LOCKY STRIKE: SMOKING THE LOCKY RANSOMWARE CODE

Floser Bacurio, Rommel Joven & Roland Dela Paz
Fortinet, Singapore

Email {fbacurio, rjoven, rdelapaz}@fortinet.com

ABSTRACT

In late January this year, an unknown TOR onion-based ransomware payment page surfaced. The new deep website didn't attract much attention; it was probably 'just another' script kiddie trying to get into the ransomware business. However, the third week of February saw a massive ransomware campaign that landed on at least 90,000 PCs per day [1] around the world – one that pointed users to the exact same TOR onion site in order to pay a ransom. The ransomware's name was 'Locky'.

At that point, not only did it become apparent that Locky was the work of experienced cybercriminals, but it was also clear that Locky was a major ransomware threat. In fact, Locky's early variants showed attributes that led us to believe it would become a prominent ransomware family alongside CryptoWall and TeslaCrypt.

In this paper, we will delve into the technical details of the Locky ransomware. We will focus on three technical aspects: its system behaviour, domain generation algorithm (DGA), and C&C communication.

Initially, we will talk about Locky's prevalence in the wild and how it behaves when it lands on a PC. We will then look at its DGA details and how we are able to simulate it in an automated fashion for C&C domain harvesting.

The paper will also explore Locky's obfuscated C&C communications, including its parameters, encryption and decryption. We will demonstrate how we successfully spoofed HTTP requests to the C&C servers in order to force them to respond with certain information, such as targeted countries.

The paper will conclude with some insights into Locky's operation and on how these findings ultimately translate to actionable threat intelligence that can be used to protect users.

1. INTRODUCTION

The Locky ransomware emerged in February this year and quickly [1] became one of the most prevalent pieces of ransomware in the wild. Initially, several users posted on forums seeking help regarding a new ransomware infection that uses the '.locky' extension. Soon after, a massive Locky spam run was observed by the security industry.

Fortinet was the first to publish in-depth technical details of the first version of the malware, in which Locky's Domain Generation Algorithm (DGA) and C&C communication and encryption were discussed [2]. While Locky's code was not complex at the time, it showed attributes that led *Fortinet*'s *FortiGuard Lion Team* researchers to believe that it would be a major threat moving forward. *FortiGuard Lion Team* kept track [3] of the threat, and the prediction turned out to be correct.

This paper will detail the results of the continuous monitoring of Locky. The paper will initially discuss Locky's prevalence in the wild using *FortiGuard Intrusion Prevention System* (IPS) telemetry. It will then delve into a technical analysis of the latest iteration of Locky's code. The paper will also discuss the timeline of Locky's code and routine updates as well as its C&C encryption and decryption process. Finally, using the technical knowledge acquired in the research, a number of intelligence-gathering approaches will be detailed that can be used in providing protection to users as quickly as possible.

2. PREVALENCE

Locky's prevalence is largely driven by an affiliate program – a program where third-party cybercriminal groups help spread the Locky binary to potential victims for a pay-per-install commission. To keep track of installs from third-party affiliates, Locky binaries have an 'affid' tag embedded in their code. This code is then sent to the Locky C&C via the malware's phone home request.

Table 1 shows a list of affiliate methods that have been observed.

affid	Method
1	Spam email containing an attached JavaScript or *MS Word* (macro) downloader
3	Spam email containing an attached JavaScript or *MS Excel* (macro) downloader
5	Spam email containing an attached JavaScript downloader
13	Compromised sites that redirect to Nuclear Exploit Kit
15	Spam email containing an attached JavaScript or HTA downloader

Table 1: Locky affiliates.

Figure 1 shows a screenshot of a spam email containing a piece of JavaScript that downloads Locky.

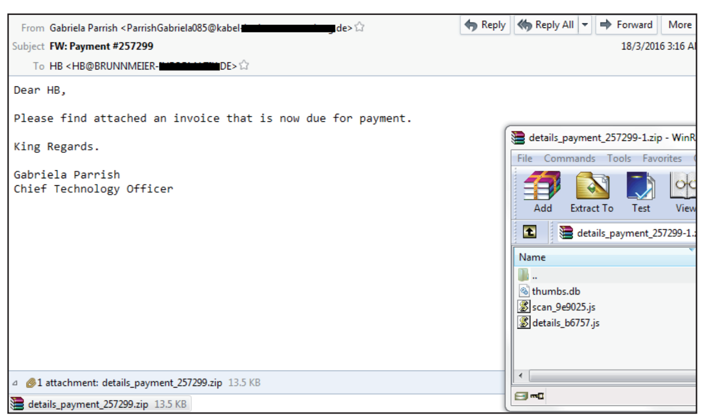

Figure 1: Spam email related to Locky.

These affiliates appear to be successful in spreading Locky. *FortiGuard Intrusion Prevention System* telemetry shows that Locky was ranked as the eighth most prevalent threat after only three months of operation. The statistics listed in Table 2 are *FortiGuard IPS* logs from 19 February 2016 to 19 May 2016.

Rank	Malware family
1	Andromeda
2	Zeroaccess
3	H-worm
4	Conficker
5	Necurs
6	Sality
7	CryptoWall
8	**Locky**
9	Ramnit
10	AAEH

Table 2: FortiGuard top 10 threats from 19 February 2016 to 19 May 2016.

Within the same timeframe, over 150 million total *FortiGuard IPS* hits from well-known ransomware families were logged.

Locky appeared as the second most prevalent ransomware family, as shown in Figure 2.

Figure 3 shows the daily activity of Locky in three months of operation. In total, *FortiGuard IPS* collected 62,599,466 hits from Locky C&C communication, averaging 687,906.2 hits per day.

The heatmap in Figure 4 shows Locky's global presence.

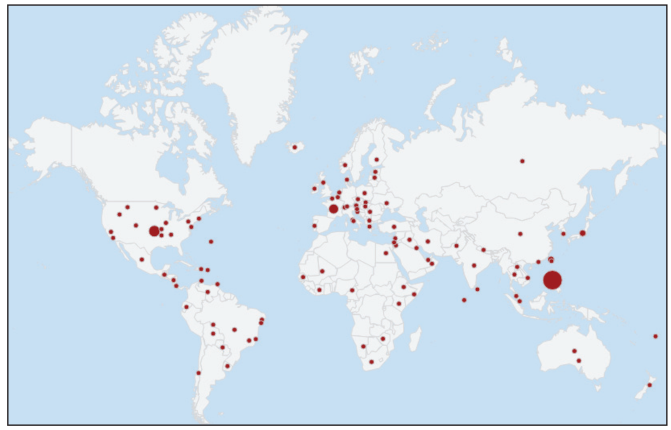

Figure 4: Heatmap of Locky infections from 19 February 2016 to 19 May 2016.

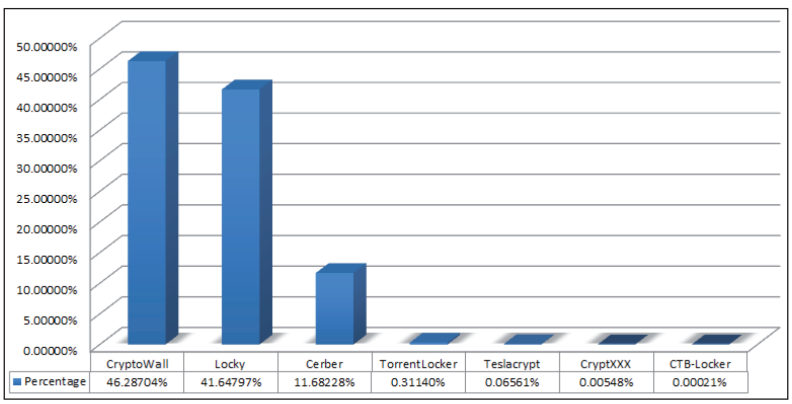

Figure 2: Ransomware prevalence from 19 February 2016 to 19 May 2016.

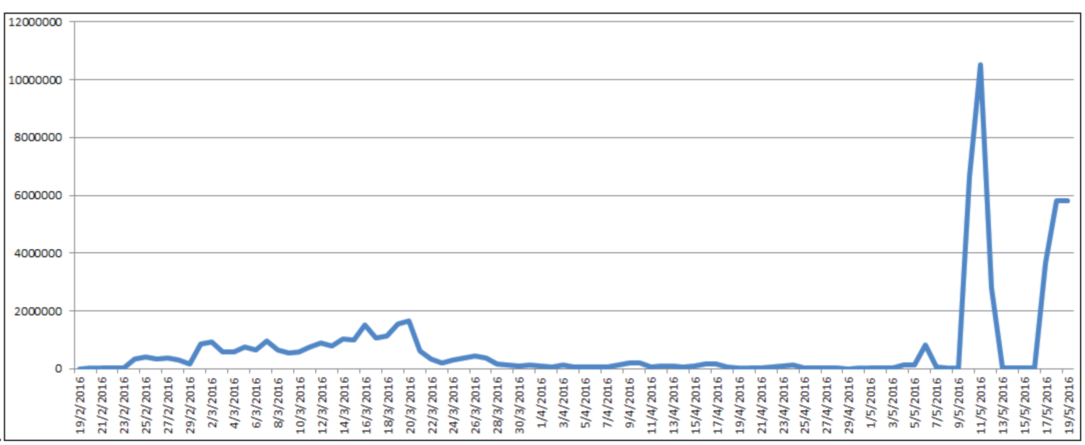

Figure 3: Locky daily activity from 19 February 2016 to 19 May 2016.

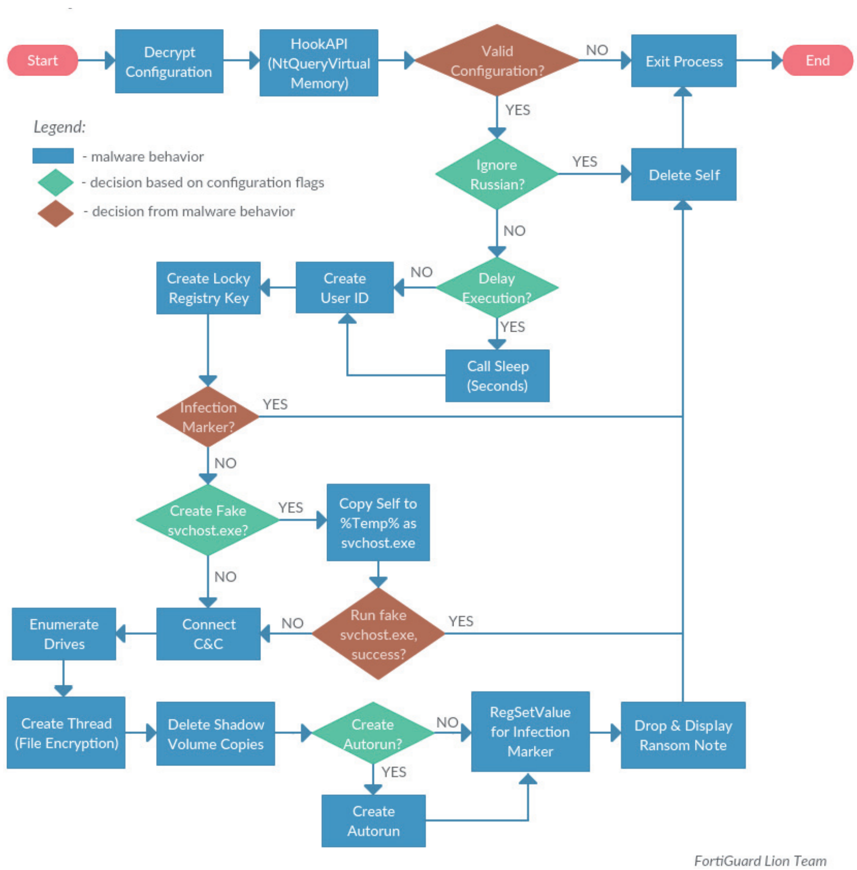

Figure 5: Locky behaviour flowchart.

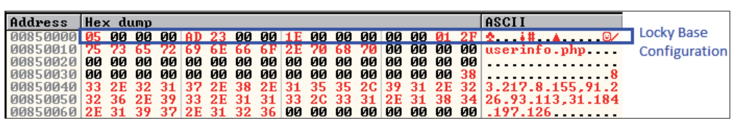

Figure 6: Locky configuration file.

3. TECHNICAL ANALYSIS

Overview

Table 3 lists the details of the sample used for analysis throughout the report.

MD5	94097c46248a187476908e3ff2cb6e97
SHA1	64917aab4c609fa62587d3f06428b0d94e1406f9
SHA256	8c73b04c6450651388d4605de113b156c39e0f22167b91c07884221a7ef767a7
Compile timestamp	2008-11-15 19:21:27
Size	147,968 bytes
File type	Win32 EXE

Table 3: Details of representative sample.

An overview of Locky's routine upon executing on a PC is shown in Figure 5.

Configuration

The malware routine begins by decrypting its configuration file and C&C (see Figure 6).

Table 4 shows Locky's configuration structure.

<table>
<tr><td>0x0</td><td>0x1</td><td>0X2</td><td>0X3</td><td>0X4</td><td>0X5</td><td>0X6</td><td>0X7</td></tr>
<tr><td colspan="4">Affiliate ID</td><td colspan="4">DGA seed</td></tr>
<tr><td colspan="4">Sleep (seconds)</td><td>Drop svchost.exe</td><td>Autorun</td><td>Check Russia</td><td>C&C offset</td></tr>
<tr><td colspan="8">URI (max length = C&C offset -1)</td></tr>
</table>

Table 4: Locky's configuration structure.

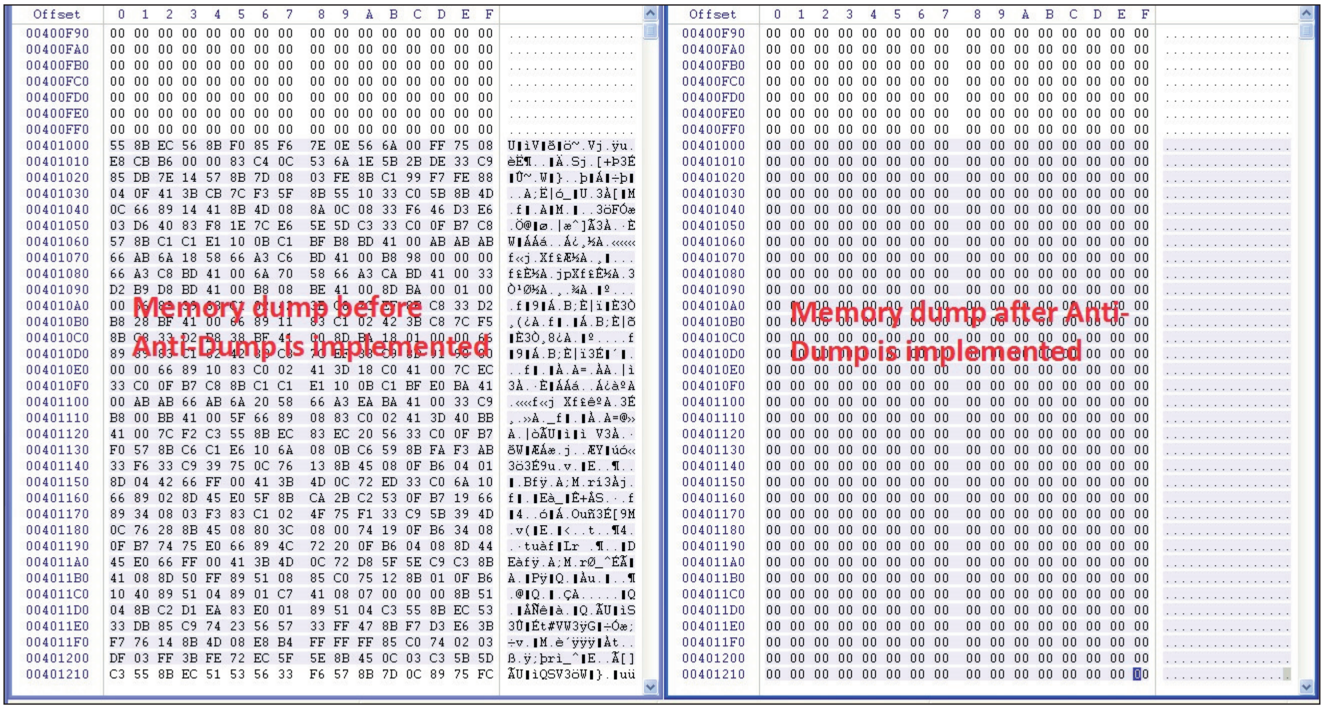

Figure 7: Locky's anti-memory dump example.

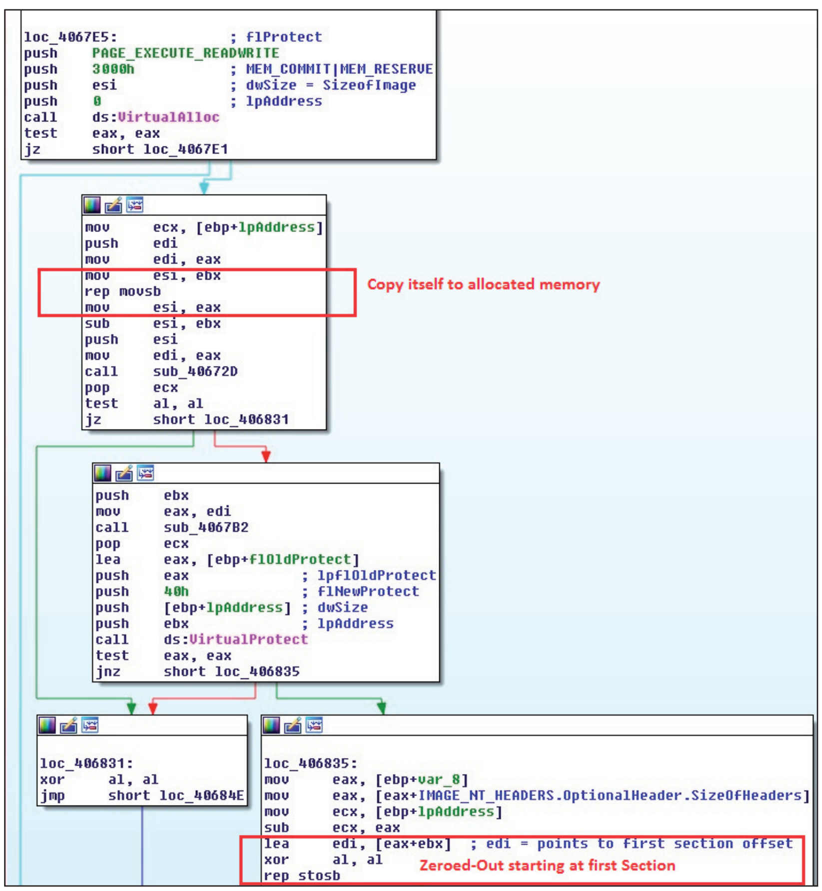

Figure 8: Code snippet for allocating memory, copying itself and zeroing out its own image.

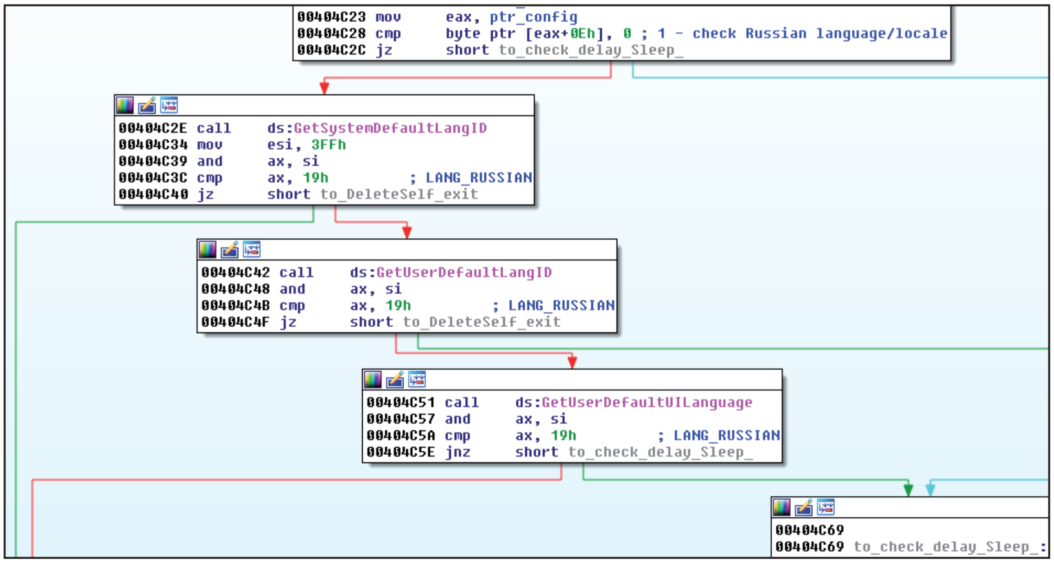

Figure 9: Code to verify if system is using Russian language.

As of the time of writing this paper, we have observed Locky to have used the following URIs for its C&C communication:

- main.php
- submit.php
- userinfo.php
- access.cgi
- /upload/_dispatch.php

Anti-memory dump

Locky employs a known technique for circumventing memory dump that has also been used by other malware families. This prevents an analyst from directly dumping the memory image of the malware while running (see Figure7).

To be able to do this, the malware allocates memory using the file's SizeOfImage value. This is to ensure there is enough memory allocated in order to successfully copy itself. It then transfers its execution code to the newly allocated memory. After that, it zeroes out the values from its own image memory, starting at the first section and continuing to the end of the allocated memory (Figure 8).

Locky then checks bases from its configuration to determine the user's language by calling the GetsystemDefaultLangID, GetUserDefaultLangID and GetUserDefaultUILanguage APIs. The malware immediately uninstalls itself if it finds itself running on a Russian-language computer.

Configuration flag(byte)	Value
0	Ignore Russian language
1	Check for Russian language

Table 5: Configuration flags for Russian computers.

Configuration offset +0x0E – check Russian language:

It continues to check its configuration to delay execution. It calls the Sleep API with a duration in seconds depending on the set value. This could be used as a technique to bypass sandbox and black-box testing.

Configuration offset +0x08 – duration of sleep (seconds):

Configuration flag(dword)	Value
0 to 0xFFFFFFFF	Sleep time in seconds

Table 6: Configuration for sleep duration.

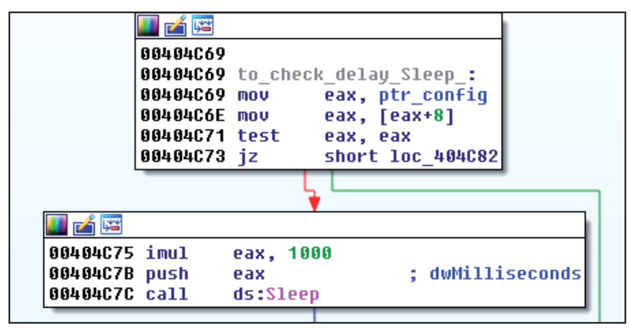

Figure 10: Code to execute sleep.

The malware then proceeds to create a unique user ID – a 16-byte-long hexadecimal string created locally:

```
Win_dir = GetWindowsDirectory

Vol_mount_point =
GetVolumeNameForVolumeMountPoint(Win_dir)

GUID = get_GUID(Vol_mount_point)

Hash_md5 = MD5(GUID)

User_id = Hash_md5.uppercase().substr(0,16)
```

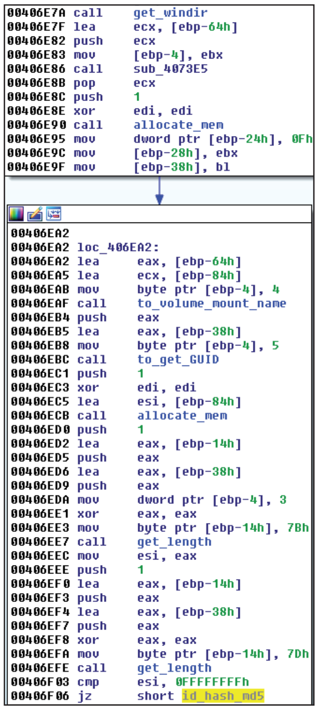

Figure 11: Unique user ID creation.

It creates a registry subkey where it will store the following encrypted data:

- RSA public key
- Ransom note in text file format
- Ransom note in HTML format
- Infection marker

It then calls the RegQueryValueExA API to get the infection marker in the registry data, decrypts the data and compares it to the string 'YES' (Figure 13).

If it finds that the user has already been infected, the malware will immediately uninstall itself. The malware once again checks its configuration to drop and run a copy of itself in the %temp% folder.

Configuration offset +0x0C – if 1, copy self as svchost.exe:

Configuration flag(byte)	Value
0	N/A
1	Create and run a copy of itself in %Temp% named as svchost.exe

Table 7: Configuration flag for svchost.exe process.

Figure 14: Code for creating svchost.exe copy.

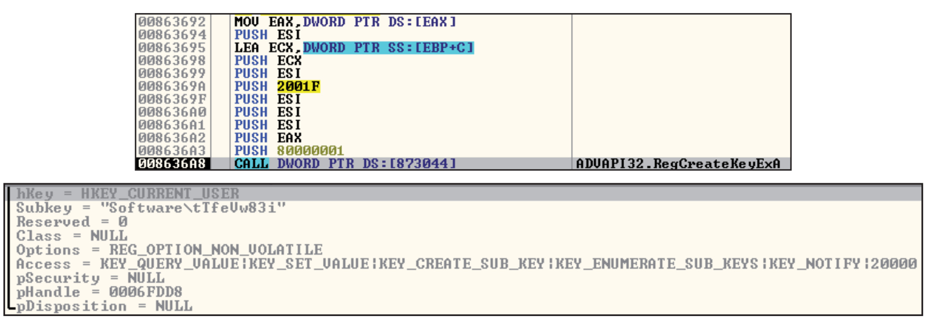

Figure 12: Registry subkey creation.

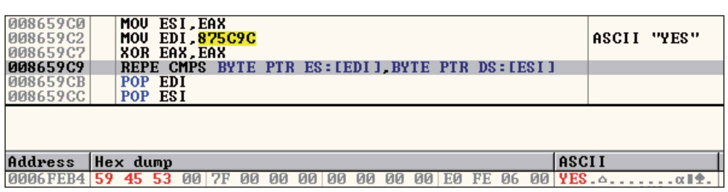

Figure 13: Infection verification.

File encryption

Locky starts by enumerating the drives in the victim machine by calling the GetDriveType API. It encrypts files on the following:

DriveType
DRIVE_REMOVABLE
DRIVE_FIXED
DRIVE_REMOTE
DRIVE_RAMDISK

Table 8: Drive types affected by Locky.

The malware then creates a thread for each logical drive seen in the victim machine with the targeted drive type. This thread's function is to encrypt the files located at the pushed root directory parameter.

```
00403EA8 ; DWORD __stdcall StartAddress(LPVOID root_dir)
00403EA8 StartAddress proc near
00403EA8
00403EA8 root_dir= dword ptr  4
00403EA8
00403EA8 mov     eax, offset loc_41217D
00403EAD call    SEH
00403EB2 sub     esp, 1Ch
00403EB5 and     dword ptr [ebp-4], 0
00403EB9 push    ebx
00403EBA push    esi
00403EBB push    edi
00403EBC mov     [ebp-10h], esp
00403EBF push    dword ptr [ebp+8] ; root_dir
00403EC2 lea     esi, [ebp-20h]
00403EC5 call    to_enumerateFiles
00403ECA pop     ecx
00403ECB mov     eax, esi
00403ECD push    eax
00403ECE push    dword ptr [ebp+8] ; root_dir
00403ED1 mov     byte ptr [ebp-4], 1
00403ED5 call    to_fileencryption_dropNote_getreport
```

Figure 15: File encryption function.

In the enumeration of files, Locky skip files where the full pathname contains one of the following strings:

_HELP_instructions.html, _HELP_instructions.bmp, _HELP_instructions.txt, _Locky_recover_instructions.bmp, _Locky_recover_instructions.txt, tmp, winnt, ApplicationData, AppData, ProgramFiles(x86), ProgramFiles, temp, thumbs.db, $Recycle.Bin, System VolumeInformation, Boot, Windows

Locky encrypts data and completely changes the filenames, adding the new extension '.locky'. It encrypts files with the following extensions:

.n64, .m4a, .m4u, .m3u, .mid, .wma, .flv, .3g2, .mkv, .3gp, .mp4, .mov, .avi, .asf, .mpeg, .vob, .mpg, .wmv, .fla, .swf, .wav, .mp3, .qcow2, .vdi, .vmdk, .vmx, .wallet, .upk, .sav, .re4, .ltx, .litesql, .litemod, .lbf, .iwi, .forge, .das, .d3dbsp, .bsa, .bik, .asset, .apk, .gpg, .aes, .ARC, .PAQ, .tar, .bz2, .tbk, .bak, .tar, .tgz, .gz, .7z, .rar, .zip, .djv, .djvu, .svg, .bmp, .png, .gif, .raw, .cgm, .jpeg, .jpg, .tif, .tiff, .NEF, .psd, .cmd, .bat, .sh, .class, .jar, .java, .rb, .asp, .cs, .brd, .sch, .dch, .dip, .pl, .vbs, .vb, .js, .h, .asm, .pas, .cpp, .c, .php, .ldf, .mdf, .ibd, .MYI, .MYD, .frm, .odb, .dbf, .db, .mdb, .sql, .SQLITEDB, .SQLITE3, .011, .010, .009, .008, .007, .006, .005, .004, .003, .002, .001, .pst, .onetoc2, .asc, .lay6, .lay, .ms11(Securitycopy), .ms11, .sldm, .sldx, .ppsm, .ppsx, .ppam, .docb, .mml, .sxm, .otg, .odg, .uop, .potx, .potm, .pptx, .pptm, .std, .sxd, .pot, .pps, .sti, .sxi, .otp, .odp, .wb2, .123, .wks, .wk1, .xltx, .xltm, .xlsx, .xlsm, .xlsb, .slk, .xlw, .xlt, .xlm, .xlc, .dif, .stc, .sxc, .ots, .ods, .hwp, .602, .dotm, .dotx, .docm, .docx, .DOT, .3dm, .max, .3ds, .xml, .txt, .CSV, .uot, .RTF, .pdf, .XLS, .PPT, .stw, .sxw, .ott, .odt, .DOC, .pem, .p12, .csr, .crt, .key, wallet.dat

Once a file to be encrypted is identified, the malware begins preparing the filename that it will be renamed as. The first 16 characters will be the unique ID of the victim and the next 16 characters will be the file ID, with the extension '.locky'.

Unique ID {16 char}	File ID{16 char}
UNICODE dump	
4DF383039AB03953	D81660EB4CADC28D.locky

Figure 16: Generated filename for encrypted file.

Below is a code snippet for generating the file ID:

```
x = [0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F]
length = 16
file_ID = []
while length > 0:
random_num = CryptGenRandom[4]
i = random_num mod 0x10
file_ID += x[i]
length --
```

The malware continues to create a file handle to the file to be encrypted; it then proceeds to call the MoveFileExW API in order to rename the file to the 32-character name (with .locky extension) that was prepared beforehand.

Using the CryptGenRandom API, it generates a random 16-byte value which will serve as the AES-128 key. Locky then uses *Intel*'s Advanced Encryption Standard Instruction (AES-NI) opcode aeskeygenassist to generate the AES round keys.

```
00401C62 loc_401C62:
00401C62 movzx   edx, ds:byte_4158E4[ecx]
00401C69 movdqa  xmm1, xmm0
00401C6D aeskeygenassist xmm2, xmm0, 0
00401C73 pslldq  xmm1, 4
00401C78 pxor    xmm0, xmm1
00401C7C movdqa  xmm1, xmm0
00401C80 pslldq  xmm0, 4
00401C85 pxor    xmm1, xmm0
00401C89 movdqa  xmm3, xmm1
00401C8D pslldq  xmm1, 4
00401C92 pshufd  xmm0, xmm2, 0FFh
00401C97 pxor    xmm3, xmm1
00401C9B pxor    xmm3, xmm0
00401C9F movd    xmm0, edx
00401CA3 pshufd  xmm0, xmm0, 0
00401CA8 pxor    xmm0, xmm3
00401CAC movdqa  xmmword ptr [eax], xmm0
00401CB0 inc     ecx
00401CB1 add     eax, 10h
00401CB4 cmp     ecx, 0Ah
00401CB7 jb      short loc_401C62
```

Figure 17: Locky AES round key generation.

The generated round keys will be used to encrypt targeted files and filenames, calling the opcode aesenc (Figure 18).

After encryption, the generated 16 bytes which served as the AES-128 key, will be encrypted by RSA-2048.

Figure 19 shows the encrypted file layout.

The malware deletes the backups by spawning this process by calling CreateProcessW: vssadmin.exe Delete Shadows /All /Quiet.

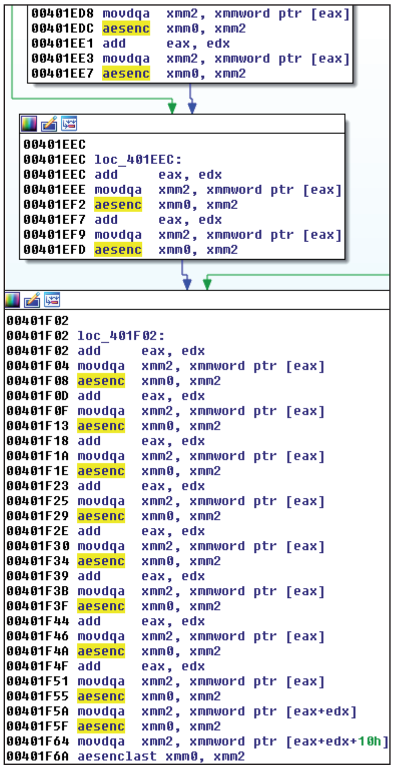

Figure 18: Locky AES round key generation via the aesenc and aesenclast instruction.

This will only work for infected users that have Administrator privileges.

Based on the configuration, the malware drops an autorun registry for the malware to run on every start up, as shown in Table 9.

Configuration flag(byte)	Value
0	N/A
1	Create autorun registry

Table 9: Configuration flags for autorun registry creation.

Configuration offset +0x0dh – autorun config.

Figure 20 shows an example of Locky's autorun registry key.

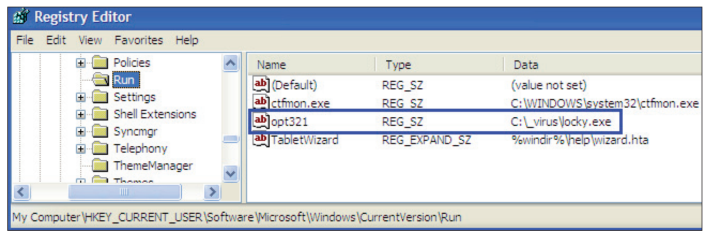

Figure 20: Locky autorun registry key.

It also creates a registry value to act as an infection marker, as shown in Figure 21.

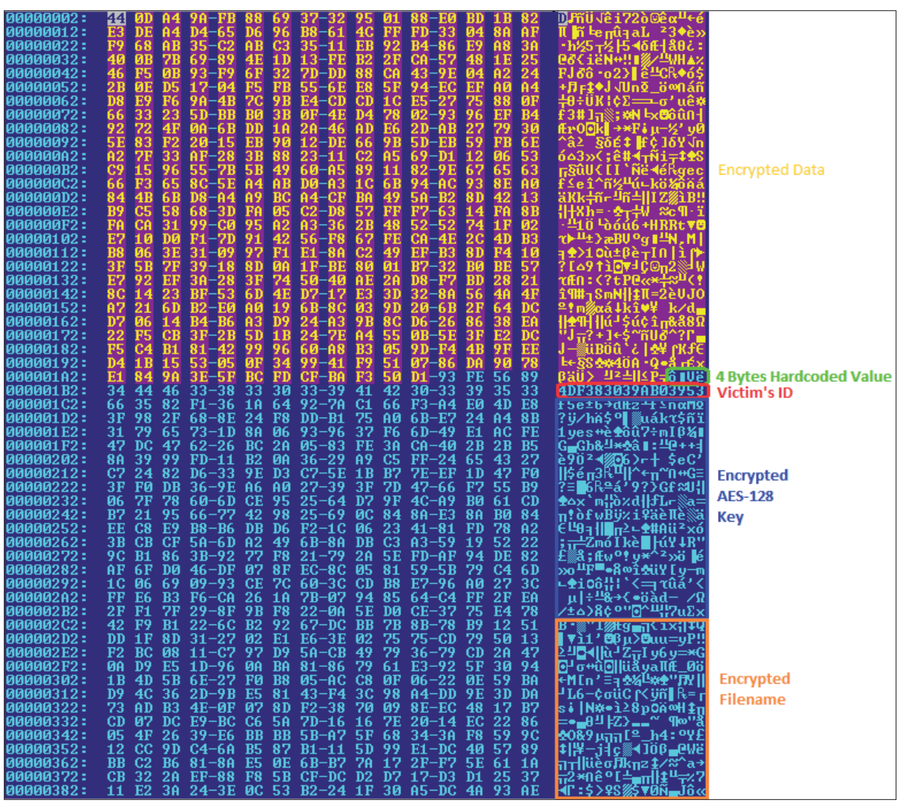

Figure 19: Encrypted file layout.

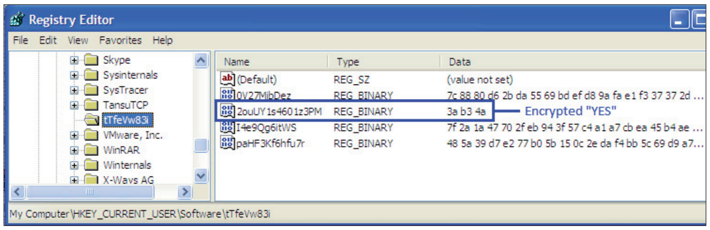

Figure 21: Locky infection marker registry.

Figure 22 shows the code that drops the HELP_instructions on the desktop.

```
004047E5 push    edi
004047E6 push    eax
004047E7 call    get_desktop_directory
004047EC xor     ebx, ebx
004047EE mov     [ebp-4], ebx
004047F1 lea     eax, [ebp-88h]
004047F7 mov     [esp+0A8h+var_A8], offset a_help_instruct ; "\\_HELP_instructions.html"
004047FE push    eax
004047FF lea     eax, [ebp-6Ch]
00404802 push    eax
00404803 call    sub_40575D
00404808 push    offset a_help_instru_0 ; "\\_HELP_instructions.bmp"
```

Figure 22: Code to drop help instructions.

Figure 23 shows the modification of wallpaper settings through the registry.

```
00404956 lea     eax, [ebp-2Ch]
00404959 push    eax
0040495A mov     ecx, offset aWallpaperstyle ; "WallpaperStyle"
0040495F mov     byte ptr [ebp-4], 7
00404963 call    set_reg_value
```

```
004049DD push    eax
004049DE mov     ecx, offset aTilewallpaper ; "TileWallpaper"
004049E3 mov     byte ptr [ebp-4], 8
004049E7 call    set_reg_value
```

Figure 23: Code to install wallpaper to the registry.

The code shown in Figure 24 sets the *Windows* wallpaper (0x14 = SPI_SETDESKWALLPAPER) and opens the dropped help_instructions file.

```
00404A09 push    ebx                     ; uiParam
00404A0A push    14h                     ; uiAction
00404A0C call    ds:SystemParametersInfoW
00404A12 cmp     dword ptr [ebp-58h], 8
00404A16 mov     eax, [ebp-6Ch]
00404A19 jnb     short loc_404A1E

00404A1B lea     eax, [ebp-6Ch]

00404A1E
00404A1E loc_404A1E:
00404A1E mov     esi, ds:ShellExecuteW
00404A24 push    1                       ; nShowCmd
00404A26 push    ebx                     ; lpDirectory
00404A27 push    ebx                     ; lpParameters
00404A28 push    eax                     ; lpFile
00404A29 mov     edi, offset Operation ; "open"
00404A2E push    edi                     ; lpOperation
00404A2F push    ebx                     ; hwnd
00404A30 call    esi ; ShellExecuteW
```

Figure 24: Code to modify wallpaper and open help instructions.

Figures 25 and 26 show screenshots of the ransom notes generated by Locky.

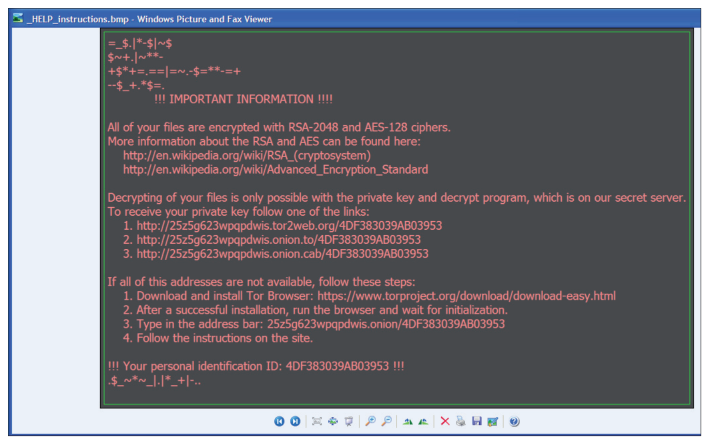

Figure 25: Locky help instructions in BMP format.

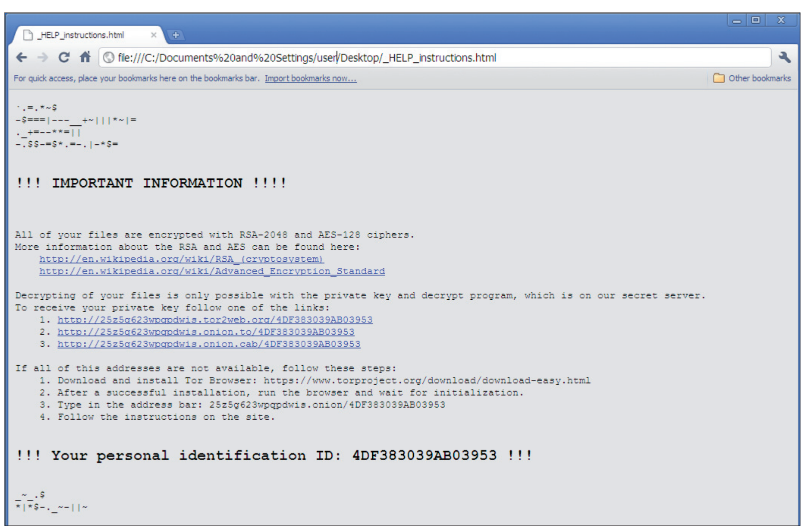

Figure 26: Locky help instructions in HTML format.

4. TIMELINE

Since Locky appeared in the wild, it has continually been updated by its perpetrators. The monitoring of Locky binaries appearing in the wild allowed the *FortiGuard Lion Team* to track code changes in the malware. Below are some of the iterations observed over time. It is important to note that the dates shown represent the earliest date that the updated Locky binary entered *FortiGuard*'s tracking system – actual code changes may have appeared earlier.

16 February 2016

- Sample is not packed
- Hard-coded configuration is not encrypted
- Hard-coded 'Locky' registry key is used
- Malware always runs as fake 'svchost.exe' in %Temp% folder
- Configuration format is as follows:

```
{
int AffiliateID;
char servers
}
```

- DGA TLD is 'rupweuinytpmusfrdeitbeuknltf'
- C&C urlPath is '/main.php'

22 March 2016

- Sample is packed
- Registry key name is generated based on affected computer's VolumeGUID
- Running as svchost.exe depends on the configuration flag
- Configuration format was updated to the following:

```
{
    int AffiliateID;
    int DGASeed;
    int delaySeconds;
    char bFakeSvchost;
    char bPersistence;
    char bIgnoreRussian;
    char[] ccServers;
}
```

- DGA TLDs are now 'ru', 'info', 'biz', 'click', 'su', 'work', 'pl', 'org', 'pw', and 'xyz'
- CC urlPath changed to '/main.php'
- DGA code is updated

31 March 2016

- Configuration is the same structure but is now encrypted
- CC urlPath is '/submit.php'

27 April 2016

- Custom encryption of HTTP communication with the C&C has been updated (details in the next section).
- Configuration now includes urlPAth with the value '/userinfo.php':

```
{
  int AffiliateID;
  int DGASeed;
  int delaySeconds;
  char bFakeSvchost;
  char bPersistence;
  char bIgnoreRussian;
  char[] urlPath;   // added update char[] ccServers;
}
```

30 May 2016

- Uses the new URI '/access.cgi'

31 May 2016

- Uses the new URI '/upload/_dispatch.php'
- Encrypted HTTP POST data is now encoded using percent encoding.

5. NETWORK BEHAVIOUR

While Locky's code was unsophisticated when it first came out, its network behaviour contained indicative signs that it was the work of experienced cybercriminals and would therefore become a major threat in the near future. Specifically, it employed a Domain Generation Algorithm, organized C&C reporting, and custom network communication encryption. This section will discuss the details of these routines.

Domain Generation Algorithm

Locky's DGA is a failover routine if the IPs listed in its configuration file are unreachable. Initially, the malware will try to connect to all IPs listed in its configuration. Failing to connect to any of the IPs will be its trigger to execute the DGA function (see Figure 27).

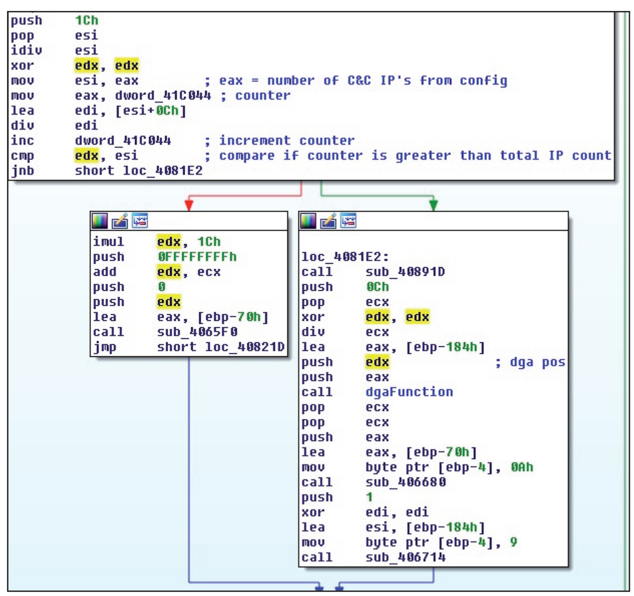

Figure 27: Locky's DGA trigger.

Figure 28 shows an opcode of the actual DGA routine. It is based on the affected machine's year, month, day, and a DGA seed value declared in its configuration file.

```
sub_410B94();
var_dgapos = *(_DWORD *)(a1 + 0xC);             // dga pos
var_dgaseed = dgaSeed;
v3 = 0;
*(_DWORD *)(a1 - 24) = 0;
GetSystemTime((LPSYSTEMTIME)(a1 - 0x28));
v4 = *(_WORD *)(a1 - 40);
v5 = (unsigned int)*(_WORD *)(a1 - 0x22) >> 1;
var_dgapos = __ROL4__(var_dgapos, 0x15);
v6 = __ROL4__(var_dgaseed, 0x11);
*(_DWORD *)(a1 - 0x18) = v6 + var_dgapos;
*(_DWORD *)(a1 - 0x14) = v5;
*(_DWORD *)(a1 - 0x10) = 7;
while ( 1 )
{
  v7 = __ROR4__(0xB11924E1 * (v4 + v3 + 7157), 7);
  v8 = (v7 + 0x27100001) ^ v3;
  v9 = __ROR4__(2971215073 * (v8 + var_dgaseed), 7);
  v10 = (v9 + 0x27100001) ^ v8;
  v11 = __ROR4__(0xB11924E1 * (v5 + v10), 7);
  v12 = 0xD8EFFFFF - v11 + v10;
  v13 = __ROR4__(0xB11924E1 * (*(_WORD *)(a1 - 38) + v12 - 0x65CAD), 7);
  v14 = v12 + v13 + 0x27100001;
  v15 = __ROR4__(0xB11924E1 * (v14 + *(_DWORD *)(a1 - 24)), 7);
  v3 = (v15 + 0x27100001) ^ v14;
  ++v4;
  v16 = (*(_DWORD *)(a1 - 16))-- == 1;
  if ( v16 )
    break;
  v5 = *(_DWORD *)(a1 - 20);
}
*(_DWORD *)(a1 - 48) = 15;
*(_DWORD *)(a1 - 52) = 0;
*(_BYTE *)(a1 - 68) = 0;
*(_DWORD *)(a1 - 24) = v3 % 0xBu + 7;
*(_DWORD *)(a1 - 4) = 0;
*(_DWORD *)(a1 - 16) = 0;
if ( v3 % 0xBu != 0xFFFFFFF9 )
{
  do
  {
    v17 = __ROL4__(v3, *(_BYTE *)(a1 - 16));
    v18 = __ROR4__(0xB11924E1 * v17, 7);
    v3 = v18 + 0x27100001;
    sub_405E30(1, v3 % 0x19u + 'a');
    ++*(_DWORD *)(a1 - 16);
  }
  while ( *(_DWORD *)(a1 - 0x10) < *(_DWORD *)(a1 - 24) );
}
```

Figure 28: Locky's DGA function.

C&C reporting

To prepare the phone home request, Locky gathers information about the victim machine and stores it in a key = value format. It collects the following information:

- Role information
- *Windows* operating system version
- User language
- Victim MD5 unique identifier

The role information is identified by making a call to the DsRoleGetPrimaryDomainInformation API with the local computer as the argument. This retrieves the state of the directory service installation and domain data, as shown in Figure 29.

By querying the return data of the API, the malware is able to determine if the computer is a server, a part of a domain or a primary domain controller. Table 10 shows the possible return values.

Integer	Computer role	
0	DsRole_ RoleStandaloneWorkstation	The computer is a workstation that is not a member of a domain
1	DsRole_ RoleMemberWorkstation	The computer is a workstation that is a member of a domain
2	DsRole_RoleStandaloneServer	The computer is a server that is not a member of a domain
3	DsRole_RoleMemberServer	The computer is a server that is a member of a domain
4	DsRole_ RoleBackupDomainController	The computer is a backup domain controller
5	DsRole_ RolePrimaryDomainController	The computer is a primary domain controller

Table 10: DsRoleGetPrimaryDomainInformation return values.

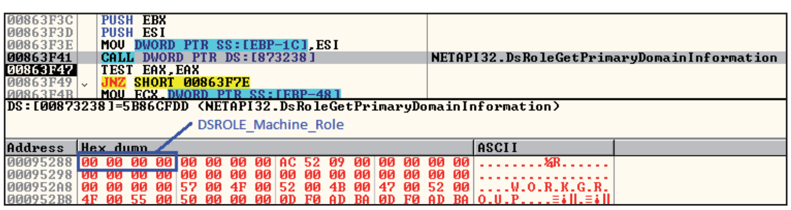

Figure 29: Code to retrieve the state of directory service installation and domain data.

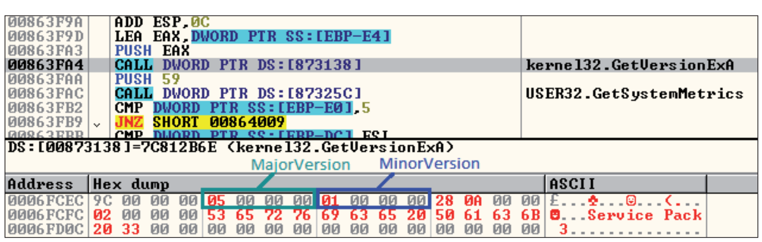

Figure 30: Code to retrieve operating system version.

The operating system version, on the other hand, is obtained by querying the OSMajorVersion and OSMinorVersion from the returned value when calling the GetVersionExA API.

The malware is able to determine the following *Windows* versions:

Windows 2000	*Windows 8*
Windows XP	*Windows Server 2012*
Windows 2003	*Windows 8.1*
Windows 2003 R2	*Windows Server 2012 R2*
Windows Vista	*Windows 10*
Windows Server 2008	*Windows Server 2016 Technical Preview*
Windows 7	Unknown
Windows Server 2008 R2	

The malware then retrieves the local language by calling the GetUserDefaultUILanguage API, which will be used to determine the language of the ransom note to be requested from the C&C, as shown in Figure 31.

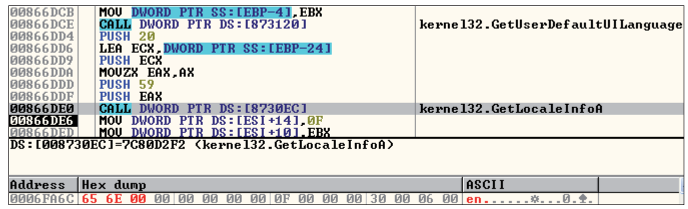

Figure 31: Code to retrieve the system's local language.

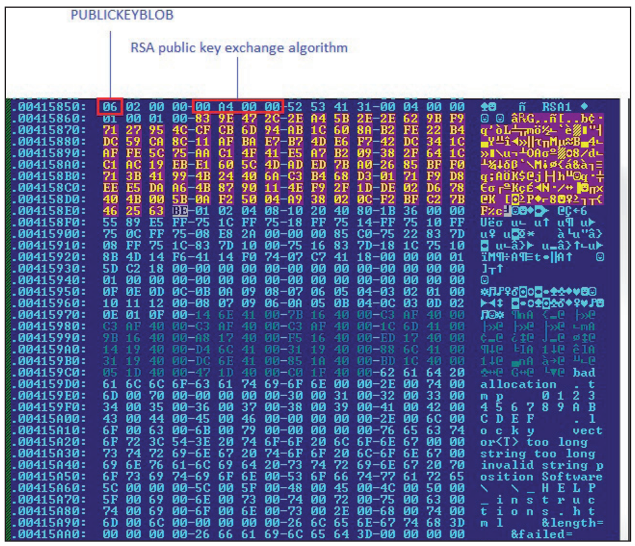

Figure 32: Public RSA-1024 key embedded in Locky binary.

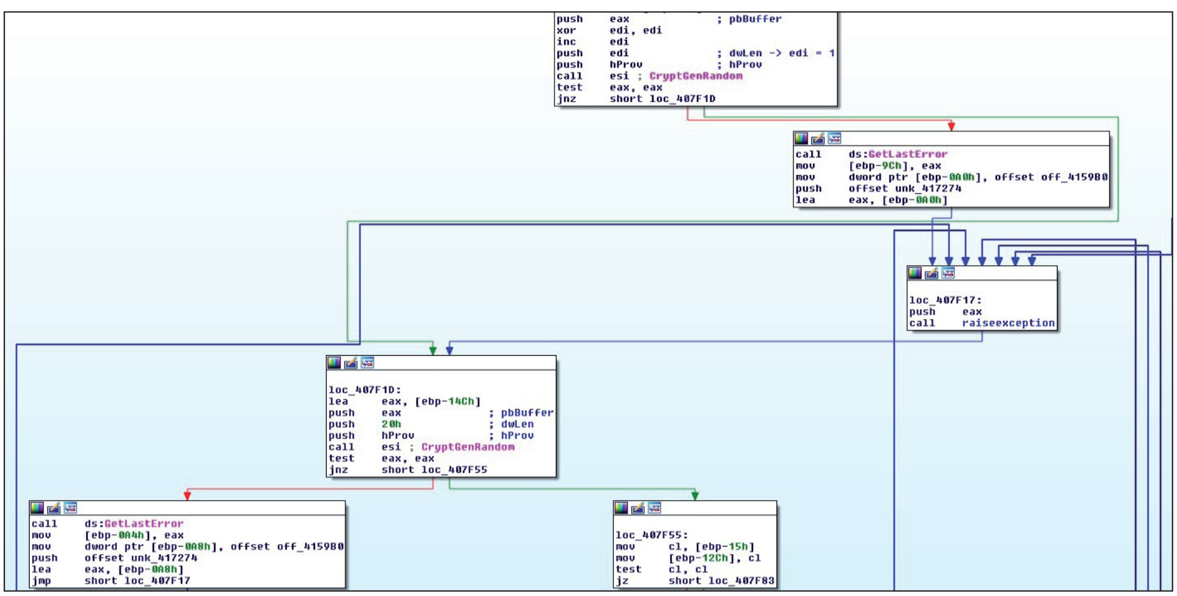

Figure 33: Code to generate random bytes for null byte size and AES-256 key generation.

Table 11 lists Locky's current C&C parameters and their descriptions.

Key	Value	Purpose
id		Victim's identification
&act	getkey gettext gethtml stats	RSA public key Ransom note in text Ransom note in HTML format Statistics of file encryption from victim's PC
&affid		
&lang	2 letter code	Victim's local language
&corp	0 1 2	Computer is not a member of a domain Computer is a member of a domain Computer is a primary domain controller
&serv	0 1	Not server Server
&os	char	*Windows* operating system version
&sp	number	Service pack
&x64	0 1	Not 64-bit 64-bit
&length	number	
&failed	number	Number of failed encrypted files
&encrypted	number	Number of successful encrypted files
&path		Root path

Table 11: Locky HTTP POST request parameters.

Network encryption – post request encryption

Initially, the malware will obtain a public RSA-1024 key embedded in the binary file to encrypt data in the following format:

> [random 32 bytes AES-256 key + random single byte (null byte size) + HMAC of plaintext request]

Using the CryptGenRandom() API, it generates a random single byte that serves as the size of null bytes to be appended to the request. It also uses this API to generate a 32-byte AES-256 key, as shown in Figures 33 and 34.

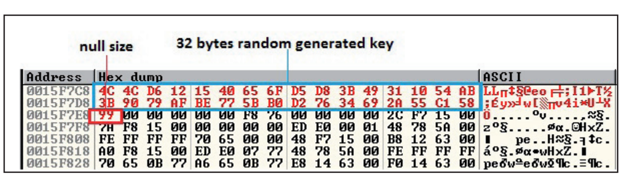

Figure 34: Generated random 32-byte AES key.

The generated 32-byte key has a dual purpose – it is used as a key for AES-256 encryption and for HMAC hash calculation.

For the HMAC hash calculation, it uses the CryptImportKey() API to create an RC2 key handle, as shown in Figure 35.

For AES-256 encryption, it uses the AES-NI extended instruction to generate encryption round keys that will be used to encrypt the plaintext request (Figures 37 and 38).

Figure 35: Code to set RC2 handle for HMAC calculation.

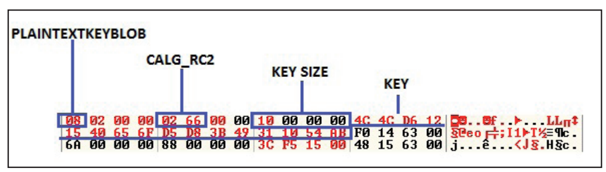

Figure 36: PUBLICKEYSTRUCT blob header.

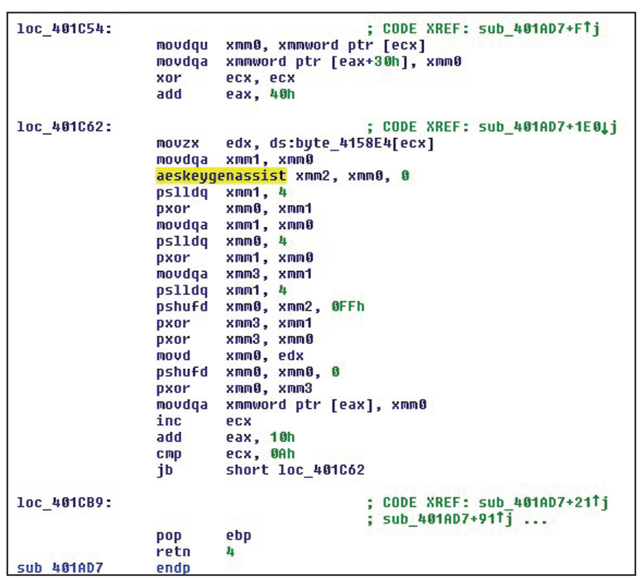

Figure 37: Encryption round keys generation routine.

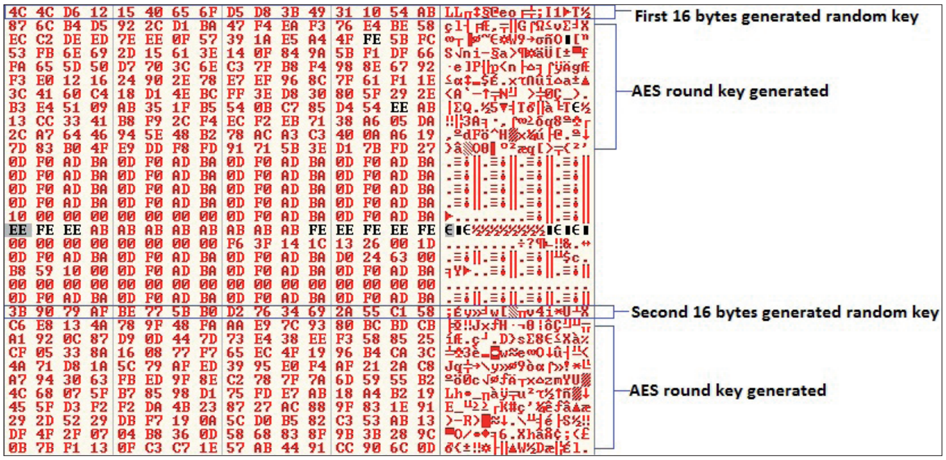

Figure 38: AES round keys generated.

Figure 39 shows a code snippet of the HMAC calculation of the plaintext request with null bytes appended. As shown in Figure 40, the result is concatenated to generated random bytes [32 bytes(AES-256 key) + single byte(null byte size)].

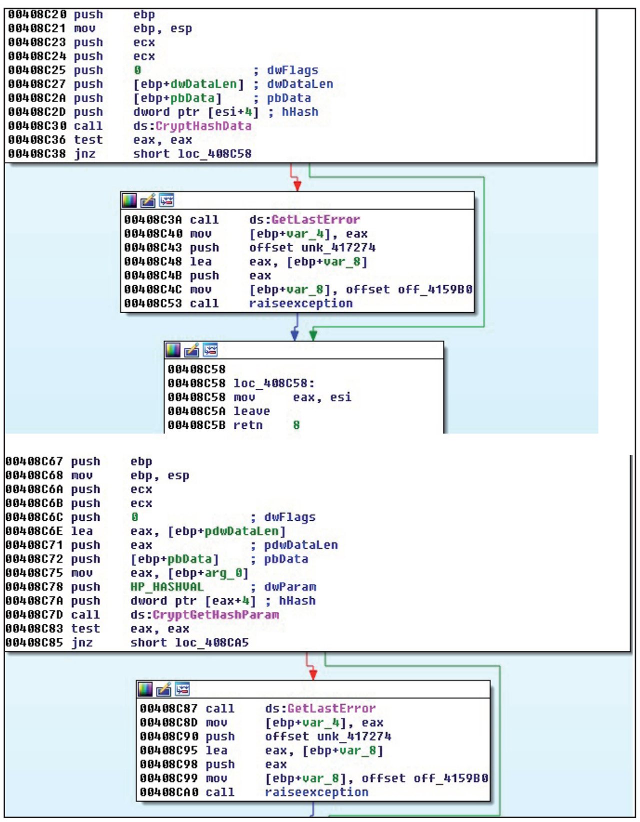

Figure 39: Calculation of HMAC of the plaintext request.

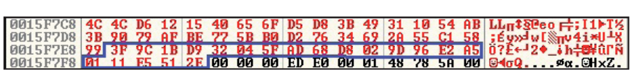

Figure 40: Concatenation of HMAC result.

Figure 41 shows the encryption of the plaintext request with null bytes appended using the generated AES round keys.

```
                movdqa  xmm5, xmmword ptr [eax]
                movdqa  xmm0, xmm1
                pshufb  xmm0, xmm2
                pxor    xmm0, xmm5
                cmp     esi, edx
                jz      short loc_401DFE
                cmp     esi, 18h
                jz      short loc_401DE8
                cmp     esi, 20h
                jnz     loc_401E6B
                add     eax, edx
                movdqa  xmm5, xmmword ptr [eax]
                aesenc  xmm0, xmm5
                add     eax, edx
                movdqa  xmm5, xmmword ptr [eax]
                aesenc  xmm0, xmm5

loc_401DE8:                             ; CODE XREF: sub_401D47+80↑j
                add     eax, edx
                movdqa  xmm5, xmmword ptr [eax]
                aesenc  xmm0, xmm5
                add     eax, edx
                movdqa  xmm5, xmmword ptr [eax]
                aesenc  xmm0, xmm5

loc_401DFE:                             ; CODE XREF: sub_401D47+7B↑j
                add     eax, edx
                movdqa  xmm5, xmmword ptr [eax]
                aesenc  xmm0, xmm5
                add     eax, edx
                movdqa  xmm5, xmmword ptr [eax]
                aesenc  xmm0, xmm5
                add     eax, edx
                movdqa  xmm5, xmmword ptr [eax]
                aesenc  xmm0, xmm5
                add     eax, edx
                movdqa  xmm5, xmmword ptr [eax]
                aesenc  xmm0, xmm5
                add     eax, edx
                movdqa  xmm5, xmmword ptr [eax]
                aesenc  xmm0, xmm5
                add     eax, edx
                movdqa  xmm5, xmmword ptr [eax]
                aesenc  xmm0, xmm5
                add     eax, edx
                movdqa  xmm5, xmmword ptr [eax]
                aesenc  xmm0, xmm5
                add     eax, edx
                movdqa  xmm5, xmmword ptr [eax]
                aesenc  xmm0, xmm5
                movdqa  xmm5, xmmword ptr [eax+edx]
```

Figure 41: Encryption routine of plaintext request.

Using the CryptEncrypt() API, it encrypts [32-bytes (AES-256 key) + single byte(null byte size) + HMAC] using the RSA public key embedded in the binary, as shown in Figures 42 and 43.

Finally, the encrypted plaintext request and [32-bytes (AES-256 key) + single byte (null byte size) + HMAC] data are combined.

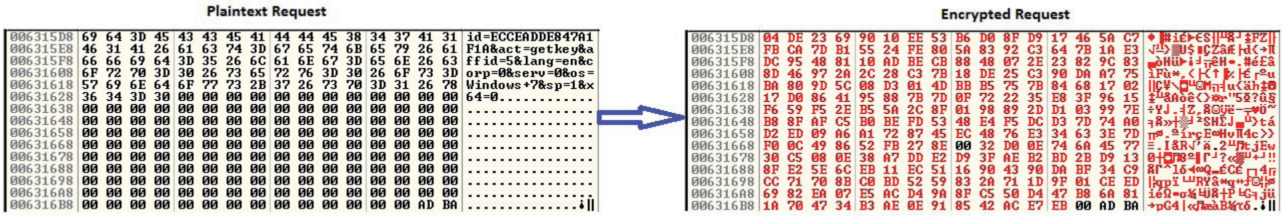

Figure 42: Encrypted plaintext request sample.

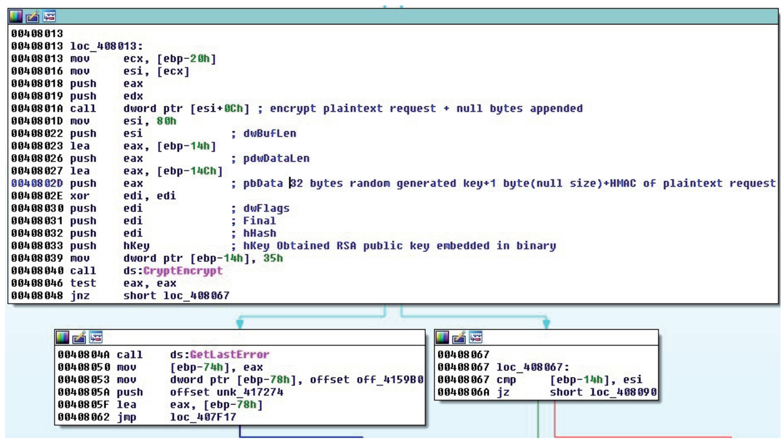

Figure 43: Encrypts [32-bytes (AES-256 key) + byte(null byte size) + HMAC].

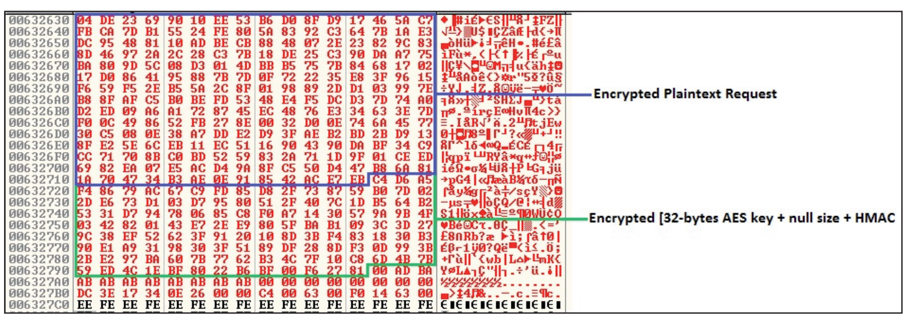

Figure 44: Encrypted plaintext request + [32-bytes (AES-256 key) + byte(null byte size) + HMAC].

6. INTELLIGENCE EXTRACTION

Apart from sourcing Locky binaries in the wild, malware metadata can be collected from Locky binaries in an automated fashion.

Collecting ransomware languages used

The very first version of Locky uses a custom algorithm to encrypt and decrypt its C&C communication. To get the ransomware note, it sends the following HTTP request format:

id={randomly generated victim ID}&act=gettext&lang={system language}

To get the system language, Locky calls the GetUserDefaultUILanguage API, which returns the language identifier for the UI language for the current user. *Microsoft*'s Language Identifier Constant and String provides a list of country codes for all supported languages.

Locky's HTTP request can then be spoofed through a script that feeds all available country codes from *Microsoft*'s website to the {system language} parameter, encrypts the request using the malware's algorithm, and then sends the encrypted request to a live Locky C&C server.

Using this approach, the C&C replies for different country codes are then hashed to identify unique ransomware notes. The following languages have been identified to be supported by Locky:

Country code	Language
de	German
en	English
es	Spanish
fr	French
it	Italian
ja	Japanese
nl	Dutch
no	Norwegian
pl	Polish
pt	Portuguese
ro	Romanian
sv	Swedish
zh	Chinese

Table 12: Locky ransomware note languages.

After identifying the above list, a script that simulates Locky's decryption algorithm is used to decrypt the ransomware notes. For unsupported country codes, the default ransomware note served is in English.

The current iteration of Locky uses a more complex C&C communication encryption. A similar approach can be used to collect the supported languages.

Collecting randomly generated domains

Similar to its network encryption, Locky's Domain Generation Algorithm can be simulated through a tool that will allow for proactive harvesting of malicious domains. The next step is to identify which of the random domains are actually used by the cybercriminals in order to block them accordingly. In addition, C&C sinkholes should be properly identified.

One approach is to send a ping request to the domains generated by the DGA tool. If there is a reply, the next verification stage can be a spoofed encrypted HTTP request made in a similar fashion with collecting ransomware notes. The size of the reply can then be compared to the *minimum* file size of the ransomware note. If the reply is smaller, then it is likely a sinkhole. Otherwise, a valid reply indicates that the domain is used by the cybercriminals.

At the time of writing this paper, using this approach *FortiGuard Lion Team* has identified many sinkholes created by security researchers. However, no actual malicious domain has been observed.

A C source code that generates random domains through Locky's DGA is available at the Appendix of this paper.

Harvesting Locky configuration files

The *FortiGuard Lion Team* has created a system that harvests Locky configuration files. The system leverages the *Cuckoo Sandbox* and is composed of three main parts: a sample collector, the *Cuckoo Sandbox*, and a database:

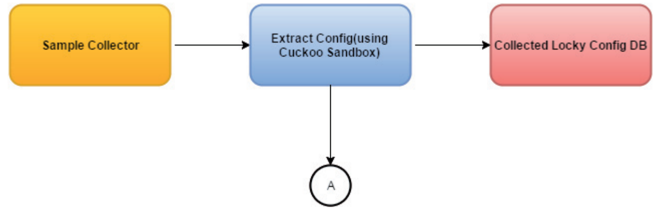

Figure 45: Overview of Locky monitoring system components.

Initially, *Cuckoo*'s 'procmemdump' flag is configured to 'yes' to enable process memory dumping. ProcMemory – a default processing module in *Cuckoo* – is then utilized to confirm Locky's presence using a Yara rule.

The same module is responsible for mapping memory dump. If Locky is confirmed to be present, the mapped memory dump will be parsed to extract Locky's configuration file.

A flowchart of this process is shown in Figure 46.

Finally, the extracted configuration file is stored in the database and extracted IPs and URIs are updated to *Fortinet* solutions.

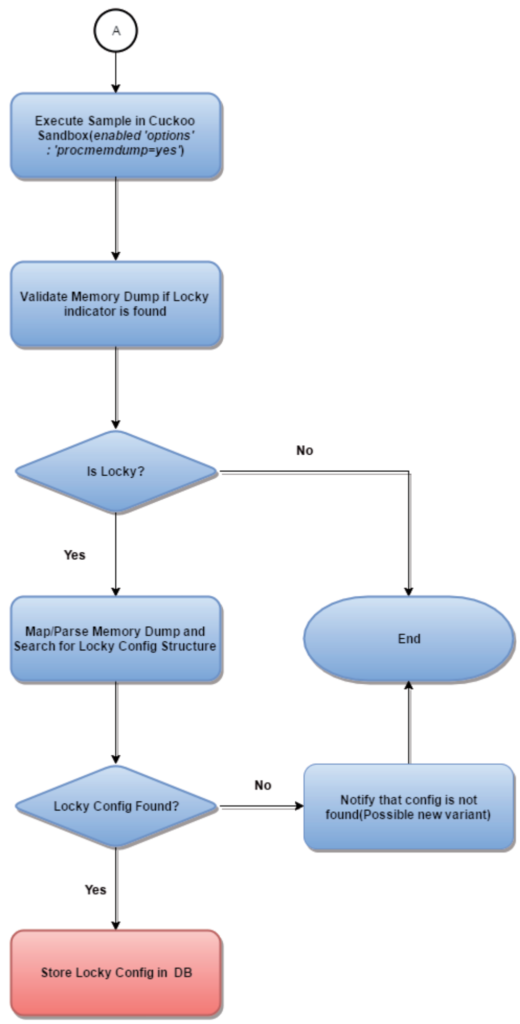

Figure 46: Flowchart for extracting Locky configuration file via Cuckoo Sandbox.

7. CONCLUSION

Today, ransomware is a major threat that affects many users and organizations worldwide. The anti-virus industry is seeing a shift in trade for many cybercriminals, both experienced and inexperienced, from other cybercrime *modus operandi* to the ransomware business. Locky ransomware is a by-product of this shift.

This research allowed the *FortiGuard Lion Team* to understand how, with the right experience and resources, cybercriminals are able to quickly dominate a specific cybercrime area, in this case, ransomware. The anti-virus industry must respond by closely monitoring these developments in order to minimize damage to users. Information sharing across the industry is essential to maximize the impact of such efforts.

In this paper, Locky's prevalence, technical analysis, developments as well as intelligence gathering approaches were detailed. The *FortiGuard Lion Team* hopes that the information shared here will contribute to the industry's collective effort in fighting the Locky ransomware.

REFERENCES

[1] Dela Paz, R. CryptoWall, TeslaCrypt and Locky: A Statiscial Perspective. Fortinet Blog. https://blog.fortinet.com/2016/03/08/cryptowall-teslacrypt-and-locky-a-statistical-perspective.

[2] Bacurio, F.; Joven, R.; Dela Paz, R. A Closer Look at Locky Ransomware. Fortinet Blog. https://blog.fortinet.com/2016/02/17/a-closer-look-at-locky-ransomware-2.

[3] Bacurio, F. U. Diligence is the Mother of Good Locky Detection. Fortinet Blog. https://blog.fortinet.com/2016/06/01/diligence-is-the-mother-of-good-locky-detection.

APPENDIX

IOCs

Added files:

%User Temp%\svchost.exe
%Desktop%_HELP_instructions.txt
%Desktop%_HELP_instructions.bmp
%Desktop%_HELP_instructions.html
{folders containing encrypted files}_HELP_instructions.txt

Added registry keys:

key:HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run value: opt321
data:"%User Temp%\svchost.exe" or {original filepath}

key:HKEY_CURRENT_USER\Software\{random characters}
value:{random characters 1}
data: {Hex values}
value:{random characters 2}
data: {Hex values}
value:{random characters 3}
data: {Hex values}
value:{random characters 4}
data: {Hex values}

key: HKCU\Control Panel\Desktop
value: Wallpaper
data: %Desktop%_HELP_instructions.bmp
Cmd command:
vssadmin.exe Delete Shadows /All /Quiet

Hashes:

A list of Locky SHA-256 hashes is available here: https://github.com/fortiguard-lion/LockyIOCs/blob/master/Locky_SHA256_hashes.txt

C&Cs:

A list of collected Locky C&Cs is available here: https://github.com/fortiguard-lion/LockyIOCs/blob/master/Locky_C2_IPs.txt

DGA tool in C source code

```
#include "stdafx.h"
#include <Windows.h>

char *tlds[] = {"ru", "info", "biz", "click", "su",
"work", "pl", "org", "pw", "xyz"};

void LockyDGA(char *domain, int pos, int seed,
SYSTEMTIME systemTime)
{
    int v1;
    int v2;
    int v3;
    int v4;
    int v8;
    int v9;
    int v10;
    int v11;
    int v12;
    int v13;
    int v14;
    int v15;
    int v17;
    int v18;
    int v19;
    int v20;
    char *v21;
    int v7;
    unsigned int v5;
    int v6;

    int var18;
    int var14;
    int var10;

    v1 = pos;
    v2 = seed;
    v3 = 0;
    v5 = systemTime.wDay >> 1;
    v4 = systemTime.wYear;
    v1 = _rotl(v1, 0x15);
    v6 = _rotl(v2, 0x11);
    var18 = v6 + v1;
    var14 = v5;
    var10 = 7;

    while (var10 > 0)
```

```
    {
      v7 = _rotr(0xB11924E1 * (v4 + v3 + 0x1BF5), 7);
      v8 = (v7 + 0x27100001) ^ v3;
      v9 = _rotr(0xB11924E1 * (v8 + v2), 7);
      v10 = (v9 + 0x27100001) ^ v8;
      v11 = _rotr(0xB11924E1 * (v5 + v10), 7);
      v12 = 0xD8EFFFFF - v11 + v10;
      v13 = _rotr(0xB11924E1 * (systemTime.wMonth + v12
- 0x65CAD), 7); v14 = v12 + v13 + 0x27100001;
      v15 = _rotr(0xB11924E1 * (v14 + var18), 7);
      v3 = (v15 + 0x27100001) ^ v14;
      ++v4;
      var10 = var10 - 1;
      v5 = var14;
    }
    var18 = v3 % 0xBu + 7;
    var10 = 0;
    if (var18 != 0)
    {
      do
      {
        v17 = _rotl(v3, var10);
        v18 = _rotr(0xB11924E1 * v17, 7);
        v3 = v18 + 0x27100001;
        domain[var10++] = v3 % 0x19u + 'a';
      } while (var10 < var18);
    }
    domain[var10++] = '.';
    v19 = _rotr(0xB11924E1 * v3, 7);
    v20 = 0;
    v21 = tlds[(v19 + 0x27100001) % (sizeof(tlds) /
    sizeof(tlds[0]))];
    do
    {
      if (!v21[v20])
      {
        break;
      }
      domain[var10++] = v21[v20++];
    } while (v20 < 5);
}
void showHelpInfo(char *s)
{
    printf("Usage : %s [-option] [argument]\n", s);
    printf("option: -h Show help information\n");
    printf(" -s Seed from Locky Config\n");
    printf(" -d Date with format [yyyy-mm-dd]\n");
    printf(" -n Max count of Domain generated\n");
    printf("Default: -d {current date} -n {7}");
}
int main(int argc, char* argv[])
{
    char domain[40];
    int pos = 0;
    SYSTEMTIME systemTime; int max = 7;
    int seed = 0;
    GetSystemTime(&systemTime);
    if (argc > 1)
    {
      for (int i = 1; i < argc; i++)
      {
        if (i + 1 > argc)
        {
          break;
        }
        if (strcmp(argv[i], "-h") == 0)
        {
          showHelpInfo(argv[0]);
          return 0;
        }
        if (strcmp(argv[i], "-d") == 0)
        {
          char *date = argv[i + 1];
          char buf[5];
          strncpy_s(buf, 5, date, 4);
          if (atoi(buf) != 0) {
            systemTime.wYear = atoi(buf); }
          memset(buf, 0, sizeof(buf));
          strncpy_s(buf, 5, date + 5, 2);
          if (atoi(buf) != 0)
          {
            systemTime.wMonth = atoi(buf); }
          memset(buf, 0, sizeof(buf));
          strncpy_s(buf, 5, date + 8, 2);
          if (atoi(buf) != 0)
          {
            systemTime.wDay = atoi(buf);
          }
        }
        if (strcmp(argv[i], "-n") == 0)
        {
          if (atoi(argv[i + 1]) != 0)
          {
            max = atoi(argv[i + 1]); }
        }
        if (strcmp(argv[i], "-s") == 0)
        {
          if (atoi(argv[i + 1]) != 0)
          {
            seed = atoi(argv[i + 1]);
          }
        }
      }
    }
    do
    {
      memset(domain, 0, sizeof(domain));
      LockyDGA(domain, pos, seed, systemTime);
      printf("DGA %d = %s\n", pos++, domain);
    } while (pos < max);
return 0;
}
```