



Aurora

Technical Analysis — Windows

REVERSE-ENGINEERED REPORT

RansomLook · ransomlook.io

File last modified: 2026-04-30

Analysis date: 2026-04-26

Sample SHA-256: **81ca5fc6b55accdbc44266d66bd72c7c4152a75b215593adc433d51250054333**

Aurora Ransomware

1. Sample Identification

Field	Value
Family	Aurora (self-identified as <code>encrypter</code>)
SHA-256	<code>81ca5fc6b55accdbc44266d66bd72c7c4152a75b215593adc433d51250054333</code>
MD5	<code>fa0913bfe83efe092998da70975b5918</code>
Type	PE32+ x86-64, Windows GUI
Size	157,696 bytes (154 KB)
Language	C/C++ compiled with Mingw-w64 (cross-platform codebase — Windows/Linux/ESXi targets share source)
Compile timestamp	<code>0x86C6787B</code> — reproducible-build file hash , not a real Unix timestamp (mingw/ld with <code>SOURCE_DATE_EPOCH</code> or <code>--build-id=sha1</code>). Same value appears in the <code>.buildid</code> section, confirming reproducible build. Cannot be used for temporal correlation.
PDB path	None (no RSDS record, no <code>.debug_info</code> — stripped mingw build)
Image base	<code>0x140000000</code>
Sections	7 in <code>objdump -h</code> : <code>.text</code> , <code>.rdata</code> , <code>.buildid</code> , <code>.data</code> , <code>.pdata</code> , <code>.tls</code> , <code>.reloc</code>
PE security	<code>DllCharacteristics = 0x8160</code> → HIGH_ENTROPY_VA + DYNAMIC_BASE (ASLR), NX_COMPAT (DEP), TERMINAL_SERVICE_AWARE . No Control-Flow-Guard, no GuardCF, no SafeSEH. No /GS cookie check typical of mingw.
.pdata entries	403 <code>RUNTIME_FUNCTION</code> records (unwind info for exception handling)
Subsystem	2 (Windows GUI) — despite the binary using <code>stdout/stderr</code> consistently via <code>__acrt_iob_func</code> + <code>fwrite</code> . This is mingw's default for GUI builds; log lines land in whatever console launched the process, or are lost when started from explorer
Functions	458 total
TLS callbacks	3 (<code>0x140006B90</code> , <code>0x14001B940</code> , <code>0x14001B9C0</code>) — standard Mingw TLS infrastructure

Aurora is a multi-platform ransomware (Windows/Linux/ESXi) shipped in this sample as a Windows x64 build. It embeds a single 4096-bit RSA public key for per-file key wrapping, uses ChaCha20 for bulk encryption via an SSE2-optimised implementation, and ships mbedTLS as the crypto library. The binary has no anti-debug, no anti-VM, no string obfuscation, and no exfiltration component. It shell-outs to `vssadmin` / `wmic` / `schtasks` for shadow copy destruction and relies on Windows privilege elevation plus ACL rewriting to bypass file permissions. The banner string `Mode: ENCRYPT (compiled-in)` indicates a build-time switch — a matching DECRYPT-mode binary presumably exists in the attacker's toolchain.

Imports (5 DLLs)

DLL	Count	Purpose
KERNEL32	48	Threading (<code>CreateThread</code> , mutexes, CS, events, TLS), file I/O (<code>CreateFileW</code> , <code>FindFirstFileW</code> , <code>ReadFile</code> , <code>WriteFile</code> , <code>SetEndOfFile</code>), drives (<code>GetLogicalDrives</code> , <code>GetDriveTypeA/W</code>), memory (<code>VirtualProtect</code> , <code>VirtualQuery</code>), env (<code>GetEnvironmentVariableW</code>), time (<code>GetTickCount</code>)
ADVAPI32	10	Privilege elevation (<code>OpenProcessToken</code> , <code>LookupPrivilegeValueW</code> , <code>AdjustTokenPrivileges</code>), ACL manipulation (<code>AllocateAndInitializeSid</code> , <code>SetEntriesInAclW</code> , <code>SetNamedSecurityInfoW</code>), RNG (<code>CryptAcquireContextA/W</code> , <code>CryptGenRandom</code>), registry (<code>RegGetValueW</code>)
MPR	3	SMB share enumeration (<code>WNetOpenEnumW</code> , <code>WNetEnumResourceW</code> , <code>WNetCloseEnum</code>) for lateral encryption
SHELL32	1	<code>CommandLineToArgvW</code> (UTF-16 CLI parsing)
MSVCRT (api-ms-win-crt-*)	~55	Standard C runtime (<code>_w fopen</code> , <code>fread</code> , <code>fwrite</code> , <code>malloc</code> , <code>wcscpy</code> , etc.)

Notably absent: - No COM/WMI imports (`OLE32` , `OLEAUT32`) → VSS deletion is shell-out only, no in-process WMI - No HTTP/socket (`WinHttp` , `WinINet` , `Ws2_32`) → **no C2, no exfiltration** - No `OpenSCManager` / `EnumServicesStatus` → **no service-stop logic** - No `CreateToolhelp32Snapshot` / third-party `TerminateProcess` targeting → **no process killer** - No `IsDebuggerPresent` , `NtQueryInformationProcess` , `CheckRemoteDebuggerPresent` → **no runtime anti-debug**

2. Infrastructure

Field	Value
Onion	<code>http://ijexszhscln27nl263lmcd7tx3jttkkm4wjhd4e3y6r4csdbfyepvid.onion</code>
Chat URL	<code>ijexszhscln27nl263lmcd7tx3jttkkm4wjhd4e3y6r4csdbfyepvid.onion</code>
Access key (per-build)	(32 hex = 16 bytes; embedded as plaintext in the note)
Note filename	<code>!!!README!!!DO_NOT_DELETE.txt</code>
Encrypted file extension	None (files kept under original name, only a 542+N byte footer appended)
Mutex	None — no <code>CreateMutexA</code> with a named mutex for single-instance; the only <code>CreateMutexA</code> call uses <code>lpName=NULL</code> inside the generic lock-init helper
Payment	Tor negotiation only

3. Ransom Note

Filename

`!!!README!!!DO_NOT_DELETE.txt` — dropped at: - `%SystemDrive%` (e.g. `C:\`) - Each drive root enumerated by `enumerate_default_paths` (fixed + network drives) - **Every directory successfully opened by the scanner threads**, unless flag `scanner_state[96] != 0` disables it

Deployed by `write_ransom_note` at `0x140002560`, using `_w fopen(path, L"w")` in text mode with LF→CRLF conversion during `fwrite` segments.

Content (extracted from `.rdata:0x14001EA60` , 226 bytes exact)

```
We have downloaded confidential information files. Your files are encrypted. Contact us via tor  
browser at http://ijexszhscln27nl263lmcd7tx3jttkkm4wjhd4e3y6r4csdbfyepavid.onion  
Your access key: [SNIP]
```

The note is a short two-liner: a data-theft claim (despite the absence of any exfiltration code in this binary), the Tor contact URL, and a 32-hex-character access key hardcoded at build time.

4. Execution Flow (`ransom_main` @ `0x1400052B0` , 2,835 bytes)

```

start (CRT) @ 0x14001B6A0
└─ sub_14001C370 (mingw wWinMain trampoline)
    └─ main_trampoline @ 0x140005260
        │ GetCommandLineW + CommandLineToArgvW
        └─ ransom_main(argc, argv) @ 0x1400052B0

ransom_main:
1. setlocale(LC_ALL, "")
2. SetConsoleCtrlHandler(ctrlc_handler, TRUE)
3. parse_args(&opts, argc, argv) @ 0x140001FA0
   └─ reads HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SystemRoot
   │ (converts \ → / for cross-platform path matching)
   └─ handles -path / -percent<N> @ sub_140001DC0
   └─ handles -f<N><K|M|G> (file size cap)
   └─ handles -threads <N>
   └─ handles -noparallel
   └─ handles -esxi
   └─ if no -path:
       └─ enumerate_default_paths(&opts) @ 0x140001830
           └─ probe A:..Z: via GetDriveTypeW (keep DRIVE_FIXED=3 + DRIVE_REMOTE=4)
           └─ WNetOpenEnumW + WNetEnumResourceW (SMB shares)
           └─ spawn up to 64 StartAddress threads (one probe per drive/share)
           └─ 5 s timeout per thread (terminate_hung_walker) @ 0x140001500
4. Privilege escalation:
   OpenProcessToken(TOKEN_ADJUST_PRIVILEGES|QUERY)
   LookupPrivilegeValueW("SeBackupPrivilege") + AdjustTokenPrivileges
   LookupPrivilegeValueW("SeRestorePrivilege") + AdjustTokenPrivileges
   g_privs_acquired = (both succeeded)
5. pk_init + pk_parse_public_key(g_rsa_pubkey_der, 550 B, .rdata:0x14001E370)
   pk_validate_rsa → exit(108) if invalid
   pk_set_padding(PKCS_V21, HASH=SHA512) @ 0x140016E20
6. drbg_random_global(&buf, 32) → sanity test
   if RNG returned all zeros: warn (continue anyway)
   if drbg call failed: warn
7. VSS / recovery destruction (skipped only in "help" mode) @ 0x1400062D0 run_shell_cmd
   a. "vssadmin delete shadows /all /quiet >nul 2>&1"
   b. "wmic shadowcopy delete >nul 2>&1"
   c. For each DRIVE_FIXED (A..Z):
       "vssadmin resize shadowstorage /for=%c: /on=%c: /maxsize=401MB >nul 2>&1"
   d. "schtasks /Change /TN \"\\Microsoft\\Windows\\SystemRestore\\SR\" /Disable >nul 2>&1"
   Each is spawned as `cmd.exe /C <cmd>` via CreateProcessA
8. Worker thread pool sizing:
   N = -threads argument if given, else max(4, 2×CPU cores)
9. thread_create(note_dropper_thread, opts) @ 0x140005E30 (1 thread)
10. thread_create(network_share_scanner_thread, opts) @ 0x140003490 (1 thread)
11. thread_create(scanner_thread, opts) × 3 @ 0x140005EE0
12. For i = 0..N-1:
    drbg_seed_thread(&worker_state[i], entropy, "thread_drbg_%d") @ 0x140011A10
    thread_create(worker_thread, &worker_state[i]) @ 0x140005210
13. WaitForMultipleObjects on all handles
14. log_stderr("Completed in %dh %dm %.1fs\n" ...)
15. Cleanup: cs_free × all, event_free × all, free worker buffers

```

4.1 Thread Architecture

- **1 note-dropper** → writes the note at `%SystemDrive%` and at each drive entry populated by `enumerate_default_paths`
- **4 directory scanners** (one of which specialises in SMB shares, spawned first with `sub_140003490 = network_share_scanner_thread`): consume a dynamic queue of 520-byte directory entries, iterate with `FindFirstFileW / FindNextFileW`, route subdirectories back onto the queue, route files into the worker queue
- **N encryption workers** (default $N = \max(4, 2 \times \text{CPU cores})$): consume 800-byte file tasks from the worker queue. Each worker owns its own CTR_DRBG context (personalization string `"thread_drbg_%d"`) to generate per-file symmetric key material

Two separate queues — directory queue (520 B entries, re-allocated at 2× capacity when full) and file queue (800 B entries, ring buffer with head/tail/count in a CS-protected tuple) — decouple traversal from encryption.

A "noparallel" flag (set by `-noparallel`) disables the large-file split-across-threads path (encryption of a single very-large file still happens, but linearly on one worker rather than striped).

4.2 Per-File Worker (`encrypt_worker_main` @ `0x140004090`, 3,081 bytes)

```

loop {
  cs_enter(queue_lock)
  pop 800 B task entry from ring buffer
  cs_leave(queue_lock)

  fh = open_file_take_ownership(path) @ 0x140004CA0
  footer_read_parse(&footer, fh) @ 0x140003580
  if footer.version==3 && footer.magic==0x4337944A:
    already encrypted → continue
  file_size = ftelli64(fh)
  chunk_plan_and_crypto_init(chacha_ctx, &footer,
    fh, file_size, opts.percent, ...) @ 0x140003D40
  └─ generate_key_rsa_wrap_chacha_init(...) @ 0x140003BB0
  │   drbg_random(&keymat, 40)
  │   mbedtls_rsa_pkcs1_encrypt(rsa_ctx, &keymat, 40, &footer.rsa_blob, 512, OAEP-SHA512)
  │   footer.version = 3, footer.magic = 0x4337944A
  │   chacha20_setkey(chacha_ctx, keymat) @ 0x140019D50
  └─ compute plan_block_size / read_block_size / percent (auto or override)
    allocate bitmap, store on-disk offset at file_size+534

  for each chunk in plan:
    chunk_read_if_not_marked(chunk_idx, total, bit_pos, mode,
      fh, &footer, file_size, &buf, chunk_size) @ 0x140003960
    if read OK:
      chacha20_starts(chacha_ctx, *buf_offset, ...) @ 0x140019D10
      chacha20_xor_stream(chacha_ctx, &chunk_data, size) @ 0x14001B080
      chunk_write_and_mark(&buf, bit_pos, &footer, fh, bitmap_fh) @ 0x140003A70
      └─ WriteFile positional at chunk_offset
      └─ set bit in in-memory bitmap AND write the updated byte to disk at
        file_size+534+byte_idx
    else:
      skip (resume-safe: bit already set)

  fclose(fh)
}

```

5. Encryption System

5.1 Key Exchange

Parameter	Value
Algorithm	RSA-OAEP (PKCS#1 v2.1)
Key size	4096 bits
MGF / Hash	MGF1-SHA512
Implementation	mbedtls (embedded statically)
Attacker public key	550-byte DER at <code>.rdata:0x14001E370</code>
Public key SHA-256	<code>3c954f9480b09fa5662e31e60f4db790d66b7f93fd2ee58eb8928c6c929ce785</code>
Modulus (first 16 B)	<code>ED 97 D7 D4 25 38 77 20 C9 56 1C 02 9A B4 EE 5C ...</code> (513 bytes total = 4096 bit + leading 0x00)

Extracted to `aurora_rsa_pubkey.der` (550 B raw DER) and `aurora_rsa_pubkey.pem` (PEM-wrapped SubjectPublicKeyInfo).

5.2 Symmetric Cipher

Parameter	Value
Algorithm	ChaCha20
Mode	Stream (XOR with keystream)
Key size	256 bits (32 B)
Nonce	64 bits (8 B) — original ChaCha20 variant, not the IETF 96-bit nonce
Counter	Starts at 0, incremented by 2 per <code>chacha20_xor_stream</code> call (the implementation processes two 64-B blocks per iteration via SSE2 interleaving)
Per-file material	40 B freshly generated by the thread-local CTR_DRBG — 32 B key
Implementation	SSE2-optimised (unambiguously identified from <code>_mm_shuffle_epi32</code> 147/78/57, rotations 16/12/8/7, 10 double-rounds, <code>_mm_add_epi64</code> counter increment by 2) — matches the SUPERCOP / Krovetz-style ChaCha20 SSE2 routine mbedtls uses on supported platforms

5.3 File Encryption Process

- Open with privilege bypass** — `open_file_take_ownership` tries `CreateFileW(GENERIC_READ|GENERIC_WRITE, SHARE_ALL, OPEN_EXISTING, FILE_FLAG_NO_BUFFERING)`; on `ERROR_ACCESS_DENIED` and if `g_privs_acquired==0` (SeBackup/SeRestore fallback path), enable `SeTakeOwnershipPrivilege`, `AllocateAndInitializeSid(BUILTIN\Administrators = S-1-5-32-544)`, `SetNamedSecurityInfoW(OWNER)`, build a new DACL granting Admins full access, `SetNamedSecurityInfoW(DACL)`, retry.
- Parse footer** — seek EOF-8, read 8-byte size qword; if `size <= (file_size >> 15) + 3`, seek back `542 + size` bytes and read the full trailer into the footer struct.
- Skip already-encrypted** — if `footer.version == 3` and `footer.magic == 0x4337944A`, return (resume only re-processes unmarked chunks).

4. **Generate fresh key material** — `drbg_random(&keymat, 40)` via the worker's per-thread CTR_DRBG.
5. **Wrap key** — `mbedtls_rsa_pkcs1_encrypt(g_rsa_pubkey, keymat, 40, &ct, 512, OAEP-SHA512, drbg_random_for_rsa)` → 512-byte ciphertext stored at `footer.rsa_blob`.
6. **Set markers** — `footer.version = 3`, `footer.magic = 0x4337944A`.
7. **Init ChaCha20** — `chacha20_setkey(ctx, keymat)` (first 32 B key, then 8 B nonce from offset 32..40, counter = 0).
8. **Compute chunk plan** — see §5.4 (auto-percent by file size or `-percent` override; sqrt-distributed positions).
9. **Allocate bitmap** — `calloc((chunks+7)/8)`, store on-disk bitmap offset = `file_size + 534`.
10. **Chunk loop** — for each planned chunk: check bitmap bit; if unset, positional `ReadFile`, `chacha20_starts(offset)`, `chacha20_xor_stream(buf, size)`, positional `WriteFile`, set bit in memory AND append the updated bitmap byte to disk at the dedicated offset.
11. **Close file** — `fclose`; on a completion flag path, `SetEndOfFile` can be invoked (code path exists but conditional — destructive truncation mode).

5.4 Intermittent Encryption (Auto Profile)

`chunk_plan_and_crypto_init` at `0x140003D40` computes the chunk plan based on raw file size unless `-percent<N>` overrides:

File size	Auto % encrypted	Plan block	Read block
< 1 KB	100 %	full	4 KB
1 KB - 1 MB	100 %	1 KB	4 KB
1 - 5 MB	80 %	512 KB	64 KB
5 - 50 MB	30 %	1 MB	64 KB
50 - 500 MB	15 %	100 MB	256 KB
500 MB - 1 GB	10 %	200 MB	256 KB / 1 MB
1 - 20 GB	5 %	500 MB	256 KB / 1 MB
> 20 GB	3 %	500 MB	1 MB

Chunk positions within the plan are **non-uniform** — a `sqrt(x)`-based recurrence (`v78 = sqrt(v77); v79 = (int)(v77 * v78 * v47)`) that concentrates chunks in the first/middle portions of the file. Bitmap is persisted to disk chunk-by-chunk → a killed process can be resumed with all previously-completed chunks skipped.

5.5 Encrypted File Format (footer, 542 + N bytes)

offset	size	field	notes
file_size	1 B	version	= 3 (current format)
+ 1	4 B	magic	= 0x4337944A (LE bytes 4A 94 37 43)
+ 5	512 B	rsa_blob	AES-4096-0AEP-SHA512 of (32 B ChaCha20 key 8 B nonce)
+517	8 B	plan_block_size	fully-encrypted span size per "block"
+525	8 B	read_block_size	I/O chunk granularity used during encryption
+533	1 B	percent	the auto or -percent value (1..100)
+534	N B	bitmap	1 bit per chunk, set after chunk is encrypted
+534+N	8 B	bitmap_size	redundant copy of N as a qword (used by footer parser)

Total footer size = 542 + N

Bitmap size $N = \text{ceil}(\text{chunk_count} / 8)$, and $\text{chunk_count} \leq (\text{file_size} \gg 15) + 3$ (sanity bound enforced by the footer parser).

5.6 Skip Rules (encryption side)

- Already-encrypted marker (footer version 3 + magic 0x4337944A) → skip entirely
- File open failure after triple-bypass → skip
- No explicit minimum file size enforced in encryption (the auto-profile handles sub-1 KB files by encrypting them fully)

6. File Targeting

6.1 Targeted Extensions

All file extensions are encrypted **except** those in the skip list (§6.3) — unless `-esxi` is given, in which case only the ESXi allowlist is encrypted.

6.2 Excluded Directories (`is_excluded_dir @ 0x1400030B0` , case-insensitive exact basename match)

Directory	Purpose
Windows	OS files
Boot	bootloader
\$Recycle.Bin	recycle bin metadata
System Volume Information	VSS / restore points
inetpub	IIS
\$WinREAgent	Windows Recovery Environment
PerfLogs	performance counters
MachineKeys	cryptographic key store

The scanner also skips `.` and `..` entries (inline compare) and, in ESXi mode, any directory whose name starts with `BOOTBANK` (inline qword compare `0x4B4E4142544F4F42`) or `OSDATA` (inline dword `0x4144534F` + word `0x4154`). The two strings `"BOOTBANK"` / `"OSDATA"` are present in `.rdata` at `0x14001EB45` / `0x14001EB4E` for the help text but have **no code xref** — the comparisons are fully inlined as immediate constants.

6.3 Excluded Extensions

`g_skip_extensions` at `.rdata:0x14001ECC0` (case-insensitive match on the file-extension tail):

- `exe`
- `dll`
- `iso`

Only 3 extensions skipped — Aurora is extremely aggressive. Documents, databases, media files, archives are all fair game.

6.4 ESXi Allowlist (only when `-esxi`)

`g_esxi_extensions` at `.rdata:0x14001EC70`:

- `vmdk` — virtual disks
- `vmx` — VM config
- `vmsd` — snapshot metadata
- `vmsn` — snapshot state
- `nvram` — BIOS/UEFI state
- `vmem` — paged-out RAM
- `vswp` — swap file
- `log` — ESXi logs

When `-esxi` is active, files **not** matching any of these are skipped entirely (allowlist mode).

6.5 Excluded Files (by exact name)

File	Function
<code>!!!README!!!</code> <code>DO_NOT_DELETE.txt</code>	Own ransom note (skipped by <code>file_filter_and_enqueue</code> at <code>0x140003170</code> via ASCII <code>strcmp</code>)
<code><own_exe_path></code>	The encrypter binary itself (skipped by <code>wcscmp</code> against <code>GetModuleFileNameW</code> result)

6.6 Drive Enumeration

`enumerate_default_paths` iterates A..Z, keeps drives returning `DRIVE_FIXED (3)` or `DRIVE_REMOTE (4)` from `GetDriveTypeW`. Then calls `WNetOpenEnumW(RESOURCE_CONNECTED, RESOURCETYPE_DISK)` and `WNetEnumResourceW` to list SMB shares and appends their UNC paths. Per drive/share, spawns a probe thread (`StartAddress` → `sub_140001070` → `FindFirstFileW` + early exit). Parent waits up to **5 seconds total** across all probes; any thread not done by then is treated as dead and its entry is excluded (avoids hangs on dead network shares). Maximum 64 drives tracked.

7. Recovery Inhibition

All shadow copy / system restore destruction is performed via **shell-out only** (`run_shell_cmd` at `0x1400062D0` wraps `CreateProcessA("cmd.exe /C <command>")`). Executed early in `ransom_main` before the worker pool spins up:

#	Command	Effect
1	<code>vssadmin delete shadows /all /quiet >nul 2>&1</code>	Delete all VSS shadow copies
2	<code>wmic shadowcopy delete >nul 2>&1</code>	Secondary VSS wipe via WMI CLI
3	<code>vssadmin resize shadowstorage /for=%c: /on=%c: /maxsize=401MB >nul 2>&1</code> (per <code>DRIVE_FIXED</code>)	Force storage eviction of any remaining shadow copies by setting a tight cap
4	<code>schtasks /Change /TN "\Microsoft\Windows\SystemRestore\SR" /Disable >nul 2>&1</code>	Disable the scheduled System Restore task

No direct in-process COM/WMI calls. No `wbadmin`, no `bcdedit`, no Windows Backup service tampering.

8. Targeted Services

None. No `OpenSCManager`, `OpenServiceW`, `ControlService`, or `EnumServicesStatusW` imports. Aurora does not stop any service before encryption.

9. Targeted Processes

None. No `CreateToolhelp32Snapshot`, `Process32FirstW`, `Process32NextW` imports. No hard-coded process blacklist. The only `TerminateProcess` use is on `GetCurrentProcess()` (self-terminate on error).

10. Persistence & Evasion

10.1 Single-Instance / Mutex

None. The only `CreateMutexA` call in the binary is inside the generic lock helper `cs_init` (at `0x140006490`), used for thread synchronisation when the caller passes flag bit 0 set. The call always uses `lpName=NULL` (unnamed mutex), so it is not a single-instance gate. Multiple copies of Aurora can run concurrently.

10.2 Privilege Elevation

- `SeBackupPrivilege` + `SeRestorePrivilege` enabled globally in `ransom_main`, result cached in `g_privs_acquired` at `0x140027054`. Together these bypass NTFS ACLs for read and write on the encryption path.
- `SeTakeOwnershipPrivilege` lazily enabled in `open_file_take_ownership` (fallback path when one of the above failed). Combined with `SetNamedSecurityInfoW` OWNER + DACL rewrite to grant `BUILTIN\Administrators` full access, then retries `CreateFileW`. Triple bypass.

10.3 ACL Rewriting

`open_file_take_ownership` at `0x140004CA0` is able to take ownership of and rewrite the DACL of any file on which the current process has at minimum `WRITE_OWNER` access. After rewriting, the file belongs to `BUILTIN\Administrators` and the original owner is gone. This is an irreversible side effect of encryption — even after a successful decryption, file ownership would not be restored without manual intervention.

10.4 Anti-Analysis

- **No anti-debug** — no `IsDebuggerPresent` , `CheckRemoteDebuggerPresent` , `NtQueryInformationProcess(ProcessDebugPort)` , PEB BeingDebugged check, INT3/INT2D trap-flag tricks, nor any `GetTickCount` -based timing detection (the two `GetTickCount` calls are for workload profiling, not evasion).
- **No anti-VM** — no `cpuid` hypervisor bit check, no BIOS/registry fingerprinting, no VMware/VBox driver presence check.
- **No anti-sandbox** — no "maltest/sandbox" username checks, no sleep-skip detection, no analyst-tool process lookup.
- **No string obfuscation** — all strings (ransom note, commands, mbedTLS error codes, API names) are plain UTF-8 / UTF-16 in `.rdata` .
- **No dynamic API resolution** — all imports statically linked; no `GetProcAddress` / `LoadLibrary` hashing.

10.5 File Operation Hygiene

- `FILE_FLAG_NO_BUFFERING` used when opening files for encryption → bypasses the OS page cache, reduces memory pressure on the victim, minimises observable I/O latency.
- Chunk bitmap persisted to disk after every chunk → resume-safe even after a hard kill.

10.6 Self-Delete

Not observed. No PowerShell self-delete, no `MoveFileExW(MOVEFILE_DELAY_UNTIL_REBOOT)` , no `DeleteFileW(GetModuleFileNameW(NULL))` . The encrypter binary persists on disk after the run completes.

10.7 Anti-Analysis Summary

Technique	Unprotect ID	Address	Description
Privilege escalation (SeBackup/SeRestore)	U0121	0x140005320..0x140005440	Bypass NTFS ACLs for read and write on encryption path
Take-ownership fallback	—	open_file_take_ownership @ 0x140004CA0	Rewrites file owner to Administrators + DACL rewrite if ACCESS_DENIED
VSS destruction (shell-out)	U0305	ransom_main @ 0x1400054F6..0x1400055DD	vssadmin / wmic / schtasks via cmd.exe /C
SMB share lateral encryption	U1016	enumerate_default_paths + network_share_scanner_thread	WNetEnumResourceW walks mapped shares, scanner treats them like local drives
FILE_FLAG_NO_BUFFERING	—	0x140004CE0, 0x140004F55	Bypass page cache on file open
Intermittent encryption (partial+sqrt distribution)	U1523	chunk_plan_and_crypto_to_init @ 0x140003D40	Non-uniform chunk positioning for fast encryption of large files
Resume-safe bitmap	—	chunk_write_and_mark @ 0x140003A70	Per-chunk bitmap persisted to disk at file_size+534
Build-time mode switch	—	banner Mode: ENCRYPT (compiled-in) @ 0x14001E5DB	Separate DECRYPT build presumed

11. Command-Line Arguments

From `parse_args` (0x140001FA0) and the embedded usage text at `.rdata:0x14001E5C0` :

Argument	Storage	Description
<code>-path <folder></code>	<code>opts+24</code> (LPWSTR → drive_entry buffer, 1144 B)	Restrict to a single folder; disables <code>enumerate_default_paths</code> . Handled via <code>parse_path_percent sub_140001DC0</code> which <code>malloc(0x478)</code> then calls <code>drive_walker_init(entry, argv[i+1])</code> ; any failure → usage error and <code>exit(101)</code>
<code>-percent<N></code>	<code>opts.percent_by te</code>	Override the auto-derived percentage (0..100). <code>0</code> keeps auto behaviour. Parsed by the same <code>parse_path_percent</code> : <code>wcsncmp(arg, L"-percent", 8)</code> ; numeric suffix extracted via <code>wcstombs(arg+16, ...)</code> + <code>atol</code> ; rejected if <code>N >= 101</code> (strict less-than-101 check, so max is 100). Passed down to <code>chunk_plan_and_crypto_init</code> as the override byte
<code>-f<N><K\ M\ G></code>	<code>opts+8</code> (qword, shifted 10/20/30 bits)	Maximum file size (K/M/G suffix). <code>0</code> = unlimited
<code>-threads <N></code>	<code>opts+44</code>	Force worker thread count (otherwise <code>max(4, 2×CPU)</code>)
<code>-noparallel</code>	<code>opts+40</code> (byte)	Disable the multi-worker split for single large files
<code>-esxi</code>	<code>opts+41</code> (byte)	Switch to ESXi allowlist mode (VM file extensions only) + enables BOOTBANK/OSDATA directory skip
<code>-allowfolders <list></code>	(Linux-only)	Listed in usage but not implemented in the Windows build — the help text tags it <code>[Linux]</code>
<code>-h / -help</code> (implicit)	returns exit 106	<code>sub_140005DE0</code> (renamed <code>usage_helper</code>) dumps the 1101-byte usage text and <code>exit(101)</code> . Invalid args path also calls it

12. Static Imports Summary

Category	Key APIs
Crypto (WinCrypt)	<code>CryptAcquireContextA</code> , <code>CryptAcquireContextW</code> , <code>CryptGenRandom</code> , <code>CryptReleaseContext</code> — used only for entropy seeding of the mbedTLS CTR_DRBG
Privileges	<code>OpenProcessToken</code> , <code>LookupPrivilegeValueW</code> , <code>AdjustTokenPrivileges</code>
ACL	<code>AllocateAndInitializeSid</code> , <code>FreeSid</code> , <code>SetEntriesInAclW</code> , <code>SetNamedSecurityInfoW</code>
Registry	<code>RegGetValueW</code> (reads <code>HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SystemRoot</code>)
File I/O	<code>CreateFileW</code> , <code>ReadFile</code> , <code>WriteFile</code> , <code>SetEndOfFile</code> , <code>_w fopen</code> , <code>fread</code> , <code>fwrite</code> , <code>fclose</code> , <code>fflush</code> , <code>ferror</code> , <code>_fseeki64</code> , <code>_ftelli64</code> , <code>_fileno</code> , <code>_get_osfhandle</code> , <code>_open_osfhandle</code> , <code>fdopen</code>
Directory walk	<code>FindFirstFileW</code> , <code>FindNextFileW</code> , <code>FindClose</code> , <code>GetModuleFileNameW</code> (self-identification)
Drives	<code>GetLogicalDrives</code> , <code>GetDriveTypeA</code> , <code>GetDriveTypeW</code> , <code>GetEnvironmentVariableW</code> (<code>SystemDrive</code>)
Network shares (MPR)	<code>WNetOpenEnumW</code> , <code>WNetEnumResourceW</code> , <code>WNetCloseEnum</code>
Process / threads	<code>GetCurrentProcess</code> , <code>GetCurrentThread</code> , <code>GetThreadId</code> , <code>CreateThread</code> , <code>ExitThread</code> , <code>GetExitCodeThread</code> , <code>TerminateProcess</code> , <code>CreateProcessA</code> , <code>WaitForSingleObject</code> , <code>WaitForMultipleObjects</code> , <code>CloseHandle</code>
Sync	<code>CreateMutexA</code> , <code>ReleaseMutex</code> , <code>CreateEventA</code> , <code>SetEvent</code> , <code>ResetEvent</code> , <code>InitializeCriticalSection</code> , <code>EnterCriticalSection</code> , <code>LeaveCriticalSection</code> , <code>TryEnterCriticalSection</code> , <code>DeleteCriticalSection</code> , <code>TlsAlloc</code> , <code>TlsFree</code> , <code>TlsGetValue</code> , <code>TlsSetValue</code>
Memory	<code>VirtualProtect</code> , <code>VirtualQuery</code> , <code>LocalFree</code> , <code>malloc</code> / <code>calloc</code> / <code>realloc</code> / <code>free</code> (CRT)
CLI	<code>GetCommandLineW</code> , <code>CommandLineToArgvW</code> , <code>GetStartupInfoA</code>
Misc	<code>SetConsoleCtrlHandler</code> (Ctrl+C handler), <code>SetUnhandledExceptionFilter</code> , <code>GetTickCount</code> , <code>Sleep</code> , <code>SleepEx</code> , <code>GetSystemInfo</code>

Absent (significant): no `ole32` / `oleaut32` (no COM/WMI), no `WinHttp` / `WinINet` / `Ws2_32` (no network C2 / exfil), no `OpenSCManager` (no services), no `CreateToolhelp32Snapshot` (no process enum).

14. Indicators of Compromise (IOCs)

14.1 Hashes

Type	Value
SHA-256 (sample)	<code>81ca5fc6b55accdbc44266d66bd72c7c4152a75b215593adc433d51250054333</code>
MD5 (sample)	<code>fa0913bfe83efe092998da70975b5918</code>
SHA-256 (embedded RSA pubkey DER)	<code>3c954f9480b09fa5662e31e60f4db790d66b7f93fd2ee58eb8928c6c929ce785</code>
MD5 (embedded RSA pubkey DER)	<code>cece77e75826d159b0a03ea821d59869</code>

14.2 Network

Type	Value
Onion (ransom site)	<code>ijexszhscln27nl263lmcd7tx3jttkkm4wjhd4e3y6r4csdbfyepavid.onion</code>

14.3 Files

Indicator	Value
Ransom note filename	<code>!!!README!!!DO_NOT_DELETE.txt</code>
Encrypted file extension	(none — original extension preserved)
Footer magic (LE)	<code>4A 94 37 43</code> at <code>EOF - 542 - N + 1</code>
Footer version byte	<code>03</code> at <code>EOF - 542 - N</code>
On-disk bitmap location	<code>file_size + 534</code> (chunk-persisted)

14.4 Registry

Key	Access	Purpose
<code>HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SystemRoot</code>	Read (<code>RegGetValue</code>)	SystemRoot path used for cross-platform path matching (backslashes replaced with forward slashes after read)

No registry writes by this sample.

14.5 Mutex

None. No single-instance mutex / named event — multiple copies run in parallel.

14.6 Behavioural

- Scans all logical fixed drives (`DRIVE_FIXED`) and network drives (`DRIVE_REMOTE`)
- Enumerates and encrypts across SMB shares via `WNetEnumResourceW`
- Runs `vssadmin delete shadows /all /quiet`, `wmic shadowcopy delete`, per-drive `vssadmin resize shadowstorage /maxsize=401MB`, `schtasks /Change .../SR /Disable` via `cmd.exe /C`
- Elevates `SeBackupPrivilege` + `SeRestorePrivilege` on startup, `SeTakeOwnershipPrivilege` lazily
- Rewrites file ownership to `BUILTIN\Administrators` (`S-1-5-32-544`) and DACL to grant Administrators full access on files that initially deny access
- Drops `!!!README!!!DO_NOT_DELETE.txt` at SystemDrive root, every drive root, and **every scanned directory**
- Creates one unnamed mutex (not for single-instance, only for general thread sync)
- Emits stderr log lines: `Using %d worker threads (%d CPU cores detected)`, `Using %d scanner threads for directory traversal`, `Completed in %dh %dm %.1fs` (or variants without hours/minutes)

14.7 Distinctive Strings

- `"We have downloaded confidential information files. Your files are encrypted."` — ransom note opener
- `"Your access key: fddc1cc4d0f43665c4a637b4d3554943"` — access key line
- `!!!README!!!DO_NOT_DELETE.txt` — note filename

- "%s/!!!README!!!DO_NOT_DELETE.txt" — note drop path template (note the forward slash — cross-platform)
- "Mode: ENCRYPT (compiled-in)" — build-time mode banner
- "Usage: ./encrypter [OPTIONS]" — usage banner (Linux-style invocation even in the Windows build)
- "WARNING: randombytes failed: " — DRBG error
- "WARNING: RNG returned all zeros, encryption keys may be weak\n" — RNG sanity check
- "thread_drbg_%d" — CTR_DRBG personalization string format
- "Using %d worker threads (%d CPU cores detected)\n" — startup banner
- "Using %d scanner threads for directory traversal\n" — scanner banner
- "Completed in %dh %dm %.1fs\n" / "Completed in %dm %.1fs\n" / "Completed in %.2fs\n" — completion banners
- "vssadmin delete shadows /all /quiet >nul 2>&1"
- "wmic shadowcopy delete >nul 2>&1"
- "vssadmin resize shadowstorage /for=%c: /on=%c: /maxsize=401MB >nul 2>&1"
- "schtasks /Change /TN "\\Microsoft\\Windows\\SystemRestore\\SR\" /Disable >nul 2>&1"
- "cmd.exe /C %s" — shell-out format
- "SystemDrive" (UTF-16)
- "SystemRoot" (UTF-16)
- "Error: could not open directory: %ls\n" — stderr warning
- "-percent", "-noparallel", "-esxi", "-path", "-threads", "-allowfolders" — CLI flag tokens
- "BOOTBANK", "OSDATA" — ESXi system volume prefixes (present in strings for help text; compared inline as qword/dword immediates)
- mbedTLS error-string set: "UNKNOWN ERROR CODE (%04X)", "SHA512", "RSASSA-PSS", "emailAddress", "rsaEncryption", "sha512WithRSAEncryption", "PEM - Unsupported key encryption algorithm", "PEM - Given private key password does not allow for correct decryption", "PEM - Unavailable feature, e.g. hashing/encryption combination", "PK - Type mismatch, eg attempt to encrypt with an ECDSA key", "PK - Given private key password does not allow for correct decryption", "RSA - The output buffer for decryption is not large enough", "Proc-Type: 4, ENCRYPTED"

14.8 Unique Artefacts

- **Footer magic** `0x4337944A` at `EOF - 542 - N + 1` — unique to this family (not a known public ransomware magic)
- **Cross-platform Linux-style usage banner** in a Windows-only build — identifies the multi-platform codebase
- **Inline BOOTBANK/OSDATA comparison** via qword/dword immediates instead of string refs — discriminant signature in the `.text` even when the strings are stripped
- **RSA-4096 public key modulus** `ED 97 D7 D4 25 38 77 20 C9 56 1C 02 9A B4 EE 5C ...` (first 16 bytes)

15. MITRE ATT&CK Mapping

ID	Technique	Implementation in this sample
T1486	Data Encrypted for Impact	ChaCha20 stream cipher, RSA-4096-OAEP-SHA512 key wrap, in-place encryption with 542+N byte trailer
T1490	Inhibit System Recovery	<code>vssadmin delete shadows</code> , <code>wmic shadowcopy delete</code> , per-drive <code>vssadmin resize shadowstorage /maxsize=401MB</code> , <code>schtasks /Change ... /SR /Disable</code>
T1134.002	Access Token Manipulation: Create Process with Token	Enables <code>SeBackupPrivilege</code> + <code>SeRestorePrivilege</code> (<code>ransom_main</code>), <code>SeTakeOwnershipPrivilege</code> (fallback in <code>open_file_take_ownership</code>)
T1222.001	File and Directory Permissions Modification (Windows)	<code>SetNamedSecurityInfoW(OWNER + DACL)</code> on files that initially deny access; rewrites owner to Administrators
T1135	Network Share Discovery	<code>WNetOpenEnumW(RESOURCE_CONNECTED, RESOURCETYPE_DISK)</code> + <code>WNetEnumResourceW</code>
T1083	File and Directory Discovery	<code>FindFirstFileW</code> / <code>FindNextFileW</code> driven directory walker
T1082	System Information Discovery	<code>GetSystemInfo</code> (CPU count), <code>GetDriveTypeW</code> , <code>GetLogicalDrives</code> , <code>GetEnvironmentVariableW("SystemDrive")</code>
T1012	Query Registry	<code>RegGetValueW(HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SystemRoot)</code>
T1106	Native API	Direct WinAPI calls throughout (no dynamic resolution)
T1059.003	Command and Scripting Interpreter: Windows Command Shell	<code>CreateProcessA("cmd.exe /C ...")</code> for all four recovery-inhibition commands
T1657	Financial Theft	Tor-based ransom negotiation
T1486 (sub)	Intermittent Encryption	Auto-profile 100/80/30/15/10/5/3 % by file size, sqrt-distributed chunk positions, per-chunk disk-persisted bitmap → resume-safe

16. Summary

Aurora is a **pragmatic, C/C++ ransomware with a cross-platform codebase** (Windows/Linux/ESXi) built on mbedTLS, shipped here as a Windows x64 Mingw binary of only ~154 KB. The sophistication is in the **encryption pipeline and privilege handling**, not in stealth — the binary is loud and trivially fingerprintable.

Crypto stack is clean and correct: each file gets a fresh random 32 B ChaCha20 key and 8 B nonce from a per-thread CTR_DRBG, wrapped with RSA-4096-OAEP-SHA512 against a single embedded attacker public key. File encryption is XOR-with-ChaCha20-keystream using an SSE2-optimised implementation (10 double-rounds, two 64 B blocks interleaved per iteration). Intermittent encryption is aggressive — a sqrt-distributed non-uniform chunk layout, 3-100 % depending on file size, with a per-chunk on-disk bitmap at `file_size + 534` making the operation fully resume-safe.

Access enforcement is aggressive: `SeBackupPrivilege` + `SeRestorePrivilege` enabled at startup, with a per-file fallback path that enables `SeTakeOwnershipPrivilege`, rewrites the file owner to `BUILTIN\Administrators`, and rewrites the DACL to grant full access. This leaves permanent ownership damage even on files that are later decrypted.

Aurora has no stealth: no anti-debug, no anti-VM, no string obfuscation, no API hashing, no process killer, no service stopper, no single-instance mutex, no self-delete, no exfiltration. The ransom note claims data theft, but the binary has no network stack (no HTTP, no sockets) — the "we have downloaded confidential files" is a pure bluff at the level of this sample (exfil would be performed out-of-band before ransomware deployment, if at all).

VSS destruction is shell-out only (`vssadmin` , `wmic` , `schtasks` via `cmd.exe /C`) — no in-process COM/WMI. **Network share discovery** via `WNet*` means any SMB share accessible to the running context will be enumerated and encrypted.

Footer layout is simple: 1 B version + 4 B magic (`0x4337944A`) + 512 B RSA blob + 16 B plan/read block sizes + 1 B percent + N B bitmap + 8 B bitmap size. The bitmap persistence means the encrypter can be interrupted mid-file and safely resumed chunk-by-chunk.

Operational risk level is high once Aurora is on a system with sufficient privileges. No anti-analysis, plain-strings IOCs, and a small, focused binary make detection straightforward — but the aggressive scope (SMB shares, all extensions except `exe/dll/iso` , every user directory except a handful of OS-level exclusions), resume-safety, and ACL rewriting make recovery painful beyond the encryption itself.

Cryptographic properties observed statically: - RSA-4096 OAEP-SHA512 wrap of a per-file (key, nonce) — no known-weakness in this construction - Per-file random key and nonce from `crypto/rand` - quality CTR_DRBG (`MBEDTLS_CTR_DRBG`), personalised per worker thread - ChaCha20 with 64-bit nonce (classic variant, not IETF-96) — no nonce reuse across files possible (each file gets a fresh `drbg_random(40)`)

- Files with only the encrypter run interrupted mid-stream retain **no plaintext key material** in the footer (the key material goes into the 512 B RSA blob during the crypto-init phase before any chunk is written)

Appendix A — Extracted artefacts

Artefact	File	Size
Original sample	<code>81ca5fc6...054333</code>	157,696 B
Attacker RSA-4096 public key (DER)	<code>aurora_rsa_pubkey.der</code>	550 B
Attacker RSA-4096 public key (PEM)	<code>aurora_rsa_pubkey.pem</code>	800 B
## Appendix B — Footer layout (542 + N bytes, bitmap-based resume)		

offset (relative to EOF - footer_size)

+0	1 B	version	0x03 (current format)
+1	4 B	magic (u32 LE)	0x4337944A (bytes 4A 94 37 43)
+5	512 B	rsa_blob	RSA-4096 OAEP-SHA512((32 B ChaCha20 key) (8 B nonce)) (Output of mbedtls_rsa_pkcs1_encrypt)
+517	8 B	plan_block_size	u64 LE (from file-size profile table)
+525	8 B	read_block_size	u64 LE (I/O granularity used during encryption)
+533	1 B	percent	auto 3..100 or -percent override
+534	N B	chunk_bitmap	1 bit per planned chunk; set after the chunk is encrypted and written. Also persisted byte-by-byte at the same file offset as each chunk completes → resume-safe.
+534+N	8 B	bitmap_size	qword N (redundant, used by footer_read_parse to locate the start of the bitmap and rsa_blob)

Total footer size = 542 + N

N bound by footer parser: $N \leq (\text{file_size} \gg 15) + 3$

Appendix C — ChaCha20 SSE2 fingerprint (recognisable pattern)

The stream cipher at `chacha20_xor_stream @ 0x14001B080` is the SSE2 interleaved-double variant: - 16 × 32-bit state held in 8 × `_mm128i` registers (2 blocks × 4 rows each) - 10 double-round loop (`do { ... --n10; } while (n10);` with `n10 = 10`) - Rotations via `_mm_or_si128(_mm_slli_epi32(x, r), _mm_srli_epi32(x, 32-r))` with `r ∈ {16, 12, 8, 7}` — the canonical ChaCha20 quarter-round rotations - Column→diagonal shuffles `_mm_shuffle_epi32(x, 0x93 / 0x4E / 0x39)` = 147/78/57 decimal - Counter increment: `_mm_add_epi64(counter_vec, _mm_cvtsi32_si128(2))` — bumps by 2 blocks per iteration - State output into `a1[0..8)` via `_mm_add_epi32(working_state, initial_state)` at the end of the 10-round loop