Lockbit5

Technical Analysis — Windows

REVERSE-ENGINEERED REPORT

RansomLook · ransomlook.io
File last modified: 2026-07-04

Sample SHA-256: `6d0166d181db1f6c381c3ff5fff4fe4106fbc17bc29316e3cf3f65136ee4803c`

LockBit 5.0 — MEGA Full Analysis (Sample 3, x64 hardened build — "Corteva/Solstice" lure)

`6d0166d181db1f6c381c3ff5fff4fe4106fbc17bc29316e3cf3f65136ee4803c`

Table of Contents

1. Executive Summary

Sample 3 is a **64-bit Windows PE** of the **modern LockBit 5.0-generation hardened build family**. It carries the full LockBit 5.0 protection stack and a **fake "Corteva / Solstice Google Photos" masquerade identity** baked into its resources.

It carries the full LockBit 5.0 protection stack: - **MBA** (Mixed Boolean-Arithmetic) control-flow rewriting; - a **process-global PRNG** whose evolving state drives **opaque predicates** and **pointer derivation**; - **per-string runtime decryption** keyed by 32-bit IDs (decryptor `sub_140002 7A0`, **28 unique IDs across 78 call sites**), backed by a decrypted-string cache; - **import-by-hash** (empty IAT).

There is **no monolithic packed payload to unpack**: the encrypted `.rdata` pool (541 KB, entropy ≈ 8.00) is **never bulk-decrypted in memory** (verified: 0 % in-place change after init emulation). Strings/config are decrypted **individually on demand**, so the configuration (onions, note, public key, target/exclusion lists) is **not statically recoverable**. The one substantive plaintext recovery is the masquerade identity in the PE resources, and emulation confirms the self-decryption mechanism (e.g. `kernel32.dll`).

2. Identity

Verdict: LockBit 5.0-generation x64 hardened build.

Binary-only identifying features:

Marker	Sample 3 `6d0166d1…`
Entry stub bytes	`E9 0B 00 00 00 CC×11 41 57 41 56 41 55 41 54 …`
Section layout	`.text` / `.rdata` (ent≈8) / `.data` (5 MB vsz, 1 KB raw) / `.rsrc` / `.reloc`
IAT	empty (import-by-hash)
String engine	ID-keyed decryptor + cache (`sub_1400027A0` , 28 unique IDs / 78 sites)
Obfuscation	MBA + global-PRNG opaque predicates
Spoofed TimeDateStamp	2010-09-27
Entry point	`0x140013250`
`.text` virtual size	`0x274B5`
`.rdata` (encrypted pool)	541 KB
Masquerade identity	**Corteva / Solstice Google Photos**

3. File Metadata

Field	Value
SHA-256	`6d0166d181db1f6c381c3ff5fff4fe4106fbc17bc29316e3cf3f65136ee4803c`
MD5	`bb7c2ca539a63f5828f053cc6ddb6680`
Size	728,968 bytes
Format	PE32+ (x86-64), console
ImageBase	`0x140000000`
Image size	`0x5C2000`
Entry point	`0x140013250` (`start` thunk → `start_0`)
TimeDateStamp	`0x4CA082AE` → 2010-09-27 11:40:30 UTC (**spoofed**)
Overall entropy	7.805

4. PE Structure

Section	VA	Virtual size	Raw size	Entropy	Role
`.text`	`0x140001000`	`0x274B5`	`0x27600`	6.37	Code (MBA) + encrypted string pool
`.rdata`	`0x140029000`	`0x841E8`	`0x84200`	**≈ 8.00**	**Encrypted pool** (~541 KB)
`.data`	`0x1400AE000`	**`0x50CEC0`**	`0x400`	0.94	~5 MB zero-init runtime working memory
`.rsrc`	`0x1405BB000`	`0x5358`	`0x5400`	2.58	**Resources — manifest + version info (plaintext lure)**
`.reloc`	`0x1405C1000`	`0xC`	`0x200`	0.10	Relocations

Structural fingerprint: empty IAT, one near-maximal-entropy `.rdata` pool, and a `.data` whose virtual size dwarfs its raw size (runtime scratch / cache).

---

5. Lure Identity

This build ships a complete **masquerade identity** in plaintext PE resources — a social-engineering disguise:

Application manifest (`requireAdministrator`):

```
name="Corteva.Solstice Google Photos"
<description>Solstice Google Photos</description>
<requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
```

Version-info block: | Field | Value | |---|---| | CompanyName | `Corteva` | | FileDescription | `Solstice Google Photos` | | ProductName | `Solstice Google Photos` | | OriginalFilename | `Solstice Google Photos.exe` | | Comments | `Solstice Google Photos Update` | | LegalTrademarks | `Corteva TM` | | FileVersion | `1,45,1078,558` | | Translation | `040904E4` (en-US, Windows-1252) |

The binary impersonates a "Corteva / Solstice Google Photos" updater (Corteva is a real agriculture-science company; "Solstice" is one of its product brands) and demands administrative elevation on launch. This is a **distribution/lure artefact** and a strong host IOC; it is plaintext in `.rsrc` and therefore the one operationally useful string recoverable without execution.

---

6. Obfuscation Architecture

Four-layer protection stack: 1. **MBA** rewriting across functions; the entry `start_0` is one MBA expression over its own return address. 2. **Global-PRNG opaque predicates** (§7). 3. **Import-by-hash** — empty IAT, no API name strings. 4. **Per-string runtime decryption** with caching (§8).

The custom API-hash and anti-analysis scanners return zero hits because every constant/hash/ID is MBA-materialised and never appears as a clean immediate.

---

7. PRNG

A process-global PRNG (`state = ROL64(accumulator + state², 32)` with a constant multiplier, `index = (state & 0xFFFFFE)+1`) seeded once at init and advanced on every access. With the state words at their load-time zero the recurrence is degenerate and every opaque predicate/decrypt path fails — i.e. **the binary is inert under naive static/dynamic inspection** until its own init seeds the PRNG in program order.

Emulation (§10) confirms the PRNG self-seeds during init.

---

8. Strings

ID-keyed getter-with-cache design:

```
cache-check (slot dword == ID) ? return cached
 -> sub_140001xxx  (working buffer)
 -> sub_1400027A0  (rcx = buffer, edx = 32-bit ID)  ->  rax = plaintext   [DECRYPTOR]
 -> post-process
 -> store into PRNG-indexed cache table in .data
```

- **Decryptor:** `sub_1400027A0` — leaf, MBA-obfuscated, reads ciphertext from the in-`.text`/pool region.
- **Coverage: 78 call sites, 28 unique 32-bit IDs.**

8.1 Recovered string-ID inventory (28)

```
12CABF7C 16715D77 1DAE8D65 229BB385 238E7433 26FAEA5F 276D26D5 2878B715
34FEA6F6 549DDAEA 658092EC 6B718914 792FFAB9 8AB8D953 A5504277 AD1356ED
B932F8E4 BDC6FA95 C4B404A5 C68BF6E4 C7355457 C85C3BF4 CA9728E3 D534D7A8
DDFEF8E4 F7E0A130 F86D55BC FD74D86A
```

(28 distinct 32-bit IDs across 78 call sites.)

---

9. "Unpacking"

This build has no monolithic packed layer to unpack. Two facts establish this: - The encrypted `.rdata` pool (541 KB, ent ≈ 8.00) is **never decrypted as a block**: after emulating init, the in-memory `.rdata` is **0.0 % changed** versus the file. - `.data` shows **no large decrypted ASCII region** after init (longest run = 2 bytes).

The protection is **per-string, on-demand decryption** into a PRNG-indexed cache. "Unpacking" therefore means **decrypting the individual strings/config**, not dumping a hidden PE. The recoverable static artefacts are: - the **plaintext resources** (lure identity, §5), and - the strings the partial init-emulation decrypts (e.g. `kernel32.dll`).

The full string/config pool is gated behind faithful in-order execution (§14). Artefacts saved under `artifacts/` (`UNPACK_NOTES.txt` , `emulator_decrypted_strings.txt`).

> **Note on scope (§§1–12 vs Part B).** Sections 1–12 describe the **file at rest** (the loader/protected outer PE). The **decrypted payload** — the inner PE this build runs — was subsequently recovered offline (*`embedded_pe/sample3_payload_decrypted.bin`* , 1.48 MB; IDB *`.i64`*). It is a **verbose / debug-symbol-rich build**, which exposed the full crypto stack, file-encryption pipeline and

> behaviour **statically, without detonation**. Those findings are in **Part B**; they supersede the loader-level "not assertable" statements in §12 for everything except the still-encrypted runtime configuration (onions/note/keys), which remains gated (§14).

---

10. Harness & Recovery

A standalone Unicorn harness (`scripts/20–22`) targets this build (PE path, `START = 0x140013250` , `DECRYPTOR = 0x1400027A0`): map sections at `0x140000000` , lazy-map touched pages, neutralise `rdtsc` / `cpuid` , intercept import-hashed (unmapped) calls, run from `start_0` .

Recovered: - the binary **self-seeds its PRNG** and **self-decrypts strings** under emulation (`kernel32.dll`); - the **28-ID decryptor inventory** (§8.1); - confirmation that `.rdata` is a **per-string pool**, not a block cipher target (§9).

Coverage is bounded by emulation fidelity: with import-hashed APIs stubbed, the run diverges after the early init, so only the first strings decrypt. The full config requires realistic API semantics during the config-build phase.

---

11. API Resolution

Empty import directory; APIs resolved by hash via PEB module walk; no DLL/API name strings. `kernel32.dll` recovered from emulation confirms the module-name-decrypt + export-hash-match pattern. (Standard constant-scan hash resolvers yield nothing because the hashes are MBA-hidden.)

---

12. Cryptography — loader, at rest

At the **loader** level (the file at rest) there are no plaintext cryptographic constants (ChaCha/Salsa sigma, AES S-box, SHA IV all absent); cipher tables/keys are MBA-materialised or live inside the encrypted `.rdata` pool. The file-encryption primitive is therefore **not assertable from the loader's static constants alone**.

> This limitation is **lifted once the payload is decrypted** — see **Part B / §B1**, where the full primitive set (X25519 + ChaCha20 + Keccak-f1600/SHAKE256) is pinned from the payload's own code.

Part B — Unpacked Payload Analysis

Target: *`embedded_pe/sample3_payload_decrypted.bin`* (1,478,656 bytes, PE32+ x64, ImageBase *`0x140000000`*). Recovered offline; the inner PE this build's loader decrypts and executes. A verbose/debug build — extensive left-in *`printf`* -style dev strings + ref-style crypto code made the analysis below possible without detonation.

B1. Payload Cryptography — X25519 + ChaCha20 + Keccak/SHAKE256

The payload carries the full modern **LockBit 5.0 / "Green"-generation** crypto stack — scalar/portable, **no AES, no SSSE3 vectorization**. Primitives are identified by their constants/structure (sigma, rotations, round count, Montgomery 121666, Keccak round constants), not validated against test vectors; the X25519→SHAKE256→ChaCha20 wiring is inferred from the call graph and KDF layout, not byte-traced through a live file encryption.

Role	Primitive	Address	Evidence
Key agreement (per-file key wrap)	**X25519** (Curve25519 ECDH)	`x25519_scalarmult` `0x140015D40` ; `fe25519_tobytes` `0x14001C880`	Montgomery-ladder constant **121666** unrolled @ `0x140016920+` ; ref10 radix-$2^{25.5}$ field-element serialization
File body cipher	**ChaCha20** (standard, 20-round, 256-bit key)	core `0x140014530` ; QR loop `0x1400148F0` ; round-counter `0x1400148D8`	sigma `"expand 32-byte k"` @ `.rdata 0x140136FB0` ; ARX rotations 16/12/8/7; `mov r8d,0x14` = 20 rounds; scalar (no `pshufb`)
Hash / KDF / per-chunk integrity	**Keccak-f1600** (SHA-3 / SHAKE)	`keccak_f1600` `0x140013B50` ; RC table `0x140139460` ; `keccak_absorb` `0x140014160` ; `keccak_squeeze` `0x1400138A0`	25 lanes, 24 rounds, ρ offsets, χ, ι round constants

Distinct from Sample 1. S1's file cipher was a customized SSSE3 ChaCha (22-round); the S3 payload uses standard scalar ChaCha20 (20-round) — confirming S3 is a different code lineage from the 3.0-base S1.

CSPRNG. `chacha20_block` (`0x140014530`) is a leaf whose only static caller is `csprng_chacha_timing` (`0x140013070`): it gathers entropy from `KUSER_SHARED_DATA` (`[0x7FFE0320]` × `[0x7FFE0004]` , InterruptTime × multiplier) in a 32-iteration `mfence` / `lfence` / `pause` -fenced loop, mixes it through the ChaCha20 permutation, and emits N random bytes — i.e. `generate_random_bytes()` . (Other ChaCha uses route through the MBA indirect-call engine and are not statically xref'd.)

Per-file/per-chunk KDF = SHAKE256. `shake256_kdf` (`0x1400136B0`):

```
rand64 = csprng_chacha_timing(64)                                  // per-file salt
key64  = SHAKE256( CONST64 ‖ rand64 ‖ S(32) ‖ domsep1 )            // S = X25519 shared secret
         // keccak_absorb (rate 0x88=136), SHAKE domain byte 0x1F, 376-byte sponge ctx
         // keccak_squeeze -> 64 bytes = ChaCha key+nonce material
```

B2. File-Encryption Pipeline

```
victim file --chunked--> encrypt_thread (0x1400A4EE0)
  per file:  S = X25519(victim/campaign pubkey, ephemeral)        // 32-byte shared secret
  per chunk (<=128 B unit):
     rand64 = csprng_chacha_timing()                              // ChaCha20 RNG, KUSER timing
seed
     key64  = SHAKE256(CONST ‖ rand64 ‖ S ‖ domsep)               // shake256_kdf 0x1400136B0
     ciphertext = ChaCha20-keystream XOR plaintext                // applied inside
chunk_crypto_dispatcher 0x14009F980
     chunk_hash = Keccak-f1600(chunk)                             // cryptographic integrity,
recorded in chunk map
  footer: write_file_meta (FILEMETA, 0x14009EDA0) -> set EOF; rand64 stored per chunk (self-
describing)
recovery = X25519 PRIVATE key required (S unrecoverable) -> config & per-file keys stay runtime-
only
```

- **Chunked / intermittent ("partial") encryption** — strings `EncryptPartly`, `InitializeChunkEncryptionMap` (`0x1400A02F0`), `DataWriteOffset` / `HashWriteOffset`, "extend file to final size".
- **Dual hashing (two distinct functions):**
- `chunk_map_hash` (`0x1400A64D0`) — a **fast NON-crypto** wyhash/mul-fold (magic consts `0x91D5E7F2C43A6B08`, `0x3A7C924FE5D86B19`, `0x5360D8D07A4A702F`; multiply-xor hi^lo of (ptr,len) → 64-bit) = the chunk-MAP key ("marked hash for unencrypted chunk").
- `keccak_f1600` (`0x140013B50`) — the **cryptographic** per-chunk integrity hash.
- **FILEMETA footer** stores per-chunk salts so each encrypted file is self-describing for the operator's decryptor.

B3. Payload Behaviour Map

Recovered from the payload's verbose dev strings + xref'd owner functions (offline):

Capability	Evidence (function / strings)	MITRE
Multi-threaded chunked encryptor	`encrypt_thread` `0x1400A4EE0`, "Encrypt Thread", progress `> %u/%u [%ls] %d%%`	T1486
RunPE process hollowing → defrag.exe / svchost	`runpe_hollow_process` `0x1400E0560` (owns all hollowing strings: create process / get-set thread context / alloc / write headers / write section %d); plaintext target string `\defrag.\svchost` @ `0x140137132`; `defrag.exe` **execution confirmed live (detonation)**	T1055.012
ETW bypass	`start` (payload EP `0x140119C6C`), `"Etw not patched!"` patched at startup	T1562.006
Network self-propagation (SMB lateral movement)	host discovery via `Iphlpapi!GetIpNetTable` (ARP) + subnet enum → `ws2_32` async **ConnectEx + IOCP** port scan → **encrypt files on `ADMIN$` /SMB shares** (`Starting search on share %ls`). Spreader `FUN_1400CC110` (`net` mode 0xC). Scanned port MBA-materialised (likely 445). See §B3.7	T1021.002 / T1135
Event-log tampering	`Channel enumeration failed`, `Failed to get channel paths`, `Failed to open EVT session` (EvtOpenSession)	T1070.001
Service / process kill	`Failed to open Service Control Manager`, `Failed to kill process %ls (PID: %lu), adding to failed cache`	T1489
Geo / locale fence	`Found in GetUserDefaultUILanguage` (+ GetUserGeoID)	T1614.001
Recovery inhibition — VSS shadow-copy deletion	`vssapi.dll` + `ole32.dll` built as inline stack-strings + dynamically loaded at `0x1401303E6` / `0x140130FA6`; COM `IVssBackupComponents` (no `vssadmin` / `wmic` shell)	**T1490**
Ransom note	`ReadMeForDecrypt.txt` (id suffix runtime-encrypted)	T1486

B3.1 Pinned function anchors (offline lea-xref)

Behaviour	Function / reference site
Encrypt thread	`encrypt_thread` `0x1400A4EE0`
RunPE hollowing	`runpe_hollow_process` `0x1400E0560` (plaintext target string `\defrag.\svchost` `0x140137132`)
Payload main / geo-fence	**`0x140113F60`** (clean prologue; gates on UI-language/Geo early)
Network spreader	ref sites `0x1400D0EB3` / `0x1400D0EFD`
Event-log clearing	ref site `0x140067213`
Service/process kill (SCM)	ref site `0x14005E9F2`; process-kill `0x14007D712`
VSS shadow delete	`0x1401303E6` / `0x140130FA6`
ETW patch	`start` `0x140119C6C`

Completeness. A full offline sweep (153 unique UTF-16 strings + ASCII) confirms the behaviour map above is complete. **No persistence / mutex / self-delete strings were found** (even encrypted-adjacent) → the build appears to rely on its masquerade/lure for delivery rather than registry-based persistence (not asserted as present). The **runtime configuration** (onion DLS, full note body, target

extension, exclusion lists, victim/campaign X25519 public key) is **visible in `.rdata` as obfuscated UTF-16 records but never plaintext** — the same per-record MBA transform as the loader. It is the one element that stays runtime-only by design (§14).

B3.3 Operation modes, geo-fence & CLI (Ghidra decompile of the 83 KB MBA monolith)

Hex-Rays cannot decompile `payload_main` (`0x140113F60` , the real EP, 83 KB MBA monolith); **Ghidra does** (708 KB of C, raised decompiler payload limit). It exposes:

- `g_operation_mode` **(** `0x14013D020` **) — master mode flag, static default** `0x0A` **.** Discrete modes gate behaviour: | Mode | Path | |---|---| | `0x0A` / `0x0B` (`&0xFE==10`) | local file/path enumeration (**default**) | | `0x0C` | **net** — network spreader `FUN_1400CC110` + network-share encryption (**confirmed live: mounted + encrypted** `N:` / `M:`) | | `0x0E` | MBA string-build path | Runtime-overridable via CLI args.
- **Geo / language fence (two gates):** aborts if `GetUserDefaultUILanguage() == 0x419` **or** `GetUserGeoID() == 0xC9` (201). The **UI-language** `0x419` **is the Russian (ru-RU) LCID** — a confirmed avoid-Russian-locale-host check, the classic CIS-exclusion operator-origin signal. The **GeoID** `0xC9` **(201) is reported as the raw compared value; its country is not independently asserted** (Windows GEOID for Russia is 203/0xCB, so 201 is a distinct, unconfirmed region — not claimed as Russia here). Logs "Found in GetUserDefaultUILanguage" / "Found in GetUserGeoID".
- **CLI / target arguments =** `;` **-delimited wide path list** (`0x3B` split; `/` and `\` normalized). The per-mode arg→flag setter lives in a sub-function; its inputs decode at runtime.

B3.4 Embedded configuration — RECOVERED (config-string crypto cracked)

The config-string crypto was reversed from the decoder `FUN_140059930` (Ghidra; it READ/WRITEs the encrypted blobs at `0x14013D390` +): **each wide-char record is XOR'd with an incrementing keystream — low byte** `(0x57 + i) & 0xFF` **, high byte 0 (UTF-16), reset at each record start.** The `.rdata` "key" at `0x140136150` is literally that `0x57,0x58,0x59…` run. Standalone offline extractor: `scripts/35_decode_config.py` → `artifacts/CONFIG_DECODED.txt` . No detonation.

Recovered operational configuration: | Category | Values | |---|---| | Target drives | all 26 letters `A:\` – `Z:\` ; `C:\` ; `\\?\` ; UNC `\\` ; SMB share `ADMIN$` | | Excluded folders | `$Recycle.Bin` , `AllUsers` , `Boot` , `chocolatey` , `Microsoft Office` , `Microsoft Visual Studio` , `Windows Kits` , `WindowsApps` , `VisualStudio` , `System Volume Information` , `Microsoft\Windows` , `Windows\Hyper-V` , `..` | | Excluded files | `iconcache.db` , `thumbs.db` | | Excluded extensions | `exe` , `lnk` , `dll` , `cpl` , `sys` | | **Targeted VM disks** | `vhdx` , `avhdx` | | Markers | `encmeta` (chunk-map), `crc` | | CLI arguments | `-d -b -m -t -n -h --help -k -w -fast -full -i -nomutex -v` | | Operation-mode keywords | `all` / `local` / `net` (drive `g_operation_mode` , §B3.3) | | Ransom-note filename | `ReadMeForDecrypt.txt` (`0x1401396A8`) | | TrustedInstaller SID | `S-1-5-80-956008885-3418522649-1831038044-1853292631-2271478464` (`0x140139610` ; privilege/impersonation for max-access encryption) |

Note body + onions = MBA-encoded, assembled at runtime (§B3.6). Unlike the operational config above (static XOR-0x57), the ransom-note template, its three onions and the version line are stored as **MBA per-ID strings** (the `FUN_14001E8A0/E990` engine) and concatenated at runtime — no `.onion` /note prose appears under any byte-level XOR of the static image. Only the internal name `ChuongDong Locke[r]` is plaintext (`.rdata 0x140136E80`). The victim ID is generated per-host at runtime. The X25519 victim/campaign public key is likewise runtime-material, not embedded.

B3.7 Network propagation (SMB lateral movement)

The `net` operation mode (`g_operation_mode 0xC`) runs spreader `FUN_1400CC110` (+ ARP-owner `0x1400D0411`): 1. **Host discovery** — `Iphlpapi.dll` → `GetIpNetTable` (ARP neighbours) + subnet enumeration; excludes own IP. 2. **Async port scan** — `ws2_32.dll` Winsock, **ConnectEx + I/O Completion Port**, dedicated host-scan / port-scan threads → massively concurrent scan of discovered hosts. 3. **Share encryption** — targets `ADMIN$` (config) and per-host shares (`Starting search on share %ls`); the share path feeds the same chunked `encrypt_thread` pipeline → **encrypts files on reachable SMB shares**.

Inline DLLs decoded from `movabs` stack-strings: `Iphlpapi` , `ws2_32.dll` , `kernel32` , `ntdll.dll` . The scanned port is MBA-materialised (no clean `htons` immediate; **likely SMB 445 given the ADMIN$ targeting, but not statically confirmed**). Model = discover LAN hosts → mount SMB shares → encrypt remote files — **no PsExec/service-copy self-replication was found, but this is absence-of-evidence: the spreader is MBA-obfuscated and a service-based spread cannot be fully excluded statically**.

Reachability (verified, Ghidra): the spreader (`FUN_1400CC110`), RunPE hollowing (`runpe_hollow_process`), and VSS shadow-delete site are **all called directly from** `payload_main` — confirmed live code paths, not dead/decoy code. (The broader behaviour map in §B3 rests on verbose dev-strings; the major capabilities are reachability-confirmed, but per-string execution under specific config flags was not exhaustively traced.)

B3.6 Ransom note & network IOCs (recovered from authorized detonation — NOT static)

> **Origin (important):** the onions and note body are **not statically extractable** — every byte-level attack (XOR-0x57 keystream, known-plaintext XOR of the onion against payload+loader, period/incrementing-key search) returned nothing; they are **MBA-assembled in memory at runtime**. They were captured two ways, both confirming the same values: (1) the dropped *`ReadMeForDecrypt.txt`* on the victim disk, and (2) **live from the malware's own memory under x64dbg** — *`bp ntdll!NtWriteFile`* conditioned on the buffer starting `"~~~ "` , dumped at write time (*`artifacts/RANSOM_NOTE_live_x64dbg.txt`*). The per-victim **ID is partly host-deterministic**: the first 16 hex are stable across runs (*`0F601778F7A96DD2`*), the last 16 are random per run. Detonation/runtime-sourced primary evidence, not static RE output.

The dropped note (full text in `artifacts/RANSOM_NOTE.txt`) yields the campaign network IOCs:

Role	Onion
DLS / leak site	`lockbitapt67g6rwzjbcxnww5efpg4qok6vpfeth7wx3okj52ks4wtad.onion`
Victim chat	`lockbitsuppyx2jegaoyiw44ica5vdho63m5ijjlmfb7omq3tfr3qhyd.onion`
Affiliate sign-up / ads	`lockbitfbinpwhbyomxkiqtwhwiyetrbkb4hnqmshaonqxmsrqwg7yad.onion`
Version tag	`ChuongDong v1.06 | x64`
Per-victim ID (this run)	`0F601778F7A96DD2BCAAA587B8B15815` (runtime-generated)

The note brands as "LockBit 5.0 … since 2019" and advertises affiliate recruitment. (Branding and the LockBit-Black-era onion reuse are operator claims in the note, not independently asserted by this analysis.)

B3.5 Detection content

A YARA ruleset is delivered alongside this report: `lockbit5_sample3.yar` (4 rules) — loader (shared entry stub + empty IAT + high-entropy `.rdata`), Corteva/Solstice lure, payload crypto+behaviour (ChaCha sigma + wyhash constants / dev strings / VSS context), and a high-precision wyhash-constant rule. Validated: loader → 2 hits, payload → 3 hits, **Sample 1 → 0 hits** (no cross-lineage false positive).

---

13. Comparison with the 3.0-base lineage (Sample 1)

Dimension	S1 `9dc03bf8` (3.0-base)	S3 `6d0166d1` (5.0-gen, this report)
Lineage	LockBit Black/3.0 leaked-builder	**LockBit 5.0 generation**
Arch / size	x86 / 158 KB	x64 / 729 KB
3.0 residue	present	none
Config	static blob (recoverable)	**static XOR-keystream (recovered, §B3.4)**
Strings	static-decodable	**per-ID runtime (28 IDs)**
Obfuscation	thunks / ROR13	**MBA + PRNG opaque pred.**
Decryptor	n/a (different scheme)	`sub_1400027A0`
Spoofed TS	2022 (real build)	**2010**
Masquerade identity	—	**Corteva / Solstice Google Photos**
Monolithic packer	APLib config blob	**none (per-string)**
File cipher (payload)	customized SSSE3 ChaCha, 22-round	**standard scalar ChaCha20, 20-round**
Key wrap (payload)	RSA-1024 e=3	**X25519 (Curve25519 ECDH)**
Hash/KDF (payload)	—	**Keccak-f1600 / SHAKE256**

The S3 payload's **ChaCha20(20)+X25519+Keccak** stack differs sharply from S1's **ChaCha(22, SSSE3)+RSA-1024** — confirming S3 belongs to the modern LockBit 5.0 toolchain, a distinct code lineage from the older 3.0-base S1.

---

14. Extraction Status & Limits

Firmly established (loader): identity as a LB 5.0-generation x64 hardened build; the obfuscation architecture; the decryptor and 28-ID inventory; the absence of a monolithic packed layer; the plaintext **Corteva/Solstice lure identity**.

Firmly established (payload, Part B — recovered offline, no detonation): the **full crypto stack** (X25519 + ChaCha20-20 + Keccak-f1600/SHAKE256) and the complete **key-derivation chain** (§B1); the **file-encryption pipeline** (chunked/partial, dual-hash, FILEMETA footer, §B2); the **behaviour map** (RunPE hollowing [target string `\defrag.\svchost`], ETW bypass, network spreader, event-log tampering, service/process kill, Geo/UI-language fence, §B3). The earlier loader-level "file-cipher not assertable" (§12) is **now resolved** by the payload.

Architecturally gated (same MBA+runtime wall): the payload entry `start` / `payload_main` (`0x140113F60`) is an **83 KB single MBA monolith** (size `0x145F2`) into which the **CLI-argument parsing and behaviour-flag dispatch are inlined**; Hex-Rays cannot decompile it and its data refs

resolve to runtime `.data` cache slots. So the **command-line interface and the config behaviour-flag map are not statically tractable** — they materialise at runtime exactly like the configuration values.

Configuration — now RECOVERED (§B3.4): the config-string crypto was cracked (XOR incrementing `0x57++` keystream). Target drive scope (A–Z + ADMIN$), exclusion lists (folders/files/extensions), VM-disk targeting (vhdx/avhdx), the full CLI argument set, operation-mode keywords (all/local/net) and the note filename are all extracted statically.

Runtime-MBA-material (not under the static XOR config): the **ransom-note body, its three onions, and the version line** are stored as MBA per-ID strings and assembled at runtime — no `.onion`/note prose appears under any byte-level XOR of the static image. They were **recovered from an authorized detonation** (dropped `ReadMeForDecrypt.txt`, §B3.6); static-only extraction would require full MBA-string-engine emulation. The **victim/campaign X25519 public key** and per-victim ID are likewise runtime material. These remain per-record MBA-obfuscated in the payload's `.rdata` (visible as ciphertext, never plaintext) and decode on-demand at runtime. Recovering them statically needs an emulator extended with realistic process/file/registry API semantics, or a contained-detonation memory capture (out of scope under the no-detonation policy). Stated plainly: **everything except the runtime config is now recovered; the config gap is architectural, re-confirmed four independent ways** (loader script-22 capture, payload script-33 capture = 2732 MBA-immediate qwords / 0 strings, MBA accessor analysis, and the encrypted-but-visible `.rdata` string sweep).

---

15. MITRE ATT&CK (evidenced)

Technique	ID	Evidence
Masquerading (legitimate name/metadata)	T1036.004 / .005	"Corteva / Solstice Google Photos" manifest + version info
Abuse Elevation Control — request admin	T1548	manifest `requireAdministrator`
Obfuscated/Compressed Files & Information	T1027	encrypted `.rdata` pool; per-string runtime crypto
Software Packing / protective transform	T1027.002	MBA rewriting
Indicator Removal: spoofed metadata	T1070	TimeDateStamp forged to 2010
Deobfuscate/Decode Information	T1140	runtime string decryptor `sub_1400027A0` + cache
Native API via dynamic resolution	T1106 / T1027.007	import-by-hash, empty IAT
Virtualization/Sandbox evasion	T1497	PRNG-gated opaque predicates render binary inert under naive inspection
Data Encrypted for Impact	**T1486**	payload `encrypt_thread` — chunked ChaCha20 + X25519 wrap (§B1–B2); note `ReadMeForDecrypt.txt`
Process Injection: Process Hollowing	**T1055.012**	`runpe_hollow_process`; target string `\defrag.\svchost` (§B3)
Impair Defenses: Indicator Blocking (ETW)	**T1562.006**	`"Etw not patched!"` — ETW patched at payload startup
Indicator Removal: Clear Windows Event Logs	**T1070.001**	EvtOpenSession channel enumeration/clear (§B3)
Service Stop / process kill	**T1489**	SCM enumeration + process-kill with failed-cache (§B3)
Lateral Movement: SMB/ Admin shares	**T1021.002**	spreader encrypts `ADMIN$` /SMB shares after host discovery + port scan (§B3.7)
Network share / system discovery	**T1135 / T1018**	ARP (`GetIpNetTable`) + subnet enum + async IOCP/ConnectEx port scan (§B3.7)
System Location Discovery: language/geo fence	**T1614.001**	`GetUserDefaultUILanguage()==0x419` (Russian LCID — confirmed CIS-exclusion) OR `GetUserGeoID()==0xC9` (raw, country unconfirmed) → abort (§B3.3)
Inhibit System Recovery (VSS shadow delete)	**T1490**	dynamic `vssapi.dll` + `ole32.dll` load, COM `IVssBackupComponents` (§B3)

16. Indicators of Compromise

Type	Value
SHA-256	`6d0166d181db1f6c381c3ff5fff4fe4106fbc17bc29316e3cf3f65136ee4803c`
MD5	`bb7c2ca539a63f5828f053cc6ddb6680`
File size	728,968 bytes
PE	x86-64, ImageBase `0x140000000`, EP `0x140013250`, spoofed TS `0x4CA082AE` (2010-09-27)
Masquerade	`Corteva` / `Solstice Google Photos` / `Solstice Google Photos.exe` / "Solstice Google Photos Update" / FileVersion `1,45,1078,558` ; `requireAdministrator`
Structural	empty IAT; `.rdata` ≈ 541 KB at ent ≈ 8.00; `.data` virtual `0x50CEC0` over raw `0x400`
String decryptor (loader)	`sub_1400027A0` (28 IDs, 78 sites) — IDs listed §8.1
Decrypted (partial)	`kernel32.dll`
Payload (decrypted PE)	`embedded_pe/sample3_payload_decrypted.bin`, PE32+ x64, 1,478,656 bytes
Payload crypto	X25519 (`0x140015D40`) + ChaCha20-20 (`0x140014530`, sigma @ `0x140136FB0`) + Keccak-f1600/SHAKE256 (`0x140013B50`, KDF `0x1400136B0`)
Payload behaviour	RunPE hollowing (`runpe_hollow_process 0x1400E0560` ; target string `\defrag.\svchost` @ `0x140137132`); ETW patch; ARP+port-scan spreader (`mpr.dll`); event-log clear; SCM/ process kill; Geo/UI-lang fence
Ransom note	`ReadMeForDecrypt.txt` (body assembled at runtime; full text in `artifacts/RANSOM_NOTE.txt`)
Onion — DLS/ leak	`lockbitapt67g6rwzjbcxnww5efpg4qok6vpfeth7wx3okj52ks4wtad.onion`
Onion — chat	`lockbitsuppyx2jegaoyiw44ica5vdho63m5ijjlmfb7omq3tfr3qhyd.onion`
Onion — affiliate	`lockbitfbinpwhbyomxkiqtwhwiyetrbkb4hnqmshaonqxmsrqwg7yad.onion`
Version tag	`ChuongDong v1.06 | x64` (internal `ChuongDong Locke[r]` @ `0x140136E80`)
Recovery inhibition	dynamic `vssapi.dll` + `ole32.dll` load → COM shadow-copy deletion (no shell cmd)
Detection	`lockbit5_sample3.yar` (4 rules; loader 2 hits / payload 3 hits / S1 0 hits)
Network (onions)	DLS `lockbitapt67g6rwzjbcxnww5efpg4qok6vpfeth7wx3okj52ks4wtad.onion` · chat `lockbitsuppyx2jegaoyiw44ica5vdho63m5ijjlmfb7omq3tfr3qhyd.onion` · affiliate `lockbitfbinpwhbyomxkiqtwhwiyetrbkb4hnqmshaonqxmsrqwg7yad.onion` (runtime-assembled; from detonation)