

Informe Código Dañino

CCN-CERT ID-15/21

HIVE ransomware



Diciembre 2021

Edita:



© Centro Criptológico Nacional, 2019

Fecha de Edición: diciembre de 2021

LIMITACIÓN DE RESPONSABILIDAD

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

AVISO LEGAL

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

ÍNDICE

1. SOBRE CCN-CERT.....	4
2. RESUMEN EJECUTIVO.....	5
3. DETALLES GENERALES	6
4. CARACTERÍSTICAS TÉCNICAS	8
5. CIFRADO	16
6. DESINFECCIÓN	19
7. MITIGACIÓN	19
8. REGLAS DE DETECCIÓN	20
8.1. REGLA DE YARA.....	20
9. REFERENCIAS	21
ANEXO A	22

1. SOBRE CCN-CERT

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

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

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

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

2. RESUMEN EJECUTIVO

El presente documento recoge el análisis de la muestra de código dañino perteneciente a la familia de *ransomware* apodada como Hive o HiveLeaks, identificada por la siguiente firma SHA-256:

d0ceb8f5170972fe737ab9cbdd6f3ee472fbe62e244cccc46d137094d33f1afc

El binario analizado está compilado para plataformas Windows y desarrollado en GO (x64).

El *actor* vinculado con este código dañino ha estado involucrado en múltiples incidentes desde junio de 2021 [1]. Este grupo se caracteriza por llevar a cabo ataques de forma indiscriminada contra todo tipo de industrias y sectores entre los que se encuentran proveedores de atención médica y hospitales [2]. Su *modus operandi* es similar al empleado hoy en día por otros grupos de atacantes que hacen uso de *ransomware*; una vez comprometida la organización exfiltran datos personales y proceden con el cifrado de los mismos para, posteriormente, presionar a sus víctimas mediante la, cada vez más común, doble extorsión [3].

A fecha de realización del presente informe, el portal administrado por este grupo, alcanzable desde el direccionamiento *.onion* donde se anuncian los *leaks* así como los tiempos de pago para presionar a las organizaciones, presenta un total de 52 empresas. Desde otro portal, accesible también desde la red *.onion*, las víctimas podrán autenticarse para negociar (vía chat con los propios atacantes) la compra del software que permite descifrar los ficheros afectados.

El vector de entrada utilizado por los atacantes es, generalmente, el uso de *spear phishing* con adjuntos maliciosos. Para la ejecución de sus TTPs (Tácticas Técnicas y Procedimientos) los atacantes suelen orquestar sus acciones desde herramientas de control remoto [4] tales como el archiconocido CobaltStrike así como el *software* comercial Connectwise/ScreenConnect (herramienta legítima utilizada en los últimos años por diversos grupos de atacantes [5][6]).

La elección de GO como lenguaje de programación para el *ransomware* proporciona determinadas ventajas a los atacantes por la propia naturaleza del mismo: permite *cross-compilation*, todas las dependencias son estáticamente compiladas en un único binario (generando un *.exe* de gran tamaño), facilidad y estabilidad a la hora de implementar concurrencia (por medio de *goroutines*), dificultad de hacer ingeniería inversa (principalmente en binarios que carecen de símbolos), etc.

3. DETALLES GENERALES

El fichero analizado se corresponde con un binario para arquitecturas de 64 bits, perteneciente a la familia de ransomware Hive. Su firma se muestra a continuación:

MD5	7802d6315bf0d45f27bd97fb48e70f8e
SHA1	3f0b80eb4c27e8a39768099c42c242da79cfab60
SHA256	d0ceb8f5170972fe737ab9cbdd6f3ee472fbe62e244cccc46d137094d33f1afc

El *sample* ha sido remitido, por primera vez, a la plataforma de análisis de malware VirusTotal [7] el 27 de noviembre de 2021 desde España presentando una tasa de detección de 33 motores AV de un total de 66. Otras muestras con características similares [8], vinculadas también con *hive*, han sido también remitidas a VirusTotal durante el mes de noviembre y diciembre de 2021.



Figura 1. VirusTotal: tasa de detección

El fichero tiene un tamaño de 2,68 MB (2.819.584 bytes) y se ha desarrollado en GO para arquitecturas de 64 bits. La siguiente imagen muestra las propiedades estáticas del mismo:

```

signature/type: PE64_EXE_image for amd64
image checksum: 0x00000000 (calc=0x00280851)
machine: 0x8664 (amd64)
subsystem: 2 (Windows GUI)
minimum os: 6.1 (Win7)
linkver: 3.0
file alignment: 0x200
section alignment: 0x1000
preferred load base: 0x00000000400000
code entrypoint: 0x0222 (EXECUTABLE_IMAGE|LARGE_ADDRESS_AWARE|DEBUG_STRIPPED)
characteristics: 0x8160 (DYNAMIC_BASE|NX_COMPAT|TERMINAL_SERVER_AWARE|HIGH_ENTROPY_VA)
DLL characteristics: <none>
debug directory: <none>
6 PE sections: .text, .rdata, .data, .idata, .reloc, .symtab
import modules: kernel32.dll
delay-import modules: <none>
export functions: <none>
version resource: <none>
load config: <none>
PE base relocations: 21794
tls directory: <none>
CLR (.NET) info: not a .NET module

data directory table has 3 entries (room for 16):

DIRECTORY      RVA      SIZE      MEM-PTR F-OFFSET POINTS-TO
-----
IMPORT          303000  00048C  000000000703000 002A4E00 .idata
BASE_RELOC     304000  00AE74  000000000704000 002A5400 .reloc
IAT            289020  000148  000000000689020 00288220 .data

section memory map / 6 entries:

SECTION  MEMORY-RANGE      SIZE  DSIZE  FILE-RANGE      SIZE  ATTR
-----
HEADERS  0000000000400000  0000000000401000  00001000 00000278  00000000-00000600  00000600  C E R
.text    0000000000401000  0000000000574000  00173000 00172EB0  00000600-00173600  00173000  I R
.rdata   0000000000574000  0000000000689000  00115000 00114AF8  00173600-00288200  00114C00  I R
.data    0000000000689000  0000000000703000  0007A000 00079420  00288200-002A4E00  0001CC00  I R W
.idata   0000000000703000  0000000000704000  00001000 0000048C  002A4E00-002A5400  00000600  I R W
.reloc   0000000000704000  000000000070F000  0000B000 0000AE74  002A5400-002B0400  0000B000  I D R
.symtab  000000000070F000  0000000000710000  00001000 00000004  002B0400-002B0600  00000200  D R

```

Figura 2. Propiedades estáticas PE

```

1 rule MALWARE_Win_Hive {
2   meta:
3     author = "ditekShen"
4     description = "Detects Hive ransomware"
5   strings:
6     $url1 = "http://hivaseekb" ascii
7     $url2 = "http://hivaseekb" ascii
8     $s1 = "encrypt_files.go" ascii
9     $s2 = "erase_key.go" ascii
10    $s3 = "kill_processes.go" ascii
11    $s4 = "remove_shadow_copies.go" ascii
12    $s5 = "stop_services_windows.go" ascii
13    $s6 = "remove_itself_windows.go" ascii
14    $x1 = "/encryptor/" ascii
15    $x2 = "HOW_TO_DECRYPT.txt" ascii
16    $x3 = "FilesEncrypted" fullword ascii
17    $x4 = "EncryptionStarted" fullword ascii
18    $x5 = "encryptFilesGroup" fullword ascii
19    $x6 = "Your data will be undecryptable" ascii
20    $x7 = "- Do not fool yourself. Encryption has perfect secrecy" ascii
21    $v1_1 = ".EncryptFiles." ascii
22    $v1_2 = ".EncryptFileName." ascii
23    $v1_3 = ")*struct ( F uintptr; autotmp_14 string )" ascii
24    $v1_4 = ")*struct ( F uintptr; data *[]uint8; seed *uint8; fnc *main.decFunc )" ascii
25    $v1_5 = "golang.org/x/sys/windows.getSystemWindowsDirectory" ascii
26    $v1_6 = "path/filepath.WalkDir" ascii
27  condition:
28    uint16(0) == 0x54ed and (all of ($url1) or all of ($s*) or 4 of ($x*) or 5 of ($v1*))
29 }

```



rdata:00000000005FC1	!goLang.org/x/sys/windows.CloseExec_0	rdata:00000000005FF01	!crypto.rsa.checkPub_0	!crypto.rsa.checkPub_0
rdata:00000000005FC2	!goLang.org/x/sys/windows.CloseServiceHandle_0	rdata:00000000005FF04	!io.ReadFull_0	!io.ReadFull_0
rdata:00000000005FC5	!goLang.org/x/sys/windows.("LazyProc").Addr_0	rdata:00000000005FF01	!crypto.rsa.PublicBd_0	!crypto.rsa.("PublicKey").Size_0
rdata:00000000005FC8	!goLang.org/x/sys/windows.ermoErr_0	rdata:00000000005FF02E	!math/big.("Int").BitLen_0	!math/big.("Int").BitLen_0
rdata:00000000005FCAC	!goLang.org/x/sys/windows.EnumServicesStatusEx_0	rdata:00000000005FF045	!math/big.("Int").SetBytes_0	!math/big.("Int").SetBytes_0
rdata:00000000005FCDD	!goLang.org/x/sys/windows.OpenSManager_0	rdata:00000000005FF05E	!crypto.rsa.Init_0	!crypto.rsa.Init_0
rdata:00000000005F004	!goLang.org/x/sys/windows.CloseHandle_0	rdata:00000000005FF06E	!crypto.sha512.Init_0	!crypto.sha512.Init_0
rdata:00000000005F005	!goLang.org/x/sys/windows.CloseHandle_0	rdata:00000000005FF083	!crypto.rsa.PublicBd_0	!crypto.rsa.("PublicKey").Size_0
rdata:00000000005F006	!goLang.org/x/sys/windows.CreateToolhelp32Snapshot_0	rdata:00000000005FF097	!crypto.sha512.Dig_0	!crypto.sha512.("digest").Reset_0
rdata:00000000005F002	!goLang.org/x/sys/windows.GetDriveType_0	rdata:00000000005FF085	!crypto.sha512.New_0	!crypto.sha512.New_0
rdata:00000000005F00A8	!goLang.org/x/sys/windows.GetLogicalDrives_0	rdata:00000000005FF0C7	!crypto.sha512.New512_224_0	!crypto/sha512.New512_224_0
rdata:00000000005F00D2	!goLang.org/x/sys/windows.GetStdHandle_0	rdata:00000000005FF0E0	!crypto.sha512.New512_1_0	!crypto/sha512.New512_256_0
rdata:00000000005F00F8	!goLang.org/x/sys/windows.GetSystemDirectory_0	rdata:00000000005FF0F9	!crypto.sha512.New384_0	!crypto/sha512.New384_0
rdata:00000000005F0E2	!goLang.org/x/sys/windows.GetSystemWindowsDirectory_0	rdata:00000000005FF10A	!crypto.rsa.PublicBd_0	!crypto.rsa.("PublicKey").Size_0
rdata:00000000005F0E5	!goLang.org/x/sys/windows.SysFile_0	rdata:00000000005FF128	!crypto.sha512.("digest").Write_0	!crypto/sha512.("digest").Write_0
rdata:00000000005F0E7E	!goLang.org/x/sys/windows_LoadLibraryEx_0	rdata:00000000005FF149	!crypto.sha512.Dig_0	!crypto/sha512.("digest").Sum_0
rdata:00000000005F0EA6	!goLang.org/x/sys/windows.OpenProcess_0	rdata:00000000005FF165	!crypto.sha512.Dig_0	!crypto/sha512.("digest").Checksum_0
rdata:00000000005F0EC8	!goLang.org/x/sys/windows.Process32First_0	rdata:00000000005FF186	!encoding.Binary.BigEndian.PutInt64_0	!encoding.Binary.BigEndian.PutInt64_0
rdata:00000000005F0EF3	!goLang.org/x/sys/windows.Process32Next_0	rdata:00000000005FF1A4	!crypto.sha512.Dig_0	!crypto/sha512.Sum512_256_0
rdata:00000000005F0F1A	!goLang.org/x/sys/windows.SetFileAttributes_0	rdata:00000000005FF1B3	!crypto.sha512.Dig_0	!crypto/sha512.Block_0
rdata:00000000005F0F4D	!goLang.org/x/sys/windows.WaitForSingleObject_0	rdata:00000000005FF1D7	!crypto.rsa.PublicBd_0	!crypto.rsa.("PublicKey").Size_0
rdata:00000000005F0F73	!goLang.org/x/sys/windows.TerminateProcess_0	rdata:00000000005FF1E9	!crypto.sha512.Dig_0	!crypto/sha512.BlockAW64_0
rdata:00000000005F0F90	!goLang.org/x/sys/windows.WaitForSingleObject_0	rdata:00000000005FF203	!crypto.sha512.Dig_0	!crypto/sha512.BlockAW2_0

USO OFICIAL

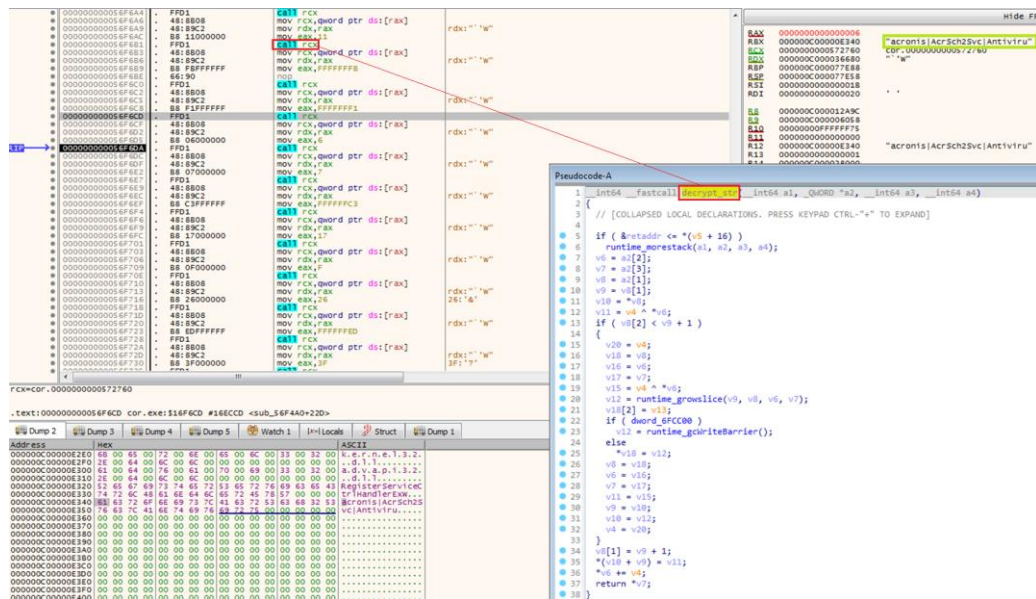
4. CARACTERÍSTICAS TÉCNICAS

El código dañino está desarrollado en GO 1.16 según la información que refleja su estructura *moduleData* [10] (tabla que recoge información relevante sobre la estructura del binario). Tras identificar su *pcln table* [11] por medio de su *magic number* es posible renombrar automáticamente el nombre de la funciones desde IDA, facilitando la ingeniería inversa.

Function name	Segment	Start	Length	Function name	Segment	Start	Length
sub_401000	.text	0000000000401000	0000004D	internal_poll_ptr_FD_PreadAutogen_EHUKL	.text	0000000000494E40	0000004C
sub_401060	.text	0000000000401060	00000016	internal_poll_ptr_FD_PreadAutogen_4OPNG6	.text	0000000000494E40	0000004C
sub_4016A0	.text	00000000004016A0	00000045	internal_poll_ptr_FD_Write	.text	0000000000494F00	00000047
sub_401B20	.text	0000000000401B20	00000018	internal_poll_ptr_FD_WriteA	.text	00000000004953C0	0000004C
sub_401B40	.text	0000000000401B40	00000011	internal_poll_ptr_FD_WriteAutogen_3GBX5Z	.text	0000000000495420	0000004C
sub_401B60	.text	0000000000401B60	00000078	internal_poll_ptr_FD_WriteConsole	.text	0000000000495480	00000055
sub_401C00	.text	0000000000401C00	00000085	internal_poll_ptr_FD_Pwrite	.text	0000000000495400	0000004E
sub_401CA0	.text	0000000000401CA0	0000006F	internal_poll_ptr_FD_PwriteAutogen_4ZCICOR	.text	0000000000495520	00000050
sub_401D20	.text	0000000000401D20	00000027	internal_poll_ptr_FD_PwriteAutogen_ZKTPTR	.text	0000000000495F00	0000004C
sub_401FE0	.text	0000000000401FE0	00000045	internal_poll_ptr_FD_FindNextFile	.text	0000000000496040	00000057
sub_402240	.text	0000000000402240	00000029	internal_poll_ptr_FD_FindNextFileA	.text	00000000004961C0	0000004C
sub_402280	.text	0000000000402280	00000022	internal_poll_ptr_FD_GetFileType	.text	0000000000496220	00000075
sub_402500	.text	0000000000402500	00000028	internal_poll_ptr_FD_GetFileTypeA	.text	00000000004963A0	0000004C
sub_402520	.text	0000000000402520	00000172	internal_poll_ptr_FD_Read_func1	.text	0000000000496400	00000052
sub_402820	.text	0000000000402820	00000019	internal_poll_ptr_FD_Write_func1	.text	0000000000496460	0000004D
sub_402840	.text	0000000000402840	0000015A	internal_poll_init	.text	00000000004964C0	00000108
sub_402860	.text	0000000000402860	00000024	internal_poll_errNetClosing_Error	.text	00000000004965E0	0000004C
sub_4028B0	.text	00000000004028B0	00000021	os_ptr_File_ReadDir	.text	0000000000496640	00000086
sub_402BA0	.text	0000000000402BA0	00000021	os_ptr_File_ReadDir	.text	00000000004966E0	0000007F
sub_402D00	.text	0000000000402D00	0000012F	os_dirEntry_Name	.text	0000000000496F00	00000008
sub_402D20	.text	0000000000402D20	00000123	os_dirEntry_IsDir	.text	0000000000496F20	0000002D
sub_402F80	.text	0000000000402F80	000000D3	os_dirEntry_Type	.text	0000000000496F60	00000083
sub_403080	.text	0000000000403080	000000D5	os_ptr_SyscallError_Error	.text	0000000000497000	00000066
sub_403180	.text	0000000000403180	00000052	os_underlyingErrors	.text	0000000000497080	00000205
sub_4031E0	.text	00000000004031E0	00000052	os_init0	.text	00000000004972A0	000000E7
sub_403240	.text	0000000000403240	000000F5	os_readNextArg	.text	00000000004973A0	0000041F
sub_403360	.text	0000000000403360	000000F5	os_commonLineToArgv	.text	0000000000497FE0	00000145
sub_403480	.text	0000000000403480	000002D2	os_executable	.text	00000000004979A0	0000002A
sub_403780	.text	0000000000403780	00000006	os_getModuleFileName	.text	0000000000497980	000000A5
sub_4037A0	.text	00000000004037A0	00000009	os_ptr_File_Name	.text	0000000000497A40	00000014
sub_4037C0	.text	00000000004037C0	0000000A	os_ptr_LinkError_Error	.text	0000000000497A60	0000010E
sub_4037E0	.text	00000000004037E0	00000008	os_ptr_File_Read	.text	0000000000497B80	000001C5
sub_403800	.text	0000000000403800	0000000A	os_ptr_File_ReadAt	.text	0000000000497D60	00000305
sub_403820	.text	0000000000403820	0000001A	os_ptr_File_Write	.text	00000000004980A0	000001F5
sub_403840	.text	0000000000403840	00000016	os_ptr_File_WriteAt	.text	00000000004982C0	00000325
sub_403860	.text	0000000000403860	00000017				
sub_403880	.text	0000000000403880	00000037				
sub_403900	.text	0000000000403900	00000039				
sub_403920	.text	0000000000403920	0000004B				
sub_403940	.text	0000000000403940	0000004D				
sub_403960	.text	0000000000403960	0000004D				
sub_403980	.text	0000000000403980	0000004D				
sub_403A20	.text	0000000000403A20	000000AE				

Figura 5. Reconstrucción nombre de funciones (*pcln table*)

El código dañino comienza reservando memoria donde decodificará algunas cadenas e información que requerirá para el proceso de infección.



The screenshot displays the IDA Pro interface. The main window shows assembly code for a function, with instructions like `mov rcx, qword ptr ds:[rax]` and `mov rax, qword ptr ds:[rax]`. The comments in Spanish describe the operations. A pseudocode window is open on the right, showing the function `runtime_growstack` which grows a stack and returns a pointer. The pseudocode includes comments in Spanish and a return statement.

Figura 6. Decodificación strings

Tras finalizar dicho proceso de *desofuscación* se encontrarán en claro: el listado de procesos y servicios a finalizar, mensajes de *logging*, *regex* con el tipo de ficheros a excluir del proceso de cifrado, contenido de los .txt que dejará en cada directorio, etc.

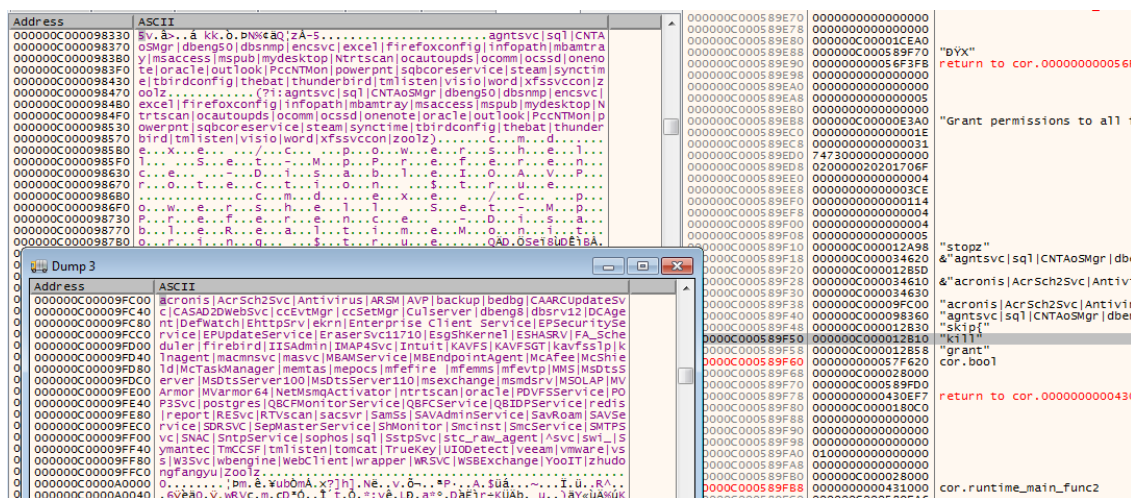


Figura 7. Cadenas en claro

El código daño escribirá en disco la clave de cifrado (cifrada con múltiples claves RSA) bajo el nombre **<random_base64>.key.<5_character>** (en la muestra actual *"fayg2"*). Posteriormente, finalizará un conjunto de servicios y procesos y empezará a recorrer el sistema de ficheros para cifrar los archivos correspondientes.

De forma opcional, el *ransomware* acepta como parámetro de entrada el *path* del directorio que se desea cifrar (de forma recursiva) así como las siguientes opciones:

- **stop**: para indicar los servicios que se quieren finalizar. Si se especifican varios servicios tendrán que indicarse con el separador "|". Por ejemplo: *"vboxservice|wscsv"*.
- **kill**: para indicar los procesos que se desean finalizar. Si se especifican varios procesos tendrán que indicarse con el separador "|". Por ejemplo: *"baretail|putty"*.

Si no se especifica ningún argumento se cifrará la unidad del sistema omitiendo el directorio *c:\Windows*. En la siguiente imagen se muestra, a modo de ejemplo, el cifrado del directorio *c:\test* además de finalizar el proceso *"baretail"* y de parar el servicio *"vboxservice"*.

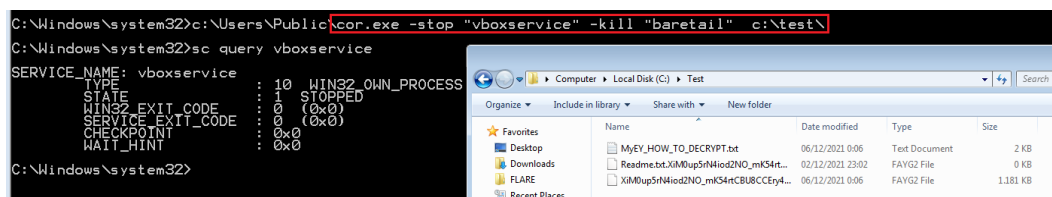


Figura 8. Cifrado directorio Test

Nota: Si el binario no puede crear en disco el fichero con la *key* no se llevará a cabo el cifrado de ficheros y se finalizará su ejecución.

Todo el proceso de infección está dividido en una serie de etapas que el *ransomware* irá reflejando a modo de *log*. La siguiente imagen muestra la función encargada de orquestar cada una de estas fases:

```

1  int64 __fastcall Bq1fecMR_ptr_DHq6niZq_RunProcess(__int64 a1, __int64 a2, __int64 a3, __int64 a4)
2  {
3      // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5      if ( &retaddr <= *(v4 + 16) )
6          runtime_morestack_noctxt(a1, a2, a3, a4);
7      result = Bq1fecMR_ptr_DHq6niZq_Init(a1, a2, a3, a4);
8      if ( !result )
9      {
10         result = Bq1fecMR_ptr_DHq6niZq_ExportKey(v7, v6, v8, v9);
11         if ( !result )
12         {
13             Bq1fecMR_ptr_DHq6niZq_Preprocess();           // StopService / KillProcesses
14             Bq1fecMR_ptr_DHq6niZq_PreNotify();
15             Bq1fecMR_ptr_DHq6niZq_ScanFiles(v11, v10, v12, v13);
16             Bq1fecMR_ptr_DHq6niZq_EncryptFiles();
17             Bq1fecMR_ptr_DHq6niZq_EraseKey();
18             Bq1fecMR_ptr_DHq6niZq_Notify();
19             Bq1fecMR_ptr_DHq6niZq_Postprocess();
20             Bq1fecMR_ptr_DHq6niZq_EraseMemory();
21         }
22     }
23     return result;
24 }
  
```

Figura 9. Fases de infección

En el *sample* analizado dicho proceso de *logging* se lleva a cabo mediante la API *WriteFile* (con un *handler* a *null*) aunque en otras versiones [2] está información es mostrada por *stdout*. Los mensajes que reflejan cada una de estas fases, acompañados de su hora, se muestran a continuación:

```

00:52:34 Exporting key
00:53:27 +export c:\<random>_key.fayg2
00:53:51 Stopping services
00:55:00 Removing shadow copies
00:55:39 Killing processes
00:56:00 Scanning files
00:56:30 Encrypting files
00:57:22 %encrypt <file_path>
00:57:22 +encrypt <file_path>
....
....
00:58:34 Removing itself
00:59:12 Erasing memory
  
```

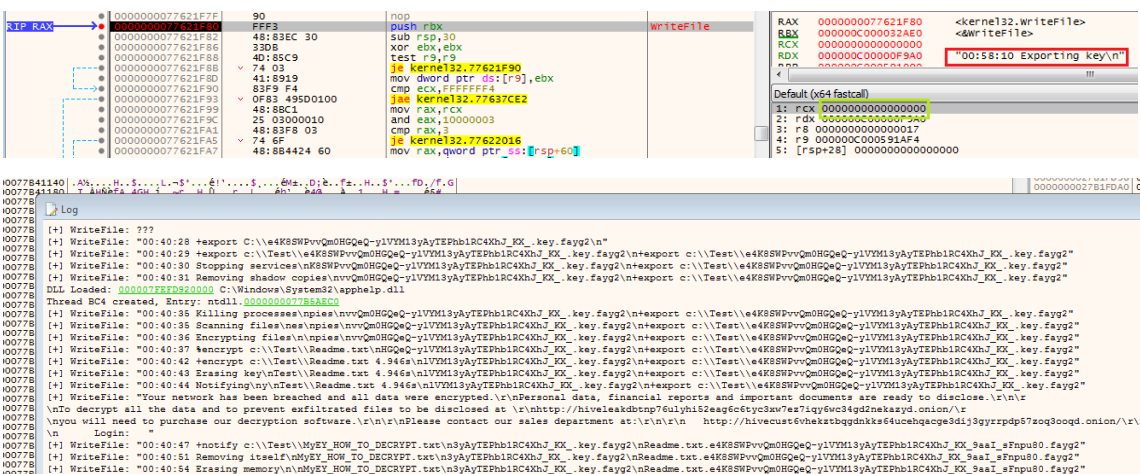


Figura 10. Fases proceso de cifrado (Ejemplo: c:\Test)

En la fase denominada como “*exporting key*” se creará una clave de 0x100000 bytes que participará en el proceso de cifrado de los ficheros. Esta clave será cifrada por medio de diversas claves RSA embebidas en el binario y, posteriormente, será guardada en disco con la extensión *.key.fayg2*. Esta clave será necesaria para la recuperación de los ficheros afectados por medio del software de descifrado que la víctima tendrá que comprar al grupo criminal.

Tras exportar la *key* el código dañino comenzará a finalizar un conjunto de procesos y servicios (véase [Anexo A](#)) que puedan interferir con el proceso de infección y de recuperación de los ficheros. La función encargada de buscar y finalizar procesos hará uso de las habituales API *CreateToolhelp32Snapshot*, *Process32First* junto a *TerminateProcess* (para aquellos cuyo nombre coincida con el listado de procesos embebido en el binario). En el caso de los servicios, éstos serán finalizados por medio del binario *net.exe* (“*net stop <nombre_servicio>*”).

```

73  v11 = runtime_convTstring(v8, v7, v9, v10);
74  *&v63 = &stru_5808E0;
75  *(&v63 + 1) = v11;
76  log_Printfln(1i64);
77  Toolhelp32Snapshot = golang_org_x_sys_windows_CreateToolhelp32Snapshot(v13, v12, v14, v15);
78  v57[0] = Bq1fecMR_ptr_DHq6niZq_KillProcesses_;
79  v57[1] = Toolhelp32Snapshot;
80  v66 = v57;
81  v58 = 0i64;
82  (loc_454365)();
83  LODWORD(v58) = 568;
84  if ( !golang_org_x_sys_windows_Process32First(v17, v16, v18, v19) )
85  {
86  do
87  {
88  v60 = golang_org_x_sys_windows_UTF16ToString(260i64, v20, v21, v22);
89  if ( regexp_ptr_Regexp_doExecute(0, v67, 0, v60, v45, v46, v47, v48, v49, v50, v51, v52, v53) )
90  {
91  v28 = v59;
92  if ( Bq1fecMR_ptr_DHq6niZq_KillProcess() )
93  {
94  v62 = (&qword_56C640[328])();
95  v56 = v28;
96  v63 = v6;

```

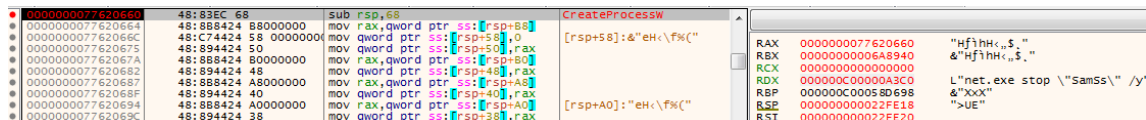


Figura 11. Finalización de procesos / servicios

Posteriormente, se deshabilitará Windows Defender haciendo cambios en determinadas entradas de registro por medio del binario *reg.exe* y deshabilitando las tareas programadas correspondientes vía *schtasks.exe*. Las instrucciones coinciden con ciertos scripts en *batch* disponibles en repositorios públicos [12] y que son comúnmente reutilizados por los atacantes para deshabilitar Windows Defender.

Figura 12. Deshabilitar Windows Defender

12

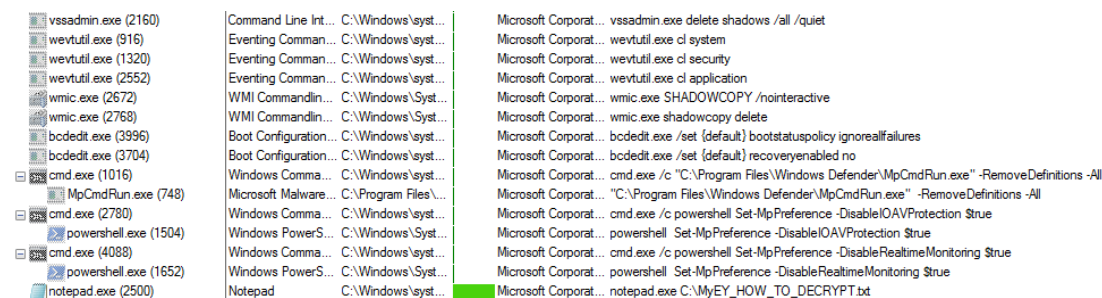


Figura 13. Deshabilitar Shadow Copies / Microsoft Malware Protection

Una vez deshabilitados los servicios anteriores, el *ransomware* comenzará con el proceso de cifrado. Para recorrer el sistema de ficheros se apoyará en la función *WalkDir (/fs/walk.go)*. Esta API acepta, como parámetro, una función que será invocada por cada fichero, a modo de *callback*. Desde esta función se comprobará si el fichero es candidato a ser cifrado por medio de una expresión regular (*Regexp.doExecute*), a modo de filtro, el cual incluye un listado de extensiones y nombres de ficheros y directorios. Si el fichero no hace *match* con dicha *regex* el archivo será cifrado.

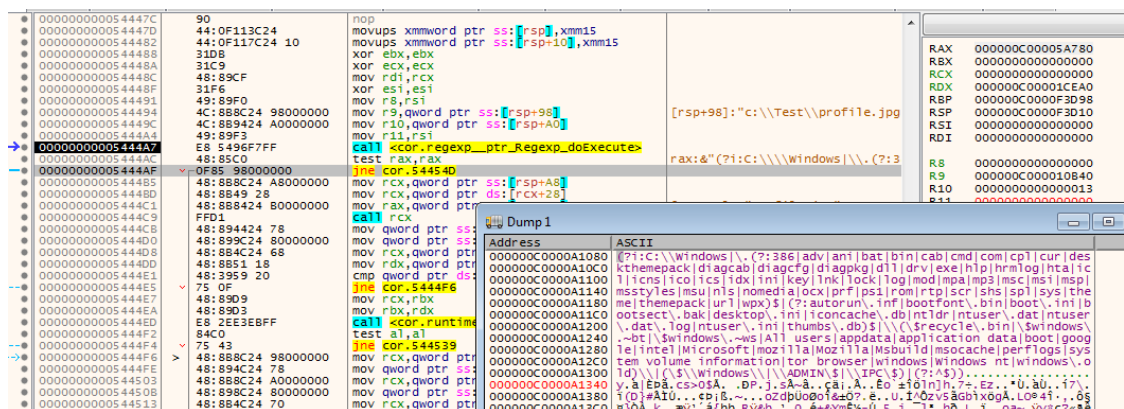


Figura 14. Filtrado de ficheros (regex)

El proceso anterior se repetirá hasta finalizar el cifrado de ficheros en todo el equipo (o en el directorio pasado como argumento al binario). Obsérvese en la *regex* de la imagen anterior que tanto el directorio *c:\Windows*, así como otro tipo de ficheros, serán excluidos del proceso de cifrado.

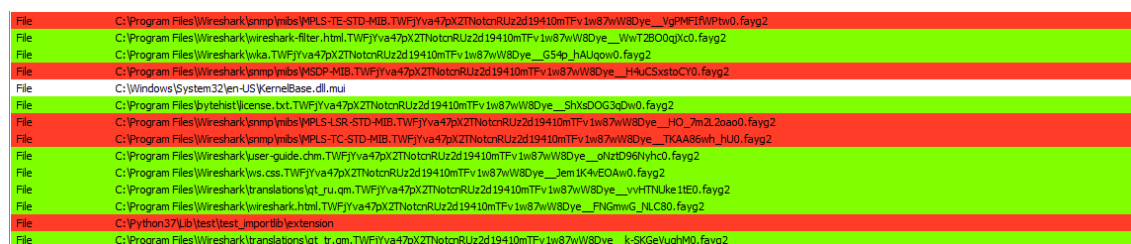


Figura 15. Cifrado de los ficheros

Por último, el *ransomware* sobrescribirá la clave de cifrado en memoria (con 0xFF), eliminará el binario de disco y limpiará parte de la memoria para eliminar todo tipo de evidencias.

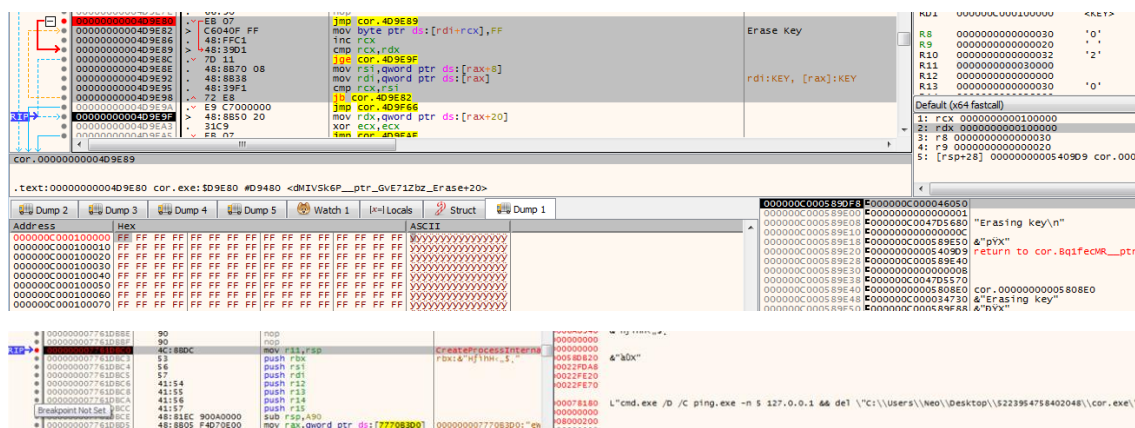


Figura 16. Elimina clave en memoria / Elimina binario de disco

Durante el proceso de cifrado, en cada directorio afectado, el ransomware dejará un archivo de texto (*MyEY_HOW_TO_DECRYPT.txt*) con las instrucciones correspondientes para la recuperación de los mismos.

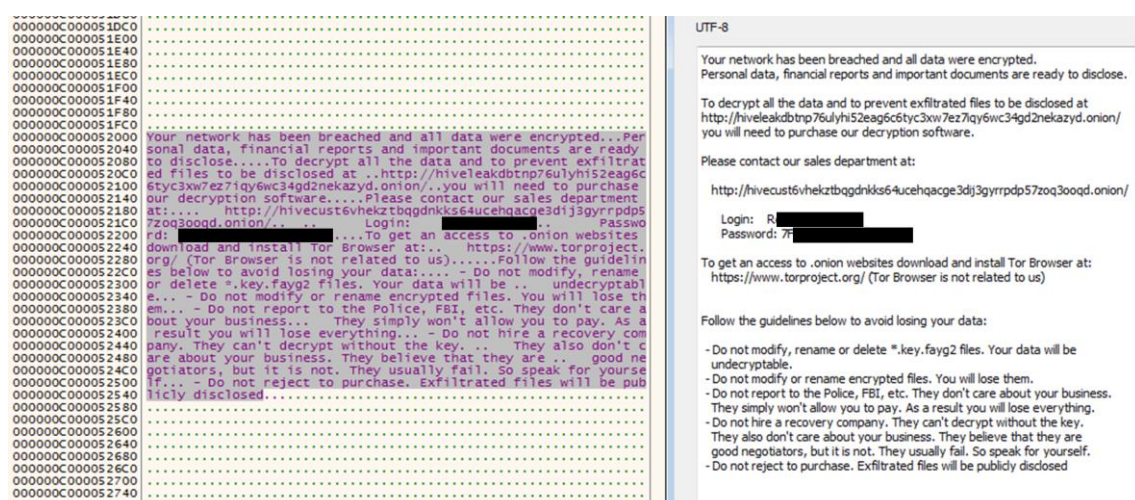


Figura 17. Contenido fichero MyEY_HOW_TO_DECRYPT.txt

En dichas instrucciones se indica, por un lado, la URL dentro del direccionamiento .onion donde se filtrará la información si no se efectúa el pago del software de descifrado:

<http://hiveleakdbtnp76ulyhi52eag6c6tyc3xw7ez7iqy6wc34gd2nekazyd.onion>

Nota: a fecha de la realización del informe, dicho portal consta de un total de 53 compañías. El primer afectado por Hive es del 23 junio de 2021 y el último el 3 de diciembre de 2021. Un total de 20 compañías cuyo pago ha vencido tienen actualmente expuestos sus ficheros por medio de enlaces de descarga desde servicios como *send.exploit.in*, *ufile.io*, *sendspace.com*, *anonfiles.com* y *mega.nz*.

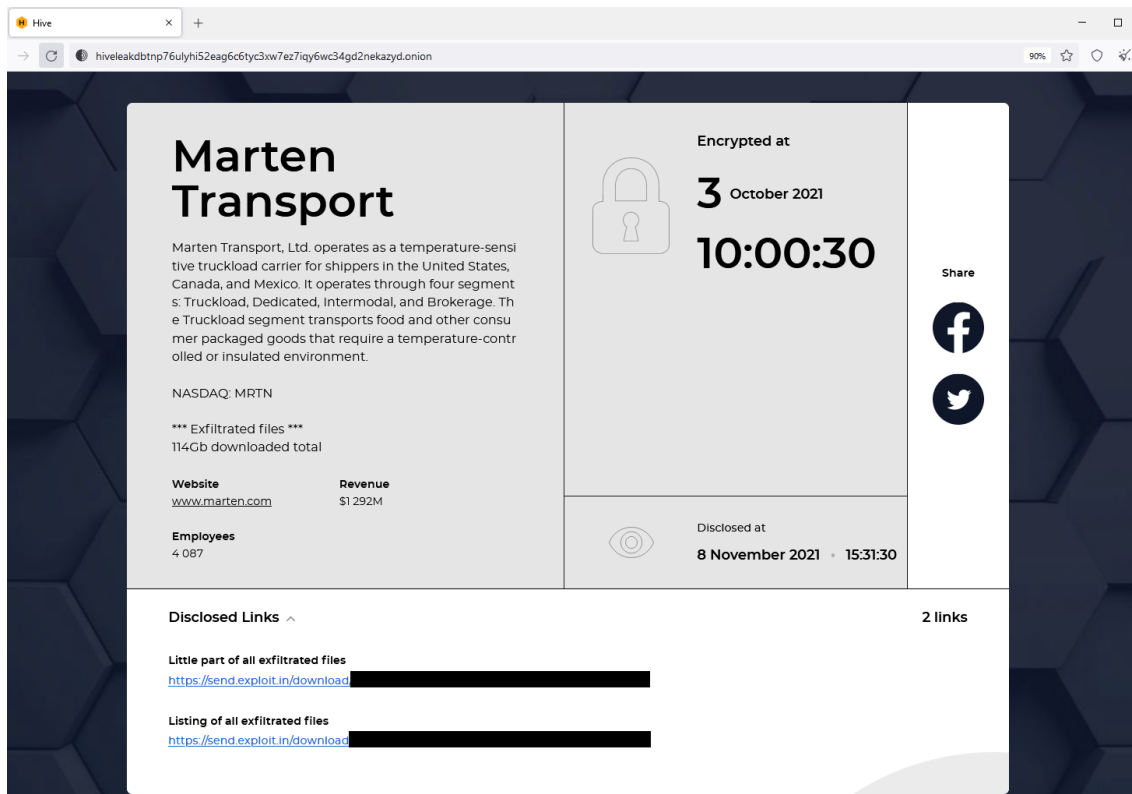


Figura 18. Portal Hive: leak información

Por otro lado, el fichero de texto también especificará las credenciales necesarias para autenticarse al portal desde el cual se negociará el pago del software de descifrado (por medio de un *Live Chat*):

<http://hivecust6vhekztbqgdnkks64ucehqacge3dij3gyrrdpdp57zoq3ooqd.onion>

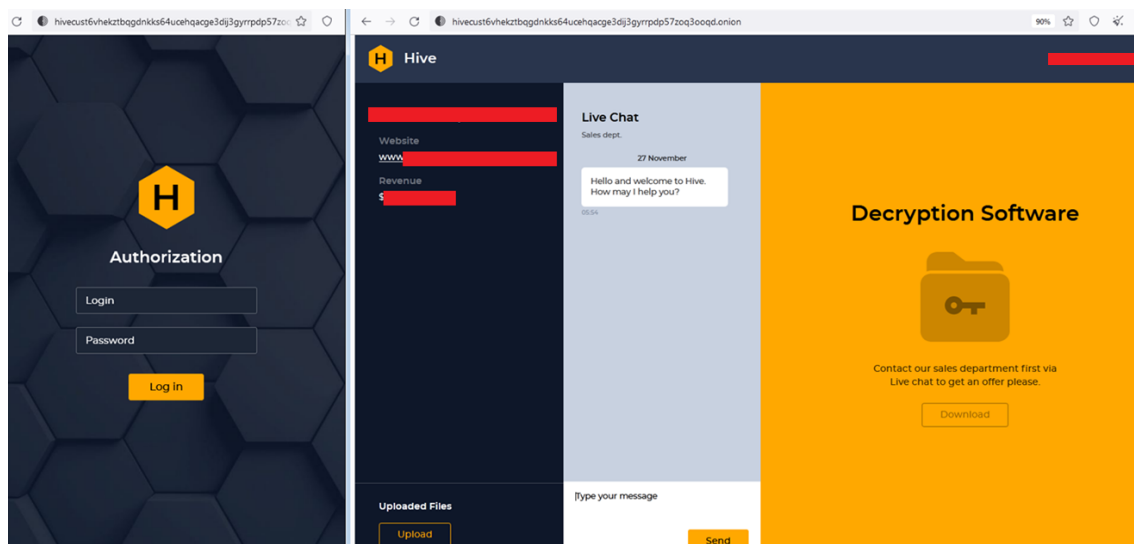


Figura 19. Portal Hive: Live Chat

5. CIFRADO

El binario generará una *key* de 0x100000 bytes que participará en el proceso de cifrado de los ficheros. Esta *key* será cifrada por medio de diversas claves RSA embebidas en el binario y, posteriormente, será guardada en disco. Para cifrar la misma, en primer lugar, se invocará la función **ParsePKCS1PublicKey** (incluida en la librería *pkcs1.go*) para *parsear* cada una de las claves públicas.

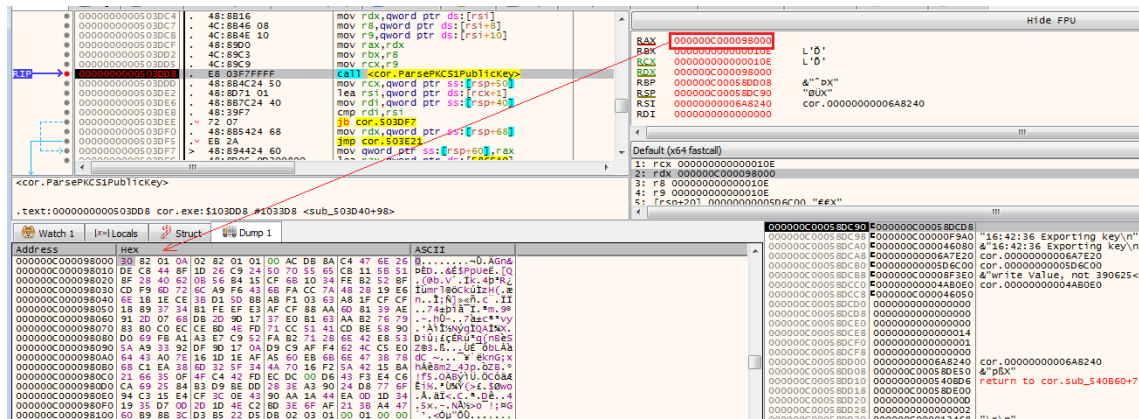


Figura 20. ParsePKCS1PublicKey

Posteriormente, comenzará el cifrado de la *key* por medio de la API **EncryptOAEP** (incluida en la librería *rsa.go*).

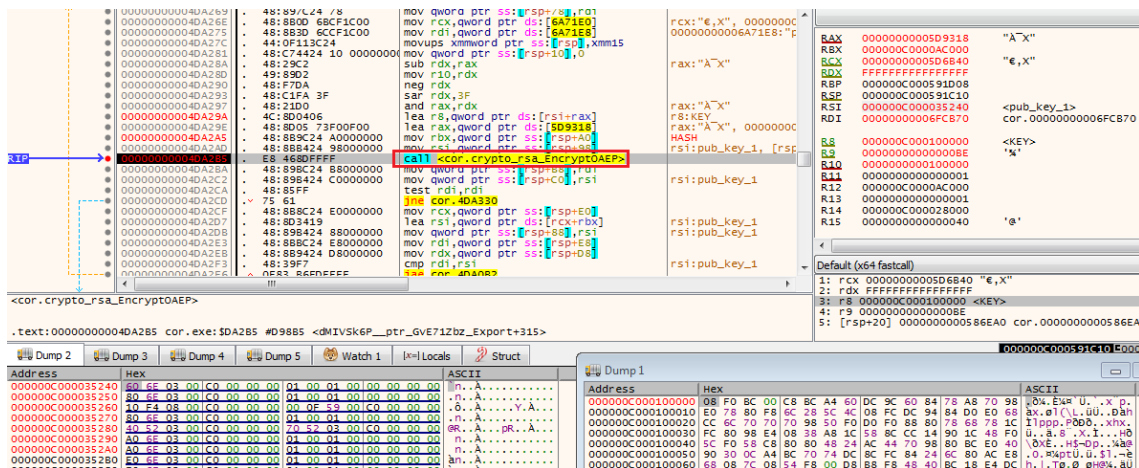
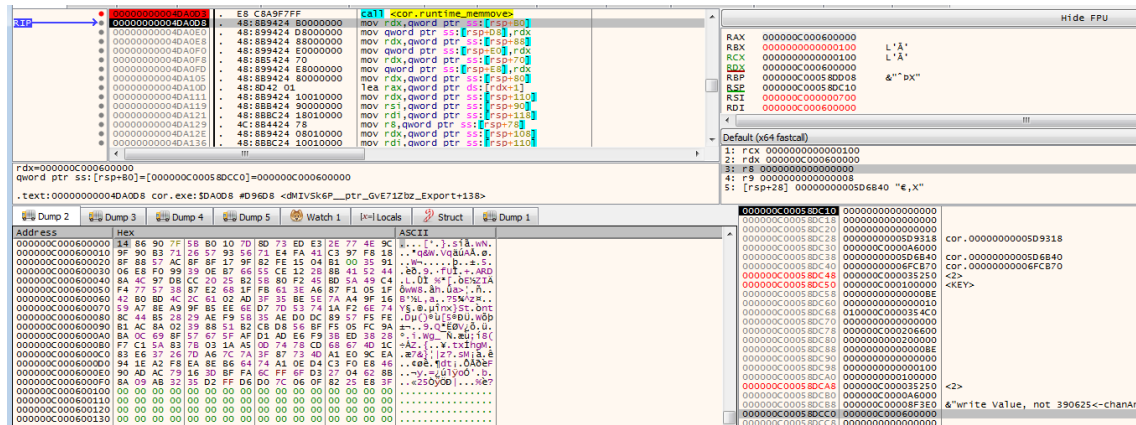


Figura 21. EncryptOAEP

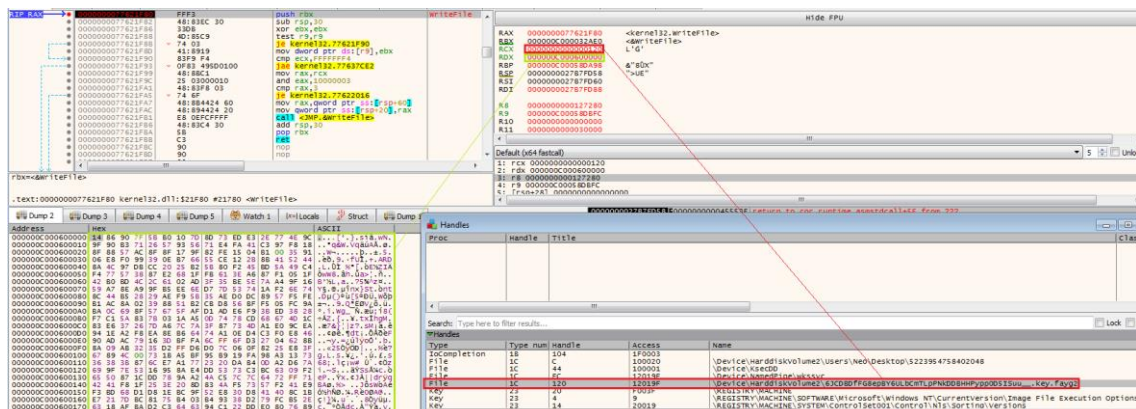
Dicho proceso irá reconstruyendo la clave cifrada en memoria (en la siguiente imagen, a partir de 0x00000000C000600000).



The screenshot shows a debugger window with assembly code on the left and a memory dump on the right. The assembly code is in x86-64 and shows a loop that reconstructs a key from memory. The memory dump shows the key being reconstructed, starting with 0x00000000C000600000.

Figura 22. Proceso de cifrado de la KEY

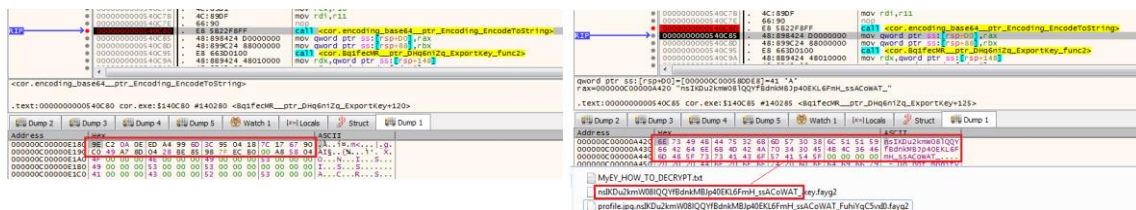
Tras finalizar el cifrado de la clave ésta será guardada en disco con extensión .key.fayg2.



The screenshot shows a debugger window with assembly code on the left and a memory dump on the right. The assembly code is in x86-64 and shows the key being written to a file. The memory dump shows the key being written to a file.

Figura 23. Escritura de la clave cifrada en disco

El nombre de la key se formará como resultado de convertir a base64 16 bytes aleatorios por medio de la API *EncodeToString*.



The screenshot shows a debugger window with assembly code on the left and a memory dump on the right. The assembly code is in x86-64 and shows the key being encoded to base64. The memory dump shows the key being encoded to base64.

Figura 24. Nombre base64 aleatorio

La siguiente imagen muestra parte de la función responsable de llevar a cabo el cifrado de ficheros el cual parece tratarse de un algoritmo personalizado.

```

000000000040AD18 > 46:0FB630E movzx r13d,byte ptr ds:[r13+r9]          alg_encrypt
000000000040AD1D > 45:31EF xor r15d,r13d
000000000040AD20 > 45:31FC xor r15d,r15d
000000000040AD23 > 44:886414 68 mov byte ptr ss:[rsp+rdx+68],r12b
000000000040AD24 > 7FC2 tnc edx
000000000040AD2A > 41:8908 mov r8d,edx
000000000040AD2D > 3900 cmp eax,edx
000000000040AD2F > 0F86 36FFFFFF jbe cor.40AE06
000000000040AD35 > 66:0F1F8400 00000000 nop word ptr ds:[rax+ax],ax
000000000040AD3E > 66:90 nop
000000000040AD40 > 48:81FA 00100000 cmp rdx,1000
000000000040AD47 > 0F83 B8000000 jae cor.40AE06
000000000040AD48 > 44:0FB6414 68 movzx r12d,byte ptr ss:[rsp+rdx+68]
000000000040AD53 > 46:802C02 rdx+8=1:"PE"
000000000040AD57 > 45:89E1 mov r15d,r13d
000000000040AD5A > 49:01ED shr r13,1
000000000040AD5D > 44:89C3 mov ebx,r8d
000000000040AD60 > 41:88 3E0AD7A3 mov r8d,A3D70A3E
000000000040AD66 > 40:0FAF6E imul r13,r8
000000000040AD6A > 45:C1ED 2F shr r13,2F
000000000040AD6E > 45:69ED 00900100 imul r13d,r13d,19000
000000000040AD75 > 44:89FE mov esi,r15d
000000000040AD78 > 45:29EF sub r15d,r13d
000000000040AD7B > 0F1F4400 00 nop dword ptr ds:[rax+ax],eax
000000000040AD80 > 40:3907 cmp r15,r10
000000000040AD85 > 46:8040006 jbe cor.40AE06
000000000040AD88 > 47:0FB62C1F movzx r13d,byte ptr ds:[r15+r11]
000000000040AD8A > 41:8F ABAAAAAA mov r15d,AAAAAAAB
000000000040AD90 > 4C:0FAFEF imul r15,r15
000000000040AD94 > 49:C1EF 2B shr r15,2B
000000000040AD98 > 45:89E1 mov r15d,qword ptr ds:[r15+r15*2]
000000000040AD9C > 41:C1E7 0A shl r15d,A
000000000040ADA0 > 44:29FE sub esi,r15d
000000000040ADA3 > 48:39CE cmp esi,r10
000000000040ADA6 > 0F82 6CFFFFFF jbe cor.alg_encrypt
  
```

Figura 25. Algoritmo de cifrado

Por cada archivo, su contenido será cargado en memoria por medio de la función *ReadFile*, se aplicará el cifrado previamente mostrado y se guardará el *ciphertext* resultante en el fichero. La siguiente imagen muestra el resultado de cifrar el fichero *profile.jpg*.

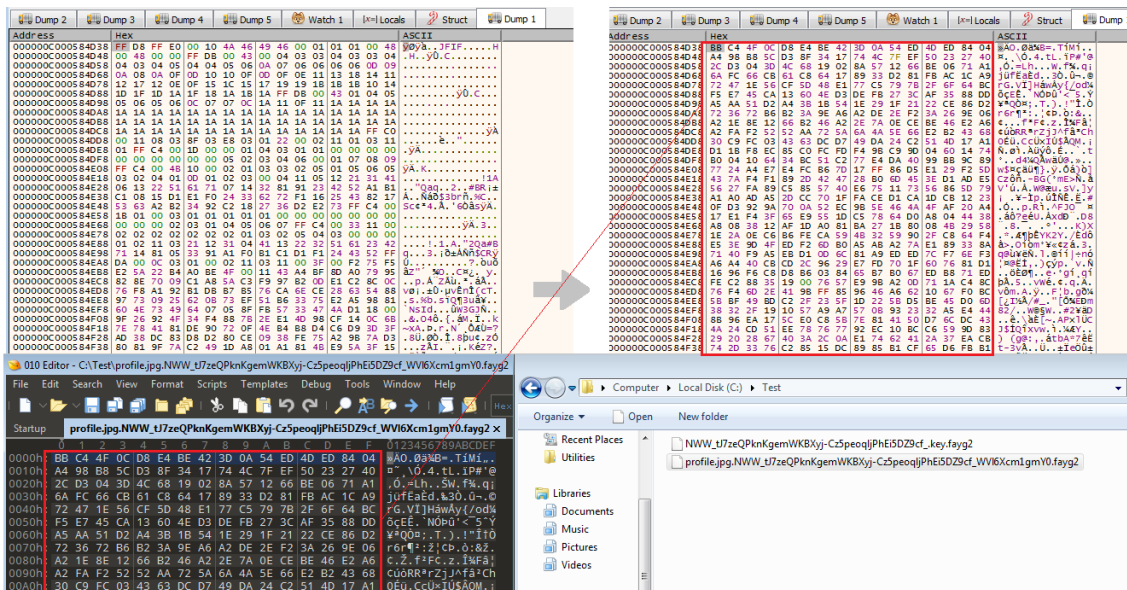


Figura 26. Cifrado del fichero profile.jpg

6. DESINFECCIÓN

Dado que la variante analizada no adquiere persistencia, los equipos afectados con Hive no requieren de una rutina de desinfección más allá de reiniciar el sistema o finalizar el proceso vinculado con el *ransomware*. Cabe destacar que, aunque el propio binario dañino no implementa funcionalidades de persistencia, el conjunto de TTPs empleadas por los atacantes para proceder con el despliegue del *ransomware* sí que puede involucrar otros componentes o técnicas para garantizar la persistencia del actor; por ejemplo, otros binarios dañinos, cuentas de usuarios en el dominio a modo de *backdoor*, *Golden tickets*, vulnerabilidades en un servicio expuesto, etc.

El proceso de respuesta a incidentes tendrá que investigar y analizar de forma meticulosa el vector de entrada, así como el conjunto de TTPs empleado por los atacantes para garantizar que la mitigación ha sido efectiva y para evitar la repetición del ataque.

Respecto al descifrado de los ficheros, el modelo de criptografía utilizado asegura que el único método factible para el descifrado sea mediante el uso de las claves privadas RSA en manos del grupo cibercriminal.

7. MITIGACIÓN

La manera más estratégica para contener y reducir el impacto del *ransomware* es trabajar de forma paralela en el aislado de redes/VLAN que conforman la entidad afectada (con el objetivo de contener los segmentos de red con equipos infectados y evitar su expansión) y en el rotado de contraseñas en el dominio. Dicho rotado debe incluir las cuentas locales de administrador (es importante que sean únicas para cada equipo) así como la cuenta KRBTGT del dominio.

Téngase en cuenta que estas medidas posiblemente interfieran con el correcto funcionamiento del dominio o redes afectadas y puedan causar diversos imprevistos (por ejemplo, el reinicio de máquinas); no obstante, permitiría contener la amenaza de una manera resolutive.

8. REGLAS DE DETECCIÓN

8.1. REGLA DE YARA

La siguiente regla, disponible en Github [\[13\]](#), permite identificar la muestra actual, así como otras variantes pertenecientes a la misma familia.

```
rule MALWARE_Win_Hive {
  meta:
    author = "ditekSHen"
    description = "Detects Hive ransomware"
  strings:
    $url1 = "http://hivecust" ascii
    $url2 = "http://hiveleakdb" ascii
    $s1 = "encrypt_files.go" ascii
    $s2 = "erase_key.go" ascii
    $s3 = "kill_processes.go" ascii
    $s4 = "remove_shadow_copies.go" ascii
    $s5 = "stop_services_windows.go" ascii
    $s6 = "remove_itself_windows.go" ascii
    $x1 = "/encryptor/" ascii
    $x2 = "HOW_TO_DECRYPT.txt" ascii
    $x3 = "FilesEncrypted" fullword ascii
    $x4 = "EncryptionStarted" fullword ascii
    $x5 = "encryptFilesGroup" fullword ascii
    $x6 = "Your data will be undecryptable" ascii
    $x7 = "- Do not fool yourself. Encryption has perfect secrecy" ascii
    $v1_1 = ".EncryptFiles." ascii
    $v1_2 = ".EncryptFilename." ascii
    $v1_3 = ")*struct { F uintptr; .autotmp_14 string }" ascii
    $v1_4 = "D*struct { F uintptr; data *[]uint8; seed *uint8; fnc *main.decFunc }" ascii
    $v1_5 = "golang.org/x/sys/windows.GetSystemWindowsDirectory" ascii
    $v1_6 = "path/filepath.WalkDir" ascii
  condition:
    uint16(0) == 0x5a4d and (all of ($url*) or all of ($s*) or 4 of ($x*) or 5 of ($v1*))
}
```


9. REFERENCIAS

[1] Malpedia: Hive

<https://malpedia.caad.fkie.fraunhofer.de/details/win.hive>

[2] Hive Ransomware: Actively Targeting Hospitals

<https://www.netskope.com/blog/hive-ransomware-actively-targeting-hospitals>

[3] Double extortion ransomware

<https://www.darktrace.com/en/blog/double-extortion-ransomware/>

[4] Hive Attacks | Analysis of the Human-Operated Ransomware Targeting Healthcare

<https://www.sentinelone.com/labs/hive-attacks-analysis-of-the-human-operated-ransomware-targeting-healthcare/>

[5] ScreenConnect MSP Software Used to Install Zeppelin Ransomware

<https://www.bleepingcomputer.com/news/security/screenconnect-msp-software-used-to-install-zeppelin-ransomware/>

[6] Yanluowang ransomware operation matures with experienced affiliates

<https://www.bleepingcomputer.com/news/security/yanluowang-ransomware-operation-matures-with-experienced-affiliates/>

[7] VirusTotal: Hive (7802d6315bf0d45f27bd97fb48e70f8e)

<https://www.virustotal.com/gui/file/d0ceb8f5170972fe737ab9cbdd6f3ee472fbe62e244cccc46d137094d33f1afc/details>

[8] Virustotal: muestra similares

<https://www.virustotal.com/gui/file/ff93136112316cea3f80218c5354d6e8c12cdfe449c40ab417002fe81dcf1dcb>

<https://www.virustotal.com/gui/file/47dbb2594cd5eb7015ef08b7fb803cd5adc1a1fbe4849dc847c0940f1ccace35/details>

[9] Intezer: Hive (7802d6315bf0d45f27bd97fb48e70f8e)

<https://analyze.intezer.com/analyses/c8ba388d-8400-4ceb-a023-fc709ffa3e69/sub/c19d3fa2-6798-4953-a361-8bd8f493f65d/code-reuse>

[10] Say Hello to Moduledata

<https://lekstu.ga/posts/hello-moduledata/>

[11] Reversing Golang

https://2016.zeronights.ru/wp-content/uploads/2016/12/GO_Zaytsev.pdf

[12] Github: disable Windows 10

https://gist.github.com/pe3zx/7c5e0080c3b0869ccba1f1dc2ea0c5e0#file-disable_windows_defender-bat

[13] Github: malware rules (Hive)

<https://github.com/ditekshen/detection/blob/master/yara/malware.yar>

ANEXO A

Listado de procesos a finalizar:

agntsvc
sql
CNTAoSMgr
dbeng50
dbsnmp
encsvc
excel
firefoxconfig
infopath
mbamtray
msaccess
mspub
mydesktop
Ntrtscan
ocautoupds
ocomm
ocssd
onenote
oracle
outlook
PccNTMon
powerpnt
sqbcoreservice
steam
synctime
tbirdconfig
thebat
thunderbird
tmlisten
visio
word
xfssvcccon
zoolz

Listado de servicios a finalizar:

acronis
AcrSch2Svc
Antivirus
ARSM
AVP
backup
bedbg
CAARCUupdateSvc
CASAD2DWebSvc
ccEvtMgr
ccSetMgr
Culserver
dbeng8
dbsrv12
DCAgent
DefWatch
EhttpSrv

ekrn
Enterprise Client Service
EPSecurityService
EPUUpdateService
EraserSvc11710
EsgShKernel
ESHASRV
FA_Scheduler
firebird
IISAdmin
IMAP4Svc
Intuit
KAVFS
KAVFSGT
kavfsslpl
klnagent
macmnsvc
masvc
MBAMService
MBEndpointAgent
McAfee
McShield
McTaskManager
memtas
mepocs
mfefire
mfemms
mfefvtp
MMS
MsDtsServer
MsDtsServer100
MsDtsServer110
msexchange
msmdsrv
MSOLAP
MVArmor
MVarmor64
NetMsmqActivator
ntrtscan
oracle
PDVFSService
POP3Svc
postgres
QBCFMonitorService
QBFCService
QBIDPService
redis
report
RESvc
RTVscan
sacsvr
SamSs
SAVAdminService
SavRoam
SAVService
SDRSVC

SepMasterService
ShMonitor
Smcinst
SmcService
SMTPSvc
SNAC
SntpService
sophos
sql
SstpSvc
stc_raw_agent
^svc
swi_
Symantec
TmCCSF
tmlisten
tomcat
TrueKey
UIODetect
veeam
vmware
vss
W3Svc
wbengine
WebClient
wrapper
WRSVC
WSBExchange
YoolT
zhudongfangyu
Zoolz