



## LockCrypt ransomware: weakness in code can lead to recovery

April 4, 2018 by [Malwarebytes Labs](#)

At the start of the year, it seemed that 2018 was going to be all about cryptominers. They so overwhelmingly dominated the landscape that it looked like no other threat had a chance. However, ransomware is not giving up the field so fast. There have been new variants popping up every couple of months, peering rather shyly around the corner.

At the moment, the most popular ransomware is GandCrab. However, a lesser-known family called LockCrypt has been creeping around under the radar since [June 2017](#). Since it is spread via RDP brute-force attacks that must be manually installed, it has never been a massive threat—and therefore had never been described in detail.

But recently we were contacted by some victims of LockCrypt, so we decided to take a closer look. Our investigation led to some interesting findings, especially when we discovered that the ransomware authors decided to ignore popular advice not to roll your own crypto. As we could easily guess, it introduced weaknesses to the code, along with the possibility to recover the data in some cases.

## Analyzed sample

[99a3d049f11474fac6844447ac2da430](https://www.virustotal.com/file/99a3d049f11474fac6844447ac2da430/analysis/)

## Behavioral analysis

In order to execute properly, the malware must be run as an Administrator. Due to the fact that it is deployed manually by attackers, it doesn't use any tricks or exploits to automatically elevate its privileges.

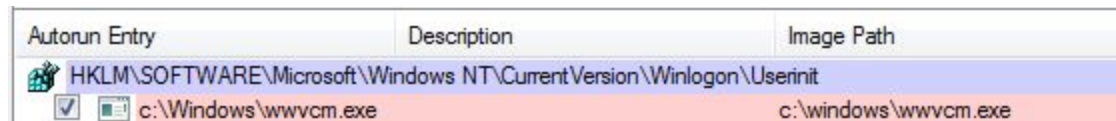
Once it is run, it deletes the original sample and drops itself in C:\Windows under the name `wwvcm.exe`:



The screenshot shows a Windows Explorer window with the address bar set to 'Local Disk (C:) > Windows >'. The main area displays a table of files and folders. A folder named 'New folder' is visible at the top. Below it, a table lists files with columns for Name, Date modified, Type, and Size. One file, 'wwvcm.exe', is highlighted. It was modified on 2018-02-05 at 23:32, is an Application, and is 13 KB in size.

Name	Date modified	Type	Size
wwvcm.exe	2018-02-05 23:32	Application	13 KB

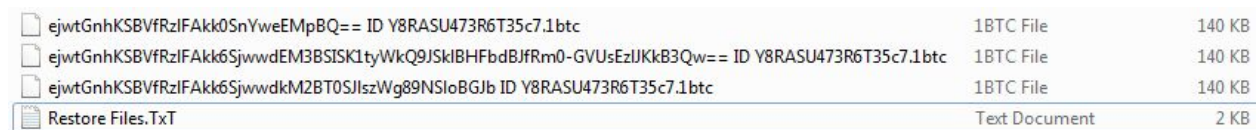
It also adds persistence using a registry key:



The screenshot shows a registry key configuration table. The columns are 'Autorun Entry', 'Description', and 'Image Path'. The entry is 'HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit', which is checked. The image path is 'c:\windows\wwvcm.exe'.

Autorun Entry	Description	Image Path
<input checked="" type="checkbox"/> HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit		c:\windows\wwvcm.exe

This ransomware encrypts all the files it can possibly reach. During the process, it enumerates and tries to terminate all running applications so that they will not be blocking access to the attacked files. Executables are also attacked.



The screenshot shows a file explorer window displaying a list of files. Three files are listed with obfuscated names and a '.1btc' extension, each 140 KB in size. A fourth file, 'Restore Files.Txt', is a 2 KB text document.

ejwtGnhKSBVfRzIFakk0SnYweEMpBQ== ID Y8RASU473R6T35c7.1btc	1BTC File	140 KB
ejwtGnhKSBVfRzIFakk6SjwwdEM3BSISK1tyWkQ9JSklBHFbdBJfRm0-GVUsEzIJKk83Qw== ID Y8RASU473R6T35c7.1btc	1BTC File	140 KB
ejwtGnhKSBVfRzIFakk6SjwwdkM2BT0SJlszWg89NSIoBGJb ID Y8RASU473R6T35c7.1btc	1BTC File	140 KB
Restore Files.Txt	Text Document	2 KB

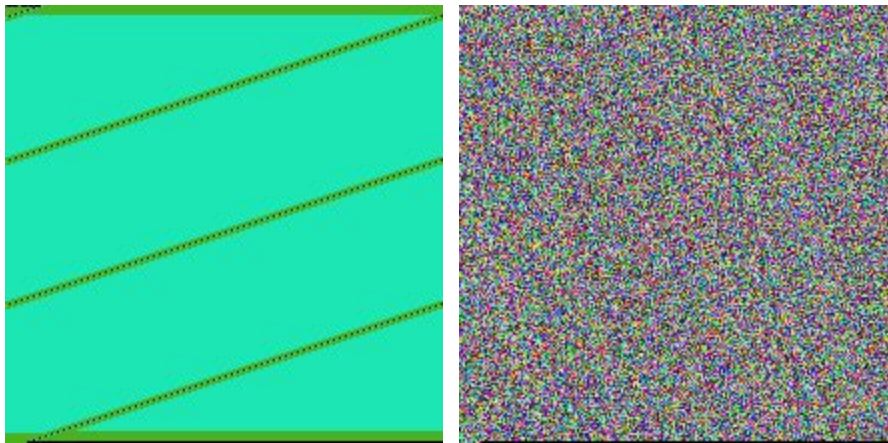
The names of the encrypted files are obfuscated—first encrypted and then converted to base64. The random ID is also a part of the name. The extension used is '1btc'.

The ransom note is dropped as a TXT file:

```
Restore Files.TXT - Notepad
File Edit Format View Help
Your ID Y8RASU473R6T35c7
All your files have been encrypted due to a security problem with your PC. If you want to restore them, write us to the e-mail support: dyamol@aol.com or dyamol@bitmessage.ch
write this ID in the title of your message
In case of no answer in 24 hours write us to these e-mails support: dyamol@aol.com or dyamol@bitmessage.ch
You have to pay for decryption in Bitcoins. The price dependson how fast you write to us. After payment we will send you the decryption tool that will decrypt all your files.
Free decryption as guarantee
Before paying you can send us up to 3 files for freedecryption. The total size of files must be less than 10mb(nonarchived), and files should not contain valuable information.
(databases,backups, large excel sheets, etc.)
How to obtain Bitcoins
The easiest way to buy bitcoins is LocalBitcoins site.Youhave to register, click 'buy bitcoins', and select the sellerbypayment method and price.
https://localbitcoins.com/buy_bitcoins
Also you can find other places to buy Bitcoins andbeginnersguide here:
http://www.coindesk.com/information/how-can-i-buy-bitcoins/
Attention!
Do not rename encrypted files.
Do not try to decrypt your data using third party software,it may cause permanent data loss.
decryption of your files with the help of third parties maycause increased price (they add their fee to our) or you can becomea victim of a scam.
```

Which pops up at the end of the execution.

Looking inside the encrypted files, we saw that they have pretty high entropy. The example below shows a BMP file before and after encryption:



Our initial assessment of the image was that the authors didn't use a trivial XOR here. It may also look like a file [encrypted by stream ciphers \(or any ciphers in CBC mode\)](#). After looking inside the code, we will know more about it.

Looking at the changes made in the registry, we found more data left there by the ransomware, such as the unique ID of the victim:

Hacked	REG_SZ	I36z7T1ILIECNXPQ
LegalNoticeCaption	REG_SZ	Attention!!! Your files are encrypted !!!
LegalNoticeText	REG_SZ	To recover files, follow the prompts in the text file "Restore Files"
PasswordExpiryWarning	REG_DWORD	0x00000005 (5)
PowerdownAfterShutdown	REG_SZ	0
PreCreateKnownFolders	REG_SZ	{A520A1A4-1780-4FF6-BD18-167343C5AF16}
ReportBootOk	REG_SZ	1
scremoveoption	REG_SZ	0
Shell	REG_SZ	explorer.exe
ShutdownFlags	REG_DWORD	0x80000027 (2147483687)
ShutdownWithoutLogon	REG_SZ	0
Userinit	REG_SZ	C:\Windows\system32\userinit.exe,c:\Windows\wwwcm.exe,
VMApplet	REG_SZ	SystemPropertiesPerformance.exe /pagefile
WinStationsDisabled	REG_SZ	0

puter\HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon

## Network communication

The malware is capable of encrypting without an Internet connection. However, if we run it on a connected machine, it beacons to its CnC. The CnC IP is [46.32.17.222 \(located in Iran\)](#).

Here's a fragment of the communication:

```
WThSQVNVNDczUjZUMzVjNycsJ1dpbmRvd3MgNyBQcm9mZXNzaW9uYWx8dGVzdGVyFEM6XFVzZXJzXHRlc3Rlc1xEZXRrdG9wXGxvY2tjcnlwdC5leGU=

/a0.Q4?XBPX...".*,Bc0>+...].+BQ$.XY.F+<.$
.F.0bE.G^...-A.=0.6/.ED&..K>- bQ.0Q_\.+A...37ZBQ.3E).Z..B./
09.<. ;?.3A&Mb..D0c6..b._?..V#.%./I.Uc:."C...ZaAN.A,..?,NXX.Q.
...H!
\9' /`Kc*...<W?.2M?.a=E.,E.?B.KP...+... [.-/W*..8bG<Y&N:.&...IW.O.$!G#SR[&BJ.#0.(9.$.^XF..6LP.
G.H@&....L.?..( .ID=4X._@P.?R.`..c `K>E9..6.a$^..Q.IcX,9R.[
]R!TS.\L$...'.6/:=. 2AZ(7,\8J28....c`J.^U."@,B@.F5.7.".\4:)...
!..8[.71E...'.X.<7.^DH5...aB.<5.CIB..Q.>DT5.W..F9..FA5A..SK8.4.3\$M,@#NQ;..MK.B?+3.JK.:
(2LYA<D0%. A
```

The bot sends base64 encoded data about the attacked machine, such as the random ID, username, operating system, and the path from where the malware was deployed. Example:

```
WThSQVNVNDczUjZUMzVjNycsJ1dpbmRvd3MgNyBQcm9mZXNzaW9uYWx8dGVzdGVyFEM6XFVzZXJzXHRlc3Rlc1xEZXRrdG9wXGxvY2tjcnlwdC5leGU=
```

Decodes to:

```
Y8RASU473R6T35c7',Windows 7 Professional|tester|C:\Users\tester\Desktop\lockcrypt.exe
```

The server sends back a block of bytes, which looks like some random or encrypted data. Its exact role we will find out by looking into the code.

## Inside the code

The sample is not packed by any external crypter, nor is it obfuscated. Once we open it, we can directly see all that it has inside.

```
void __noreturn start()
{
    DWORD ThreadId; // [sp+0h] [bp-4h]@1

    SetLastError(1u);
    WSASStartup(0x101u, &stru_405048);
    check_wvcm();
    adjust_token();
    CreateThread(0, 0, searching_processes, 0, 0, &ThreadId);
    registry_read();
    if ( should_encrypt != 1 )
    {
        socket_communicate();
        encrypt_drives();
    }
    while ( dword_404004 )
        Sleep(1000u);
    encryption_done = 1;
    popup_note();
    ExitProcess(0);
}
```

At the beginning, the ransomware checks the folder from which it is running. It tries to make a copy in the Windows folder and redeploys itself from that location.

Then, it creates a thread that continuously enumerates all the running processes and tries to terminate them.

It reads the registry to check if it was already deployed. Finding the appropriate keys can stop the infection—the malware will recognize the machine as already attacked. Otherwise, it will proceed further.

## Encryption

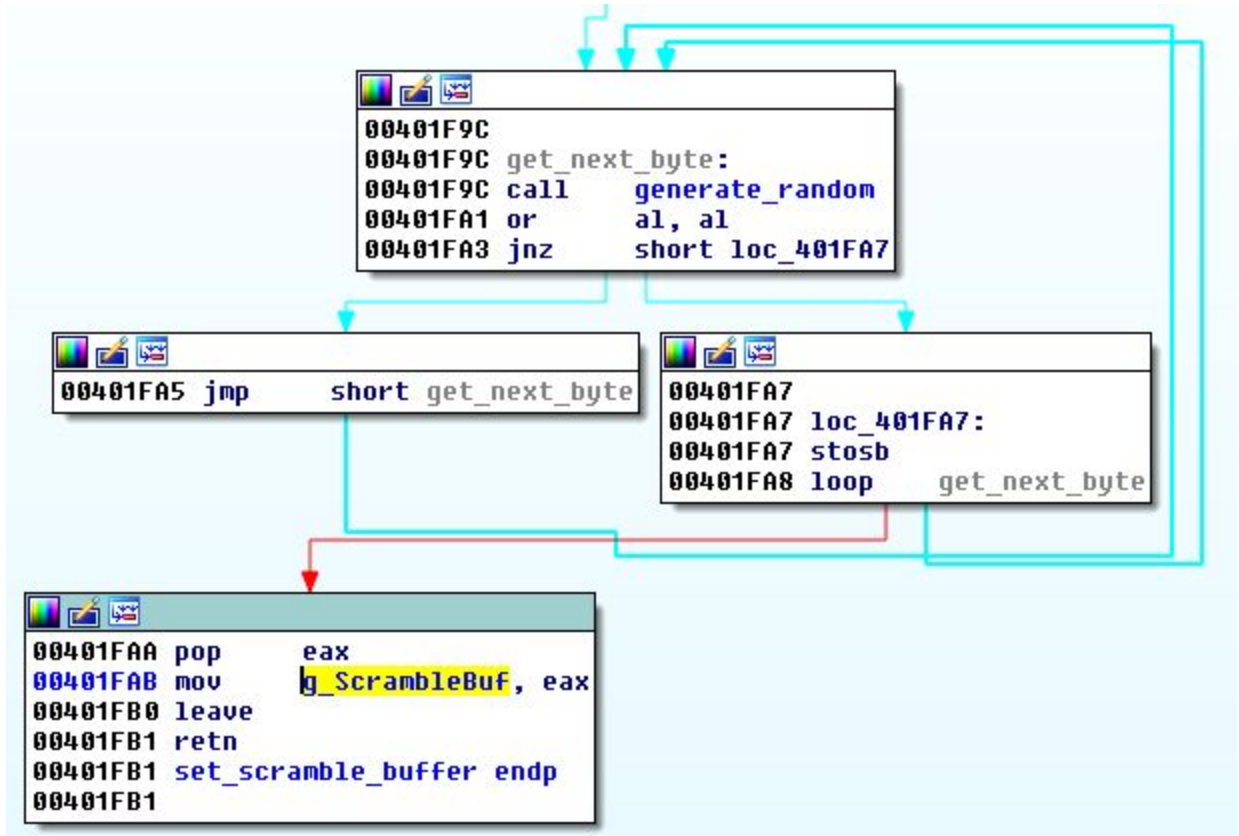
The infection starts from the attempt to communicate with the CnC.

Looking inside this function, we could now understand the role of the mysterious buffer of bytes seen during the behavioral analysis. The downloaded buffer is validated by its CRC32 checksum. Then, it sets in a global variable for the further use of the encryption routine.

```
    v4 = 25004;
    while ( 1 )
    {
        v5 = recv(s, v3, v4, 0);
        v10 += v5;
        if ( !v5 || v10 >= 0x61AC )
            break;
        v4 = 25004 - v10;
        v3 = &buf[v10];
    }
    if ( v10 < 0x61AC || (checksum = crc32(buf + 4, 25000), downl_buf = (_DWORD*)(v8 - 4), *downl_buf != checksum) )
    {
        LABEL_18:
        closesocket(s);
        WSACleanup();
        result = set_scramble_buffer((int)buf);
    }
    else
    {
        g_ScrambleBuf = (int)(downl_buf + 1);
        closesocket(s);
        result = WSACleanup();
    }
    return result;
}
```

It turns out that this buffer is like a pad used for the encryption schema. The authors probably wanted to achieve something like a [one-time-pad encryption](#). However, they reused the buffer, and because of this, they made their algorithm vulnerable for a plain text attack.

If for some reason downloading the buffer from the Internet is not possible, it is generated by a simple, pseudo-random algorithm:



The authors did not make the best choice for the random generator. Rather than using a cryptographically strong one, they went for the GetTickCount function.

Looking inside the encryption routine, we can see that the file is scrambled by a pretty simple function:

```

004017A3 .: $ PUSH EBP
004017A4 .: MOV EBP,ESP
004017A6 .: MOV ECX,[ARG_2]
004017A9 .: SHR ECX,0x2
004017AC .: SHL ECX,1
004017AE .: MOV EBX,DWORD PTR DS:[0x405044] scrambling_buffer
004017B4 .: MOV EDX,EBX
004017B6 .: ADD EBX,0x61A8
004017BC .: MOV ESI,[ARG_1]
004017BF .: MOV EDI,ESI
004017C1 .: CLD
004017C2 .: > LODS DWORD PTR DS:[ESI]
004017C3 .: XOR EAX,DWORD PTR DS:[EDX]
004017C5 .: STOS DWORD PTR ES:[EDI]
004017C6 .: DEC ESI
004017C7 .: DEC EDI
004017C8 .: DEC ESI
004017C9 .: DEC EDI
004017CA .: ADD EDX,0x4
004017CD .: CMP EDX,EBX
004017CF .: JNZ SHORT lockcryp.004017D7
004017D1 .: MOV EDX,DWORD PTR DS:[0x405044] scrambling_buffer
004017D7 .: > Y LOOPD SHORT lockcryp.004017C2
004017D9 .: MOV ECX,[ARG_2]
004017DC .: SHR ECX,0x2
004017DF .: MOV EBX,DWORD PTR DS:[0x405044] scrambling_buffer
004017E5 .: MOV EDX,EBX
004017E7 .: ADD EBX,0x61A8
004017ED .: MOV ESI,[ARG_1]
004017F0 .: MOV EDI,ESI
004017F2 .: CLD
004017F3 .: > LODS DWORD PTR DS:[ESI]
004017F4 .: ROL EAX,0x5
004017F7 .: XOR EAX,DWORD PTR DS:[EDX]
004017F9 .: BSWAP EAX
004017FB .: STOS DWORD PTR ES:[EDI]
004017FC .: ADD EDX,0x4
004017FF .: CMP EDX,EBX
00401801 .: JNZ SHORT lockcryp.00401809
00401803 .: MOV EDX,DWORD PTR DS:[0x405044]
00401809 .: > Y LOOPD SHORT lockcryp.004017F3
0040180B .: LEAVE
0040180C .: RETN 0x8

```

Address	Hex dump	ASCII
002C3F80	38 06 02 34 55 21 51 0A 09 18 43 51 54 13 2B 09	8*04U*0..+CQT!!+.
002C3F90	2F 1F 47 24 03 10 01 3B 0C 59 07 4D 05 49 26 61	/7G\$*0;.Y.M*I&a
002C3FA0	52 4D 33 37 5D 58 55 60 2B 01 5F 5A 0F 1F 09 57	RM37DXU~+0_2
002C3FB0	27 47 19 1C 03 51 60 10 36 23 40 15 11 5E 4C 25	'G+L*Q~>6#0S4^L?
002C3FC0	0D 13 59 05 5B 35 62 45 36 63 13 05 2E 09 14 08	..!!Y*[5bE6c!!#.9
002C3FD0	5A 30 1C 5A 34 0D 1A 58 2A 1D 2C 43 2E 2B 1B 59	Z0LZ4.+X**,.C.+Y
002C3FE0	14 13 20 43 1C 10 2A 0D 0E 57 2B 57 4B 52 62 19	9!! CL>*.0U+UkRb+
002C3FF0	49 2A 05 19 3A 45 41 39 52 06 60 5E 0C 06 0E 19	I**+:EA9R*~^*#+
002C4000	0D 22 46 08 37 52 63 56 48 5C 61 10 5F 48 55 23	."7RcUH\>_KU#
002C4010	4F 31 20 36 17 08 4C 47 55 46 2B 30 45 23 55 4F	01 6#0LGUF+0E#U0

The scrambling algorithm has two different rounds. The reconstructed code of both rounds can be seen below.



## Round 1

```
void lockcrypt::level1_crypt(void *buf, size_t buf_size, void *key, size_t
key_size)
{
    size_t dwsz = buf_size >> 2;

    for (size_t i = 4, k = 0; i < buf_size - 6; i += 2, k +=4) {
        if (k > KEY_SIZE) {
            k = 0;
        }
        DWORD *dwlist = (DWORD *) ((ULONGLONG)buf + i);
        DWORD *keydw = (DWORD *) ((ULONGLONG)key + k);

        DWORD inp = *dwlist;
        DWORD out = inp ^ (*keydw);
        (*dwlist) = out;
    }
}
```

[view raw lockcrypt\\_round1.cpp](#) hosted with ❤ by [GitHub](#)

This round uses only XOR operation, but there is a twist that prevents you from recovering the original key. Although the DWORD from the input is XORed with a DWORD from the key, the input is also tainted with the previous output. On every step, the first half of the input DWORD is taken from the previous output, while only the second half is fresh. That makes it a simple stream cipher.

## Round 2

```
void lockcrypt::level2_crypt(void *buf, size_t buf_size, void *key, size_t
key_size)
{
    size_t dwsz = buf_size >> 2;

    for (size_t i = 4, k = 0; i < buf_size - 6; i += 4, k +=4) {
        if (k > KEY_SIZE) {
            k = 0;
        }
        DWORD *dwlist = (DWORD *) ((ULONGLONG)buf + i);
        DWORD *keydw = (DWORD *) ((ULONGLONG)key + k);

        DWORD inp = *dwlist;
        inp = rol32(inp, 5);
        DWORD out = inp ^ (*keydw);
        out = bswap32(out);
        (*dwlist) = out;
    }
}
```

[view raw lockcrypt\\_round2.cpp](#) hosted with ❤ by [GitHub](#)

This round looks more complicated—Not only is XOR operation used here, but also ROL and bitwise swap. However, there is no input tainting this time, so it is easily reversible.

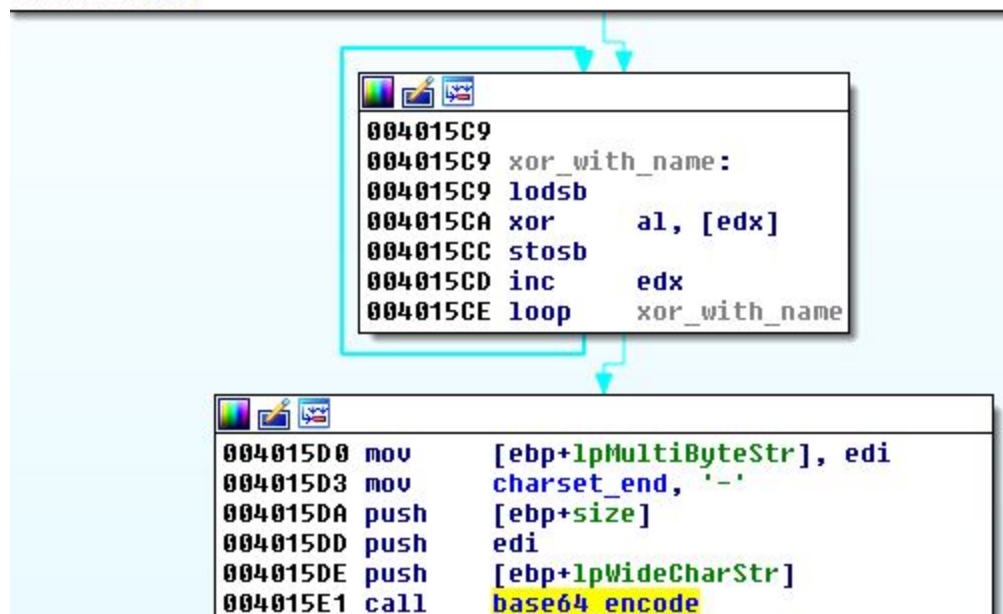
Those two simple rounds, together with the “pad” buffer that is 2,500 bytes long, were able to generate the output with pretty high entropy.

## File names obfuscation

The names of the files are first XORed with the pad buffer, and then base64 encoded. The offset of the XOR key is 1111 characters from the beginning of the buffer.

The part of code responsible for this:

```
004015B1 mov     [ebp+lpWideCharStr], eax
004015B4 mov     edi, eax
004015B6 mov     ecx, [ebp+size]
004015B9 mov     esi, [ebp+lpString]
004015BC mov     edx, g_ScrambleBuf
004015C2 add     edx, 1111
004015C8 cld
```



## Conclusion

LockCrypt is an example of yet another simple ransomware created and used by unsophisticated attackers. Its authors ignored well-known guidelines about the proper use of cryptography. The internal structure of the application is also unprofessional.

Sloppy, unprofessional code is pretty commonplace when ransomware is created for manual distribution. Authors don't take much time preparing the attack or the payload. Instead, they're rather focused on a fast and easy gain, rather than on creating something for the long run. Because of this, they could easily be defeated.

