDEuMFxlbi1VU1xIZWxwX01L
V0RfQmVzdEJldC5IMVc=')
b'\\Assistance\\Client\\1.0\\en-US\\Help_MKWD_BestBet.H1W'
>>> base64.b64decode('Ny04RTdGLUJBM0YyNDczMkQ5NX0uSDFR')
b'7-8E7F-BA3F24732D95}.H1Q'
>>>
base64.b64decode('zpcUHJvZ3JhbURhdGFcTWljcm9zb2Z0XE5ldHdvcmtcRG93b
mxvYWRlclxxbWdyMS5kYXQ=NzQyLTRkOTYtYTUwYS0xNzc1ZmIxYTdhNDJ9XHByaW5
0X3F1ZXVlLmljbw==')
b'\xce\x97\x14\x1c\x9b\xd9\xdc\x98[Q\x18]\x18W\x13ZX\xdc\x9b\xdc\x
db\xd9\x9d\x17\x13\x99]\x1d\xdb\xdc\x9a\xd7\x11\x1b\xdd\xdb\x9b\x1
b\xd8Y\x19\\\x97\x1c[Y\xdc\x8cK\x99\x18]\x03sC"\xd3FC\x93b\xd6\x13
S\x06\x12\xd3\x13ssVf#\x16\x13v\x13C\'\xd5\xc7\x07&\x96\xe7E\xf7\x
17VWVR\xe6\x966\xf0'
```

Written by Malware Dancer

Follow

3 Followers

Low-level programmer, like to see how software works underneath
https://malwaredancer.com