Informe Código Dañino
CCN-CERT ID-12/20

Sadogo Locker

Julio 2020

Edita:

GOBIERNO DE ESPAÑA MINISTERIO DE DEFENSA

CCN centro criptológico nacional

Fecha de Edición: julio de 2020

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 BEB8D880D619CA9A3275950F03631116**, perteneciente a la familia de ransomware **Sadogo Locker**. El principal objetivo de esta muestra es cifrar los ficheros del sistema afectado para, posteriormente, solicitar el pago de un rescate en bitcoins a cambio de la herramienta de descifrado.

3. SADOGO LOCKER

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:

NOMBRE FICHERO	MD5
Desconocido	BEB8D880D619CA9A3275950F03631116

La fecha de compilación es el 28 de agosto de 2019, 06:37:59 (UTC), sin embargo, esta información no es del todo fiable, ya que se puede alterar fácilmente:

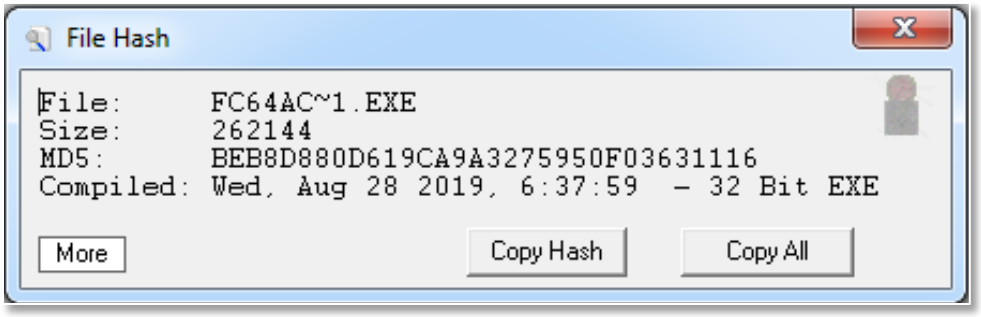


Figura 1. Fechas de compilación de la muestra.

La muestra no presenta propiedades del fichero.

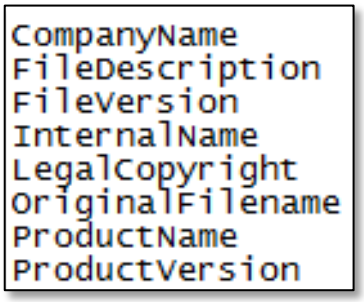


Figura 2. Las propiedades de los ficheros están vacías.

3.2 ANÁLISIS TÉCNICO

El código dañino se encuentra empaquetado, por lo que la tarea del código original es realizar el desempaquetado del verdadero código dañino. El desempaquetado se realiza mediante un shellcode, que es inyectado en su propio espacio de memoria. El shellcode extrae el código dañino sobrescribiendo la memoria del código dañino original con el código dañino desempaquetado.

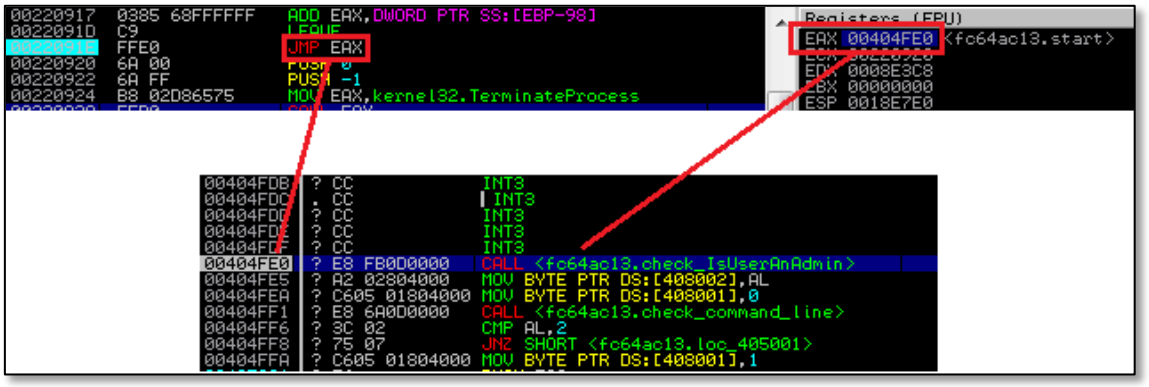


Figura 3. Salto desde el shellcode al código desempaquetado.

El código dañino utiliza AES CBC con IV nulo (todo ceros) y una clave de 256 bits para cifrar las cadenas que va a utilizar durante su ejecución. La clave utilizada es la siguiente:

AES KEY 256 BITS
406F227E19E96F8A8D73CBA599ED00E4B2C017D8378CDB7912B4B54C8C6F4534

El código dañino comienza su ejecución comprobando si se está ejecutando con privilegios de administrador. Para ello utiliza la función **IsUserAnAdmin ()** si **MajorVersion** es menor que 6 (anteriores a Vista) o comprobando si el tipo de elevación del Token de proceso es **TokenElevationTypeFull**, para versiones posteriores de Windows.

```
bool check_IsUserAnAdmin()
{
  DWORD dwVersion_; // eax
  HANDLE processHandle; // eax
  HANDLE TokenHandle; // [esp+0h] [ebp-Ch]
  int TokenInformation; // [esp+4h] [ebp-8h]
  DWORD ReturnLength; // [esp+8h] [ebp-4h]

  LOBYTE(dwVersion_) = dwVersion;
  if ( !dwVersion )
  {
    dwVersion_ = GetVersion();
    dwVersion = dwVersion_;
  }
  if ( (unsigned __int8)dwVersion_ < 6u )       // dwMajorVersion < 6
    return IsUserAnAdmin();
  processHandle = GetCurrentProcess();
  if ( !OpenProcessToken(processHandle, 0xF01FFu, &TokenHandle) )
    return 0;
  GetTokenInformation(TokenHandle, TokenElevation, &TokenInformation, 4u, &ReturnLength);
  CloseHandle(TokenHandle);
  return TokenInformation != 0;
}
```

Figura 4. Verificación de privilegios de administrador.

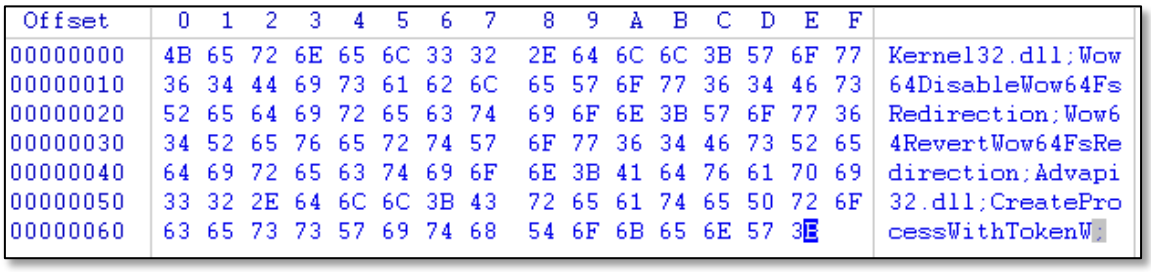
Siguiendo con la ejecución, procede a la comprobación de los argumentos de la línea de comandos. El código dañino admite un argumento con alguno de los siguientes formatos:

- nPID – Donde PID debe de ser el ID de un proceso. En este caso el código dañino buscará recursos de red remotos y cifrará los ficheros.
- ePID – Donde PID debe de ser el ID de un proceso. Aunque el código dañino comprueba si el formato del argumento es este, no tiene ninguna funcionalidad, ya que el código dañino se comporta como si no se le hubiese pasado ningún argumento.

El código dañino embebe una clave pública RSA cifrada con AES, que será utilizada para cifrar las claves AES de cifrado de los ficheros. Una vez descifrada, la clave es la siguiente:

RSA PUBLIC KEY BLOB
0602000000A400005253413100040000010001008507 3CC2EDE98B68EFCC87C1FA2A952C05B5E373278D86EB4FFD2DD23FB79AEAE90086045E3AE62D6A7C8A6AA7AC8D0CB9AE45BAA219DDC8B98EE0CE7523B7BCE9B163CCF1F64F73BD020A59072BA5DBE35466AB34FE2C04455B8056D3409C8081E4F21F1D84DCE32B8B36A23BAF690423CF0AB2B0FDD350D7927AC31F68CFDE
-----BEGIN PUBLIC KEY----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDez2gfw3qS11DT/bCyCs8jBGmvO6I2iyvj3IQdH/LkgYCcQNNWgFtFBCz+NKtmVOPbpSsHWQoCvXNP9vHMY7HpvLcjdc7gjrnI3RmiukWuuQyNrKdqinxqLeY6XgSGAOnqmrc/0i39T+uGjSdz47UFLJUq+sGHzO9oi+ntwjwHhQIDAQAB -----END PUBLIC KEY-----

Seguidamente procede a descifrar una lista de nombres de Dlls y de funciones API, que importará en el programa para utilizarlas durante su ejecución.



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	4B	65	72	6E	65	6C	33	32	2E	64	6C	6C	3B	57	6F	77	Kernel32.dll;Wow
00000010	36	34	44	69	73	61	62	6C	65	57	6F	77	36	34	46	73	64DisableWow64Fs
00000020	52	65	64	69	72	65	63	74	69	6F	6E	3B	57	6F	77	36	Redirection;Wow6
00000030	34	52	65	76	65	72	74	57	6F	77	36	34	46	73	52	65	4RevertWow64FsRe
00000040	64	69	72	65	63	74	69	6F	6E	3B	41	64	76	61	70	69	direction;Advapi
00000050	33	32	2E	64	6C	6C	3B	43	72	65	61	74	65	50	72	6F	32.dll;CreatePro
00000060	63	65	73	73	57	69	74	68	54	6F	6B	65	6E	57	3B		cessWithTokenW;

Figura 5. Lista de funciones API a importar.

```
imports = decrypt_data_by_id((int)phProv__, 26, &pdwDataLen);
imports_ = imports;
if ( imports )
{
  split_imports(imports, &v31, pdwDataLen);
  v13 = (int)v32;
  if ( v32 && v31 > 1 )
    v14 = v32[1];
  else
    v14 = 0;
  if ( v32 && v31 > 0 )
    v15 = *v32;
  else
    v15 = 0;
  v25 = v14;
  v16 = GetModuleHandleA(v15);
  Wow64DisableWow64FsRedirection = (int (__stdcall *)(_DWORD))GetProcAddress(v16, v25);
  if ( v13 && v31 > 2 )
    v17 = *(const CHAR **)(v13 + 8);
  else
    v17 = 0;
  if ( v13 && v31 > 0 )
    v18 = *(const CHAR **)v13;
  else
    v18 = 0;
  v26 = v17;
  v19 = GetModuleHandleA(v18);
  Wow64RevertWow64FsRedirection = (int (__stdcall *)(_DWORD))GetProcAddress(v19, v26);
```

Figura 6. Código de importación de las APIs.

Como otros muchos lockers, el código dañino evita infectar si la configuración del lenguaje del equipo es algún lenguaje cirílico.

```
nullsub_1();
memset_0(&locale_info, 0, sizeof(locale_info));
if ( !GetLocaleInfoW(LOCALE_SYSTEM_DEFAULT, LOCALE_FONTSIGNATURE, (LPWSTR)&locale_info, 32)
  || (result = locale_info.lsUsb[0], !(locale_info.lsUsb[0] & 0x200))
  || (locale_info.lsUsb[0] & 0x80000000) != 0 )
{
  result = prepare_folders();
```

Figura 7. Comprobación del lenguaje del equipo.

El código dañino crea el siguiente mutex: **m23071644** con la finalidad de evitar ejecuciones múltiples del código dañino. Este mutex podría utilizarse para detener la

ejecución del código dañino, ya que, si el mutex ya existe en el sistema, el código dañino finalizará su ejecución sin cifrar ningún fichero.

```
char create_mutex_m23071644()
{
  char v0; // bl
  const CHAR *m23071644; // eax
  CHAR *v2; // esi
  HANDLE v4; // eax
  DWORD pdwDataLen; // [esp+8h] [ebp-4h]

  v0 = 0;
  m23071644 = (const CHAR *)decrypt_data_by_id((int)phProv__, 29, &pdwDataLen);
  v2 = (CHAR *)m23071644;
  if ( !m23071644 )
    return 1;
  CreateMutexA(0, 1, m23071644);
  if ( GetLastError() == ERROR_ALREADY_EXISTS )
    v0 = 1;
  v4 = GetProcessHeap();
  HeapFree(v4, 0, v2);
  return v0;
}
```

Figura 8. Creación del mutex.

El código dañino instala persistencia en el sistema mediante la clave de registro **HKCU\Software\Microsoft\Windows\CurrentVersion\Run\1**, la cual apunta al ejecutable del código dañino.

```
LSTATUS __usercall set_reg_value_key@<eax>(const WCHAR *a1@<eax>, const BYTE *a2@<edi>)
{
  const WCHAR *v2; // esi
  LSTATUS result; // eax
  char *v4; // eax
  __int16 v5; // dx
  LSTATUS v6; // esi
  HKEY phkResult; // [esp+4h] [ebp-4h]

  v2 = a1;
  result = RegOpenKeyExW(HKEY_CURRENT_USER, a1, 0, 0x20106u, &phkResult);
  if ( result == 2 )
    result = RegCreateKeyExW(HKEY_CURRENT_USER, v2, 0, 0, 0, 0x20106u, 0, &phkResult, 0);
  if ( !result )
  {
    v4 = (char *)a2;
    do
    {
      v5 = *(_WORD *)v4;
      v4 += 2;
    }
    while ( v5 );
    v6 = RegSetValueExW(phkResult, L"1", 0, 1u, a2, 2 * ((v4 - (char *)a2) >> 1));
    RegCloseKey(phkResult);
    result = v6;
  }
  return result;
}
```

Figura 9. Instalación del método de persistencia.

A continuación, obtiene el valor **ProductId** de la clave de registro **HKLM\Software\Microsoft\Windows NT\CurrentVersion\ProductId** y el **Volume Serial Number** del volumen raíz del equipo, donde Windows se encuentra instalado. Ambos valores son concatenados de la siguiente forma: **ProductID-VolumeSerialNumber** y se calcula el **CRC32** de la cadena resultante. Este valor será utilizado como un identificador de la infección, y formará parte de la extensión de los ficheros cifrados (el valor hexadecimal dentro de los corchetes).

```
Windows_NT_CurrentVersion = (CHAR *)decrypt_data_by_id((int)phProv__, 16, &pdwDataLen);
ProductId = decrypt_data_by_id((int)phProv__, 17, &pdwDataLen);
v2 = (CHAR *)ProductId;
cbData = 1024;
if ( !Windows_NT_CurrentVersion
  || !ProductId
  || RegOpenKeyExA(HKEY_LOCAL_MACHINE, Windows_NT_CurrentVersion, 0, 0x20119u, &phkResult)
  || (v3 = RegQueryValueExA(phkResult, v2, 0, 0, (LPBYTE)malware_id, &cbData), RegCloseKey(phkResult), v3) )
{
  malware_id[0] = 0;
}
if ( Windows_NT_CurrentVersion )
{
  v4 = GetProcessHeap();
  HeapFree(v4, 0, Windows_NT_CurrentVersion);
}
if ( v2 )
{
  v5 = GetProcessHeap();
  HeapFree(v5, 0, v2);
}
vol_serial_number = get_SystemDrive_Volume_Serial_Number();
```

Figura 10. Cálculo del CRC32 de ProductID y Volume Serial Number.

Si el código dañino se está ejecutando con privilegios elevados, el código dañino se ejecutará a si mismo con un parámetro, compuesto por la letra **"n"** y su ID de proceso. **Ejemplo: malware.exe n1234**, donde 1234 es el ID del proceso del código dañino. Este nuevo proceso se encargará de buscar y cifrar ficheros en cualquier recurso o capeta compartida de red. Para ello intenta enumerar cualquier tipo de recurso de red.

```
DWORD __stdcall network_shares_seek_and_encrypt_Wrapper(LPVOID lpThreadParameter)
{
  unsigned int i; // esi
  DWORD dwScope; // [esp+8h] [ebp-14h]
  MACRO_RESOURCE_CONNECTED v4; // [esp+Ch] [ebp-10h]
  MACRO_RESOURCE_CONNECTED v5; // [esp+10h] [ebp-Ch]
  MACRO_RESOURCE_CONNECTED v6; // [esp+14h] [ebp-8h]
  MACRO_RESOURCE_CONNECTED v7; // [esp+18h] [ebp-4h]

  dwScope = RESOURCE_CONNECTED;
  v4 = RESOURCE_RECENT;
  v5 = RESOURCE_CONTEXT;
  v6 = RESOURCE_REMEMBERED;
  v7 = RESOURCE_GLOBALNET;
  for ( i = 0; i < 5; ++i )
    network_shares_seek_and_encrypt(*(&dwScope + i), 0, 128, (int)lpThreadParameter);
  return 0;
}
```

Figura 11. Infección de los recursos de red.

La siguiente acción es la creación de 2 mensajes de rescate, readme-warning.txt en el directorio raíz donde Windows se encuentra instalado, es decir **c:\readme-warning.txt** y también en el escritorio: **c:\Users\[USERNAME]\Desktop\readme-warning.txt**. El contenido del mensaje de rescate se ha incluido en el Apéndice I. El mensaje hace referencia a la siguiente dirección: **http://reco3zanpd2ijycv.onion/**, que está relacionada con otras muestras de **Sadogo Locker**.

```
WCHAR SystemDrive[262]; // [esp+18h] [ebp-20Ch]

read_warning_txt = decrypt_data_by_id((int)phProv__, 6, &pdwDataLen);// readme-warning.txt
YOUR_FILES_ARE_ENCRYPTED = decrypt_data_by_id((int)phProv__, 7, &pdwDataLen);// YOUR_FILES_ARE_ENCRYPTED
ransom_message = decrypt_data_by_id((int)phProv__, 8, &pdwDataLen);// ransom message
ransom_message_ = ransom_message;
if ( YOUR_FILES_ARE_ENCRYPTED && read_warning_txt && ransom_message )
{
  v4 = sub_402D50();
  if ( v4 )
    ((void (__userpurge *)(int@<eax>, int, int, int))sub_402BB0)(
      v4,
      (int)read_warning_txt,
      (int)YOUR_FILES_ARE_ENCRYPTED,
      pdwDataLen);
  else
    nullsub_1();
  SystemDrive_string = (const WCHAR *)decrypt_data_by_id((int)phProv__, 22, &v10);
  if ( SystemDrive_string )
  {
    if ( GetEnvironmentVariableW(SystemDrive_string, SystemDrive, 0x104u) )
      drop_ransom_message((__int16 *)SystemDrive, (int)ransom_message_, (int)YOUR_FILES_ARE_ENCRYPTED);
    ((void (*)(void))heap_cleaner_1)();
  }
  write_content_2_file((int)read_warning_txt, (int)pszPath_Desktop, pdwDataLen, ransom_message_);
  v6 = GetProcessHeap();
  HeapFree(v6, 0, ransom_message_);
  v7 = GetProcessHeap();
  HeapFree(v7, 0, YOUR_FILES_ARE_ENCRYPTED);
  v8 = GetProcessHeap();
  ransom_message = (BYTE *)HeapFree(v8, 0, read_warning_txt);
}
return ransom_message;
```

Figura 12. Creación de los ficheros con el mensaje de rescate.

El código dañino cifra los ficheros mediante AES CBC, utilizando una clave de 256 bytes generadas aleatoriamente. Durante su ejecución, el código dañino genera 2 claves AES aleatorias, una para ficheros mayores de 0x180000 y otra para el resto, es decir, ficheros menores de 0x180000. Estas claves son embebidas en dos bloques de datos, cifrados mediante RSA con clave pública mencionada anteriormente. Estos datos cifrados serán incluidos al final de cada fichero cifrado, como veremos más adelante. Cada bloque tiene la siguiente estructura:

Offset	Tamaño	Descripción
0x00	4 bytes	Número mágico – **0xADADEBA0**
0x04	4 bytes	CRC32(ProductID-VolumeSerialNumber)
0x08	4 bytes	VolumeSerialNumber
0x0C	4 byte	Tipo del dispositivo, obtenido mediante GetDriveTypeW
0x10	32 bytes	AES Key
0x30	4 bytes	CRC32 de los datos anteriores

El código dañino comprobará si alguno de los siguientes servicios existe en el sistema. En caso afirmativo los borrará:

sc delete vmickvpexchange	sc delete MSSQLServerADHelper100	sc delete KLIF
sc delete vmicguestinterface	sc delete MSSQLServerOLAPService	sc delete klpd
sc delete vmicshutdown	sc delete MsDtsServer100	sc delete klflt
sc delete vmicheartbeat	sc delete ReportServer	sc delete klbackupdisk
sc delete vmicrdv	sc delete "SQLTELEMETRY$HL"	sc delete klbackupflt
sc delete storflt	sc delete TMBMServer	sc delete klkbdflt
sc delete vmictimesync	sc delete "MSSQL$PROGID"	sc delete klmouflt
sc delete vmicvss	sc delete "MSSQL$WOLTERSKLUWER"	sc delete klhk
sc delete MSSQLFDLauncher	sc delete "SQLAgent$PROGID"	sc delete "KSDE1.0.0"
sc delete MSSQLSERVER	sc delete "SQLAgent$WOLTERSKLUWER"	sc delete kltap
sc delete SQLSERVERAGENT	sc delete "MSSQLFDLauncher$OPTIMA"	sc delete TmFilter
sc delete SQLBrowser	sc delete "MSSQL$OPTIMA"	sc delete TMLWCSService
sc delete SQLTELEMETRY	sc delete "SQLAgent$OPTIMA"	sc delete tmusa
sc delete MsDtsServer130	sc delete "ReportServer$OPTIMA"	sc delete TmPreFilter
sc delete SSISTELEMETRY130	sc delete "msftesql$SQLEXPRESS"	sc delete TMSmartRelayService
sc delete SQLWriter	sc delete "postgresql-x64-9.4"	sc delete TMiCRCScanService
sc delete "MSSQL$VEEAMSQL2012"	sc delete WRSVC	sc delete VSApiNt
sc delete "SQLAgent$VEEAMSQL2012"	sc delete ekrn	sc delete TmCCSF
sc delete MSSQL	sc delete klim6	sc delete tmlisten
sc delete SQLAgent	sc delete "AVP18.0.0"	sc delete TmProxy
sc delete ntrtscan	sc delete ofcservice	

El código dañino eliminará las copias shadow del sistema, mediante el siguiente comando:

vssadmin.exe Delete Shadows /All /Quiet

El código contiene una lista de **133 procesos**, que serán terminados si se encuentran ejecutándose en el sistema. La lista, por su extensión, se ha incluido en el Apéndice II.

El proceso de cifrado de ficheros es ejecutado en diferentes *threads*, de forma que el primer *thread* es el encargado de cifrar los ficheros del directorio desde donde se está ejecutando el código dañino. El código dañino comienza el cifrado en el directorio desde el que se está ejecuta y a continuación empieza a recorrer y cifrar de forma recursiva los ficheros del volumen seleccionado. El proceso de cifrado de cada fichero es el siguiente:

1. Añadir, al final del fichero, los 0x00 necesarios para alinearlo a 0x10.
2. Insertar, al final del fichero, el siguiente bloque de datos, cifrado mediante AES.

Offset	Tamaño	Descripción
0x00	4 bytes	Tamaño de los bloques cifrados
0x04	8 bytes	Tamaño del fichero / 3
0x0C	8 bytes	Tamaño del fichero
0x14	4 bytes	Tamaño del nombre del fichero en Unicode
0x18	X bytes (Tamaño del nombre del fichero en Unicode)	Nombre del fichero
0x18 + 0x0X	4 bytes	CRCR32 de los datos anteriores

3. Insertar, al final del fichero, el tamaño de los datos cifrados en el punto 2.
4. Insertar, al final del fichero, el IV.
5. Insertar, al final del fichero, el bloque de datos cifrados, mediante RSA, que contiene la clave AES (el bloque que comenzaba con el número mágico **0xADADEBA0**).
6. Insertar 8 bytes, al final del mensaje, con el offset al final del fichero original, incluido el *padding* añadido en el punto 1.
7. Añadir la extensión **".[INFECTION_ID].Encrypted"**, donde **INFECTION_ID** es el **CRC32** de "**ProductID-VolumeSerialNumber**"

```
v19 = (BYTE *)(v26 == 1 ? (*v17)->aes_key1 : (*v17)->aes_key2);
if ( import_aes_key_and_set_iv(&hKey, v19, (BYTE *)IV) )
{
  v20 = 0;
  while ( 1 )
  {
    if ( v20 == 1 )
    {
      v34.QuadPart = file_size_divided_by_3;
    }
    else if ( v20 == 2 )
    {
      v34.QuadPart = FileSize.QuadPart + (unsigned int)v32 - (unsigned __int64)(unsigned int)a3[6];
    }
    if ( !cipher_0x40000_bytes_of_data(&v34, (int)a3, h_file_, hKey, &NewFilePointer) )
      break;
    if ( ++v20 >= 3 )
    {
      CryptDestroyKey(hKey);
      hKey = 0;
      if ( SetFilePointerEx(h_file_, NewFilePointer, 0, 0) && SetEndOfFile(h_file_) )
        v27 = 1;
      break;
    }
  }
}
```

Figura 13. Cifrado de 3 bloques de 0x40000 bytes del fichero.

El código dañino cifrará los ficheros que tengan alguna de las siguientes extensiones:

EXTENSIONES DE FICHEROS
4dd;4dl;abs;abx;accdb;accdc;accde;adb;adf;ckp;db;db-journal;db-shm;db-wal;db2;db3;dbc;dbf;dbs;dbt;dbv;dcb;doc;docx;dp1;eco;edb;epim;fcd;gdb;mdb;mdf;ldf;myd;ndf;nwdb;nyf;sqlitedb;sqlite3;sqlite;xls;xlsx;xlsm;

El código dañino contiene una lista de exclusiones, de forma que no cifrará ficheros con alguno de los siguientes nombres o extensiones:

NOMBRES Y EXTENSIONES EXCLUIDOS
boot.ini, bootfont.bin, ntldr, ntdetect.com, io.sys, readme-warning.txt
Makop, CARLOS, shootlock, shootlock2, Encrypted, exe, dll

Algunas de estas extensiones que son excluidas son las que utilizan otras familias de lockers.

4. PERSISTENCIA

Esta muestra de código dañino instala persistencia en la siguiente clave de registro:

HKCU\Software\Microsoft\Windows\CurrentVersion\Run\1

apuntando al propio ejecutable del código dañino.

5. YARA

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

RAGNAR

```
import "pe"

rule sadogo_locker {
  meta:
    description = "Ransomware Sadogo Locker"
    date = "2020-07-12"
    hash1 = "BEB8D880D619CA9A3275950F03631116"

  strings:
    $s1 = { 4D 65 6D 65 72 65 79 34 44 75 7A 69 63 6F 6E 69 62 65 20 68 69 67 69 67 65 70
75 68 6F 68 69 74 61 20 77 65 76 65 66 69 20 67 75 64 75 63 61 6E 61 6C 69 72 75 63 20 67
6F 72 75 6C}

  condition:
    uint16(0) == 0x5a4d and
    filesize < 300KB and
    $s1 or
    pe.imphash() == "60503daed530fc880ba25eaedc918315"
}
```

6. IOCS

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

	IOCs
MD5	BEB8D880D619CA9A3275950F03631116
Nombre de fichero	c:\readme-warning.txt c:\Users\[USERNAME]\Desktop\readme-warning.txt
Extensión de fichero	.[HEX].Encrypted
URL	http://reco3zanpd2ijycv.onion
Mutex	m23071644 Este mutex puede ser utilizado para detener la infección, ya que, si el mutex existe en el sistema, el código dañino detiene su ejecución.

7. APÉNDICE I

El código dañino presenta el siguiente mensaje de rescate.

MENSAJE DE RESCATE
Dear user! Your computer is encrypted! To decrypt your computer, you need to download the TOR browser at https://www.torproject.org/download/ Install it and visit our website for further action **http://reco3zanpd2ijycv.onion/** Also from your servers files, documents, databases SQL, PDF were uploaded to our cloud storage After we agree, you will receive a decryption program, valuable advice in order not to fall into this situation in the future, as well as all your files on our server will be deleted. Otherwise, they will fall into the open access of the Internet!

8. APÉNDICE II

Lista de procesos que el código dañino terminará su ejecución.

LISTA DE PROCESOS	
sqlbrowser.exe	unsecapp.exe
sqlwriter.exe	TodoBackupService.exe
sqlservr.exe	MediaButtons.exe
msmdsrv.exe	IAStorDataMgrSvc.exe
MsDtsSrvr.exe	jhi_service.exe
sqlceip.exe	LMS.exe
fdlauncher.exe	DDVDataCollector.exe
Ssms.exe	DDVCollectorSvcApi.exe
SQLAGENT.EXE	TeamViewer.exe
fdhost.exe	tv_w32.exe
fdlauncher.exe	tv_x64.exe
ReportingServicesService.exe	Microsoft.Photos.exe
msftesql.exe	MicrosoftEdge.exe
pg_ctl.exe	ApplicationFrameHost.exe
postgres.exe	browser_broker.exe
UniFi.exe	MicrosoftEdgeSH.exe
armsvc.exe	MicrosoftEdgeCP.exe
IntelCpHDCPSvc.exe	RtkNGUI64.exe
OfficeClickToRun.exe	RAVBg64.exe
DellOSDService.exe	WavesSvc64.exe
DymoPnpService.exe	OneDrive.exe
Agent.exe	DYMO.DLS.Printing.Host.exe
FJTWMKSV.exe	FtLnSOP.exe
IPROSetMonitor.exe	FjtwMkup.exe
IRMTService.exe	FTPWREVT.exe
MBCloudEA.exe	FTErGuid.exe
QBCFMonitorService.exe	qbupdate.exe
QBIDPService.exe	QBWebConnector.exe
RstMwService.exe	ShellExperienceHost.exe
TeamViewer_Service.exe	RuntimeBroker.exe
dasHost.exe	IAStorIcon.exe
IntelCpHeciSvc.exe	PrivacyIconClient.exe

LISTA DE PROCESOS	
RAVBg64.exe	SupportAssistAgent.exe
vds.exe	SecurityHealthService.exe
LSCNotify.exe	taskhostw.exe
SelfServicePlugin.exe	taskhosta.exe
wfcrun32.exe	wijca.exe
HPNETW~1.EXE	ktfwswe.exe
HPScan.exe	HeciServer.exe
taskhost.exe	mdm.exe
Teams.exe	ULCDRSvr.exe
AuthManSvr.exe	WLIDSVC.EXE
WLXPhotoGallery.exe	WLIDSVCM.EXE
OUTLOOK.EXE	GoogleCrashHandler.exe
prevhost.exe	GoogleCrashHandler64.exe
EXCEL.EXE	RAVCpl64.exe
chrome.exe	igfxtray.exe
AcroRd32.exe	hkcmd.exe
RdrCEF.exe	igfxpers.exe
vssadmin.exe	PsiService_2.exe
WmiPrvSE.exe	UNS.exe
unsecapp.exe	taskeng.exe
RAVCpl64.exe	AdobeARM.exe
ScanToPCActivationApp.exe	rdpclip.exe
BrStMonW.exe	LenovoReg.exe
BrCtrlCntr.exe	LMS.exe
concentr.exe	dwm.exe
redirector.exe	taskeng.exe
BrccMCtl.exe	wuauclt.exe
BrYNSvc.exe	armsvc.exe
Receiver.exe	avp.exe
BrCcUxSys.exe	OfficeClickToRun.exe
unsecapp.exe	FBService.exe
RAVCpl64.exe	Jhi_service.exe
ScanToPCActivationApp.exe	LBAEvent.exe
BrStMonW.exe	PDFProFiltSrvPP.exe
BrCtrlCntr.exe	avpsus.exe
concentr.exe	IAStorDataMgrSvc.exe
redirector.exe	klnagent.exe
BrccMCtl.exe	vapm.exe
BrYNSvc.exe	UNS.exe
Receiver.exe	BrCcUxSys.exe