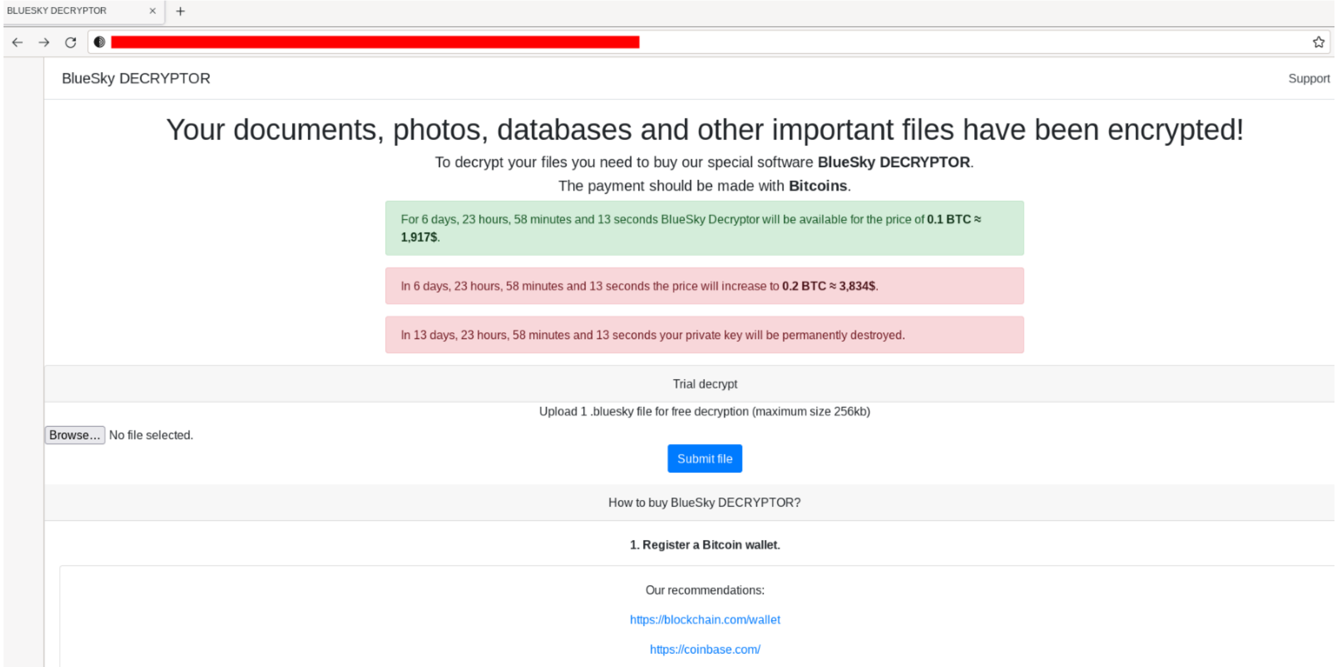
Dissecting BlueSky Ransomware Payload

yoroi.company/research/dissecting-bluesky-ransomware-payload/

September 30, 2022



BlueSky DECRYPTOR

Your documents, photos, databases and other important files have been encrypted!

To decrypt your files you need to buy our special software **BlueSky DECRYPTOR**.

The payment should be made with **Bitcoins**.

For 6 days, 23 hours, 58 minutes and 13 seconds BlueSky Decryptor will be available for the price of **0.1 BTC ≈ 1,917$**.

In 6 days, 23 hours, 58 minutes and 13 seconds the price will increase to **0.2 BTC ≈ 3,834$**.

In 13 days, 23 hours, 58 minutes and 13 seconds your private key will be permanently destroyed.

Trial decrypt

Upload 1 .bluesky file for free decryption (maximum size 256kb)

Browse… No file selected.

Submit file

How to buy BlueSky DECRYPTOR?

1. Register a Bitcoin wallet.

Our recommendations:

https://blockchain.com/wallet

https://coinbase.com/

09/30/2022

Introduction

BlueSky is a ransomware firstly spotted in May 2022 and it gained the attention of the threat researchers for two main reasons: the first one is that the group behind the ransomware doesn't adopt the double-extortion model; the second one is that their targets are even normal users because the ransomware has been discovered inside cracks of programs and videogames.

For these reasons, we at Yoroi malware ZLab decided to keep track of the threat, following the distribution of the samples, and we decided to provide a technical analysis of the ransomware payload.

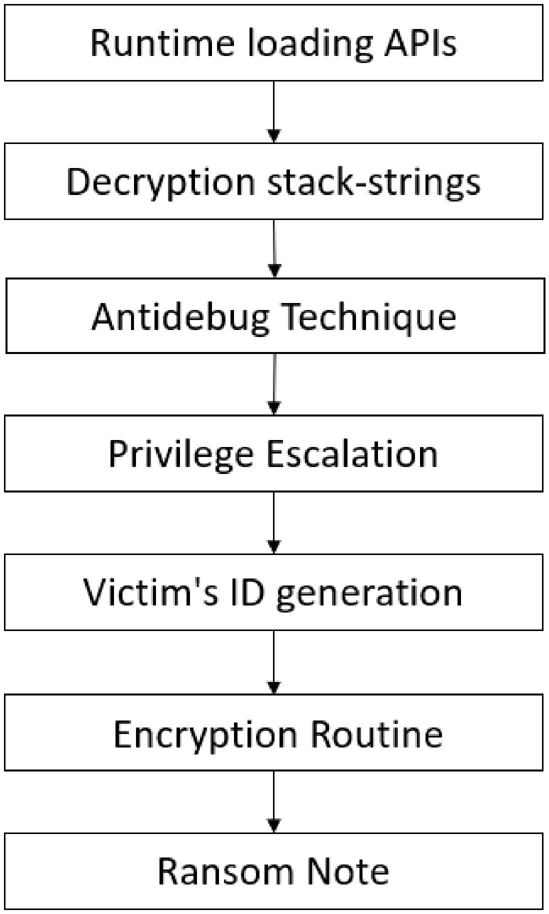


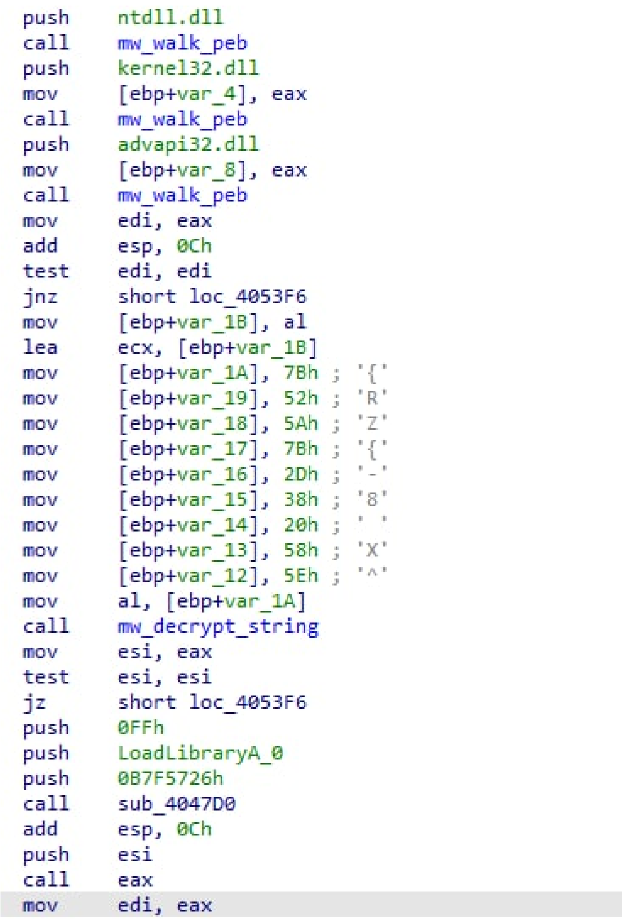
Figure 1: Bluesky Control Flow

Technical Analysis

Hash	9e302bb7d1031c0b2a4ad6ec955e7d2c0ab9c0d18d56132029c4c6198b91384f
Threat	Ransomware
Brief Description	BlueSky Ransomware
SSDEEP	1536:G+5geBR2Q+a8M124Zl2i5SADBDg8trv4t9MBY5ySvV:GDeBgQ+a8M12Y2i59hrvWMBGvV

The API Loading Scheme

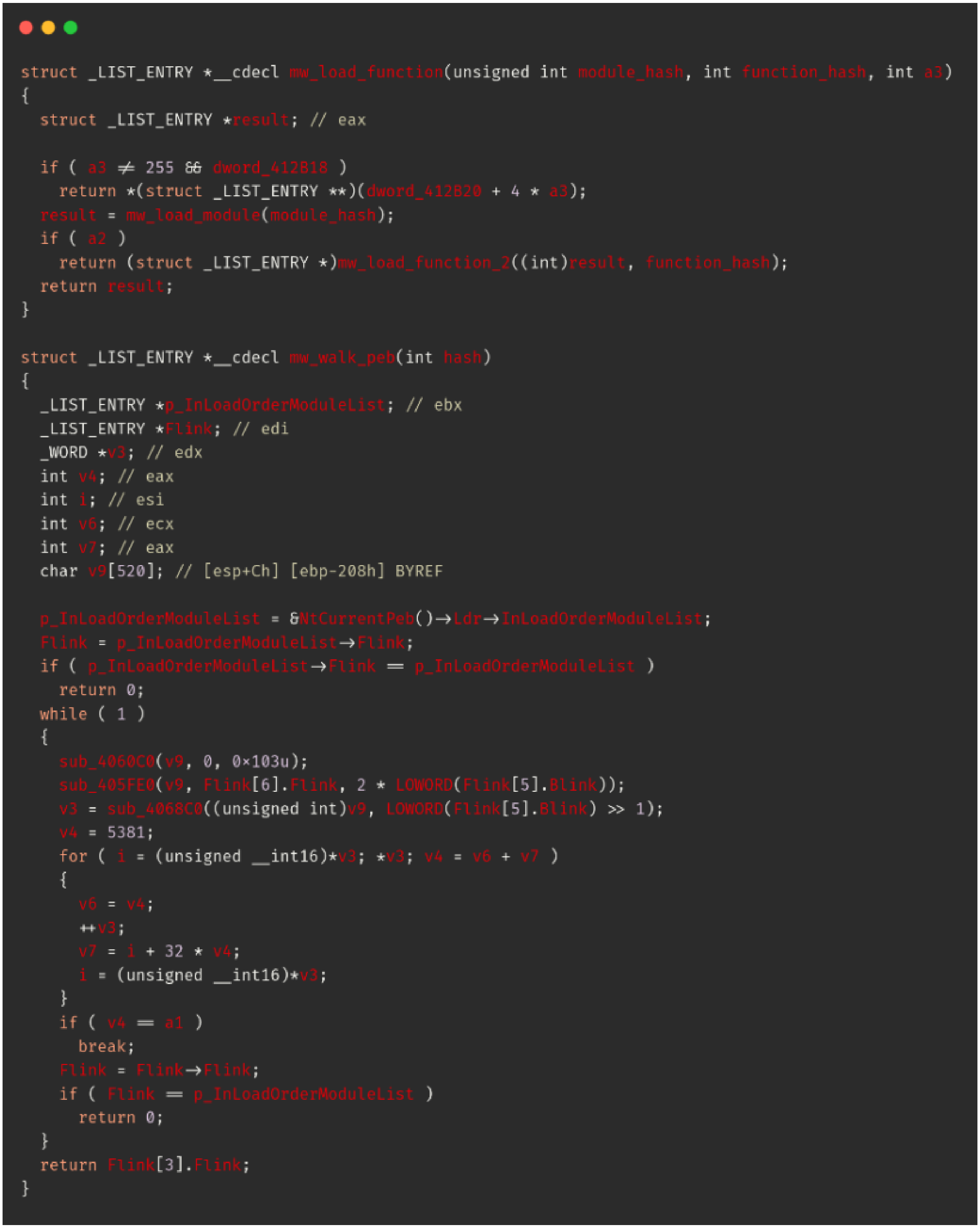
The sample starts by walking the PEB (Process Environment Block) to dynamically load the APIs. It is a common technique to not statically show them in the import table, it walks one of the three linked lists located in the **PEB_LDR_DATA** such as **InLoadOrderModuleList.** In this way, the sample is able to enumerate the modules contained inside the linked list and to compare them with the hashed names hidden inside the code in order to correctly import the desired ones. In this case, the APIs are hashed with ***djb2 algorithm***.



```
push    ntdll.dll
call    mw_walk_peb
push    kernel32.dll
mov     [ebp+var_4], eax
call    mw_walk_peb
push    advapi32.dll
mov     [ebp+var_8], eax
call    mw_walk_peb
mov     edi, eax
add     esp, 0Ch
test    edi, edi
jnz     short loc_4053F6
mov     [ebp+var_1B], al
lea     ecx, [ebp+var_1B]
mov     [ebp+var_1A], 7Bh ; '{'
mov     [ebp+var_19], 52h ; 'R'
mov     [ebp+var_18], 5Ah ; 'Z'
mov     [ebp+var_17], 7Bh ; '{'
mov     [ebp+var_16], 2Dh ; '-'
mov     [ebp+var_15], 38h ; '8'
mov     [ebp+var_14], 20h ; ' '
mov     [ebp+var_13], 58h ; 'X'
mov     [ebp+var_12], 5Eh ; '^'
mov     al, [ebp+var_1A]
call    mw_decrypt_string
mov     esi, eax
test    esi, esi
jz      short loc_4053F6
push    0FFh
push    LoadLibraryA_0
push    0B7F5726h
call    sub_4047D0
add     esp, 0Ch
push    esi
call    eax
mov     edi, eax
```

Figure 2: Dynamically loading APIs

The following figure shows the routines to dynamically load the function:

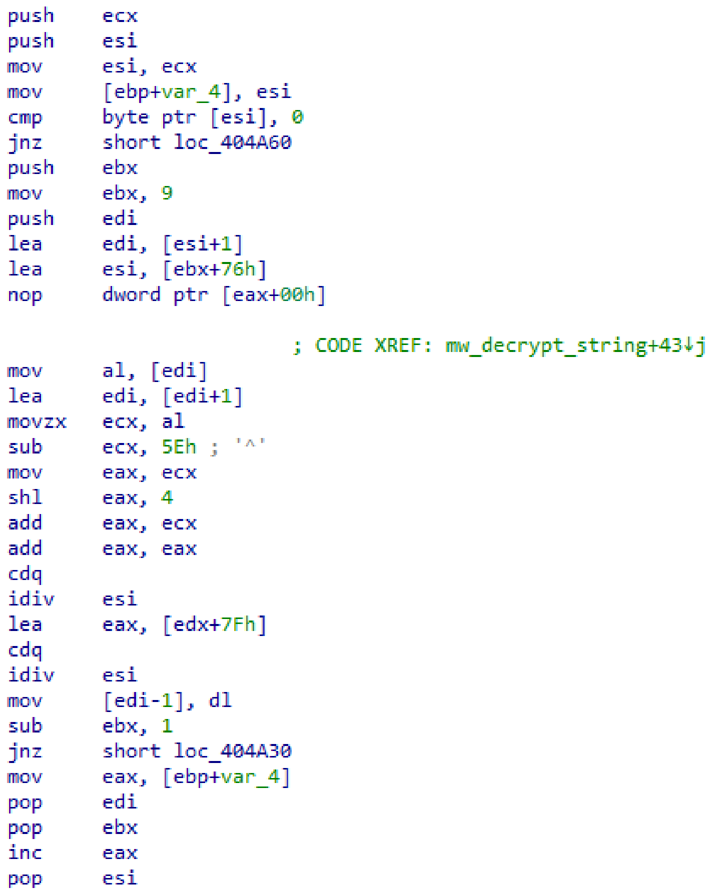


```
struct _LIST_ENTRY *__cdecl mw_load_function(unsigned int module_hash, int function_hash, int a3)
{
  struct _LIST_ENTRY *result; // eax

  if ( a3 != 255 && dword_412B18 )
    return *(struct _LIST_ENTRY **)(dword_412B20 + 4 * a3);
  result = mw_load_module(module_hash);
  if ( a2 )
    return (struct _LIST_ENTRY *)mw_load_function_2((int)result, function_hash);
  return result;
}

struct _LIST_ENTRY *__cdecl mw_walk_peb(int hash)
{
  _LIST_ENTRY *p_InLoadOrderModuleList; // ebx
  _LIST_ENTRY *Flink; // edi
  _WORD *v3; // edx
  int v4; // eax
  int i; // esi
  int v6; // ecx
  int v7; // eax
  char v9[520]; // [esp+Ch] [ebp-208h] BYREF

  p_InLoadOrderModuleList = &NtCurrentPeb()->Ldr->InLoadOrderModuleList;
  Flink = p_InLoadOrderModuleList->Flink;
  if ( p_InLoadOrderModuleList->Flink == p_InLoadOrderModuleList )
    return 0;
  while ( 1 )
  {
    sub_4060C0(v9, 0, 0x103u);
    sub_405FE0(v9, Flink[6].Flink, 2 * LOWORD(Flink[5].Blink));
    v3 = sub_4068C0((unsigned int)v9, LOWORD(Flink[5].Blink) >> 1);
    v4 = 5381;
    for ( i = (unsigned __int16)*v3; *v3; v4 = v6 + v7 )
    {
      v6 = v4;
      ++v3;
      v7 = i + 32 * v4;
      i = (unsigned __int16)*v3;
    }
    if ( v4 == a1 )
      break;
    Flink = Flink->Flink;
    if ( Flink == p_InLoadOrderModuleList )
      return 0;
  }
  return Flink[3].Flink;
}
```

Figure 3: "mw_load_function routine"

The obfuscated Stack Strings

Instead, other critical strings are obfuscated through the stackstrings method and a simple routine to encrypt them



```
push    ecx
push    esi
mov     esi, ecx
mov     [ebp+var_4], esi
cmp     byte ptr [esi], 0
jnz     short loc_404A60
push    ebx
mov     ebx, 9
push    edi
lea     edi, [esi+1]
lea     esi, [ebx+76h]
nop     dword ptr [eax+00h]

                        ; CODE XREF: mw_decrypt_string+43↓j
mov     al, [edi]
lea     edi, [edi+1]
movzx   ecx, al
sub     ecx, 5Eh ; '^'
mov     eax, ecx
shl     eax, 4
add     eax, ecx
add     eax, eax
cdq
idiv    esi
lea     eax, [edx+7Fh]
cdq
idiv    esi
mov     [edi-1], dl
sub     ebx, 1
jnz     short loc_404A30
mov     eax, [ebp+var_4]
pop     edi
pop     ebx
inc     eax
pop     esi
```

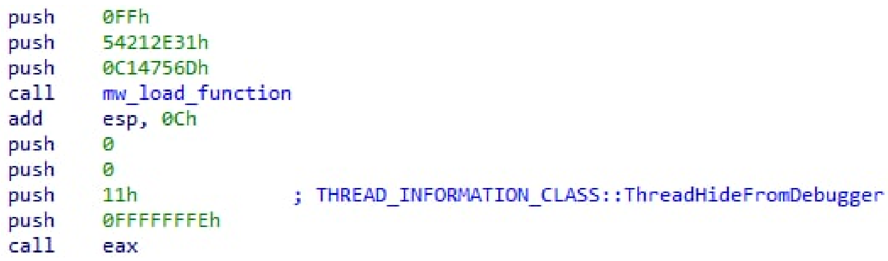
Figure 4: Strings Decryption Routine

However, the algorithm is easy to revert, and we developed an easy script to decrypt the stackstrings:

```
string = [123,82,90,123,45,56,32,88,94]

decrypted = ""

for i in string:

    decrypted += chr((34 * (i - 94) % 127 + 127) % 127)

print(decrypted)
```

Anti-Debug Technique

Once resolved the first functions, the sample calls **NtSetInformationThread** with **ThreadHideFromDebugger** hiding the thread and if any breakpoint is placed causing the crash of the process, you can read more about this anti-debug technique here



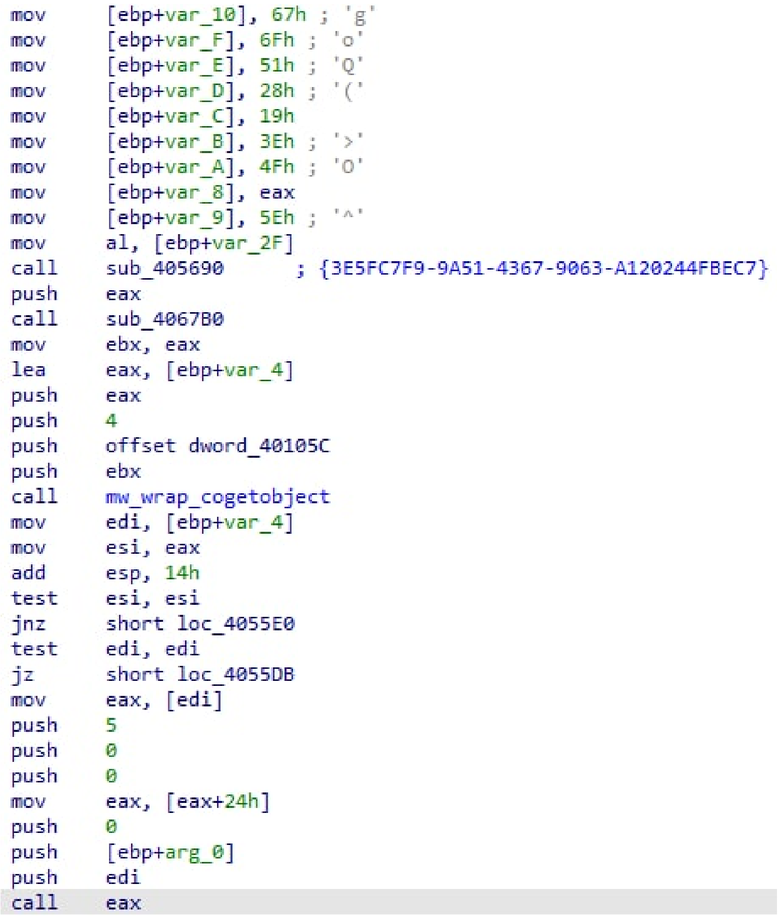
```
push    0FFh
push    54212E31h
push    0C14756Dh
call    mw_load_function
add     esp, 0Ch
push    0
push    0
push    11h             ; THREAD_INFORMATION_CLASS::ThreadHideFromDebugger
push    0FFFFFFFEh
call    eax
```

Figure 5: NtSetInformationThread

anti-debug

Privilege Escalation

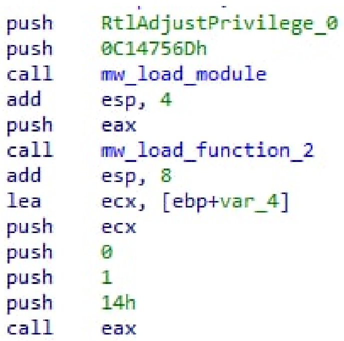
While analyzing the sample, we also found similarities with Conti Ransomware in how the strings are obfuscated and some other routines, like how BlueSky removes the shadow copies through the WMI COM Interface. It abuses the "*ICMLuaUtil COM Interface (3E5FC7F9-9A51-4367-9063-A120244FBEC7)*". However, this technique is a well-known and documented technique publicly available on the internet, adopted both in intrusion and malware development operations.



```
mov     [ebp+var_10], 67h ; 'g'
mov     [ebp+var_F], 6Fh ; 'o'
mov     [ebp+var_E], 51h ; 'Q'
mov     [ebp+var_D], 28h ; '('
mov     [ebp+var_C], 19h
mov     [ebp+var_B], 3Eh ; '>'
mov     [ebp+var_A], 4Fh ; 'O'
mov     [ebp+var_8], eax
mov     [ebp+var_9], 5Eh ; '^'
mov     al, [ebp+var_2F]
call    sub_405690      ; {3E5FC7F9-9A51-4367-9063-A120244FBEC7}
push    eax
call    sub_4067B0
mov     ebx, eax
lea     eax, [ebp+var_4]
push    eax
push    4
push    offset dword_40105C
push    ebx
call    mw_wrap_cogetobject
mov     edi, [ebp+var_4]
mov     esi, eax
add     esp, 14h
test    esi, esi
jnz     short loc_4055E0
test    edi, edi
jz      short loc_4055DB
mov     eax, [edi]
push    5
push    0
push    0
mov     eax, [eax+24h]
push    0
push    [ebp+arg_0]
push    edi
call    eax
```

Figure 6: Bypassing UAC via ICMLuaUtil

The sample calls RtlAdjustPrivilege API call with the token "SeDebugPrivilege", in order to gain the privilege to arbitrary manipulate every file and process.



```
push    RtlAdjustPrivilege_0
push    0C14756Dh
call    mw_load_module
add     esp, 4
push    eax
call    mw_load_function_2
add     esp, 8
lea     ecx, [ebp+var_4]
push    ecx
push    0
push    1
push    14h
call    eax
```

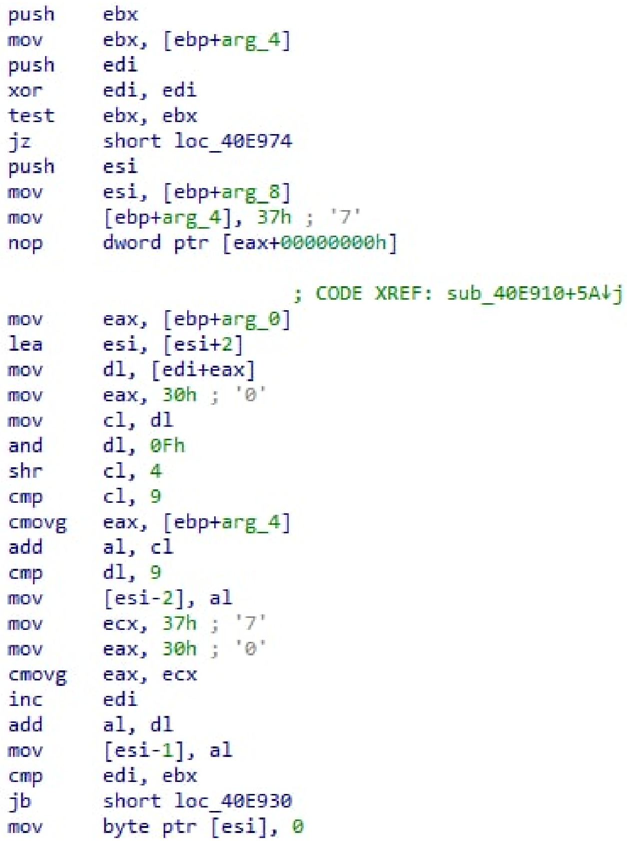
Figure 7: Evidence of privilege escalation method

Generating the Victim ID

BlueSky proceeds by generating the victim ID by hashing with MD5 the following system info:

- MachineGuid (4 Bytes)
- DigitalProductId
- InstallDate
- C:\ Serial Number

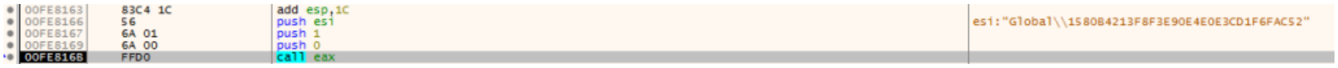
Then the hash is passed to the following custom routine:



```
push    ebx
mov     ebx, [ebp+arg_4]
push    edi
xor     edi, edi
test    ebx, ebx
jz      short loc_40E974
push    esi
mov     esi, [ebp+arg_8]
mov     [ebp+arg_4], 37h ; '7'
nop     dword ptr [eax+00000000h]

                        ; CODE XREF: sub_40E910+5A↓j
mov     eax, [ebp+arg_0]
lea     esi, [esi+2]
mov     dl, [edi+eax]
mov     eax, 30h ; '0'
mov     cl, dl
and     dl, 0Fh
shr     cl, 4
cmp     cl, 9
cmovg   eax, [ebp+arg_4]
add     al, cl
cmp     dl, 9
mov     [esi-2], al
mov     ecx, 37h ; '7'
mov     eax, 30h ; '0'
cmovg   eax, ecx
inc     edi
add     al, dl
mov     [esi-1], al
cmp     edi, ebx
jb      short loc_40E930
mov     byte ptr [esi], 0
```

Figure 8: Hash custom routine

The sample proceeds creating a mutex “Global\\{generated_id}” in this case being “Global\1580B4213F8F3E90E4E0E3CD1F6FAC52”

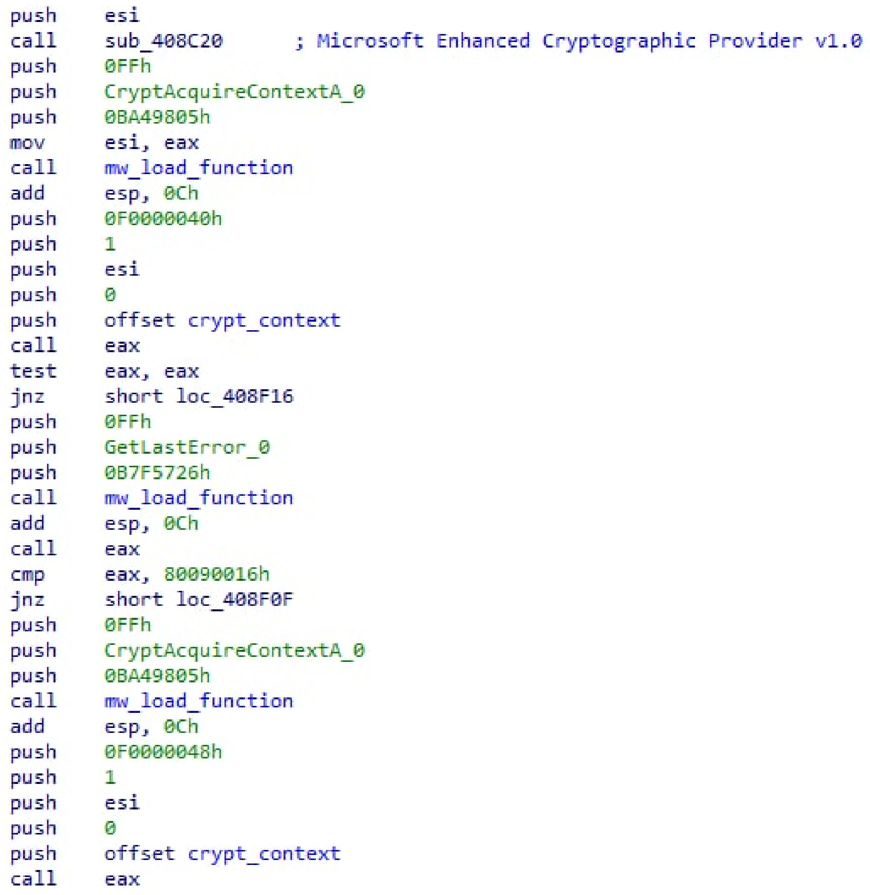


```
00FE8163  83C4 1C    add esp,1C
00FE8166  56         push esi          esi:"Global\\1580B4213F8F3E90E4E0E3CD1F6FAC52"
00FE8167  6A 01      push 1
00FE8169  6A 00      push 0
00FE816B  FFD0       call eax
```

Figure 9: Mutex Creation

The Encryption Routine

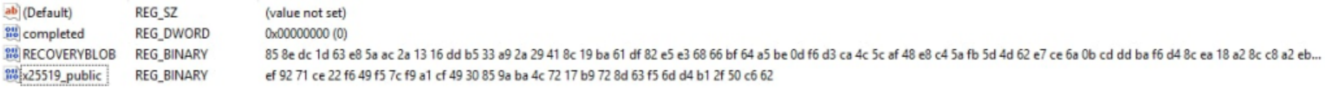
Now it’s time to encrypt the files. The first operation of the sample is to aquire a handle to the cryptographic provider **PROV_RSA_FULL** by calling **CryptAcquireContextA**:



```
push    esi
call    sub_408C20      ; Microsoft Enhanced Cryptographic Provider v1.0
push    0FFh
push    CryptAcquireContextA_0
push    0BA49805h
mov     esi, eax
call    mw_load_function
add     esp, 0Ch
push    0F0000040h
push    1
push    esi
push    0
push    offset crypt_context
call    eax
test    eax, eax
jnz     short loc_408F16
push    0FFh
push    GetLastError_0
push    0B7F5726h
call    mw_load_function
add     esp, 0Ch
call    eax
cmp     eax, 80090016h
jnz     short loc_408F0F
push    0FFh
push    CryptAcquireContextA_0
push    0BA49805h
call    mw_load_function
add     esp, 0Ch
push    0F0000048h
push    1
push    esi
push    0
push    offset crypt_context
call    eax
```

Figure 10: Acquiring a handle to PROV_RSA_FULL

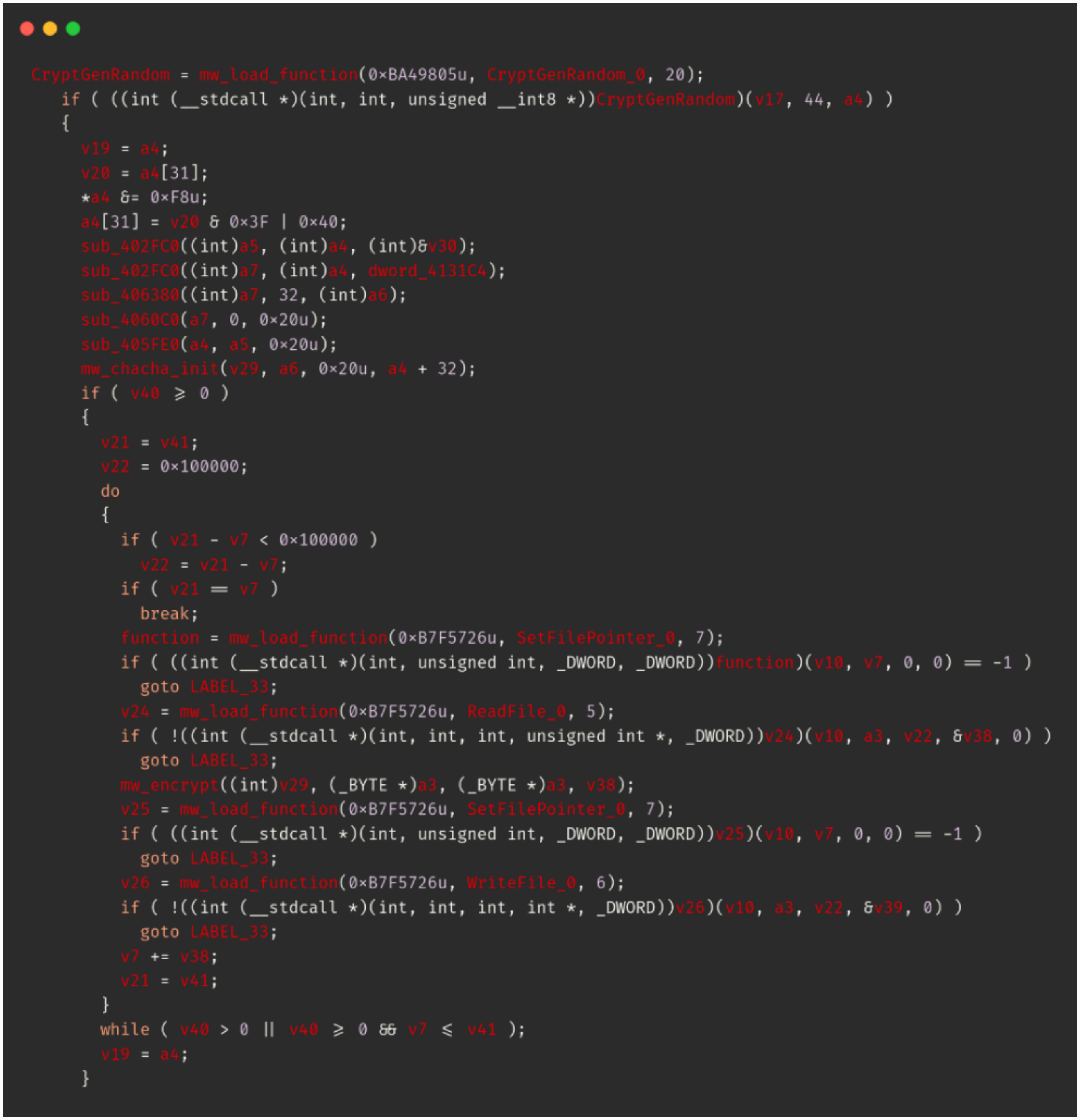
BlueSky stores the information related to the encryption, in the registry key “HKCU\SOFTWARE\1580B4213F8F3E90E4E0E3CD1F6FAC52\”. To store the recovery information, it uses “**ChaCha20 + Curve25519 + RC4 (on RECOVERYBLOB)**”, meanwhile “**ChaCha20 + Curve25519**” for the encryption



```
(Default)      REG_SZ        (value not set)
completed      REG_DWORD     0x00000000 (0)
RECOVERYBLOB   REG_BINARY    85 8e dc 1d 63 e8 5a ac 2a 13 16 dd b5 33 a9 2a 29 41 8c 19 ba 61 df 82 e5 e3 68 66 bf 64 a5 be 0d f6 d3 ca 4c 5c af 48 e8 c4 5a fb 5d 4d 62 e7 ce 6a 0b cd dd ba f6 d4 8c ea 18 a2 8c c8 a2 eb...
x25519_public  REG_BINARY    ef 92 71 ce 22 f6 49 f5 7c f9 a1 cf 49 30 85 9a ba 4c 72 17 b9 72 8d 63 f5 6d d4 b1 2f 50 c6 62
```

Figure 11: BlueSky Recovery Information

Below the encryption routine:



```
CryptGenRandom = mw_load_function(0xBA49805u, CryptGenRandom_0, 20);
   if ( ((int (__stdcall *)(int, int, unsigned __int8 *))CryptGenRandom)(v17, 44, a4) )
   {
     v19 = a4;
     v20 = a4[31];
     *a4 &= 0xF8u;
     a4[31] = v20 & 0x3F | 0x40;
     sub_402FC0((int)a5, (int)a4, (int)&v30);
     sub_402FC0((int)a7, (int)a4, dword_4131C4);
     sub_406380((int)a7, 32, (int)a6);
     sub_4060C0(a7, 0, 0x20u);
     sub_405FE0(a4, a5, 0x20u);
     mw_chacha_init(v29, a6, 0x20u, a4 + 32);
     if ( v40 >= 0 )
     {
       v21 = v41;
       v22 = 0x100000;
       do
       {
         if ( v21 - v7 < 0x100000 )
           v22 = v21 - v7;
         if ( v21 == v7 )
           break;
         function = mw_load_function(0xB7F5726u, SetFilePointer_0, 7);
         if ( ((int (__stdcall *)(int, unsigned int, _DWORD, _DWORD))function)(v10, v7, 0, 0) == -1 )
           goto LABEL_33;
         v24 = mw_load_function(0xB7F5726u, ReadFile_0, 5);
         if ( !((int (__stdcall *)(int, int, int, unsigned int *, _DWORD))v24)(v10, a3, v22, &v38, 0) )
           goto LABEL_33;
         mw_encrypt((int)v29, (_BYTE *)a3, (_BYTE *)a3, v38);
         v25 = mw_load_function(0xB7F5726u, SetFilePointer_0, 7);
         if ( ((int (__stdcall *)(int, unsigned int, _DWORD, _DWORD))v25)(v10, v7, 0, 0) == -1 )
           goto LABEL_33;
         v26 = mw_load_function(0xB7F5726u, WriteFile_0, 6);
         if ( !((int (__stdcall *)(int, int, int, int *, _DWORD))v26)(v10, a3, v22, &v39, 0) )
           goto LABEL_33;
         v7 += v38;
         v21 = v41;
       }
       while ( v40 > 0 || v40 >= 0 && v7 <= v41 );
       v19 = a4;
     }
```

Figure 12: Encryption routine

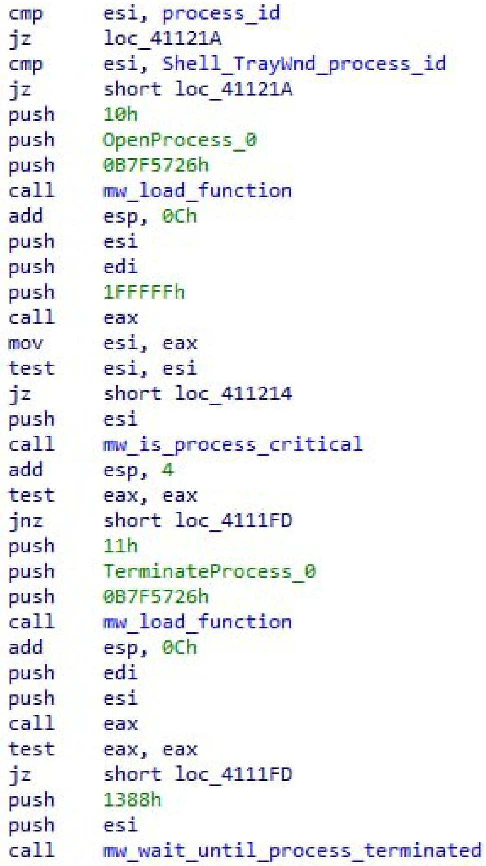
BlueSky creates a list of the **excluded files** inside the code. The list is the following:

Extensions (ldf, scr, icl, 386, cmd, ani, adv, theme, msi, rtp, diagcfg, msstyles, bin, hlp, shs, drv, wpx, bat, rom, msc, lnk, cab, spl, ps1, msu, ics, key, msp, com, sys, diagpkg, nls, diagcab, ico, lock, ocx, mpa, cur, cpl, mod, hta, exe, ini, icns, prf, dll, bluesky, nomedia, idx)

- **Directories** ($recycle.bin, $windows.~bt, $windows.~ws, boot, windows, windows.old, system volume information, perflogs, programdata, program files, program files (x86), all users, appdata, tor browser)
- **Filenames** (# decrypt files bluesky #.txt, # decrypt files bluesky #.html, ntuser.dat, iconcache.db, ntuser.dat.log, bootsect.bak, autorun.inf, bootmgr, ntldr, thumbs.db)

Exception Handling and other features

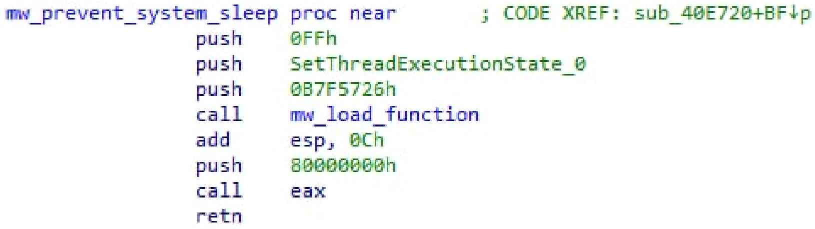
The sample implements also some interesting Exception Handling features in order to avoid the system crash. In detail, before proceeding to the encryption BlueSky checks if after calling **CreateFileW** the LastErrorValue is **ERROR_SHARING_VIOLATION**if true, the sample calls **NtQueryInformatonFile** retrieving the **FileProcessIdsUsingFileInformation** which contains a list of the PIDs which use the file. If the PID isn’t equal to itself or the PID of explorer.exe retrieved before, it calls **NtQueryInformatonProcess** with **ProcessInformationClass** set to 29 (**ProcessBreakOnTermination**) to retrieve a value indicating whether the process is considered critical. In this case, the malware skips that file and keeps encrypting others.



```
cmp     esi, process_id
jz      loc_41121A
cmp     esi, Shell_TrayWnd_process_id
jz      short loc_41121A
push    10h
push    OpenProcess_0
push    0B7F5726h
call    mw_load_function
add     esp, 0Ch
push    esi
push    edi
push    1FFFFFh
call    eax
mov     esi, eax
test    esi, esi
jz      short loc_411214
push    esi
call    mw_is_process_critical
add     esp, 4
test    eax, eax
jnz     short loc_4111FD
push    11h
push    TerminateProcess_0
push    0B7F5726h
call    mw_load_function
add     esp, 0Ch
push    edi
push    esi
call    eax
test    eax, eax
jz      short loc_4111FD
push    1388h
push    esi
call    mw_wait_until_process_terminated
```

Figure 13: Checking file availability

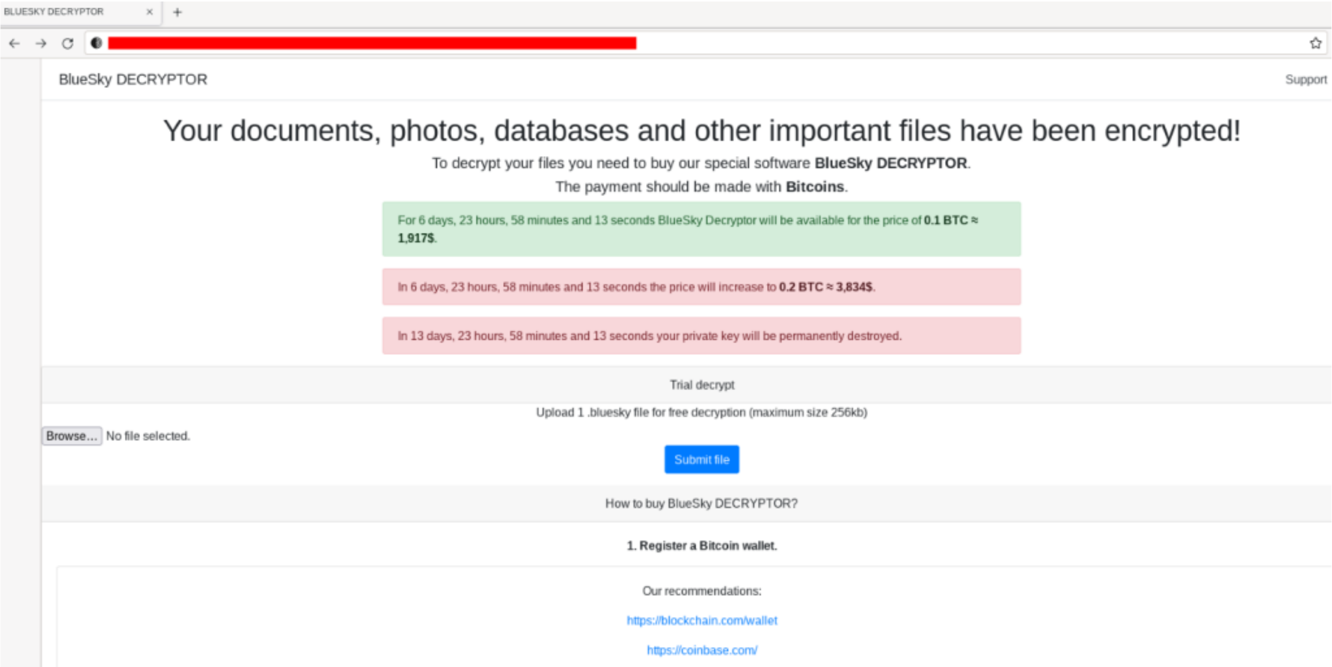
The sample can prevent the system from entering sleep or turning off the display by calling **SetThreadExecutionState** to **ES_CONTINUOUS**



```
mw_prevent_system_sleep proc near       ; CODE XREF: sub_40E720+BF↓p
                push    0FFh
                push    SetThreadExecutionState_0
                push    0B7F5726h
                call    mw_load_function
                add     esp, 0Ch
                push    80000000h
                call    eax
                retn
```

Figure 14: Preventing sleep mode

At the end of the encryption, the ransom note points to the blog of the attackers:



[redacted]

BlueSky DECRYPTOR

Your documents, photos, databases and other important files have been encrypted!

To decrypt your files you need to buy our special software **BlueSky DECRYPTOR**.

The payment should be made with **Bitcoins**.

For 6 days, 23 hours, 58 minutes and 13 seconds BlueSky Decryptor will be available for the price of **0.1 BTC ≈ 1,917$**.

In 6 days, 23 hours, 58 minutes and 13 seconds the price will increase to **0.2 BTC ≈ 3,834$**.

In 13 days, 23 hours, 58 minutes and 13 seconds your private key will be permanently destroyed.

Trial decrypt

Upload 1 .bluesky file for free decryption (maximum size 256kb)

No file selected.

How to buy BlueSky DECRYPTOR?

1. Register a Bitcoin wallet.

Our recommendations:

https://blockchain.com/wallet

https://coinbase.com/

Figure 15: BlueSky Ransomware Website

Conclusion

Blusky ransomware is a proof that even nowadays cyber criminals use basic and highly effective social engineering techniques. When we are looking for a cracked software, we have to know that there is always a price and in this case it’s a ransomware with a high ransom.

So, it is necessary to sensibilize people to avoid installing cracked software, not only inside the company perimeter, but also inside the home devices. It is a simple but effective preventive measure to defend against similar threats.

The attention for emerging threats is one of the core activities of Yoroi and we think that BlueSky needs to be observed with attention.

Yara Rules

```
rule bluesky_ransomware

{

  meta:

    author = "Yoroi Malware ZLab"

    description = "Rule for BlueSky Ransomware"

    last_updated = "2022-09-14"

    tlp = "WHITE"

    category = "informational"

    hash = "9e302bb7d1031c0b2a4ad6ec955e7d2c0ab9c0d18d56132029c4c6198b91384f"



  strings:

     //sub_00407a30

    $1 = {55 8b ec 83 ec ?? 56 e8 ?? ?? ?? ?? 85 c0 0f 84 ?? ?? ?? ?? 0f 10 05 ?? ?? ?? ?? 68 ?? ?? ??
?? 68 ?? ?? ?? ?? 0f 11 4? ?? 68 ?? ?? ?? ?? 0f 10 05 ?? ?? ?? ?? c7 4? ?? ?? ?? ?? ?? c7 4? ?? ?? ??
?? ?? 0f 11 4? ?? e8 ?? ?? ?? ?? 0f 10 4? ?? 83 c4 ?? 8b d0 8d 4? ?? 50 83 ec ?? 8b cc 6a ?? 6a ?? 83
ec ?? 0f 11 01 8b c4 0f 10 4? ?? 0f 11 00 ff d2 85 c0 0f 88 ?? ?? ?? ?? 68 ?? ?? ?? ?? 68 ?? ?? ?? ??
68 ?? ?? ?? ?? e8 ?? ?? ?? ?? 83 c4 ?? 8d 4? ?? 51 ff d0 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? 68 ?? ?? ?? ??
e8 ?? ?? ?? ?? 83 c4 ?? 8d 4? ?? 51 ff d0 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? e8 ?? ?? ?? ??
83 c4 ?? 8d 4d c8 51 ff d0 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? e8 ?? ?? ?? ?? 83 c4 ?? 8d 4?
?? 51 ff d0 0f 10 4? ?? 8b 4? ?? 83 ec ?? 8b c4 83 ec ?? 8b 11 0f 11 00 8b c4 83 ec ?? 0f 10 4? ?? 0f
11 00 8b c4 83 ec ?? 0f 10 4? ?? 0f 11 00 8b c4 0f 10 4? ?? 51 0f 11 00 ff 52 28 68 ?? ?? ?? ?? 68 ??
?? ?? ?? 68 ?? ?? ?? ?? 8b f0 e8 ?? ?? ?? ?? 83 c4 ?? 8d 4? ?? 51 ff d0 68 ?? ?? ?? ?? 68 ?? ?? ?? ??
68 ?? ?? ?? ?? e8 ?? ?? ?? ?? 83 c4 ?? 8d 4? ?? 51 ff d0 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? 68 ?? ?? ?? ??
e8 ?? ?? ?? ?? 83 c4 ?? 8d 4? ?? 51 ff d0 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? e8 ?? ?? ?? ??
83 c4 ?? 8d 4? ?? 51 ff d0 85 f6 78 ?? 8b 4? ?? 8d 5? ?? 52 68 ?? ?? ?? ?? 50 8b 08 ff 5? ?? 85 c0 78
?? 8b 4? ?? 6a ?? ff 7? ?? 8b 08 50 ff 5? ?? 8b 4? ?? 85 c9 74 ?? 8b 01 51 ff 5? ?? 8b 4? ?? 85 c9 74
?? 8b 01 51 ff 50 08 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? e8 ?? ?? ?? ?? 83 c4 ?? ff d0 5e 8b
e5 5d c3}



  condition:

    uint16(0) == 0x5A4D and $1

}
```

This blog post was authored by Luigi Martire, Carmelo Ragusa of Yoroi Malware ZLAB