

PAPERMILL: Tracking an Emerging China-Nexus Malware Factory

 jumpsec.com/guides/an-emerging-silver-fox-like-cluster-papermill

lewis@tcmarketing.co.uk

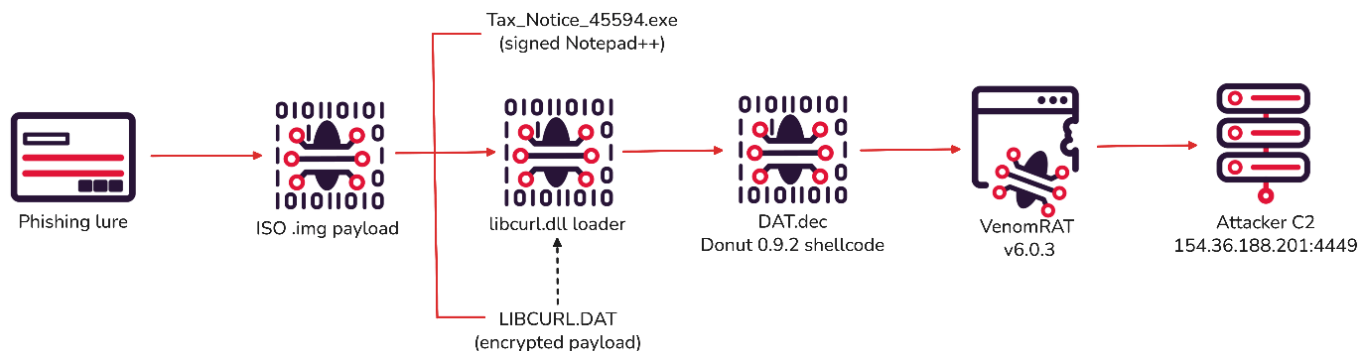
September 15, 2026



Executive Summary

JUMPSEC's Detection and Response Team (DART) recently escalated a phishing email after it reached a client's inbox despite passing standard email authentication checks. On the surface, this appeared to be another routine phishing campaign with a tax-related pretext and a seemingly unremarkable attachment.

However, rather than delivering malware directly, the operator had constructed a multi-stage execution chain designed to bypass trust mechanisms, abuse trusted software, evade automated analysis, and ultimately deliver a remote access trojan with minimal scrutiny. What followed was a journey through ISO-based delivery, DLL sideloading, encrypted shellcode loaders, and ultimately a VenomRAT payload linked to a wider ecosystem of related activity.



A high-level overview of the PAPERMILL execution chain from delivery, loading, and evasion to the final payload stages.

The sample became the starting point for a wider investigation into what appears to be an emerging Silver Fox-like cluster, a family of China-nexus campaigns known for using tax-themed lures, trusted software abuse, and multi-stage malware delivery chains.

We have dubbed this cluster **PAPERMILL**, a temporary designation until the relationship between this activity and known Silver Fox operations, if any, becomes clearer through future observations, reporting, and research.

Initial Access: A Trusted Container

At first glance the attachment appears unremarkable. In reality, the choice of file format is one of the most important parts of the attack chain.

The extension says disk image; the bytes say optical disc. First 32 KB are zero-filled (the ISO system area), then the primary volume descriptor shows up right on cue at sector 16 – 01 43 44 30 30 31 01 = \x01 “CD001” \x01 at offset 0x8001.

So Tax_Notice_45594.img is an **ISO 9660** image wearing an .img extension. This is the entire reason it’s a container and not a plain ZIP: files carried inside a mounted ISO **do not inherit Mark-of-the-Web**. This technique has become popular because it exploits a gap in how Windows handles downloaded archives versus mounted disk images. From a defender’s perspective, the attacker is effectively converting an internet download into something that appears local and trusted. The malware itself hasn’t changed but the security context around it has. The browser tags the downloaded .img with a zone identifier, but when the user double-clicks it Windows mounts it as a virtual CD, and the Tax_Notice_45594.exe *inside* comes out clean – no MOTW, so SmartScreen and the Office/Explorer “downloaded from the internet” warnings never fire.

How it was built is stamped right in the descriptor’s application-ID field: **PyCddlib (C) 2015-2020 Chris Lalancette**, with blank volume and system IDs (the pycddlib defaults nobody bothered to set). The ISO was generated programmatically from a Python script, not authored in any disc tool.

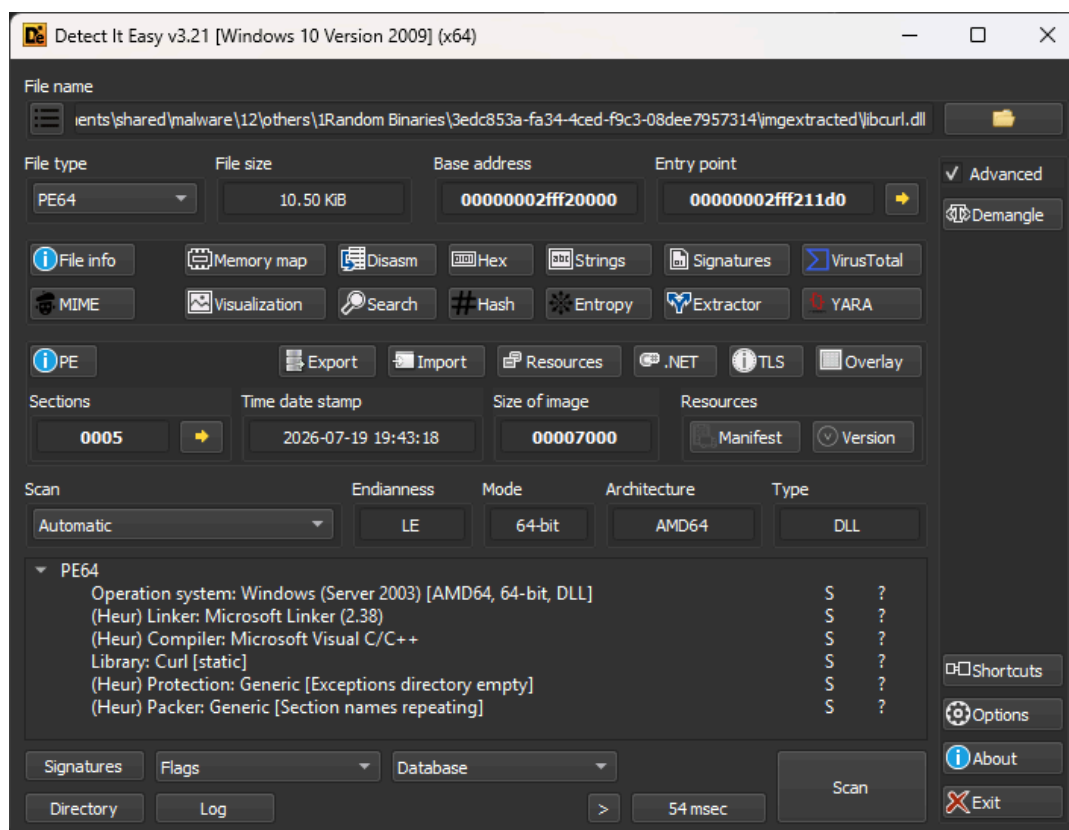
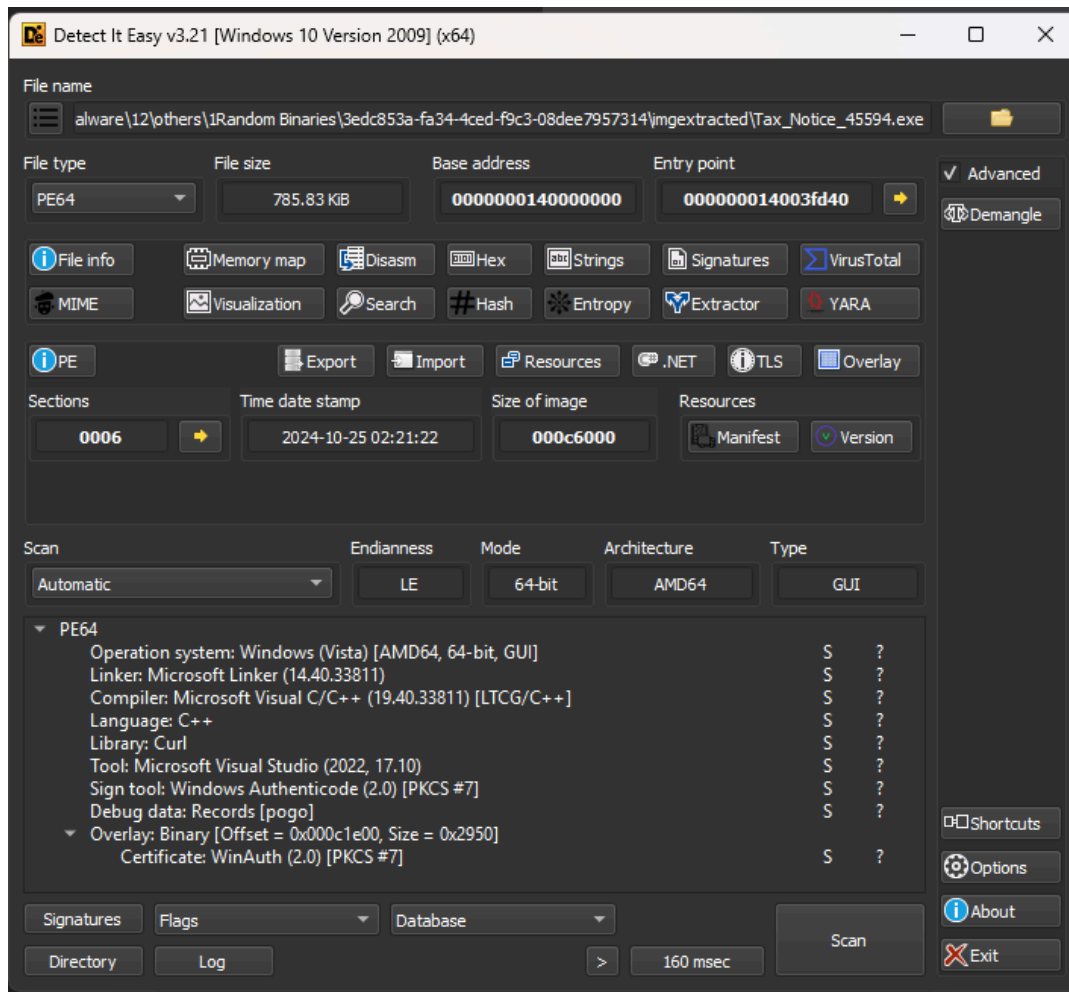
	Tax_Notice_45594.img															
Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00007FC0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00007FD0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00007FE0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00007FF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00008000	01	43	44	30	30	31	01	00	20	20	20	20	20	20	20	20
00008010	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008020	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008030	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008040	20	20	20	20	20	20	20	20	20	00	00	00	00	00	00	00
00008050	E6	01	00	00	00	00	01	E6	00	00	00	00	00	00	00	00
00008060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00008070	00	00	00	00	00	00	00	00	00	01	00	00	01	01	00	00
00008080	00	08	08	00	0A	00	00	00	00	00	0A	15	00	00	00	00
00008090	00	00	00	00	00	00	00	17	00	00	00	00	22	00	1D	00
000080A0	00	00	00	00	00	1D	00	08	00	00	00	00	08	00	7E	07
000080B0	17	02	28	36	00	02	00	00	01	00	00	01	01	00	20	20
000080C0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000080D0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000080E0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000080F0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008100	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008110	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008120	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008130	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008140	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008150	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008160	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008170	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008180	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008190	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000081A0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000081B0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000081C0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000081D0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000081E0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000081F0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008200	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008210	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008220	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008230	20	20	20	20	20	20	20	20	20	20	20	20	20	20	50	79
00008240	43	64	6C	69	62	20	28	43	29	20	32	30	31	35	2D	32
00008250	30	32	30	20	43	68	72	69	73	20	4C	61	6C	61	6E	63
00008260	65	74	74	65	20	20	20	20	20	20	20	20	20	20	20	20
00008270	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008280	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008290	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000082A0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000082B0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000082C0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000082D0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000082E0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
000082F0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008300	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00008310	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
Offset: 0																Overwrite

The ISO 9660 volume descriptor confirms the attachment is a disk image rather than a conventional archive.

When mounted, it holds exactly three files: “Tax_Notice_45594.exe” (not to be confused with the original Tax_Notice_45594.img), “libcurl.dll”, and “LIBCURL.DAT”. Which is where it gets fun.

The Sideload

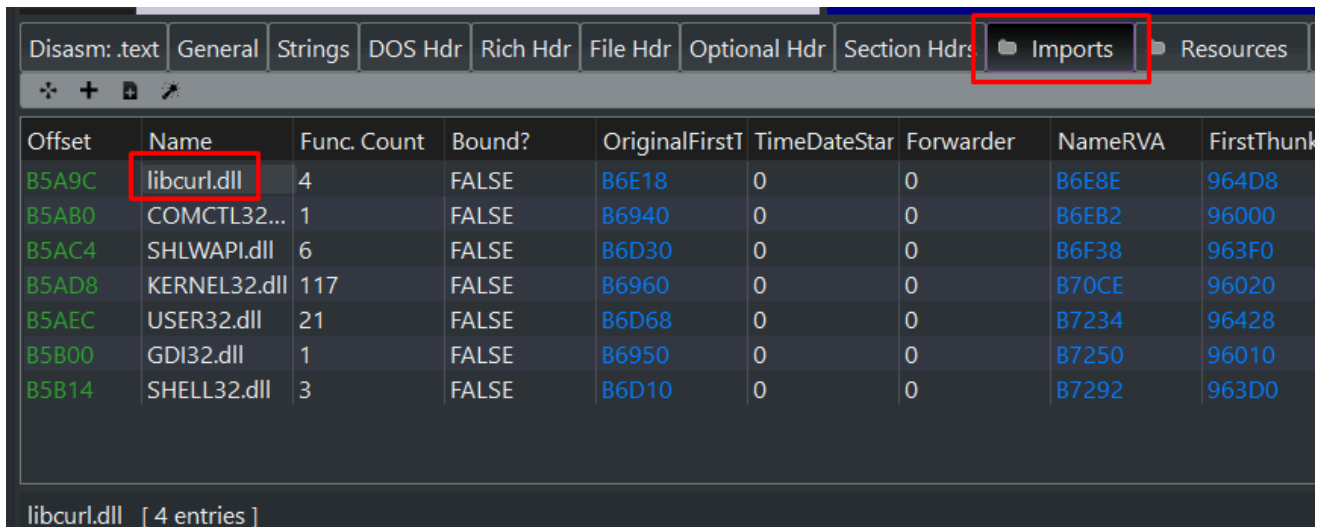
By running DIE across the three payload files, a toolchain mismatch tells the whole story straight away. The EXE is **MSVC 2022 (19.36.33811) and Authenticode-signed**; the DLL sitting next to it is a **MinGW/GNU-Id (Binutils 2.38)** build. A real libcurl.dll shipped alongside a Microsoft-signed app would not be a 10 KB GCC artifact.



Toolchain differences between the signed executable and neighbouring DLL immediately suggest sideloading activity.

The “malware” EXE is a legitimately signed Notepad++ binary. The attacker didn’t trojanize it – they renamed it to Tax_Notice_45594.exe and dropped it as a launcher. This is a good example of why defenders should be cautious about treating signed binaries as inherently trustworthy. Nothing about the Notepad++ executable itself is malicious. Instead, the trust associated with the signed binary is being borrowed by a neighbouring DLL. Notepad++ uses libcurl.dll for its plugin-admin/updater, so it will happily load a libcurl.dll from its own directory. That’s the hook.

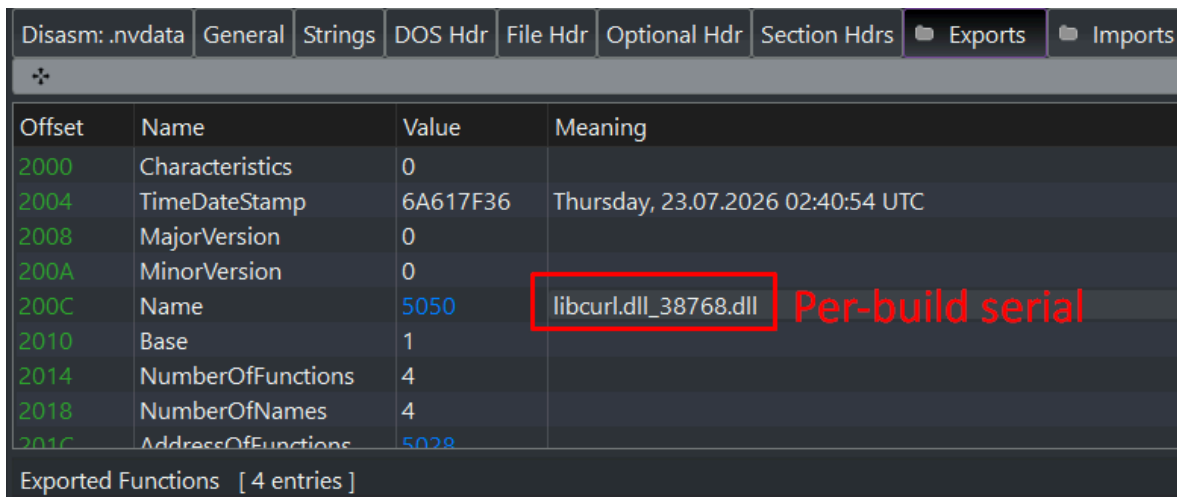
The EXE imports four functions from libcurl.dll – ‘curl_easy_setopt’, ‘curl_easy_cleanup’, ‘curl_easy_init’, ‘curl_easy_perform’ – and the malicious DLL exports **exactly those four and nothing else**. This is a clear indication of DLL Proxying, where the loader re-exports the precise API surface the host asks for, so Notepad++ resolves its imports, runs, and looks completely normal to the user while the DLL does its business on the side.



The screenshot shows the Disasm Imports tab in a debugger. The 'Imports' tab is selected and highlighted with a red box. Below the tabs, a table lists imported DLLs. The first entry, 'libcurl.dll', is highlighted with a red box. Below the table, it says 'libcurl.dll [4 entries]'.

Offset	Name	Func. Count	Bound?	OriginalFirstT	TimeDateStar	Forwarder	NameRVA	FirstThunk
B5A9C	libcurl.dll	4	FALSE	B6E18	0	0	B6E8E	964D8
B5AB0	COMCTL32...	1	FALSE	B6940	0	0	B6EB2	96000
B5AC4	SHLWAPI.dll	6	FALSE	B6D30	0	0	B6F38	963F0
B5AD8	KERNEL32.dll	117	FALSE	B6960	0	0	B70CE	96020
B5AEC	USER32.dll	21	FALSE	B6D68	0	0	B7234	96428
B5B00	GDI32.dll	1	FALSE	B6950	0	0	B7250	96010
B5B14	SHELL32.dll	3	FALSE	B6D10	0	0	B7292	963D0

libcurl.dll [4 entries]



The screenshot shows the Disasm Exports tab in a debugger. The 'Exports' tab is selected. Below the tabs, a table lists exported functions. The entry 'libcurl.dll_38768.dll' is highlighted with a red box. To the right of this entry, the text 'Per-build serial' is written in red. Below the table, it says 'Exported Functions [4 entries]'.

Offset	Name	Value	Meaning
2000	Characteristics	0	
2004	TimeStamp	6A617F36	Thursday, 23.07.2026 02:40:54 UTC
2008	MajorVersion	0	
200A	MinorVersion	0	
200C	Name	5050	libcurl.dll_38768.dll
2010	Base	1	
2014	NumberOfFunctions	4	
2018	NumberOfNames	4	
201C	AddressOfFunctions	5028	

Exported Functions [4 entries]

The proxy DLL exports only the functions required by the renamed Notepad++ executable.

Inside the loader

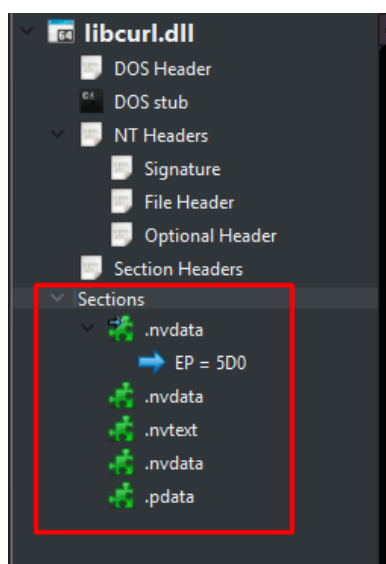
The first IDA surprise is that the section table is a lie. Every section has been deliberately named to suggest it contains something else entirely.

That’s interesting because most malware doesn’t need to defeat analysts in this way. Commodity loaders generally assume they’ll only ever be inspected briefly, if at all. The author has spent a lot of effort actively degrading the analyst experience, which suggests they expect defenders to take a closer look. It’s a small change technically but a very useful indicator of intent.

Section	What it actually holds
.nvdata	The real .text
.nvdata	The real .data
.nvtext	The real .rdata
.nvdata	The real .edata
.pdata	The real .idata

The .nvtext/.nvdata swap is the obvious one: if your disassembler only disassembles sections named .text, it finds nothing, because the code is in the section called .nvdata and .nvtext is read-only data.

Alongside this, .pdata is a trap for reversers, there is no exception directory in this binary at all. The section named .pdata is the import table. Any disassembler that takes apart this binary will essentially output garbage.



Misleading section names are used to complicate static analysis and reverse engineering.

Once IDA is pointed at the right section, there are only 57 functions to analyse, these are the main ones:

- `loader\_DllMain` – everything happens here on DLL_PROCESS_ATTACH before Notepad++ ever calls the first curl_easy_* function
- `apc\_run\_payload` – the APC callback that eventually executes the decrypted shellcode
- `installer\_thread` – establishes persistence through Registry Key edits
- `is\_admin` – Self-explanatory
- `elevation\_watchdog` – background loop that keeps popping UAC until it wins.

What makes this loader interesting is that much of its execution logic is concentrated into a single function. Rather than spreading responsibility across dozens of components, privilege checks, persistence, anti-analysis routines, payload decryption, and shellcode execution are orchestrated from a single control path.



High-level control flow showing how the loader progresses from initial execution through anti-analysis checks, persistence, and ultimately payload execution.

The loader's decision point

`apc\_run\_payload` at `0x2fff21a10` is what actually runs the decrypted blob.

```

1 // APC callback. DPI -> is_admin -> (runas + watchdog + exit) OR (installer_thread + 7-check anti-VM) -> VirtualProtect RX -> call blob
2 void __fastcall apc_run_payload(ULONG_PTR Parameter)
3 {
4     HMODULE LibraryW; // rax
5     FARPROC ProcAddress; // rax
6     void (__stdcall *v3)(HMODULE, LPWSTR, DWORD); // rsi
7     HANDLE v4; // rcx
8     void (__stdcall *v5)(DWORD); // rsi
9     HANDLE Thread; // rcx
10    ULARGE_INTEGER FreeBytesAvailableToCaller; // [rsp+10h] [rbp-2F8h] BYREF
11    struct tagPOINT Point; // [rsp+18h] [rbp-2F0h] BYREF
12    struct tagPOINT v9; // [rsp+20h] [rbp-2E8h] BYREF
13    WCHAR DirectoryName[4]; // [rsp+28h] [rbp-2E0h] BYREF
14    struct _SYSTEM_INFO SystemInfo; // [rsp+30h] [rbp-2D8h] BYREF
15    SHELLEXECUTEINFO pExecInfo; // [rsp+60h] [rbp-2A8h] BYREF
16    WCHAR LibFileName[284]; // [rsp+00h] [rbp-238h] BYREF
17
18    if ( lpAddress )
19    {
20        wcsncpy(LibFileName, L"shcore.dll");
21        LibraryW = LoadLibraryW(LibFileName);
22        if ( LibraryW )
23        {
24            ProcAddress = GetProcAddress(LibraryW, "SetProcessDpiAwareness");
25            if ( ProcAddress )
26                ((void (__fastcall *) (__int64))ProcAddress)(2);
27        }
28        SetProcessDPIAware();
29        v3 = (void (__stdcall *) (HMODULE, LPWSTR, DWORD))GetModuleFileNameW;
30        if ( !(unsigned int)is_admin() )
31        {
32            v3 = (void (__stdcall *) (HMODULE, LPWSTR, DWORD))GetModuleFileNameW;
33            GetModuleFileNameW(nullptr, LibFileName, 0x104u);
34            *(_QWORD *)&pExecInfo.cbSize = 0x80000000000070LL;
35            memset(&pExecInfo.hwnd, 0, 0x68u);
36            pExecInfo.lpFile = LibFileName;
37            pExecInfo.lpVerb = L"runas";
38            if ( ShellExecuteExW(&pExecInfo) )

```

The APC callback is responsible for executing the decrypted payload in memory.

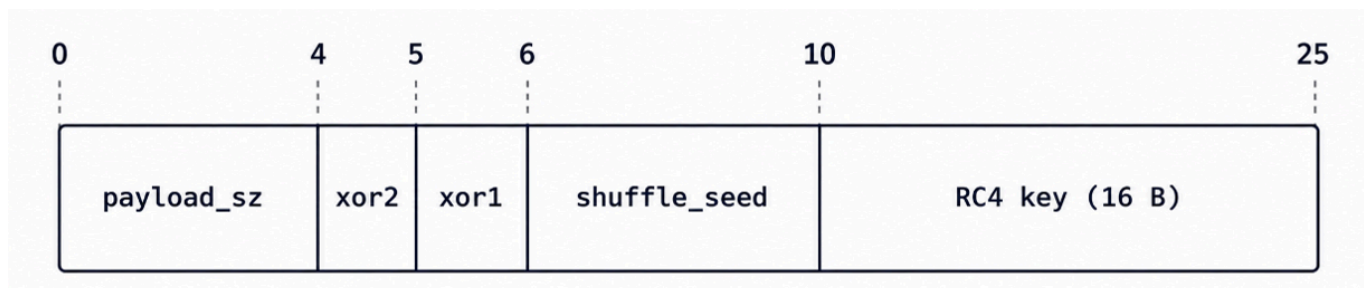
Execution summary: DPI-aware (it LoadLibraryWs shcore.dll and resolves SetProcessDpiAwareness dynamically, with the statically-imported USER32 SetProcessDPIAware as a fallback) → check admin → if not admin, relaunch itself with the runas verb, spawn a watchdog thread that keeps re-popping UAC until the user gives in, and exit → if admin, spawn installer_thread to drop and persist, then run the anti-VM screen. (MAKE ILLUSTRATION)

Check	Constant	Fails if
CPU cores	<= 1	Single-core
Physical RAM	<= 0xFFFFFFFF	< 4GB
Free disk on C:	<= 0xEFFFFFFF	< ~60GB
Uptime	cmp rax, 0x927BF	< 10 min
Cursor moved over 3s	GetCursorPos (x2)	Didn't move
Screen resolution	cmp eax, 0x3FF / 0x2FF	< 1024×768
Last user input	cmp eax, 0x493E0	None in 5 min

The battery itself is normal. What's smart is the failure mode: instead of exiting when a check fails, which is exactly what a naive sandbox logs as suspicious, it falls into Sleep(0x493E0) (300,00 ms = 5 minutes) and then continues anyway. Most automated sandboxes have shorter analysis windows than that, so the sample looks benign and gets scored clean, this reduces the likelihood of the malware being fully observed in automated analysis pipelines. Rather than fighting the sandbox, it simply waits it out.

Payload Decryption & Shellcode Recovery

The whole custom cipher fits into one Python script. The last 26 bytes of the file are a parameter block:



The final bytes of LIBCURL.DAT contain the parameters required for payload decryption.

Then three passes over the body – `^= xor1`, bitwise NOT, `^= xor2` followed by a DWORD level Fisher–Yates de-shuffle and finally RC4 with the 16-byte key applied. The de-shuffle PRNG is a bog-standard LCG, and it's all sitting in the disassembly in plain sight:

```
PS C:\Users\resetti\Documents\shared\malware\12\others\1Random Binaries\3edc853a-fa34-4ced-f9c3-08dee7957314\analysis\scripts> python decrypt_dat.py
hared\malware\12\others\1Random Binaries\3edc853a-fa34-4ced-f9c3-08dee7957314\imgextracted\LIBCURL.DAT" .\LIBCURL.DAT.dec
[+] file size : 114,462
[+] footer.payload_size: 0x1bf04 (114,436)
[+] footer.xor1 : 0x0f
[+] footer.xor2 : 0x12
[+] footer.shuffle_seed: 0x1bf02
[+] footer.rc4_key : c2666f54f679214b28330d39c79496c3
[+] dword-deshuffle applied; real payload size = 0x1bf02 (114,434)
[+] wrote 114,434 bytes to LIBCURL.DAT.dec
[+] entropy of decrypted blob : 7.811
[+] first 64 bytes: e88b6101008861010095459a0bc3a0acbd5d39fac534503228f150f974690ebb723d7be78c0405824af000000001205911a46b63142b63003bb45676a45644832
[+] first 16 chars: b'\xe8\x88a\x01\x00\x88a\x01\x00\x95E\x9a\x0b\xc3\xa0\xac'
PS C:\Users\resetti\Documents\shared\malware\12\others\1Random Binaries\3edc853a-fa34-4ced-f9c3-08dee7957314\analysis\scripts> |
```

Decrypting LIBCURL.DAT reveals an additional encrypted execution stage.

The interesting part is that the “shuffle seed” isn’t a seed at all, it’s the plaintext length. 0x1BF02 = 114,434 is the exact size of the decrypted output.

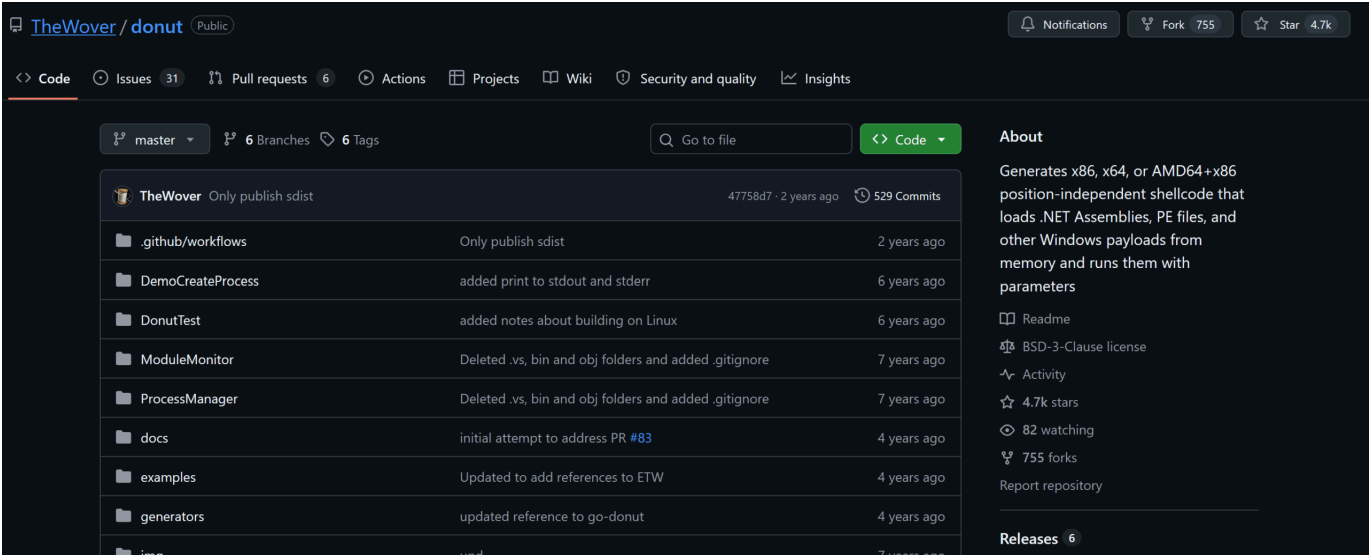
Entropy after decryption is still 7.811, which is a red flag that we haven’t hit the final stage, so I looked at the file in HxD and found what looked like shellcode:

```
Detect It Easy v3.21 [Windows 10 Version 2009] (x64)
File name: hared\malware\12\others\1Random Binaries\3edc853a-fa34-4ced-f9c3-08dee7957314\imgextracted\LIBCURL.DAT.dec
File type: Binary, File size: 111.75 KB, Base address: 00000000, Entry point: 00000000
Scan results:
- Binary (Heur) Format: Decal (Background) file [by extension]
- Shellcode: Donut x86 + AMD64
Scan took: 160 msec
```

Persistent high entropy indicates further packing or shellcode-based execution.

Splitting the blob by entropy, and the layout reveals itself, the first 90KB is an encrypted blob, and the last 24KB looks like a stub at the end. The initial CALL pushes the address of the encrypted blob onto the stack, then the stub starts with `pop rcx` to grab it.

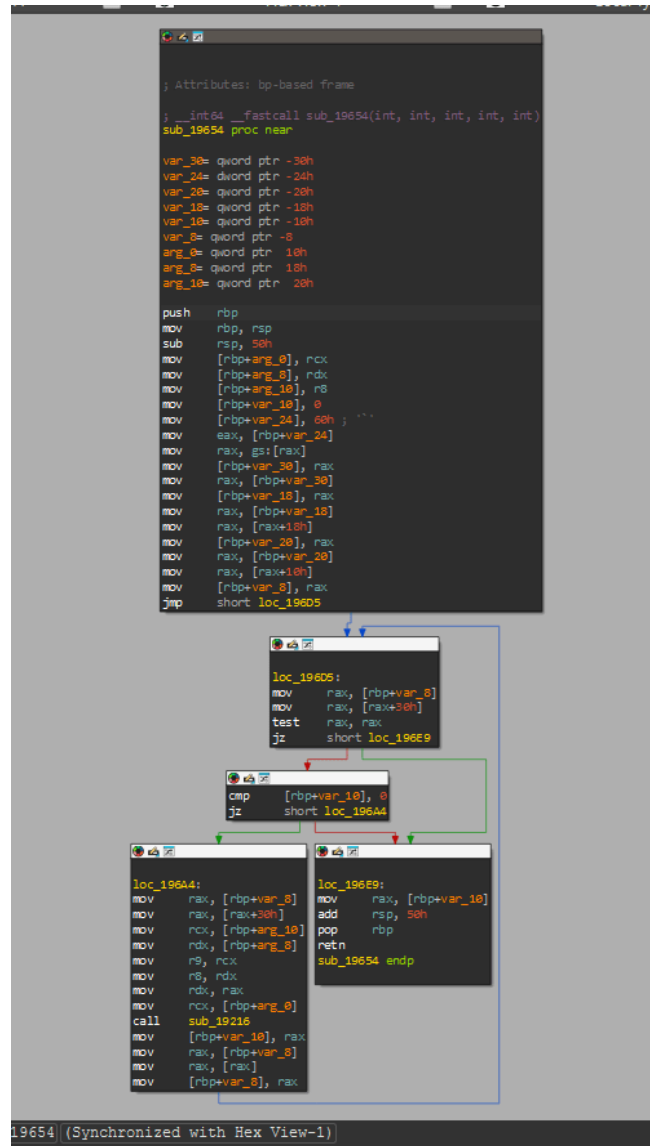
Structurally this looks like Donut loader, an open-source tool that can generate position-independent shellcode that loads anything from .NET assemblies to PE files into memory and then executes them.



The recovered shellcode shares characteristics with the open-source Donut loader framework.

The Donut Layer

The stub at 0x1618d opens with a **textbook PEB walk**: ``gs:[0x60]`` for the PEB, ``+0x18`` for ``PEB.Ldr``, ``+0x10`` for the module list, then ``+0x30`` for ``DllBase``. That's the exact prologue at the top of every Donut PIC.



A textbook PEB walk helps identify the payload as Donut-generated shellcode.

We used an open source tool to decrypt the Donut shellcode: <https://github.com/volexity/donut-decryptor>

```

INFO - Parsing donut from file: C:\Users\resetti\Documents\shared\malware\12\others\1Random Binaries\3edc853a-fa34-4ced-f9c3-08dee7957314\SHOTS\LIBCURL.DAT.dec
INFO - Using 0.9.2_64, and instance version: 0.9.2
INFO - Locating instance in file: C:\Users\resetti\Documents\shared\malware\12\others\1Random Binaries\3edc853a-fa34-4ced-f9c3-08dee7957314\SHOTS\LIBCURL.DAT.dec
INFO - Found instance at: 0x5
INFO - Writing module to: C:\Users\resetti\Documents\shared\malware\12\others\1Random Binaries\3edc853a-fa34-4ced-f9c3-08dee7957314\SHOTS\mod_LIBCURL.DAT.dec
INFO - Writing instance meta data to: C:\Users\resetti\Documents\shared\malware\12\others\1Random Binaries\3edc853a-fa34-4ced-f9c3-08dee7957314\SHOTS\inst_LIBCURL.DAT.dec
INFO - Parsed: 1 of 1 attempted files

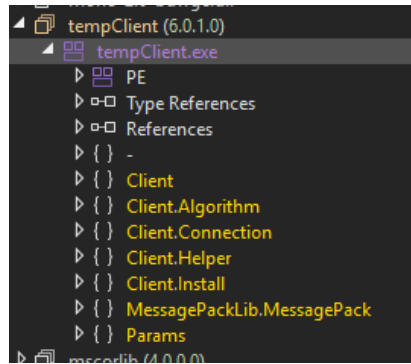
-- instance metadata -----
{
  "File": "C:\\Users\\resetti\\Documents\\shared\\malware\\12\\others\\1Random Binaries\\3edc853a-fa34-4ced-f9c3-08dee7957314\\SHOTS\\LIBCURL.DAT.dec",
  "Instance Type": "DONUT_INSTANCE_EMBED",
  "Module Type": "DONUT_MODULE_NET_DLL"
}

-- extracted module -----
file      : mod_LIBCURL.DAT.dec
size      : 82216 bytes
sha256    : 947221d0f1e2c9c09028491997406dd9f0cb5b65258556c86d201ec1ca538f5f
magic     : 4d5a (MZ)

```

Using Donut-specific tooling exposes the underlying .NET payload.

The `MODULE\_NET\_DLL` indicates to us that this is a .NET DLL, reflectively loaded into the process. The binary is 82KB and the namespace layout is very similar to the ASyncRAT/Quasar malware family.



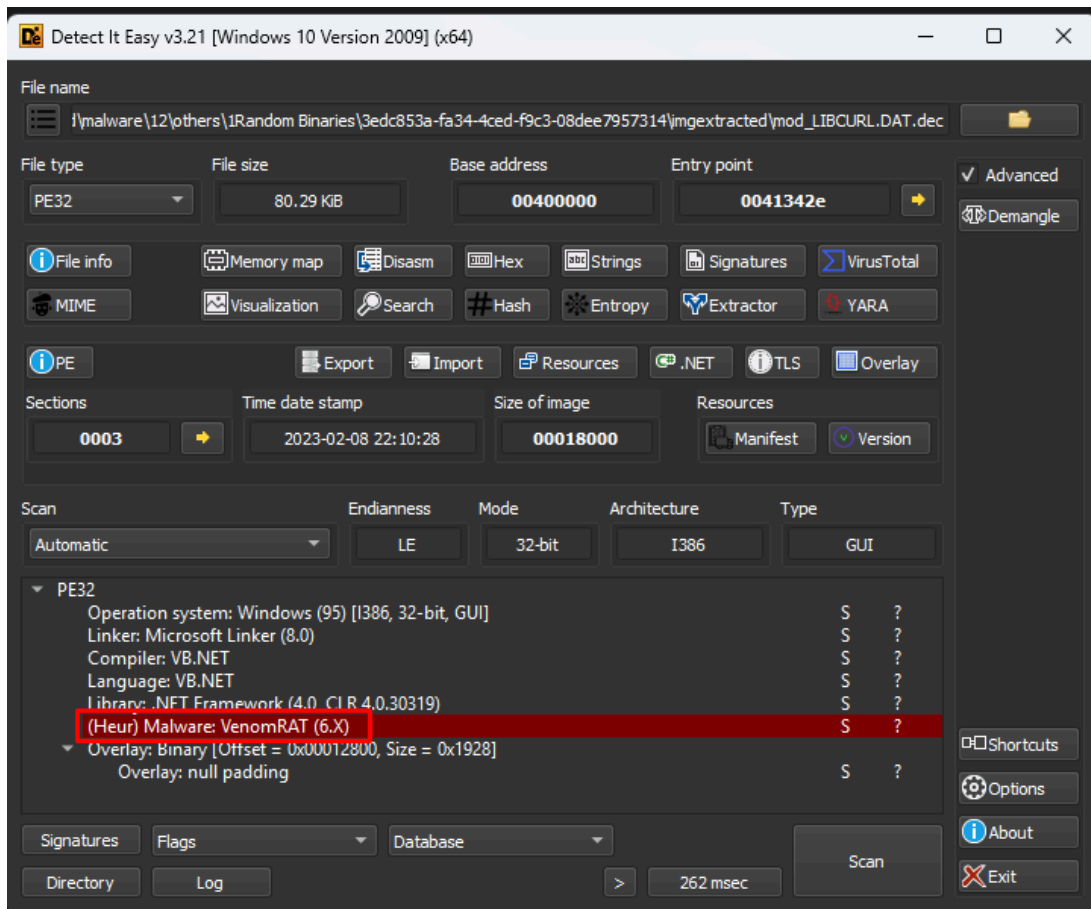
The recovered namespace structure resembles the ASyncRAT and Quasar malware families.

Final Payload: VenomRAT

We can tell the difference between this being an ASyncRAT/Quasar payload and pin it to VenomRAT specifically due to a few indicators:

- `Client.Helper.DInvokeCore` + `Delegates.NtProtectVirtualMemory` – VenomRAT ships D/Invoke for hooked-API bypass
- `Client.Helper.ProcessCritical` – calls `RtlSetProcessIsCritical`, so killing the process BSODs the box
- `Client.Program.StopHVNC` – kills `cvtres` processes, VenomRAT's HVNC worker name
- `Client.Algorithm.Aes256.Salt = Encoding.ASCII.GetBytes("VenomRATByVenom")` – the literal PBKDF2 salt string, checked in from the builder repo

DetectItEasy also detects it instantly after decryption:



Post-decryption analysis identifies the payload as VenomRAT.

The process to decrypt the VenomRAT config is well known, so we won't go through it here, the master key is 'pYkMMX7nP7idsNYX2UZf4GQmQG26h2cV' and the derived AES key is 'e55cf0745105d264db810ffc87a6f65f88686cb476498f081abb35c06383df13', using this information with a quick Python script, we get the decrypted config:

```
PS C:\Users\resetti\Documents\shared\malware\12\others\1Random Binaries\3edc853a-fa34-4ced-f9c3-08dee7957314\analysis\scripts> python .\decrypt_venomrat_config.py
[+] Settings Key (b64) : cFlrTU1YN25QN2Lkc85ZNDJWkY0R1FtUUCyNmgyY1Y=
[+] Master key (decoded ASCII): 'pYkMMX7nP7idsNYX2UZf4GQmQG26h2cV'
[+] Derived AES-256 key : e55cf0745105d264db810ffc87a6f65f88686cb476498f081abb35c06383df13
[+] Derived HMAC key : 54ce6ba854641e7781ad32c2d80e2af5...fa9d321acc5df4b5

Port_s      = '4449'
Hos_ts      = '154.36.188.201'
Ver_sion    = 'Venom RAT + HVNC + Stealer + Grabber v6.0.3'
In_stall    = 'false'
MTX         = 'xkoqvgcojtqgi'
Paste_bin   = 'null'
BS_OD       = 'false'
Group       = 'Default'
Anti_Process = 'false'
An_ti       = 'false'
Server_signa_ture = 'NDuphx4rX1LNo01EF+JPoy+0zhpc83KAKj4LLwRl/zmuGZUhd2uweq3904P40faauQ2wF39m0ocS2Y45FPC6k5VX50+nhx78L7bXZ5PMmEgVTHhNg2raChiuAoz/P2bF1RYFprUhzBTBSJ/4RvSi+XyY
GUnETxHPA3897nqBV0U='

[+] Server certificate : 571 bytes -> server_cert.der
subject : CN=VenomRAT
issuer : C=CN,L=SH,O=VenomRAT By qwqdanchun,OU=qwqdanchun,CN=LMTEAM Server
not_before : 2022-08-26 09:24:50+00:00
not_after : 2033-06-04 09:24:50+00:00
serial : 9eb75dc5944b01d47dc5f31ffc5f52f6e7ec0b03
sig alg : 1.2.840.113549.1.1.13
PS C:\Users\resetti\Documents\shared\malware\12\others\1Random Binaries\3edc853a-fa34-4ced-f9c3-08dee7957314\analysis\scripts> |
```

Decrypting the configuration reveals hard-coded command-and-control details and campaign metadata.

Field	Value
Hos_ts	154.36.188.201
Por_ts	4449
Ver_sion	Venom RAT + HVNC + Stealer + Grabber v6.0.3
MTX	xkoqvgcojtqgi
Group	Default
In_stall / BS_OD / An_ti / Anti_Process	false
Paste_bin	

(Note the version banner has **two** spaces before v6.0.3 – matters if you’re grepping for it.)

Every anti-analysis/install/process-kill toggle is false. The RAT config is *stripped* because the loader stage already ran anti-VM, established persistence, and elevated. The crypter handles evasion + install, VenomRAT handles C2 + capabilities.

And the version banner is an indicator too: that’s the builder banner from the “qwqdanchun” redistribution of VenomRAT on GitHub.

Beyond the Initial Sample

Static analysis stopped at the C2. The next question was whether that C2 leads back to a named actor. Short answer is **almost**. The chain slots neatly into a well-documented China-nexus playbook targeting Indian tax entities this season, but the specific tools don’t match any single named group’s arsenal, so this is a cluster-adjacent attribution.

The most productive pivot was **the VenomRAT builder cert** (SHA1 009a05eaf082d20ae13a65abda12afd959cd76de). Feeding that thumbprint to VT Intelligence returned five siblings – three ISO loaders (Tax_Notice_16695.img, ITDENF2026-4281.img, Tax_436454367.img), one loose VenomRAT payload, and one 2023 outlier.

behaviour:"009a05eaf082d20ae13a65abda12afd959cd76de"

Save search Build your Search

IOCS ~37 COMMENTS 0

Collapse filters

Sort by Export Tools

Filters

IoC type

Files ~37

Examples

Have 5 or more detections

Have distribution vectors

Have threat network infrastructure

Have sandbox detonation report

Have community comments

Seen in the last month

Matches - 20/~37 Files	Detections	First seen	Last seen	Submitters
75bed93fbb1161646652337929148e10b8011df6db3c7289d0b100a3f609066a Tax_Notice_82335.img isoimage spreader contains-pe malware	22 / 62	2026-08-03 18:50:39	2026-08-03 18:50:39	1
454f585b6276a2bf24aeac35e4593b2ca6f94623e6eb111a24a6a7f862f0c708 सुना क्रमांक ITDENF2026-4281/Tax_Notice_89574.img isoimage spreader detect-debug-environment calls-wmi contains-pe checks-cpu-name ...	30 / 56	2026-07-13 06:56:53	2026-07-29 14:53:19	5
be5d73b5ce22e1de26d926defaddc02fd049dbdffdeb204676615ab947646fa सुना क्रमांक ITDENF2026-4282/Tax_Notice_73540.img isoimage malware contains-pe spreader	34 / 62	2026-07-24 12:50:15	2026-07-24 12:50:15	1
0c0e4935f8df04e86056798d417bf3f485c640fa0322756c0a860af2a6bd0f1d Decrypted.exe peexe 64bits checks-network-adapters spreader assembly	39 / 70	2023-11-08 00:08:37	2026-07-23 09:22:29	6
72a321802d738e8bc6a0ab9d9d24b380be944ffffbfc85094b889690b679a73c2 ClientAny.exe peexe calls-wmi malware assembly checks-user-input payload	61 / 70	2026-07-23 08:47:02	2026-07-23 08:47:02	1
268a90d07cf02b16c091ef6cc52e3dbe17638bb72221807bc833c4c1299a8a9 Tax_Notice_16695 (1).img isoimage spreader contains-pe malware detect-debug-environment long-sleeps	22 / 62	2026-07-23 08:41:50	2026-07-23 08:41:50	1

Pivoting on the builder certificate uncovers additional related samples and infrastructure.

All three ISOs landed on VT the same day as mine, probably several people pulling the link on the same day. The 2023 sibling is useful, because it's Chinese (*"November payment, please check receipt [PC version]"*): the same qwqdanchun builder has been in service across Chinese-language crimeware since at least late 2023.

Pivoting on the C2 IP 154.36.188.201 turned up two more Tax_Notice_*.img samples, and the passive-DNS history is unambiguous – the IP has hosted Chinese sites since 2021 (1.ccew.cc, 0769mybj.com, 3158mz.com; "0769" is Dongguan's area code). A Chinese VPS reused for VenomRAT C2.

154.36.188.201			
2026-06-16	0 / 61	HTML	python-aur-api-client
Communicating Files (5) ⓘ			
Scanned	Detections	Type	Name
2026-09-08	34 / 62	ISO image	Tax_Notice_23665.img
2026-09-05	51 / 71	Win32 DLL	nvdahelperremote.dll
2026-09-03	39 / 71	Win32 DLL	nvdaHelperRemote.dll
2026-07-23	17 / 70	Win32 DLL	rcdll.dll
2026-07-23	7 / 62	ISO image	Tax_Notice_99674.img
Files Referring (7) ⓘ			
Scanned	Detections	Type	Name
2026-09-08	32 / 63	ZIP	D:\auto_black_abuse\resources\unzi
2026-09-08	34 / 62	ISO image	Tax_Notice_23665.img
2026-09-05	51 / 71	Win32 DLL	nvdahelperremote.dll
2026-09-03	39 / 71	Win32 DLL	nvdaHelperRemote.dll
2026-07-27	42 / 69	Win32 DLL	C:\Windows\67ucf.exe
2026-07-15	14 / 62	Win32 DLL	C:\Windows\paj05pgeh.exe
2026-09-04	0 / 61	XML	xl/sharedStrings.xml

Additional samples were identified through infrastructure associated with the campaign's command-and-control server.

Silver Fox? Where Does PAPERMILL Fit?

This is where attribution becomes frustrating. Technically, we have a well-understood chain. We know how execution occurs, we know how the payload is protected, and we know where command-and-control communication will eventually land. What we don't have is the evidential bridge between infrastructure and a confidently attributed threat actor.

This is an increasingly common situation in modern malware analysis. Operators frequently share tooling, borrow builders, reuse loaders, and copy tradecraft from one another. As a result, understanding *what happened* is often significantly easier than proving *who did it*. In this case, the campaign sits close enough to known activity to be interesting, but not close enough to make a defensible attribution claim

Public reporting on tax-notice DLL-sideload campaigns is dense and converges on one broad cluster. Silver Fox (also tracked as SwimSnake / UTG-Q-1000 / Void Arachne / "Valley Thief"), a China-nexus cybercrime group active since 2022 is a likely connection here. Seqrite/Mallory attributed a look-alike sample (Tax_Assessment.exe + libsvcs.dll > XWorm/DcRAT) to Silver Fox with medium-to-high confidence, and CYFIRMA's Operation DragonReturn (May-Jul 2026) tracked the same class of activity with possible overlap.

So: is *this* sample Silver Fox, a splinter, or a separate opportunist riding the same wave?

Element	Silver Fox (documented)	This sample
---------	-------------------------	-------------

Sideload host	thunder.exe (Xunlei download manager)	Notepad++ GUP.exe
Sideload DLL	libexpat.dll	libcurl.dll
Container	.zip + NSIS installer, or .vhdx	.img ISO9660 built with pycdlib
Middle stage	Direct memory injection / explorer.exe hollow	Donut 0.9.2 x64 PIC loader
Final stage	ValleyRAT / Winos 4.0	VenomRAT v6.0.3 (qwqdanchun/LMTEAM build)
Delivery domains	ggwk.cc, govtop.one, harivo.vip, kjfuwyce.love	dsfgssd.uk.cc, dgdskfds.uk.cc, .xin cluster
C2	223.26.63.40, 103.97.131.179, 103.231.12.27	154.36.188.201:4449, 103.119.15.189

Every row is different. The class of activity (India ITD lure + container-based MOTW bypass + DLL sideload + in-memory .NET RAT + Chinese-nexus infrastructure) is a Silver Fox family fingerprint, but the specific implementation is a variant, not a repetition.

Our best-supported call, in decreasing order of confidence:

- **China-nexus, financially-motivated commodity crimeware, targeting Indian entities** with High confidence. Chinese bulk mailer, Chinese-preferred registrar, Chinese-hosted C2 with a five-year history of Chinese resolutions, Shanghai-authored VenomRAT builder, and a builder cert previously seen in a Chinese-language finance phish. Every third-party control-plane touch point in this operation is Chinese-language-native.
- **Silver Fox-adjacent / same tax-lure playbook** with Moderate-to-high confidence. Silver Fox's canonical India campaign uses the “कर लेखापरीक्षा” (Tax Audit) lure theme, a direct match to this subject line. The whole delivery-and-loading class matches; only the specific tools are substituted.
- **Silver Fox splinter or post-arrest successor** with Moderate confidence. A substitute stack, VenomRAT via a public builder, a different sideload pair, is exactly the shift a surviving operator would make after the 2025 arrests, and the 5-month gap since the last documented Silver Fox India wave fits reconstitution. This is supported fairly-well by pattern, but unfortunately not provable from static analysis.
- **Independent opportunist riding the wave** with Low-to-moderate confidence. This stack is all public and commodity, technically, any Chinese-language operator could set it up. The mailer slips ({{seq}}, and the localhost beacon), suggest someone running a rented toolkit rather than a development shop.

We have dubbed this cluster **PAPERMILL**, because a lot of the stages to this campaign are generated by pre-made builder tools, like for example, the final VenomRAT payload. The actors simply just have to churn out new payloads for new campaigns.

The honest takeaway is that this is broadly Silver Fox-like activity. However, we cannot yet confidently attribute it to a specific actor, and so we are assigning the cluster a new designation. Whether this represents a splinter group, a successor operation, or an entirely separate actor drawing from the same ecosystem remains unclear. We will continue tracking related activity as it emerges and hope the indicators and analysis presented here help others further develop the picture.

Conclusion

The chain hides five layers deep behind a legitimately signed binary the operator did not modify by a single byte, and every layer does exactly one job, but the interesting thing in this campaign in my opinion is the timestamp. Two numbers three days

apart in a PE header led to the ISO's creation date matching the DLL's export directory to the second, which proved the container is minted **per download**, three of the five file hashes I'd have happily handed a client as an IOC package are unique to my own fetch and would never have matched a second victim.

The builder in this campaign is what wins in a sense, hash-based detection loses by default against it. Since everything is unique, therefore, the cert thumbprint, the behaviour, and detection engineering like YARA rules are what survive.

Indicators of Compromise

Email / delivery

Type	Indicator
Sender address / DKIM domain	dfgfasd@hsaii[.]cc
Sending IP	155.94.154.195
Sending MTA (HELO/PTR)	mos1.17dlz[.]cn
Bulk-mailer auth host	smtp.smtpman[.]cn
Payload/landing URL	hxxps://dsfgssd[.]uk[.]cc/ → hxxps://dsfgssd[.]uk[.]cc/download[.]php
Tracking-beacon path	/api/maier/open?task=<GUID>&t=<token> (host left as localhost:6688)
Lure subject	कर लेखापरीक्षा परिणाम No. ITD/SCNU/2026-<seq>
Mailer tell	unrendered {{seq}} in From display name

Malicious File Hashes

File	SHA256
Tax_Notice_45594.img (ISO container)	f945b3f2f29d62099683cbbf069ed5bc952d089c16a85fb314ac8fe0f0c10d2b
Tax_Notice_45594.exe (signed Notepad++ launcher)	fe0ddd8686324e0a8b07ed5ffdb4f56397c70b1b5a19d2916c8eb192583f950c
libcurl.dll (proxy loader)	8c1ba078598e09294d72293a42dcd878e183b6e5a207315f7bf6ef74a966cbb5
LIBCURL.DAT (encrypted payload)	2ff898c1a4bb0dd48687bbbc8cca646a896dc9ed24780a49523dfbf9fb94a4ec
stage2.dll (VenomRAT .NET DLL, post-Donut)	947221d0f1e2c9c09028491997406dd9f0cb5b65258556c86d201ec1ca538f5f

Host / behavioral

Type	Indicator
Install directory	%APPDATA%\Microsoft\Crypto\RuntimeBroker\
VenomRAT runtime dir	%APPDATA%\MyData\

Dropped files (loader)	RuntimeBroker.exe, libcurl.dll, libcurl.dat
Staging temp names	*_dbg_src.tmp, *_dbg_dst.tmp
Persistence key	HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce → ... \RuntimeBroker\RuntimeBroker.exe
Masquerade	RuntimeBroker.exe (impersonates Windows Runtime Broker)
Internal DLL name	libcurl.dll_38768.dll — hunt the pattern libcurl.dll_<digits>.dll, the number is per-build
Loader module base (memory)	maps at exactly 0x00000002FFF20000 (no relocations) inside GUP.exe / RuntimeBroker.exe
Loader section names	.nvdata ×3, .nvtext, .pdata — no exception directory despite the .pdata name
VenomRAT mutex	xkoqvgcojtggi
VenomRAT HVNC worker	Kills any cvtres.exe process on reconnect
VenomRAT anti-kill	RtlSetProcessIsCritical(TRUE) — BSODs on kill if BS_OD=true (off in this build)

C2 / network

Type	Indicator
C2 endpoint	154.36.188.201:4449
VenomRAT campaign group	Default
VenomRAT master key (b64)	cFIRtU1YN25QN2lk05ZWDJvWkY0R1FtUUcyNmgyY1Y= → ASCII pYkMMX7nP7idsNYX2UZf4GQmQG26h2cV
PBKDF2 salt	VenomRATByVenom (50000 iters, SHA-1 HMAC)
Version banner	Venom RAT + HVNC + Stealer + Grabber v6.0.3 (two spaces before the version)

Builder attribution / pivot

Type	Indicator
Server X.509 subject / issuer	CN=VenomRAT / C=CN, L=SH, O=VenomRAT By qwqdanchun, OU=qwqdanchun, CN=LMTEAM Server
SHA1 thumbprint (pivot)	009a05eaf082d20ae13a65abda12afd959cd76de – same across every sample built with this qwqdanchun-modded v6.0.3 builder
SHA256 thumbprint	f72bf7c1fd132262715820d839e4874c97d927b10072fb4709a1321927fea66d

Sibling samples (same builder cert, uploaded 2026-07-23)

SHA256	Filename	VT label
72a321802d738e8bc6a0ab9d9d24b380be944ffbfbc85094b889690b679a73c2	ClientAny.exe / skkr3.exe	trojan.asyncrat/msil (VenomRAT)

268a90d07cf02b16c091ef6cc52e3dbe17638bb722221807bc833c4c1299a8a9	Tax_Notice_16695 (1).img	trojan.shellcode/dllhijack
f23708ae470904490cb25d370185ca89c0d37e33fa5a3c652e4b10e65d0b08e3	ITDENF2026- 4281.img	trojan.dllhijack/shellcode
d768222934f6014bf17c3d0a1ef4c35e02aa6fe1e89564e252ca5e7b7c7fe237	Tax_436454367.img	trojan.dllhijack/shellcode
0c0e4935f8df04e86056798d417bf3f485c640fa0322756c0a860af2a6bd0f1d	11月已付款,回单请查 收[电脑版].exe (2023)	trojan.msil/dinvoke
38ec1f5e23f65b10ae3027beabfa0bf7f9fb686355a9e33c7e7e44e6a998e04c	Tax_Notice_23665.img (2026-07-15, contacts our C2)	trojan.shellcode/misc
a4098fe9ed421a6244c9fd4d3d1a632ff5f2ee4ffa204dff0587548932936d38	Tax_Notice_99674.img (2026-07-23, contacts our C2)	trojan.misc/donut

Sister delivery infrastructure (July 2026 waves)

Type	Indicator
Sister lure domain	dgdsksfds[.]uk[.]cc (Cloudflare, 17 VT malicious, waves 2026-07-09→23)
Sister delivery hosts	aidenllc[.]com (AWS ap-south-1), gov-xnui[.]com, xjxfxn[.]com (Tencent HK), jfbcea[.]com (BGPNET SC)
.xin DGA cluster (SG)	tzawccsw[.]xin, zxizusuy[.]xin, zasudtytw[.]xin, zixhasda[.]xin, pzisiauywa[.]xin, gisudyawz[.]jink — all resolve to 103.23.172.15
Sister lure title	कर दंड सूचना – भारत सरकार (“Tax Penalty Notice — Government of India”)
Secondary C2 IP (sibling)	103.119.15.189 (Turing Group AS140869)
Related NEET-phish cert	ekl1-neettr[.]bond on sender IP 155.94.154.195

Contributors

Jack Lewis Threat Research & Analysis

Sean Moran Editorial Review

Dhaval Mukeshkumar Sample Discovery & Initial Investigation

The intelligence presented in this report originated from the detection of previously undocumented malware by JUMPSEC’s MXDR team. The sample was initially identified and investigated by Dhaval Mukeshkumar before being escalated for deeper threat research and analysis. Further information on JUMPSEC’s incident response, incident exercising, and detection and response services is provided below.



Jack Lewis

Research & Analysis

Jack is a security researcher with a strong focus on malware analysis, tracking new threat actors and campaigns, reverse engineering, patch diffing, and proactive threat hunting.

[Managed Detection and Response \(MDR\) Services](#)

Managed Detection and Response (MDR) Services

Read More

[JUMPSEC announces its Assured Service Provider status for NCSC's Cyber Incident Exercising \(CIE\) Scheme](#)

JUMPSEC announces its Assured Service Provider status for NCSC's Cyber Incident Exercising (CIE) Scheme

JUMPSEC today announces it has become one of only a...

Read More

[Guide to Incident Response](#)

Guide to Incident Response

Why do you need Cyber Incident Response? A cybersecurity incident...

Read More

[Load More](#)