Informe Código Dañino
CCN-CERT ID-11/22

RansomEXX_Linux ransomware

Octubre 2022

Edita:

GOBIERNO DE ESPAÑA MINISTERIO DE DEFENSA

CCN centro criptológico nacional

Fecha de Edición: octubre de 2022

LIMITACIÓN DE RESPONSABILIDAD

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

AVISO LEGAL

ÍNDICE

1. SOBRE CCN-CERT

El CCN-CERT es la Capacidad de Respuesta a Incidentes de Seguridad de la Información del Centro Criptológico Nacional (CCN) del 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 311/2022, de 3 de mayo, por el que se regula el Esquema Nacional de Seguridad.

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

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

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

2. RESUMEN EJECUTIVO

El presente documento recoge el análisis de la muestra de código dañino identificada por la firma MD5 377c6292e0852afeb4bd22ca78000685 compilada para plataformas Linux (x64) y desarrollada en lenguaje Rust. El binario analizado pertenece a la familia de *ransomware* apodada como EXX o RansomEXX, también conocido como Defray777 o RansomX. Este *ransomware*, vinculado con el actor GOLD DUPONT [1], ha estado involucrado en múltiples incidentes desde 2018 [2] afectando a organismos gubernamentales y empresas críticas.

En mayo de 2020 el sistema judicial [3] así como el Departamento de Transporte del estado de Texas (TxDOT) [4] se vio afectado por este *ransomware*. La multinacional japonesa Konica Minolta [5] así como el sistema judicial de Brasil [6] fueron también objetivo de RansomEXX en julio y noviembre, respectivamente, de ese mismo año. La empresa brasileña Embraer, considerada como uno de los mayores fabricantes de aviones civiles de la actualidad, no solo se vio afectada [7] por RansomEXX a finales de 2020, sino que sus datos fueron filtrados después de que el fabricante de aviones se negara a ceder ante la, cada vez más común, doble extorsión [8] . En el último año, los ataques vinculados a este ransomware han continuado incrementándose viéndose afectadas entidades como Ferrari, BRP (*Bombardier Recreational Products*) o el FNDE (*National Fund for Educational Development)* de Brasil.

Los atacantes se caracterizan por utilizar como vía de entrada ciertas vulnerabilidades en productos VMWare ESXi [9] para cifrar sus discos duros virtuales con EXX.

El ransomware hace uso del cifrado por bloques AES-256 para cifrar los ficheros de la víctima. A diferencia de las variantes de RansomEXX previamente identificadas [10] la muestra actual se corresponde con una nueva versión del código dañino desarrollada en lenguaje Rust, uniéndose así a la tendencia empleada por las familias de ransomware Hive y BlackCat/ALPHV [11] las cuales han migrado también su código a este lenguaje.

3. DETALLES GENERALES

El fichero analizado se corresponde con una muestra de ransomware para plataformas Linux (ELF64) de la familia RansomEXX cuya firma se muestra a continuación:

MD5	377c6292e0852afeb4bd22ca78000685
SHA1	4fd0a5d1f5dd5c4de31c372e23dc148982ac153d
SHA256	a7ea1e33c548182b8e56e32b547afb4b384ebe257ca0672dbf72569a54408c5c

El *sample* ha sido remitido, por primera vez, a plataformas de análisis de malware el 7 de octubre de 2022 sin mostrar ninguna alerta por parte de los motores antivirus.

Cabe destacar que para que el ransomware proceda con el cifrado de ficheros es necesario ejecutarlo con ciertos argumentos. Entornos de análisis de malware automatizados que ejecuten el binario sin el *command-line* correspondiente tendrán dificultades para identificar acciones dañinas a partir de la heurística del binario. Este hecho explicaría la nula detección por parte de los diversos motores antivirus.

En la siguiente imagen se ha ejecutado el código dañino por medio de *strace* sin ningún argumento. Puede observarse que el proceso rápidamente finaliza su ejecución sin llevar a cabo acciones dañinas (no se observan *syscalls* relevantes tales como *open, read, write*).

```
execve("./mlw", ["./mlw"], 0x7ffeac1e89e0 /* 27 vars */) = 0
mmap(NULL, 400, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f09acf47000
arch_prctl(ARCH_SET_FS, 0x7f09acf470b0) = 0
set_tid_address(0x7f09acf55fd8)         = 18034
poll([{fd=0, events=0}, {fd=1, events=0}, {fd=2, events=0}], 3, 0) = 0 (Timeout)
rt_sigaction(SIGPIPE, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x7f09acd36e53}, {sa_handler=SIG_DFL, sa_mask=[], sa_flag
s=0}, 8) = 0
rt_sigaction(SIGSEGV, NULL, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RT_1 RT_2], NULL, 8) = 0
rt_sigaction(SIGSEGV, {sa_handler=0x7f09acd033d0, sa_mask=[], sa_flags=SA_RESTORER|SA_ONSTACK|SA_SIGINFO, sa_restorer=0x7f09acd36e53}, NULL, 8) = 0
rt_sigaction(SIGBUS, NULL, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGBUS, {sa_handler=0x7f09acd033d0, sa_mask=[], sa_flags=SA_RESTORER|SA_ONSTACK|SA_SIGINFO, sa_restorer=0x7f09acd36e53}, NULL, 8) = 0
sigaltstack(NULL, {ss_sp=NULL, ss_flags=SS_DISABLE, ss_size=0}) = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f09acf44000
mprotect(0x7f09acf44000, 4096, PROT_NONE) = 0
sigaltstack({ss_sp=0x7f09acf45000, ss_flags=0, ss_size=8192}, NULL) = 0
brk(NULL)                               = 0x555555af2000
brk(0x555555af3000)                     = 0x555555af3000
rt_sigprocmask(SIG_BLOCK, ~[RTMIN RT_1 RT_2], [], 8) = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
sigaltstack({ss_sp=NULL, ss_flags=SS_DISABLE, ss_size=8192}, NULL) = 0
munmap(0x7f09acf44000, 12288)           = 0
exit_group(1)                           = ?
+++ exited with 1 +++
```

Figura 1. Ejecución vía strace

El binario presenta un tamaño de 592.264 bytes y ha sido compilado (*statically linked*) para arquitecturas de 64 bits. La siguiente imagen muestra su *ELF Header* así como algunas de las propiedades estáticas del mismo.

```
┌──(root㉿CCNLAB)-[~]
└─# file mlw
mlw: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, stripped
┌──(root㉿CCNLAB)-[~]
└─# ldd mlw
        statically linked
┌──(root㉿CCNLAB)-[~]
└─# readelf -s mlw

Symbol table '.dynsym' contains 1 entry:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
┌──(root㉿CCNLAB)-[~]
└─# readelf -h mlw
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x8b3b
  Start of program headers:          64 (bytes into file)
  Start of section headers:          590600 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         7
  Size of section headers:           64 (bytes)
  Number of section headers:         26
  Section header string table index: 25
┌──(root㉿CCNLAB)-[~]
└─# objdump -s --section .comment mlw

mlw:     file format elf64-x86-64

Contents of section .comment:
 0000 4743433a 2028474e 55292039 2e322e30  GCC: (GNU) 9.2.0
 0010 00                                   .
```

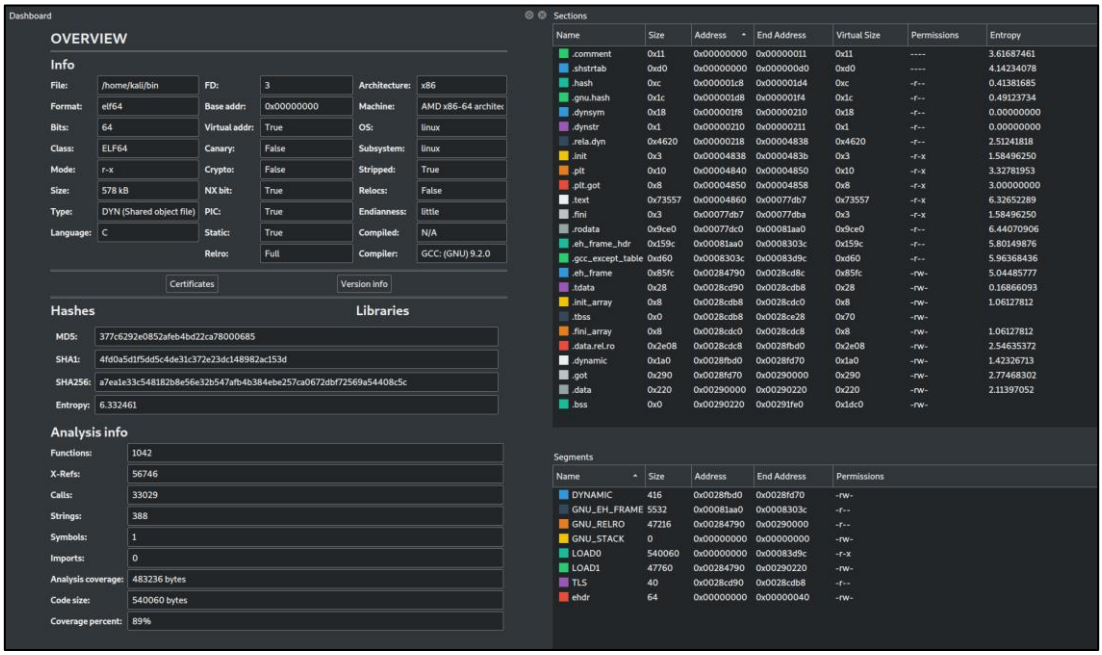

Figura 2. Información estática ELF64

A partir de las múltiples referencias al gestor de paquetes "Cargo" en los *strings* del binario ("*.cargo/registry/src*") se puede inferir que Rust ha sido el lenguaje empleado por los atacantes para su desarrollo. Dichos *paths* reflejan las dependencias externas requeridas por el código (obsérvese, por ejemplo, la referencia a la implementación de AES en Rust).

```
┌──(root💀CCNLAB)-[/tmp]
└─# rabin2 -zzz bin  | grep 'cargo/registry'
7221 0x00079e94 0x00079e94 583 584 (.rodata) ascii /mnt/c/Users/1/.cargo/registry/src/github.com-1ecc6299db9ec823/rayon-core-1.9.3/src/job.
aproject/ransomexx2/ransomexx/src/parallel_iter.rs/mnt/c/Users/1/.cargo/registry/src/github.com-1ecc6299db9ec823/aes-0.8.1/src/soft/fixslic
atting trait implementation returned an errorlibrary/alloc/src/fmt.rs) should be < len (is removal index (is /mnt/c/Users/1/.cargo/registry
tifier(
7271 0x0007af30 0x0007af30 446 447 (.rodata) ascii Layoutalign/mnt/c/Users/1/.cargo/registry/src/github.com-1ecc6299db9ec823/crossbeam-epoc
-1ecc6299db9ec823/crossbeam-epoch-0.9.11/src/sync/once_lock.rsPRIVATE [] (primitiveconstructedCONTEXT-SPECIFIC [APPLICATION [BMPStringVisib
UTF8StringENUMERATEDOBJECT IDENTIFIEROCTET STRINGBIT STRINGINTEGERBOOLEANTag(0x
7274 0x0007b410 0x0007b410 522 523 (.rodata) ascii /rustc/a55dd71d5fb0ec5a6a3a9e8c27b2127ba491ce52/library/alloc/src/collections/btree/map/
src/collections/btree/node.rsassertion failed: edge.height == self.height - 1assertion failed: src.len() == dst.len().debug_types.debug_str
c.debug_line_str.debug_line.debug_info.debug_aranges.debug_addr.debug_abbrev/cargo/registry/src/github.com-1ecc6299db9ec823/gimli-0.25.0/sr
7275 0x0007b66d 0x0007b66d  85  86 (.rodata) ascii /cargo/registry/src/github.com-1ecc6299db9ec823/miniz_oxide-0.4.0/src/inflate/core.rs
7279 0x0007b742 0x0007b742  94  95 (.rodata) ascii /cargo/registry/src/github.com-1ecc6299db9ec823/miniz_oxide-0.4.0/src/inflate/output_buf
7280 0x0007b7b0 0x0007b7b0 1645 1646 (.rodata) ascii attempt to divide by zeroassertion failed: self.len() ≤ isize::MAX as usizedestinatio
enexplicit panicdivide by zero!/mnt/c/Users/1/.cargo/registry/src/github.com-1ecc6299db9ec823/num-bigint-dig-0.8.1/src/monty.rsassertion fa
1ecc6299db9ec823/num-bigint-dig-0.8.1/src/algorithms/add.rs/mnt/c/Users/1/.cargo/registry/src/github.com-1ecc6299db9ec823/num-bigint-dig-0.
```

Figura 3. Cadenas (Cargo: *Rust package manager*)

Otras cadenas, relacionadas también con los diversos *paths* de los ficheros en Rust, referencian a la propia familia RamsomEXX.

```
der-0.5.1/src/document.rs/mnt/c/Users/1/.cargo/registry/src/github.com-1ecc6299db9ec823/rsa-0.6.1/src/pkcs1v15.rs/rustc/a55dd71d5fb0ec5a6a3a9e8c27b2127ba491
Once has panicked/mnt/c/Users/1/.cargo/registry/src/github.com-1ecc6299db9ec823/spin-0.5.2/src/once.rsfrom_entropy failed: /mnt/c/Users/1/.cargo/registry/sr
es256_impl.rs{
B+)S"\k
:xn?m
%l'5
mOv!
ERROR_RENAME_LIMIT_REACHEDransomexx/src/footer.rsransomexx/src/logic.rs
attempt to divide by zero
ransomexx/src/ransom_data.rsL
$$<,
```

Figura 4. Referencia RansomEXX (Rust)

En cada directorio afectado por el proceso de cifrado el código dañino creará un fichero de texto bajo el nombre "!_WHY_FILES_ARE_ENCRYPTED_!.txt" donde puede

encontrarse una referencia a la dirección *.onion* vinculada con este grupo criminal (*rnsm777cdsjrsdlbs4v5qoeppu3px6sb2igmh53jzrx7ipcrbjz5b2ad.onion/.onion*)

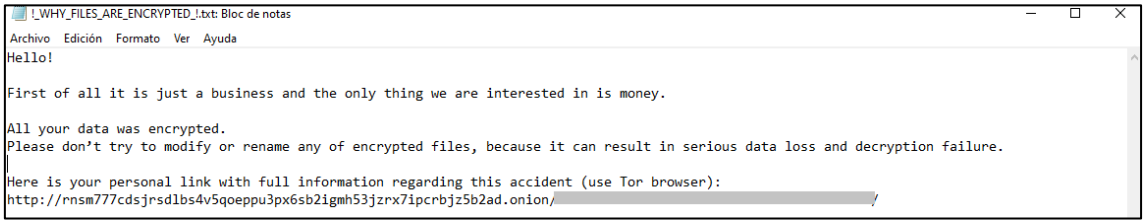

Figura 5. Contenido del fichero "!_WHY_FILES_ARE_ENCRYPTED_!.txt"

Hasta la fecha, dicho portal recoge un total de 12 entidades comprometidas durante el año 2022 desde donde se puede descargar los ficheros vinculados a cada *leak.*

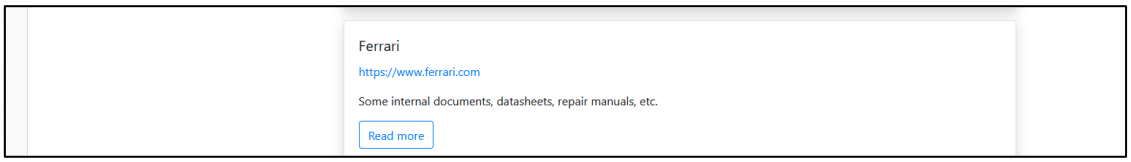

Figura 6. Empresas afectadas por RansomEXX (2022)

4. CARACTERÍSTICAS TÉCNICAS

La principal novedad de esta nueva versión de RansomEXX es su implementación en Rust. Este lenguaje no solo ofrece a los atacantes notables ventajas desde un punto de vista de seguridad y rendimiento (mejor gestión de la concurrencia y la memoria, gran velocidad de cifrado, etc.) sino que dificulta significativamente la ingeniería inversa por parte de los analistas. Este hecho está relacionado con la forma en la que Rust gestiona los errores y la gran cantidad de *checks* de seguridad que se añaden al código durante el proceso de compilación para garantizar un uso seguro de la memoria. La carencia de una ABI estable, el uso de determinados *flags* de compilación (por ejemplo, *link-time optimization*) así como otras características del propio lenguaje complican en gran medida la comprensión del código generado.

Para que el binario comience con el proceso de cifrado de ficheros es necesario proporcionar, por medio de su *command-line,* el parámetro "**--do**" seguido de los directorios/ficheros que se desean cifrar. Por ejemplo, la siguiente ejecución permitirá cifrar de forma recursiva los directorios "*/var/log/*" y "*/opt/java/*" además del fichero "*/etc/passwd*: **mlw --do /var/log/ /opt/java/ /etc/passwd**".

El siguiente bloque de código muestra el proceso de *desofuscación* de dicho argumento (la cadena "**--do**") seguido de la comprobación del primer parámetro proporcionado al proceso.

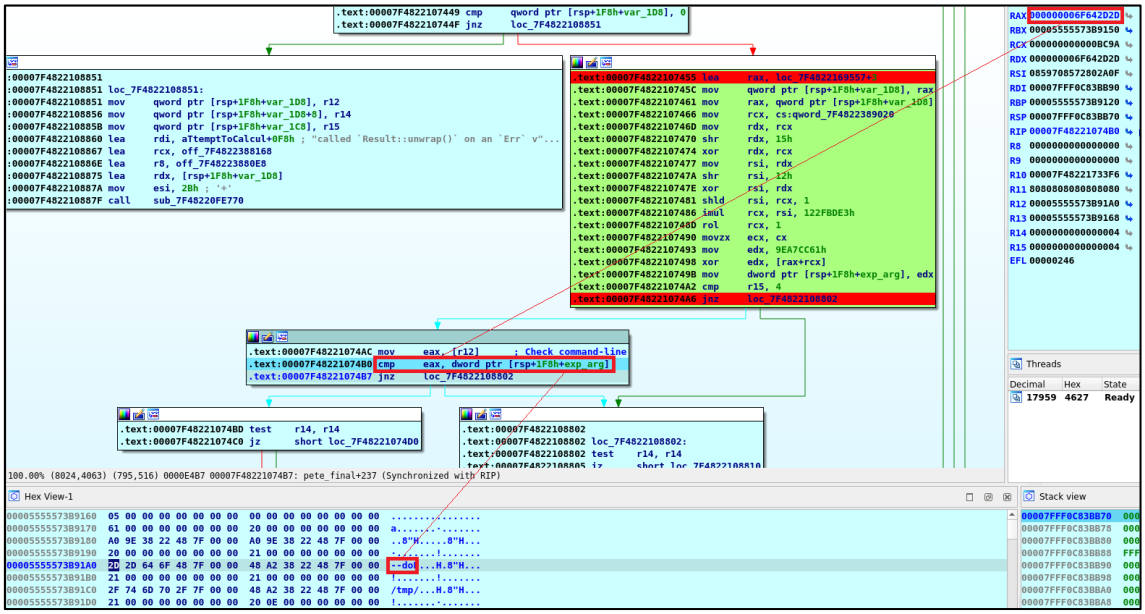

Figura 7. Comprobación *command-line* (--do)

El resto de cadenas requeridas por el ransomware se encontrarán también cifradas mediante diversos algoritmos *custom*. En la siguiente imagen se observa el bloque de código responsable de recuperar el nombre de fichero de texto "!_WHY_FILES_ARE_ENCRYPTED_!.txt"

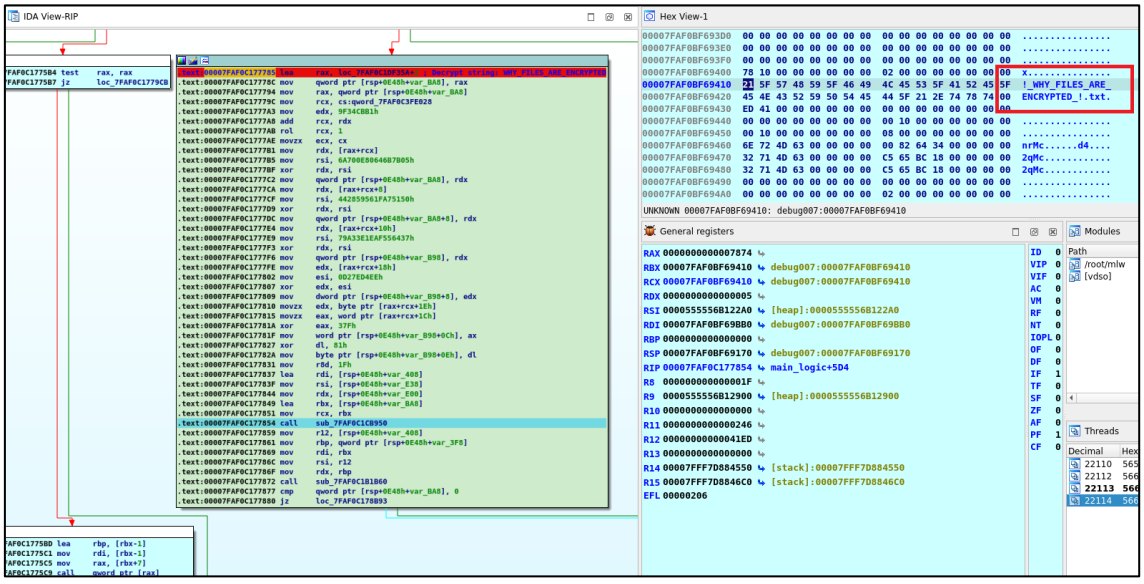

Figura 8. String !_WHY_FILES_ARE_ENCRYPTED_!.txt'

Este nombre será asignado al fichero .txt creado en cada directorio afectado durante el proceso de cifrado. La siguiente imagen muestra la creación de dicho fichero en disco.

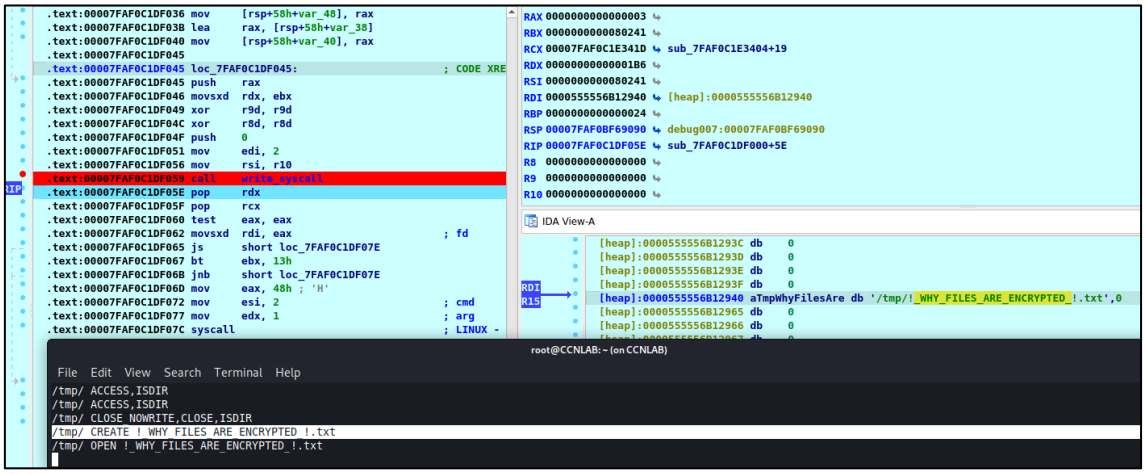

Figura 9. Creación fichero !_WHY_FILES_ARE_ENCRYPTED_!.txt' / inotify

El contenido del mismo referencia la URL en ".onion", perteneciente al grupo criminal en donde se encuentran las instrucciones para poder recuperar, supuestamente, los ficheros afectados.

Figura 10. Dirección .onion ransomEXX

Cada fichero cifrado será renombrado con la extensión única según cada víctima (nombre que hace referencia a la institución afectada). Esta cadena se encuentra ofuscada también por medio de cierto algoritmo *custom*.

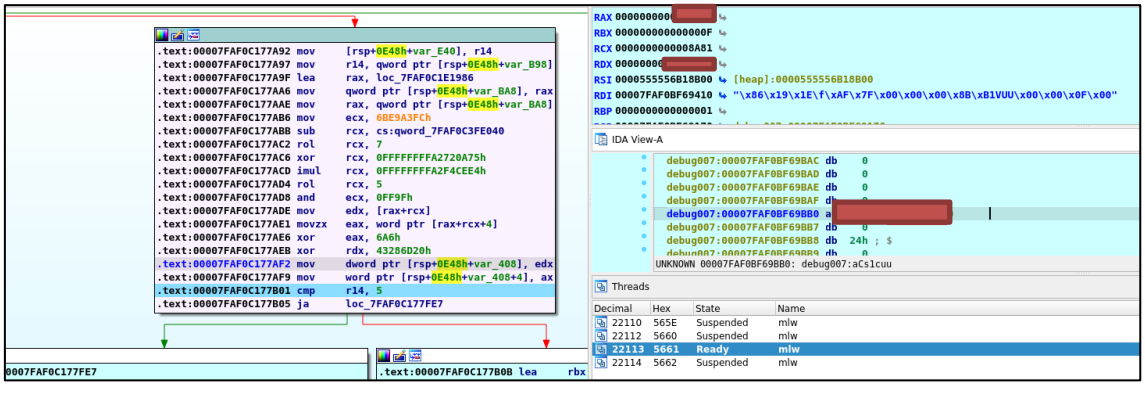

Figura 11. Extensión .<según_víctima> / Renombre fichero

Para llevar a cabo el cifrado de los ficheros, el código dañino hace uso del algoritmo AES 256 utilizando el conjunto de instrucciones AES-NI (*aesenc/aesenclast*)[12] introducido en la familia de procesadores Intel® Core™ 2010. En la siguiente imagen se muestra la función que implementa dicha lógica:

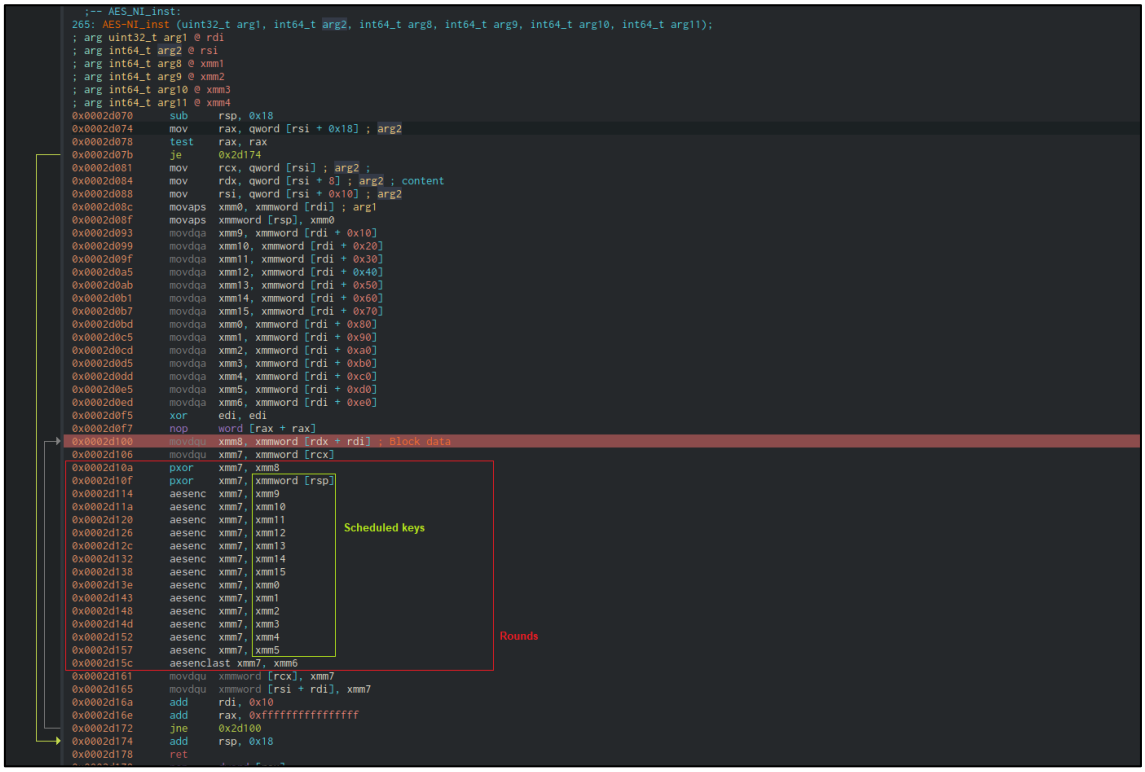

Figura 12. Instrucciones AES-NI (cifrado AES)

La clave AES se generará de manera aleatoria al inicio de la ejecución del proceso dañino y se utilizará para cifrar todos los ficheros indicados a través del *command line*. Posteriormente, y siguiendo el mismo *modus operandi* que las variantes de EXX anteriores, la clave AES será cifrada utilizando una clave RSA-4096 embebida, de forma ofuscada, en el binario.

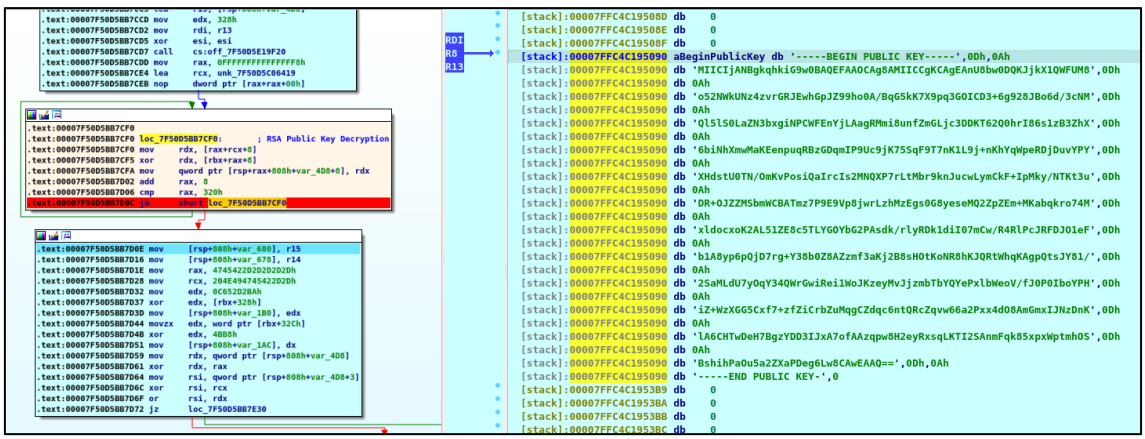

Figura 13. Descifrado clave pública

El *ciphertext* resultante (clave AES cifrada con RSA) será añadido al final del contenido previamente cifrado.

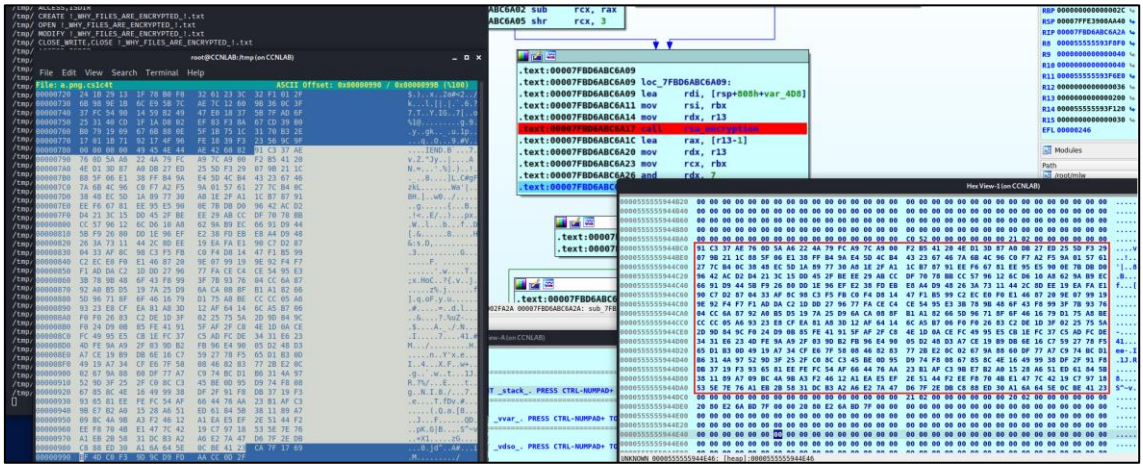

Figura 14. Ciphertext (clave AES cifrada con RSA)

Finalmente se generará un ID de infección de 16 bytes (derivado, entre otros, de la extesión según víctima) que es añadido al final del fichero (véanse los últimos 16 bytes de la imagen anterior, justo después del conjunto de bytes resaltados, en la parte izquierda).

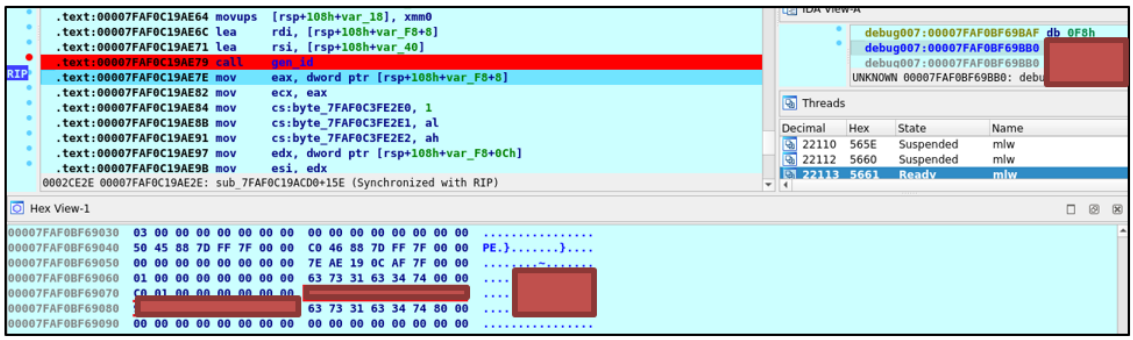

Figura 15. Generación ID campaña (16 bytes)

La siguiente imagen recoge el proceso de infección descrito previamente (con las *syscalls* más relevantes) tras ejecutar RansomExx. El código dañino recorrerá de forma recursiva los directorios proporcionados como argumento (en el ejemplo, "/tmp"), renombrando cada fichero con la extensión *según victima* y posteriormente cifrando su contenido según el procedimiento expuesto.

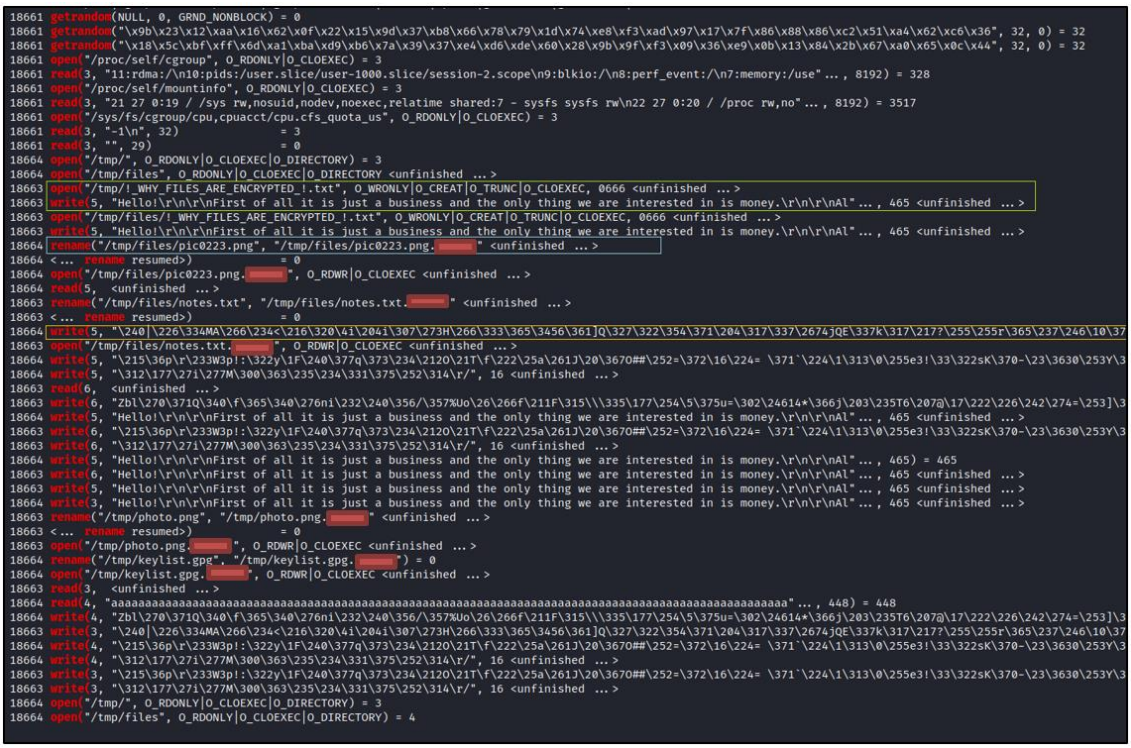

Figura 16. Ejecución EXX (strace)

5. 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 (entornos Windows). 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 resolutiva.

6. REGLAS DE DETECCIÓN

6.1 REGLA YARA

```
rule RansomEXX_Lin_Rust {
 meta:
   description = "Ransomware ransomEXX (Linux)"
   date = "2022-10-20"
   author = "CCN-CERT"

strings:
 $x1 = "DeadlockTimedOutNotFound.zdebug_" fullword ascii
 $x2 = "ransomexx/src/" ascii
 $x3 = "/mnt/c/Users/1/.cargo/"
 $x4 = "ciphers/aes256_impl.rs" ascii
 $y1 = { 48 69 CE E3 BD 2F 12 48 D1 C1 0F B7 C9 BA 61 CC A7 9E }    // imul    rcx, rsi, 122FBDE3h
                                                                    // rol     rcx, 1
                                                                    // movzx   ecx, cx
                                                                    // mov     edx, 9EA7CC61h
 $y2 = { 48 BE 05 7B 6B 64 80 0E 70 6A 48 31 F2 }                   // mov     rsi, 6A700E80646B7B05h
                                                                    // xor rdx, rsi
 $y3 = { 48 BE 50 51 A7 1F 56 59 28 44 48 31 F2 }                   // mov     rsi, 442859561FA75150h
                                                                    // xor     rdx, rsi
 condition:
   (uint32(0) == 0x464C457F) and ((filesize > 400KB) and (filesize < 700KB)) and ((2 of ($x*)) or (any of ($y*)))
}
```