SOPHOS
Cybersecurity made simple.

BTCWare Ransomware

By Anand Ajjan and Dorka Palotay, SophosLabs

Last years' news headlines were dominated by ransomware attacks like Wannacry and Petya, the constant presence of Cerber, the disappearance and return of Locky, and the growing popularity of Ransomware-as-a-Service (RaaS) and smaller campaigns like Jaff and BadRabbit.

Introduction

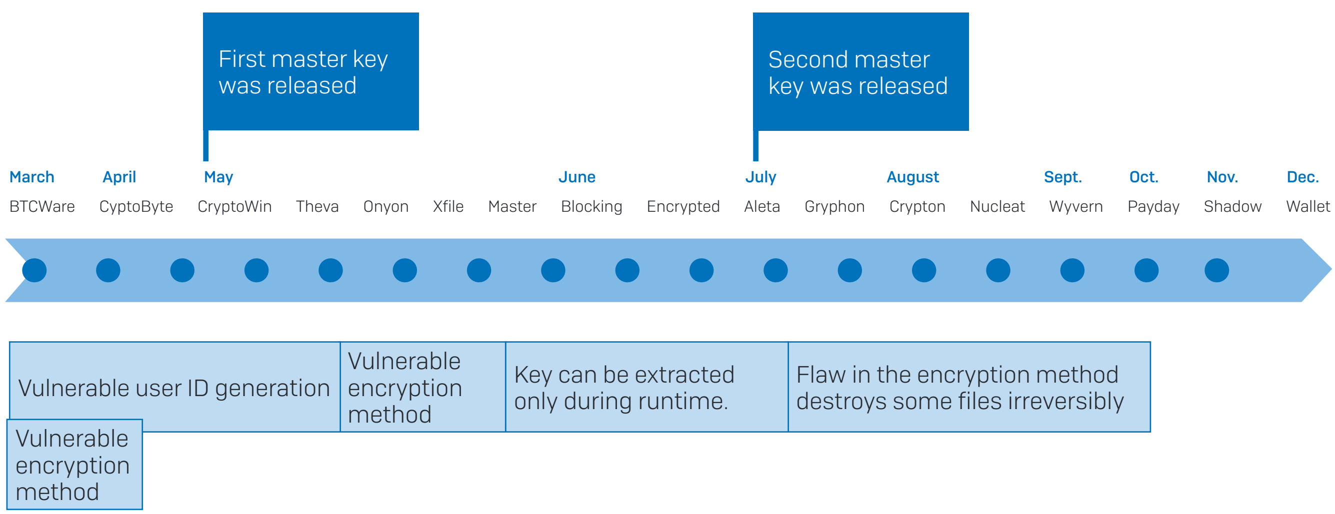
But there are many ransomware families that didn't get much attention, even though they were also active. One such example is BTCWare, which was among the 10 most commonly intercepted ransomware families of the year.

We've collected and analyzed 17 different variants of this family, from its first appearance in March 2017 until today. While looking into these variants, we can follow the development process of the BTCWare ransomware family and see how the developers have made mistakes, learned from them, and tried to improve their code over time.

The authors of this family experimented with different key generation and encryption methods. Looking through these solutions, we will introduce a few implementation failures and the route which led from a completely vulnerable method to a cryptographically secure file encryption. Although the analysis of the file encryption method will be a focus of this paper, we will cover some other features of the BTCWare family and highlight the similarities and differences of different variants.

We refer to the variants by the new extension they add to the encrypted file names, but it is important to note that there might be differences within variants as well. It is possible that, for example, the change in the encryption method didn't happen at the same time as the change in the extension.

On the timeline below, we have summarized the appearance of the different variants by month:



Infection vector

The BTCWare ransomware family targets Windows machines and is primarily distributed by brute-forcing weak passwords of the Remote Desktop Protocol (RDP) and manually installing the malware.

A few variants of BTCWare were distributed by spam emails as well. The so-called "Blank Slate" malspam campaign is responsible for spreading different ransomware families targeting Windows (e.g. Cerber, Locky, GlobeImposter) [1].

In a few cases, BTCWare variants were distributed via this campaign. [2 3] We observed a big spike in August that spread the Gryphon variant. Previously, in July, Aleta was distributed using this method.

These spam emails arrive with no subject and no content. The attachment is either a Zip file containing another Zip archive with a JavaScript file inside or a Microsoft Word document. Once JavaScript or a malicious macro is executed, the BTCWare payload is downloaded.

Execution

Apart from continuous updates to the file encryption method, BTCWare performs additional changes to the system such as:

- Adding an auto run entry in the registry under Run key value,
- Deleting volume shadow copies using a vssadmin tool,
- Disabling recovery settings using a bcdedit tool.

Below is the summary of various system changes done by different BTCWare variants:

Variant	Ransom note name	Added to HKCU\Software\Microsoft \Windows\CurrentVersion\Run registry key	Mutex name
BTCWare	#_HOW_TO_FIX.inf	Value Name: btq Value Data: %Appdata%\mfskSkfkls.exe (this is a copy of the original executable) Deleted after encryption is done.	BTCWXXX
Cryptobyte	#_HOW_TO_FIX.inf	Value Name: btq Value Data: %Appdata%\mfskSkfkls.exe (this is a copy of the original executable) Deleted after encryption is done. Value Name: insbtq Value Data: %Appdata%\#_HOW_TO_FIX.inf	BTCWXXX
Cryptowin	#_HOW_TO_FIX.inf	Value Name: btq Value Data: %Appdata%\mfskSkfkls.exe (this is a copy of the original executable) Deleted after encryption is done. Value Name: insbtq Value Data: %Appdata%\#_HOW_TO_FIX.inf	BTCWXXX
Theva	#_README_#.inf	Value Name: xcz Value Data: %Appdata%\zmsksddfff.exe (this is a copy of the original executable) Deleted after encryption is done. Value Name: info Value Data: %Appdata%\#_README_#.inf	CRYNXXX
Onyon	!#_DECRYPT_#!.inf	Value Name: DECRYPTINFO Value Data: %Appdata%\ !#_DECRYPT_#!.inf	ONYONLOCK
Xfile	!#_DECRYPT_#!.inf	Value Name: DECRYPTINFO Value Data: %Appdata%\ !#_DECRYPT_#!.inf	ONYONLOCK
Master	!#_RESTORE_FILES_#!.inf	Value Name: DECRYPTINFO Value Data: %Appdata%\ !#_RESTORE_FILES_#!.inf	ONYONLOCK or MASTERLOCK

Encrypted	!#_RESTORE_FILES_#!.inf	Value Name: DECRYPTINFO Value Data: %Appdata%\ !#_RESTORE_FILES_#!.inf	MASTERLOCK
Blocking	!#_RESTORE_FILES_#!.inf	Value Name: DECRYPTINFO Value Data: %Appdata%\ !#_RESTORE_FILES_#!.inf	MASTERLOCK
Aleta	!#_READ_ME_#!.inf !#_READ_ME_#!.hta %Appdata%\Info.hta	Value Name: DECRYPTINFO Value Data: %Appdata%\!#_READ_ME_#!.inf Value Name: DECRYPTINFO Value Data: %Appdata%\ Info.hta	MASTERLOCK
Gryphon	!## DECRYPT FILES##!.txt %Appdata%\HELP.txt	Value Name: DECRYPTINFO Value Data: %Appdata%\HELP.txt	no mutex or GIVEMEBTC
Crypton	!## DECRYPT FILES##!.txt %Appdata%\HELP.txt	Value Name: DECRYPTINFO Value Data: %Appdata%\HELP.txt	no mutex or GIVEMEBTC
Nuclear	HELP.hta	Value Name: DECRYPTINFO Value Data: %Appdata%\HELP.hta Value Name: HELPINFO Value Data: %Appdata%\HELP.hta	NUCLEAR
Wyvern	!FILES ENCRYPTED.txt %Appdata%\HELP.hta	Value Name: DECRYPTINFO Value Data: %Appdata%\HELP.hta	NUCLEAR
Payday	!! RETURN FILES !!.txt %Appdata%\payday.hta	Value Name: payday Value Data: %Appdata%\payday.hta Value Name: baby Value Data: %Appdata%\payday.hta	PAYDAYDAYPAY
Shadow	!! RETURN FILES !!.txt %Appdata%\payday.hta	Value Name: 1payday Value Data: %Appdata%\payday.hta Value Name: 2baby Value Data: %Appdata%\payday.hta	PAYDAYDAYPAY
Wallet	! How Decrypt Files.txt %Appdata%\payday.hta	Value Name: 1payday Value Data: %Appdata%\payday.hta Value Name: 2baby Value Data: %Appdata%\payday.hta	PAYDAYDAYPAY

Table 1

When BTCWare is executed, it starts by checking a specific mutex. If the mutex exists, the process ends. Otherwise it creates the mutex. The different mutex names can be found in Table 1.

In the case of the first few variants, it creates a copy of itself in the %Appdata% folder and adds this to the HKCU\Software\Microsoft\Windows\CurrentVersion\Run registry key to ensure it runs after reboot. After the file encryption is completed, the created registry values are deleted.

All the variants save the ransom note in the %Appdata% folder and add this to the HKCU\Software\Microsoft\Windows\CurrentVersion\Run registry key. This way, every time the infected computer is rebooted, the ransom note will be opened. From the last two variants (Shadow and Wallet), this information is added to the HKLM\Software\Microsoft\Windows\CurrentVersion\Run registry key as described in the above Table 1.

To reduce the chance of recovering the encrypted files, BTCWare attempts to delete volume shadow copies using the vssadmin.exe tool with the following command:

vssadmin.exe Delete Shadows /All /Quiet

Using bcdedit.exe, BTCWare disables recovery and boot failures using the two commands below:

bcdedit.exe /set {default} bootstatuspolicy ignoreallfailures
bcdedit.exe /set {default} recoveryenabled No

```
0000000000403AC3
0000000000403AC3                          loc_403AC3:
0000000000403AC3 8B 35 DC 81 42 00        mov     esi, ds:ShellExecuteW
0000000000403AC9 6A 00                    push    0               ; nShowCmd
0000000000403ACB 6A 00                    push    0               ; lpDirectory
0000000000403ACD 68 48 52 43 00           push    offset Parameters ; "/c vssadmin.exe Delete Shadows /All /Qu"...
0000000000403AD2 68 A0 52 43 00           push    offset File     ; "cmd.exe"
0000000000403AD7 68 B0 52 43 00           push    offset Operation ; "open"
0000000000403ADC 6A 00                    push    0               ; hwnd
0000000000403ADE FF D6                    call    esi ; ShellExecuteW
0000000000403AE0 6A 00                    push    0               ; nShowCmd
0000000000403AE2 6A 00                    push    0               ; lpDirectory
0000000000403AE4 68 C0 52 43 00           push    offset aCBcdeditExeSet ; "/c bcdedit.exe /set {default} recoverye"...
0000000000403AE9 68 A0 52 43 00           push    offset File     ; "cmd.exe"
0000000000403AEE 68 B0 52 43 00           push    offset Operation ; "open"
0000000000403AF3 6A 00                    push    0               ; hwnd
0000000000403AF5 FF D6                    call    esi ; ShellExecuteW
0000000000403AF7 6A 00                    push    0               ; nShowCmd
0000000000403AF9 6A 00                    push    0               ; lpDirectory
0000000000403AFB 68 28 53 43 00           push    offset aCBcdeditExeSet_0 ; "/c bcdedit.exe /set {default} bootstatu"...
0000000000403B00 68 A0 52 43 00           push    offset File     ; "cmd.exe"
0000000000403B05 68 B0 52 43 00           push    offset Operation ; "open"
0000000000403B0A 6A 00                    push    0               ; hwnd
0000000000403B0C FF D6                    call    esi ; ShellExecuteW
0000000000403B0E 6A 00                    push    0               ; nShowCmd
0000000000403B10 6A 00                    push    0               ; lpDirectory
0000000000403B12 68 B0 53 43 00           push    offset aCVssadminExeDe_0 ; "/c vssadmin.exe delete shadows /all /qu"...
0000000000403B17 68 A0 52 43 00           push    offset File     ; "cmd.exe"
0000000000403B1C 68 B0 52 43 00           push    offset Operation ; "open"
0000000000403B21 6A 00                    push    0               ; hwnd
0000000000403B23 FF D6                    call    esi ; ShellExecuteW
```

In the first few variants, the strings related to the above mentioned registry keys, commands and ransom note were not obfuscated or encrypted at all in the binary.

With the Aleta variant, the ransom note was stored base64 encoded. Later all the above-mentioned strings were base64 encoded as well: from Nuclear the email address, from Payday all the strings related to the registry entries and the commands above.

Encryption

BTCWare kept evolving by making constant updates to it encryption method.

In Table 2, we have summarized a few details of the file encryption method of the different BTCWare variants.

Variant	New extension	User ID	File encryption algorithm	Encrypted size	Leaked master key	Where is the encrypted user ID stored?
BTCWare	.[< email address >].btcware	<0x19 random alphanumeric characters>	RC4	Whole file in 1000 byte long chunks	yes	key.dat
Cryptobyte	[< email address >].cryptobyte	OBAMA-<0x19 random alphanumeric characters>-<YYYY>-<MM>-<DD> XZD-<0x19 random alphanumeric characters>-<YYYY>-<MM>-<DD>	RC4	Whole file in 1000 byte long chunks	yes	Ransom note
Cryptowin	.[< email address >].cryptowin	ADM-<0x19 random alphanumeric characters>-<YYYY>-<MM>-<DD>	AES-192	Whole file in 992 byte long chunks	yes	Ransom note
Theva	.[< email address >].theva	SX-<random alphanumeric string>-<YYYY>-<MM>-<DD>	AES-192	Whole file in 992 byte long chunks	yes	Ransom note
Onyon	.[< email address >].onyon or .onyon	SX/DN-<random alphanumeric string>-<YYYY>-<MM>-<DD> or OBAMA-<random alphanumeric string>-<YYYY>-<MM>-<DD>	RC4	Encrypted length max 0xA00000 bytes	yes	Ransom note
Xfile	.[< email address >].xfile or .xfile	SX-<random alphanumeric string>-<YYYY>-<MM>-<DD>	RC4	Encrypted length max 0xA00000 bytes	yes	Ransom note

Master	.[< email address >].master	<number>-<random alphanumeric string>-<YYYY>-<MM>-<DD>	AES-256	Encrypted length max 0xA00000 bytes	yes	Ransom note
Encrypted	.[< email address >].encrypted	<number>-<random alphanumeric string>-<YYYY>-<MM>-<DD>	AES-256	Encrypted length max 0xA00000 bytes	no	Ransom note
Aleta	.[< email address >].aleta	<number>--<random alphanumeric string>-<YYYY>-<MM>-<DD>	AES-256	Encrypted length max 0xA00000 bytes	no	Ransom note or key.aleta
Blocking	.[< email address >].blocking	<number>-<random alphanumeric string>-<YYYY>-<MM>-<DD>	AES-256	Encrypted length max 0xA00000 bytes	no	Ransom note
Gryphon	.[< email address >].gryphon	User ID is generated using function addresses and the result of GetTickCount	AES-256	Encrypt first 0xFA00 bytes	no	Ransom note
Crypton	.[< email address >].crypton	User ID is generated using function addresses and the result of GetTickCount	AES-256	Encrypt first 0xFA00 bytes	no	Ransom note
Nuclear	.[< email address >].nuclear	User ID is generated using function addresses and the result of GetTickCount	AES-256	Encrypted length max 0xA00000 bytes	no	At the end of the encrypted file
Wyvern	.[< email address >]-id-< id >.wyvern	User ID is generated using function addresses and the result of GetTickCount	AES-256	Encrypted length max 0xA00000 bytes	no	At the end of the encrypted file
Payday	.[< email address >]-id-< id >.payday	CryptGenRandom is used for key generation	AES-256	Encrypted length max 0xA00000 bytes	no	At the end of the encrypted file
Shadow	.[< email address >]-id-< id >.shadow	CryptGenRandom is used for key generation	AES-256	Encrypted length max 0xA00000 bytes	no	At the end of the encrypted file
Wallet	.[< email address >]-id-< id >.wallet or .[< email address > or < email address >]-id-< id >.wallet	CryptGenRandom is used for key generation	AES-256	Encrypted length max 0xA00000 bytes	no	At the end of the encrypted file

Table 2

In each case, a new extension is appended to the encrypted file name. This extension usually contains an email address, which can be used to contact the attackers. Although the details of the key generation and the used encryption algorithm vary over different variants, the general process of file encryption is very similar in most variants.

BTCWare uses the combination of asymmetric and symmetric encryption. Each sample contains a hard-coded RSA-1024 public key, which means that this ransomware can encrypt files offline without communicating with a command-and-control server.

The file encryption process can be summarized in four steps:

1. User ID generation
2. User ID encryption
3. Symmetric key derivation
4. File encryption

In the next four sections, we will examine these steps and highlight the similarities and differences between different variants.

User ID generation

The first step is to generate a user ID, which later will be used for symmetric key generation. The list of the user IDs can be found in the third column of Table 2. We can group the variants into six different categories based on the used method to generate the user ID. Let's look at these one by one.

BTCWare

The first variant generates 0x19 random alphanumeric characters using the __time64, _srand and _rand functions. The first call to _rand determines if a number, lowercase letter or uppercase letter will be generated and the next call to _rand generates the character.

Example: 1HYvsz449Uhn4gs68N1vy0U5a

The return value of __time64 in the EAX register is used as seed for the pseudorandom number generation. This is a 32 bit value, which can be easily brute-forced. This and the fact that the same key was used to encrypt all the files make it possible to decrypt the encrypted files without paying the ransom.

```
.text:004036F0 generate_rand   proc near               ; CODE XREF: WinMain(x,x,x,x)+FF↓p
.text:004036F0                 push    ebx
.text:004036F1                 push    esi
.text:004036F2                 push    edi
.text:004036F3                 push    0               ; Time
.text:004036F5                 call    __time64
.text:004036FA                 push    eax             ; unsigned int
.text:004036FB                 call    _srand
.text:00403700                 push    1Ah             ; unsigned int
.text:00403702                 call    ??_U@YAPAXI@Z   ; operator new[](uint)
.text:00403707                 add     esp, 0Ch
.text:0040370A                 mov     edi, eax
.text:0040370C                 xor     esi, esi
.text:0040370E                 lea     ebx, [esi+1Ah]
.text:00403711
.text:00403711 loc_403711:                             ; CODE XREF: generate_rand+6E↓j
.text:00403711                 call    _rand
.text:00403716                 cdq
.text:00403717                 mov     ecx, 3
.text:0040371C                 idiv    ecx
.text:0040371E                 sub     edx, 0
.text:00403721                 jz      short loc_403747
.text:00403723                 sub     edx, 1
.text:00403726                 jz      short loc_40373A
.text:00403728                 sub     edx, 1
.text:0040372B                 jnz     short loc_40375A
.text:0040372D                 call    _rand
.text:00403732                 cdq
.text:00403733                 idiv    ebx
.text:00403735                 add     dl, 61h ; 'a'   ; lowercase letters
.text:00403738                 jmp     short loc_403757
.text:0040373A ; ---------------------------------------------------------------------------
.text:0040373A
.text:0040373A loc_40373A:                             ; CODE XREF: generate_rand+36↑j
.text:0040373A                 call    _rand
.text:0040373F                 cdq
.text:00403740                 idiv    ebx
.text:00403742                 add     dl, 41h ; 'A'   ; uppercase letters
.text:00403745                 jmp     short loc_403757
.text:00403747 ; ---------------------------------------------------------------------------
.text:00403747
.text:00403747 loc_403747:                             ; CODE XREF: generate_rand+31↑j
.text:00403747                 call    _rand
.text:0040374C                 cdq
.text:0040374D                 mov     ecx, 0Ah
.text:00403752                 idiv    ecx
.text:00403754                 add     dl, 30h ; '0'   ; number 0-9
```

Figure 1: BTCWare random generation

Cryptobyte, Cryptowin, Theva

0x19 random values are generated the same way as previously, but other characters are added to them.

Examples:

XZD-1ZCgy2V1tLUA22weX2Pw0K9lc-2018-01-11

ADM-5i051f4066j6JaK04JA8qfLt6-2018-01-11

OBAMA-GiaGHJ03PTyN07J9V011F5pH5-2018-01-11

In case of Theva variant, the number of random alphanumeric characters is calculated using the hour and minute value from the result of GetLocalTime function.

Example:

SX-lMJ93SpGA01sr29UK04M6f5TziTbjgQtH2R03358oh4k8Kf3YYt7BTL3Gcy7WD567FOm-2018-01-11.

Only the random alphanumeric characters will be used to derive the file encryption key.

The Cryptobyte and Cryptowin user IDs can be brute-forced the same way as in case of the BTCWare variant. Meanwhile the changes in Theva only multiply the number of possible keys by 1440, which means that brute-forcing is still possible.

```
000000000040408A
000000000040408A                       loc_40408A:
000000000040408A 8D 85 08 E5 FF FF     lea     eax, [ebp-1AF8h]
0000000000404090 66 89 7C 55 A0        mov     [ebp+edx*2-60h], di
0000000000404095 50                    push    eax             ; lpSystemTime
0000000000404096 FF 15 A0 90 42 00     call    ds:GetLocalTime
000000000040409C 6A 00                 push    0               ; Time
000000000040409E E8 C5 E1 00 00        call    __time64
00000000004040A3 83 C4 04              add     esp, 4
00000000004040A6 50                    push    eax             ; unsigned int
00000000004040A7 E8 A0 C2 00 00        call    _srand
00000000004040AC 0F B7 85 10 E5 FF FF  movzx   eax, word ptr [ebp-1AF0h] ; minute
00000000004040B3 83 C4 04              add     esp, 4
00000000004040B6 0F B7 8D 12 E5 FF FF  movzx   ecx, word ptr [ebp-1AEEh] ; hour
00000000004040BD 83 C0 1E              add     eax, 1Eh
00000000004040C0 03 C8                 add     ecx, eax
00000000004040C2 E8 89 FD FF FF        call    generate_rand
00000000004040C7 0F B7 8D 0E E5 FF FF  movzx   ecx, word ptr [ebp-1AF2h] ; day
00000000004040CE 51                    push    ecx
00000000004040CF 0F B7 8D 0A E5 FF FF  movzx   ecx, word ptr [ebp-1AF6h] ; month
00000000004040D6 51                    push    ecx
00000000004040D7 0F B7 8D 08 E5 FF FF  movzx   ecx, word ptr [ebp-1AF8h] ; year
00000000004040DE 51                    push    ecx
00000000004040DF 50                    push    eax
00000000004040E0 68 70 88 43 00        push    offset aSx      ; "SX"
00000000004040E5 68 78 E0 43 00        push    offset aSSD02d02d ; "%s-%s-%d-%02d-%02d"
00000000004040EA 68 10 3E 44 00        push    offset user_id  ; LPSTR
00000000004040EF A3 C0 29 44 00        mov     lpString, eax
00000000004040F4 FF 15 3C 92 42 00     call    ds:wsprintfA
00000000004040FA B9 10 3E 44 00        mov     ecx, offset user_id
00000000004040FF C7 85 90 E6 FF FF 00 00+mov   dword ptr [ebp-1970h], 0
0000000000404109 83 C4 1C              add     esp, 1Ch
000000000040410C C7 85 94 E6 FF FF 0F 00+mov   dword ptr [ebp-196Ch], 0Fh
0000000000404116 C6 85 80 E6 FF FF 00  mov     byte ptr [ebp-1980h], 0
000000000040411D 8D 51 01              lea     edx, [ecx+1]
```

Figure 2: Theva user ID generation (zmsksddfff.exe)

Onyon, Xfile

To create the user ID, these variants first generate a random string (string1) using GetLocalTime, the address of a few functions and a similar random generation method as previously. Then a string like "SX" or "DN" is added to the front of the user ID and the date to the end.

Example: DN-1967722968a38qpA50x6avM3TRT7lHe3xm82bKW913712A19677236001966031880YSJy9752FCEE9-2018-01-11

The generated random string (string1) is used to derive the file encryption key. Adding the address of certain functions is an unusual way to increase randomness, but this, with using the random generation twice, ensures that the previous brute-force method cannot be used anymore.

```
00000000004035CA
00000000004035CA                        loc_4035CA:
00000000004035CA FF 35 EC 71 42 00      push    ds:GetClipboardOwner
00000000004035D0 51                     push    ecx
00000000004035D1 B9 05 00 00 00         mov     ecx, 5
00000000004035D6 E8 A5 F7 FF FF         call    generate_rand
00000000004035DB 8B 8D C8 F1 FF FF      mov     ecx, [ebp-0E38h]
00000000004035E1 83 C4 04               add     esp, 4
00000000004035E4 50                     push    eax
00000000004035E5 FF 35 E0 71 42 00      push    ds:GetActiveWindow
00000000004035EB 0F B7 85 D8 F1 FF FF   movzx   eax, word ptr [ebp-0E28h] ; second
00000000004035F2 FF 35 48 70 42 00      push    ds:GetCurrentThreadId
00000000004035F8 50                     push    eax
00000000004035F9 0F B7 85 DA F1 FF FF   movzx   eax, word ptr [ebp-0E26h] ; millisecond
0000000000403600 50                     push    eax
0000000000403601 0F B7 85 D4 F1 FF FF   movzx   eax, word ptr [ebp-0E2Ch] ; hour
0000000000403608 83 C0 05               add     eax, 5
000000000040360B 03 C8                  add     ecx, eax
000000000040360D E8 6E F7 FF FF         call    generate_rand
0000000000403612 50                     push    eax
0000000000403613 FF 35 60 70 42 00      push    ds:GetCurrentProcessId
0000000000403619 68 64 3B 43 00         push    offset aDSDXDDSX ; "%d%s%d%X%d%d%s%X"
000000000040361E FF 35 70 DC E3 00      push    string1         ; LPSTR
0000000000403624 FF 15 E8 71 42 00      call    ds:wsprintfA
000000000040362A 0F B7 85 D2 F1 FF FF   movzx   eax, word ptr [ebp-0E2Eh] ; day
0000000000403631 50                     push    eax
0000000000403632 0F B7 85 CE F1 FF FF   movzx   eax, word ptr [ebp-0E32h] ; month
0000000000403639 50                     push    eax
000000000040363A 0F B7 85 CC F1 FF FF   movzx   eax, word ptr [ebp-0E34h] ; year
0000000000403641 50                     push    eax
0000000000403642 FF 35 70 DC E3 00      push    string1
0000000000403648 68 E0 37 43 00         push    offset aSx      ; "SX"
000000000040364D 68 78 3B 43 00         push    offset aSSD02d02d ; "%s-%s-%d-%02d-%02d"
0000000000403652 68 48 B5 E3 00         push    offset user_id  ; LPSTR
0000000000403657 FF 15 E8 71 42 00      call    ds:wsprintfA
000000000040365D B9 48 B5 E3 00         mov     ecx, offset user_id
0000000000403662 C7 85 C0 F1 FF FF 00 00+mov    dword ptr [ebp-0E40h], 0
000000000040366C 83 C4 44               add     esp, 44h
000000000040366F C7 85 C4 F1 FF FF 0F 00+mov    dword ptr [ebp-0E3Ch], 0Fh
0000000000403679 C6 85 B0 F1 FF FF 00   mov     byte ptr [ebp-0E50h], 0
0000000000403680 8D 51 01               lea     edx, [ecx+1]
```

Figure 3: Onyon user ID generation

Even though the user-ID generation flaw is fixed in this variant, they introduced a flaw in the encryption method in the updated variant which we discuss below.

Master, Encrypted, Aleta, Blocking

This is very similar to the previous method, but these variants add the result of GetTickCount to the string as well, and the beginning of the user ID is a number or combination of numbers (e.g. "1", "2-4" or "201").

Example:

1-774A0371FF5858963D5C1F1A359B8A461AFB7928975491450752F4408C4AA9CB4752FCEE9-2018-01-11

The generated random string (string1) is used to derive the file encryption key.

```
00000000004036B8
00000000004036B8                          loc_4036B8:
00000000004036B8 E8 D3 59 00 00           call    ?_Random_device@std@@YAIXZ ; std::_Random_device(void)
00000000004036BD 8B F0                    mov     esi, eax
00000000004036BF E8 CC 59 00 00           call    ?_Random_device@std@@YAIXZ ; std::_Random_device(void)
00000000004036C4 8B F8                    mov     edi, eax
00000000004036C6 E8 C5 59 00 00           call    ?_Random_device@std@@YAIXZ ; std::_Random_device(void)
00000000004036CB FF 35 FC 71 42 00        push    ds:GetClipboardOwner
00000000004036D1 0F B7 8D 8C F2 FF FF     movzx   ecx, word ptr [ebp-0D74h] ; minute
00000000004036D8 56                       push    esi
00000000004036D9 FF 35 F0 71 42 00        push    ds:GetActiveWindow
00000000004036DF 89 85 C8 F1 FF FF        mov     [ebp-0E38h], eax
00000000004036E5 FF 35 4C 70 42 00        push    ds:GetCurrentThreadId
00000000004036EB 0F B7 85 8E F2 FF FF     movzx   eax, word ptr [ebp-0D72h]
00000000004036F2 51                       push    ecx
00000000004036F3 50                       push    eax
00000000004036F4 57                       push    edi
00000000004036F5 51                       push    ecx
00000000004036F6 E8 25 F7 FF FF           call    generate_rand
00000000004036FB 83 C4 04                 add     esp, 4
00000000004036FE 50                       push    eax
00000000004036FF FF B5 C8 F1 FF FF        push    dword ptr [ebp-0E38h]
0000000000403705 FF 15 6C 70 42 00        call    ds:GetTickCount
000000000040370B 8B 35 F8 71 42 00        mov     esi, ds:wsprintfA
0000000000403711 50                       push    eax
0000000000403712 FF 35 5C 70 42 00        push    ds:GetCurrentProcessorNumber
0000000000403718 68 B8 3B 43 00           push    offset aXXXXXDXXXXX ; "%X%X%X%X%X%d%X%X%X%X%X"
000000000040371D FF 35 78 DC 83 01        push    string1         ; LPSTR
0000000000403723 FF D6                    call    esi ; wsprintfA
0000000000403725 0F B7 85 86 F2 FF FF     movzx   eax, word ptr [ebp-0D7Ah] ; day
000000000040372C 83 C4 34                 add     esp, 34h
000000000040372F 50                       push    eax
0000000000403730 0F B7 85 82 F2 FF FF     movzx   eax, word ptr [ebp-0D7Eh] ; month
0000000000403737 50                       push    eax
0000000000403738 0F B7 85 80 F2 FF FF     movzx   eax, word ptr [ebp-0D80h] ; year
000000000040373F 50                       push    eax
0000000000403740 FF 35 78 DC 83 01        push    string1
0000000000403746 68 9C 33 43 00           push    offset unk_43339C
000000000040374B 68 D0 3B 43 00           push    offset aSSD02d02d ; "%s-%s-%d-%02d-%02d"
0000000000403750 68 50 B5 83 01           push    offset user_id  ; LPSTR
0000000000403755 FF D6                    call    esi ; wsprintfA
0000000000403757 B9 50 B5 83 01           mov     ecx, offset user_id
000000000040375C C7 85 C0 F1 FF FF 00 00+mov      dword ptr [ebp-0E40h], 0
0000000000403766 83 C4 1C                 add     esp, 1Ch
0000000000403769 C7 85 C4 F1 FF FF 0F 00+mov      dword ptr [ebp-0E3Ch], 0Fh
0000000000403773 C6 85 B0 F1 FF FF 00     mov     byte ptr [ebp-0E50h], 0
000000000040377A 8D 51 01                 lea     edx, [ecx+1]
000000000040377D 0F 1F 00                 nop     dword ptr [eax]
```

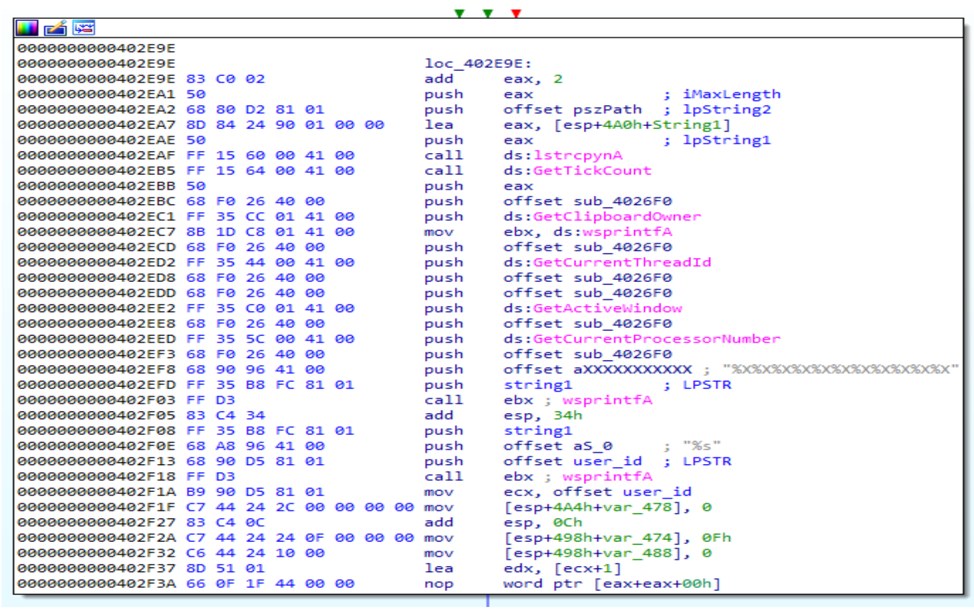
Figure 4: Master user ID generation

In this case, the user ID generation is safe thanks to the combination of the usual random generation and the result of the GetTickCount function. However, there is another flaw in these variants, which can be used in certain cases to decrypt the encrypted files. These variants keep scanning the machine for new files to encrypt. This means that if the victim's computer hasn't been rebooted after the ransomware attack and the ransomware process hasn't been killed by the victim, then the encryption key can be retrieved from memory.

Gryphon, Crypton, Nuclear, Wyvern

These variants do not use the random generator method (with the _srand and _rand functions), which was used by all the previous variants. The user ID is generated using the result of GetTickCount and the address of a few functions. No other characters (like the date) are appended to it.

Example: 13828F0774A037113828F0752F440813828F013828F07549145013828F0752FCEE913828F0103EDA0

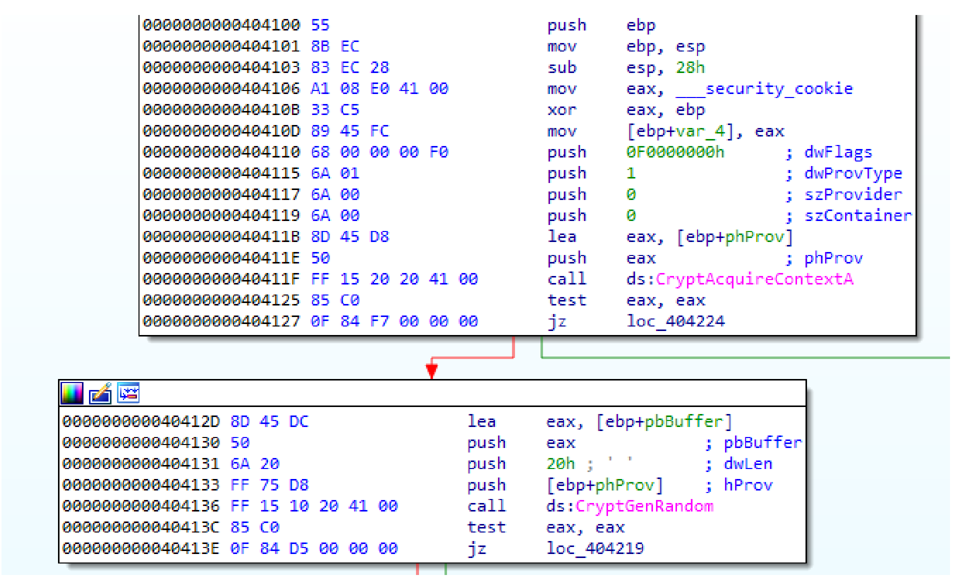


```
0000000000402E9E
0000000000402E9E                          loc_402E9E:
0000000000402E9E 83 C0 02                 add     eax, 2
0000000000402EA1 50                       push    eax             ; iMaxLength
0000000000402EA2 68 80 D2 81 01           push    offset pszPath  ; lpString2
0000000000402EA7 8D 84 24 90 01 00 00     lea     eax, [esp+4A0h+String1]
0000000000402EAE 50                       push    eax             ; lpString1
0000000000402EAF FF 15 60 00 41 00        call    ds:lstrcpynA
0000000000402EB5 FF 15 64 00 41 00        call    ds:GetTickCount
0000000000402EBB 50                       push    eax
0000000000402EBC 68 F0 26 40 00           push    offset sub_4026F0
0000000000402EC1 FF 35 CC 01 41 00        push    ds:GetClipboardOwner
0000000000402EC7 8B 1D C8 01 41 00        mov     ebx, ds:wsprintfA
0000000000402ECD 68 F0 26 40 00           push    offset sub_4026F0
0000000000402ED2 FF 35 44 00 41 00        push    ds:GetCurrentThreadId
0000000000402ED8 68 F0 26 40 00           push    offset sub_4026F0
0000000000402EDD 68 F0 26 40 00           push    offset sub_4026F0
0000000000402EE2 FF 35 C0 01 41 00        push    ds:GetActiveWindow
0000000000402EE8 68 F0 26 40 00           push    offset sub_4026F0
0000000000402EED FF 35 5C 00 41 00        push    ds:GetCurrentProcessorNumber
0000000000402EF3 68 F0 26 40 00           push    offset sub_4026F0
0000000000402EF8 68 90 96 41 00           push    offset aXXXXXXXXXXX ; "%X%X%X%X%X%X%X%X%X%X%X%X"
0000000000402EFD FF 35 B8 FC 81 01        push    string1         ; LPSTR
0000000000402F03 FF D3                    call    ebx ; wsprintfA
0000000000402F05 83 C4 34                 add     esp, 34h
0000000000402F08 FF 35 B8 FC 81 01        push    string1
0000000000402F0E 68 A8 96 41 00           push    offset aS_0     ; "%s"
0000000000402F13 68 90 D5 81 01           push    offset user_id  ; LPSTR
0000000000402F18 FF D3                    call    ebx ; wsprintfA
0000000000402F1A B9 90 D5 81 01           mov     ecx, offset user_id
0000000000402F1F C7 44 24 2C 00 00 00 00  mov     [esp+4A4h+var_478], 0
0000000000402F27 83 C4 0C                 add     esp, 0Ch
0000000000402F2A C7 44 24 24 0F 00 00 00  mov     [esp+498h+var_474], 0Fh
0000000000402F32 C6 44 24 10 00           mov     [esp+498h+var_488], 0
0000000000402F37 8D 51 01                 lea     edx, [ecx+1]
0000000000402F3A 66 0F 1F 44 00 00        nop     word ptr [eax+eax+00h]
```

Figure 5: Nuclear user ID generation

Payday, Shadow, Wallet

From the Payday variant, the generation of the key has changed completely. These variants use the CryptGenRandom function to generate 0x20 random bytes.



```
0000000000404100 55                 push    ebp
0000000000404101 8B EC              mov     ebp, esp
0000000000404103 83 EC 28           sub     esp, 28h
0000000000404106 A1 08 E0 41 00     mov     eax, ___security_cookie
000000000040410B 33 C5              xor     eax, ebp
000000000040410D 89 45 FC           mov     [ebp+var_4], eax
0000000000404110 68 00 00 00 F0     push    0F0000000h      ; dwFlags
0000000000404115 6A 01              push    1               ; dwProvType
0000000000404117 6A 00              push    0               ; szProvider
0000000000404119 6A 00              push    0               ; szContainer
000000000040411B 8D 45 D8           lea     eax, [ebp+phProv]
000000000040411E 50                 push    eax             ; phProv
000000000040411F FF 15 20 20 41 00  call    ds:CryptAcquireContextA
0000000000404125 85 C0              test    eax, eax
0000000000404127 0F 84 F7 00 00 00  jz      loc_404224

000000000040412D 8D 45 DC           lea     eax, [ebp+pbBuffer]
0000000000404130 50                 push    eax             ; pbBuffer
0000000000404131 6A 20              push    20h ; ' '       ; dwLen
0000000000404133 FF 75 D8           push    [ebp+phProv]    ; hProv
0000000000404136 FF 15 10 20 41 00  call    ds:CryptGenRandom
000000000040413C 85 C0              test    eax, eax
000000000040413E 0F 84 D5 00 00 00  jz      loc_404219
```

Figure 6: Payday random generation

We can see from these examples how the developers of the BTCWare ransomware family were experimenting with the different solutions. From a completely vulnerable method they managed to reach a cryptographically secure solution. In every step, they tried to make their implementation more secure. They started with a very common mistake, using the _srand and _rand functions for random generation, which is vulnerable to brute force attack. Later, they tried to add other values to increase the randomness using GetLocalTime, function addresses and GetTickCount. By the last couple of versions, they completely removed the _srand and _rand functions and started to use the most popular and secure solution, the CryptGenRandom function.

User ID encryption

Once the user ID is generated, the next step is to encrypt it with the hard-coded RSA public key. This method is the same across all variants. The main functions used during encryption are the following:

CryptStringToBinaryA, CryptDecodeObjectEx, CryptImportPublicKeyInfo, CryptEncrypt.

These read, decode, and import the hard-coded RSA public key, and then encrypt the generated user ID. The encrypted user ID is saved to either a specific file, the ransom note itself, or at the end of the encrypted files. The last column of Table 2 shows this location by variants.

Symmetric key derivation

The symmetric key, used for file encryption, is derived from the user ID. The general process, which is almost exactly the same across all variants, is the following:

- The MD5 hash of the user ID (or some part of it) is calculated using the CryptCreateHash and CryptHashData functions.
- From this hash the symmetric key is derived using the CryptDeriveKey function. The ALG_ID input of this function identifies the symmetric encryption algorithm for which the key is to be generated. The fourth column of Table 2 contains the used encryption algorithm by different variants.
- Finally, if the encryption algorithm is AES, using the CryptSetKeyParam function the IV, padding and encryption mode are set.

The first few variants have a backup method for symmetric key generation. If for some reason the user ID generation was unsuccessful, the encryption key will be generated using the CryptGenKey, CryptGetUserKey, and CryptExportKey functions.

Three different encryption algorithms were used by the different variants: RC4, AES-192, and AES-256. The changes across these methods also show how the developers of the BTCWare family were experimenting with the different solutions. They started with RC4 with an implementation mistake. They use the same key for each file, which makes this method vulnerable to known plaintext attack. In the end they reached the secure AES-256 encryption in CBC-mode.

File encryption

The file encryption method is very similar for all variants. A certain number of bytes is read from the file using the ReadFile function. These bytes are then encrypted by CryptEncrypt and written to file by the WriteFile function. Either a new file is created for the encrypted content and the original is deleted after encryption, or the original file will be simply overwritten and renamed by MoveFileExA.

All the variants use specific white list of folders as mentioned below, which will be skipped during encryption.

$recycle.bin, program files, program files (x86), programdata, windows, nvidia, intel, appdata, temp, msocache, inetpub.

The first few variants used a list of extensions to select which files to encrypt, but later variants encrypt all files regardless their extension, except in the listed folders. In most of the cases the ransom note and the already encrypted files are also skipped, but we have seen a few exceptions.

For example, one of the Aleta samples encrypted its own ransom note, which makes it impossible for the victims to read the attackers' message.

Some variants also have a list of folders where they do encrypt despite of the previous exclusion:

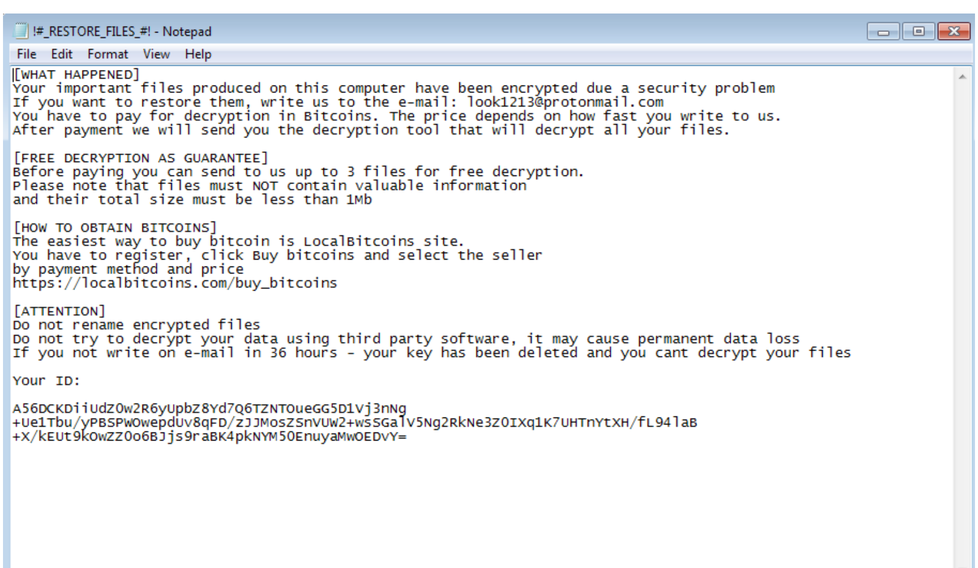
- C:\Program Files\MySQL\
- C:\Program Files (x86)\Firebird\
- C:\Program Files (x86)\MSSQL.1\

It's not just the flaws in the implementation of the cryptographic solutions that made it possible to decrypt certain variants of this ransomware family. The developers of BTCWare have made two of their master secret keys public.

The first release made possible to decrypt the BTCWare, Cryptobyte, Cryptowin variants, while the second key belongs to the Theva, Onyon, Xfile, and Master variants. [4 5] (There are some exceptions among these variants which use these extensions, but a different private key.) Every time the key was released a new variant appeared with a different private key to keep the business running.

Ransom demand

The ransom note is almost exactly the same for all variants. The file format and design changed a little bit over time, but the main content remained the same. The victims have to contact the attackers using a given email address. They can send a couple of files to the attackers who will decrypt those for free.


!#_RESTORE_FILES_#! - Notepad

File Edit Format View Help

```
[WHAT HAPPENED]
Your important files produced on this computer have been encrypted due a security problem
If you want to restore them, write us to the e-mail: look1213@protonmail.com
You have to pay for decryption in Bitcoins. The price depends on 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 to us up to 3 files for free decryption.
Please note that files must NOT contain valuable information
and their total size must be less than 1Mb

[HOW TO OBTAIN BITCOINS]
The easiest way to buy bitcoin is LocalBitcoins site.
You have to register, click Buy bitcoins and select the seller
by payment method and price
https://localbitcoins.com/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
If you not write on e-mail in 36 hours - your key has been deleted and you cant decrypt your files

Your ID:
```

A56DCKDiiUdZ0w2R6yUpbZ8Yd7Q6TZNTOueGG5D1Vj3nNg
+Ue1Tbu/yPBSPWOwepdUv8qFD/zJJMosZSnVUW2+wsSGalV5Ng2RkNe3Z0IXq1K7UHTnYtXH/fL94laB
+X/kEUt9kOwZZ0o6BJjs9raBK4pkNYM50EnuyaMwOEDvY=

Figure 7: Master ransom note

In a few cases we have seen different ransom notes but with much less content.

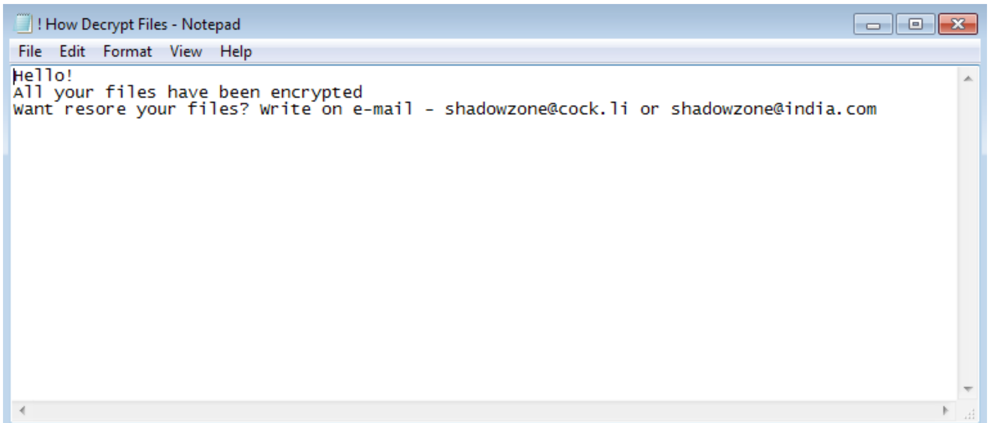


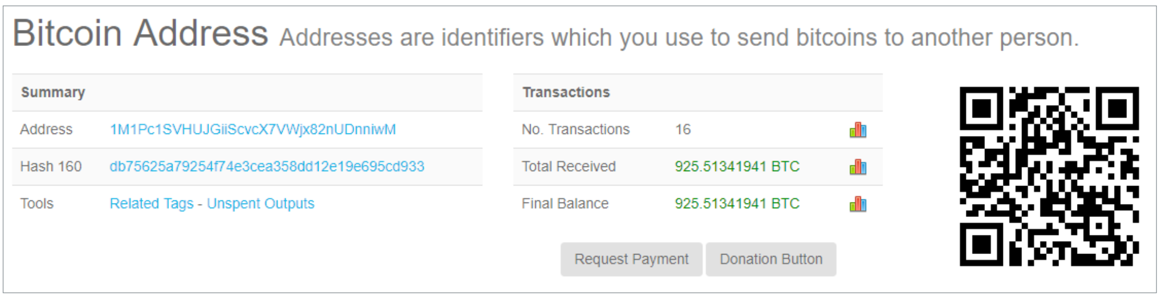
Figure 8: Wallet ransom note

Figure 9: PayDay ransom note

We have found a few examples in which the attackers provided a TOR address to the victims instead of an email address. For example, the address hxxp://cr7icbfqm64hixta[.]onion is used for payment instead of contacting the author via email ID.

Ransom payment

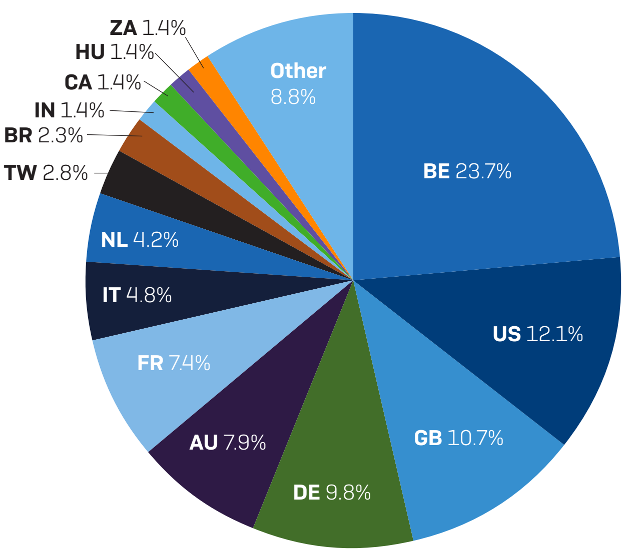
If a victim contacts the attackers using the given email address, the victim will receive a Bitcoin address and the amount of Bitcoin demanded. We tried to collect Bitcoin addresses, mainly from forums, where the victims shared the information they received from the attackers. Following one Bitcoin address we found on the Bleeping Computer support forum, we reached a Bitcoin address in two steps with more than 925 Bitcoins.



Statistics

Although most of the BTCWare attacks targeted European countries with Belgium in the lead, we have seen attacks all around the world. The United States was the second most affected country and we can find Taiwan, India, Canada, and the South African Republic represented among the targets as well.

In the second half of 2017 BTCWare has shown a steady presence, with relatively low number of customer lookups, thanks to the fact that it was distributed using RDP.



Protection

Sophos detects BTCWare ransomware using the following detections:

Mal/Btcware-A, Troj/Btcware-*, HPmal/BTCWare-A.

Additionally, Sophos Intercept X proactively prevents the malware from attacking your data, as the CryptoGuard component stops the ransomware from scrambling your files.

Summary

Ransomware in general is a big problem and users need to apply good security practices to avoid falling victim to it.

For more information about ransomware and best security practices, read "How to stay protected against ransomware."

References

1. Unit 42. [Online] https://researchcenter.paloaltonetworks.com/2017/03/unit42-Blank-Slate-campaign-takes-advantage-hosting-providers-spread-ransomware/.

2. Malware-Traffic-Analysis.net. [Online] http://www.malware-traffic-analysis.net/2017/08/02/index4.html.

3. Malware-Traffic-Analysis.net. [Online] http://www.malware-traffic-analysis.net/2017/07/29/index.html.

4. BleepingComputer. [Online] https://www.bleepingcomputer.com/news/security/btcware-ransomware-master-key-released-free-decrypter-available/.

5. Bleeping Computer. [Online] https://www.bleepingcomputer.com/news/security/new-btcware-ransomware-decrypter-released-for-the-master-variant/.

Appendix A

Here are some of the email addresses to contact the attackers mentioned in the ransom notes:

3bitcoins[at]protonmail.com	kekin[at]cock.li
alekstraza[at]bigmir.net	keyforyou[at]tuta.io
assistance[at]firemail.cc	lavandos[at]dr.com
atf13[at]tuta.io	lockers[at]tutamail.com
avalona.toga[at]aol.com	lockers[at]protonmail.com
averia[at]tuta.io	mail[at]aleta.cc
averiasw[at]qq.com	Merlin[at]aolonline.top
badking[at]india.com	mia.kokers[at]aol.com
barbarossol051223[at]tutanota.com	microcost[at]bigmir.net
black.block[at]qq.com	moneymaker2[at]india.com
black.mirror[at]qq.com	mortalis_certamen[at]aol.com
black.world[at]tuta.io	nicecrypt[at]india.com
blacklandfat[at]qq.com	nkr.siger[at]protonmail.ch
bravobravo[at]cock.li	nuclear[at]cryptmaster.info
Checkzip[at]india.com	pakhomovsemen60[at]gmail.com
chines34[at]protonmail.ch	payfordecrypt[at]qq.com
crypt24[at]protonmail.com	predatorthre[at]bigmir.net
darkwaiderr[at]cock.li	prt.nyke[at]protonmail.ch
darkwaiderr[at]tutanota.com	realsapport[at]bigmir.net
decr[at]cock.li	roggers[at]bigmir.net
decryptorx[at]cock.li	stopstorage[at]qq.com
decryptyourfileshereee1[at]cock.li	support[at]fbamasters.com
garryhelpyou[at]qq.com	torettoxyx[at]gmail.com
gladius_rectus[at]aol.com	tsce308[at]tutanota.com
goliaf[at]tuta.io	tullakump[at]tutanota.com
help[at]onyon.info	universe1[at]protonmail.ch
hostname[at]bigmir.net	usermsd[at]cock.li
info[at]kraken.cc	vargbt[at]tutanota.com
info[at]zayka.pro	webmafia[at]asia.com
irmagetstein[at]india.com	wyvern[at]cryptmaster.info

United Kingdom and Worldwide Sales
Tel: +44 (0)8447 671131
Email: sales@sophos.com

North American Sales
Toll Free: 1-866-866-2802
Email: nasales@sophos.com

Australia and New Zealand Sales
Tel: +61 2 9409 9100
Email: sales@sophos.com.au

Asia Sales
Tel: +65 62244168
Email: salesasia@sophos.com

18-04 WP-NA (NP)

SOPHOS