Umbra

Technical Analysis — Cti Report

REVERSE-ENGINEERED REPORT

RansomLook · ransomlook.io
File last modified: 2026-08-09

Sample SHA-256: `976ea6c54c8eea1b7c3d1d5227c50fd16f301518fd659a9ee4a770568850f553`

Umbra Ransomware - CTI Report

Family: Umbra **Reference sample:**
`976ea6c54c8eea1b7c3d1d5227c50fd16f301518fd659a9ee4a770568850f553`

1. Executive Summary

Umbra is a Rust ransomware payload targeting Windows x86-64. It encrypts whole files with **ChaCha20-Poly1305** under per-file keys wrapped with **RSA-2048**, renames them to `.umbra`, drops a templated ransom note and replaces the desktop wallpaper with an embedded cartoon-styled ransom screen.

Two characteristics stand out for defenders:

- **The payload itself performs no network I/O.** No networking library or API is imported, no `LoadLibrary` variant is present, and no networking DLL or API name exists anywhere in the file, so it opens no connection of its own. It does, however, carry a command-execution path whose command comes from the configuration (§2.2), and this build supplies none - so nothing leaves the host here, but the design leaves room for a child process to do it.
- **No contact channel is published.** The ransom note instructs the victim to open a "contact portal" in Tor Browser but names none. No `.onion` host, URL, e-mail address, TOX ID or cryptocurrency address is present in the note, in the configuration, or in the binary. The configuration format reserves ten string slots and **three are left empty** in this build. The image also carries 73 obfuscated strings, **all of which have now been recovered** (§2.2) - none is a contact address. The finding is therefore complete for this build.

The note nonetheless asserts that data "will be published" if the victim does not make contact. Nothing in the sample supports that capability.

The configuration is an **appended plaintext trailer**, not compiled into the image - the payload is operator-configurable per build without recompilation.

2. Capability Assessment

Derived from call sites rather than from the import table or from configuration strings - both mislead here, in opposite directions (§2.1, §2.2).

Present

Capability	Basis
Drive and directory enumeration	`GetLogicalDrives`, `GetDriveTypeW`, `FindFirstFileExW`, `FindNextFileW`
File read/write via memory mapping	`CreateFileMappingA`, `MapViewOfFile`, `NtReadFile`, `NtWriteFile`
Rename to the encrypted extension	`MoveFileExW`
File deletion	`DeleteFileW`
Suppression of automatic timestamp updates	`SetFileTime` called with `lpLastWriteTime` = `0xFFFFFFFFFFFFFFFF` and the creation / access parameters `NULL` - the documented idiom for freezing a handle's timestamps, not falsifying them
Cryptographic randomness	`BCryptGenRandom`, `SystemFunction036`
Desktop wallpaper replacement	`SystemParametersInfoA` called with `uiAction = SPI_SETDESKWALLPAPER`
Child command execution	`WinExec`
Ransom-note writing	dedicated routine holding the sole `{DECRYPTION_ID}` reference, called from the top-level function at three sites
Self-deletion	dedicated routine calling `GetModuleFileNameW` then the `WinExec` wrapper; last call in the top-level function

Every entry above was confirmed at a real call site, not from the presence of an import. See §2.1 - a third of the import table is never invoked.

Absent

Capability	Basis
Network I/O of any kind	no `ws2_32`, `wsock32`, `wininet`, `winhttp`, `urlmon`, `dnsapi`, `iphlpapi`, `mswsock`; no `socket`, `WSAStartup`, `InternetOpen*`, `WinHttpOpen*` - egress by child process remains possible, see §2.2
Dynamic loading of a networking DLL	no `LoadLibrary*` imported; no networking DLL name present in the file
Low-level socket access	no `\Device\Afd`, `NtDeviceIoControl` or `NtCreateFile`
-	(process termination, service control, shadow-copy destruction and defence evasion were previously listed here as absent. That was wrong - see §2.2.)
Registry persistence	no `RegSetValueEx`, `RegCreateKey` - registry access is read-only
Scheduled task or service installation	no `schtasks`; no service-control API
Privilege escalation	manifest requests `asInvoker`; no token manipulation API
Second-stage delivery	exactly one valid PE header in the file; no embedded archive
Registry access of any kind	`RegOpenKeyA`, `RegQueryValueA`, `RegCloseKey` imported but never invoked (§2.1)
Window, message loop, tray icon, drag-and-drop	the entire `USER32` window set, all of `GDI32` and all of `SHELL32` imported but never invoked (§2.1)
COM usage	all of `ole32` imported but never invoked (§2.1)

Operational consequence: this payload is a host-local file encryptor that does not spread, does not persist and opens no connection of its own. It **does** carry a full recovery-destruction and defence-evasion arsenal, disabled by configuration flags in this build but present in the code (§2.2).

2.1 A third of the import table is never invoked

A 591-byte function containing no call instruction takes the address of 48 imports (`lea rax, <API>` followed by a stack store, repeated). Its only effect is to keep the linker from discarding them.

Counting only references that are a `call` to the import or to its thunk, **48 of the 150 imports have no caller anywhere in the image**: all of `GDI32` (11), all of `SHELL32` (6), all of `ole32` (5), 16 of 17 `USER32` and 4 of 5 `ADVAPI32`.

Two practical consequences:

- **Capability must be read from call sites, not from the import table.** An analysis based on imports alone would credit this sample with a GUI, a tray icon, drag-and-drop handling, registry access and COM usage. It has none of them.
- `imphash` **characterises the declared imports, not the behaviour.** It remains usable as a pivot for this build, but it is inflated by the padding and should not be read as a behavioural fingerprint.

2.2 A full command arsenal, obfuscated in the image and disabled by flags

The top-level routine dispatches on individual configuration flag bytes. Six of them gate routines that all call the `WinExec` wrapper. The commands those routines run are **compiled into the binary and obfuscated**, not supplied by the operator. Recovered, they are:

Purpose	Commands
Shadow-copy and backup destruction	`vssadmin delete shadows /all /quiet`, `wmic shadowcopy delete`, `wbadmin delete catalog -quiet`
Boot and recovery tampering	`bcdedit /set {default} recoveryenabled No`, `bcdedit /set {default} bootstatuspolicy ignoreallfailures`, `bcdedit /set {current} safeboot networkminimal`
Defender neutralisation	`powershell ... Set-MpPreference -DisableRealtimeMonitoring $true -DisableBehaviorMonitoring $true`, `powershell ... Add-MpPreference -ExclusionPath C:\`
Event-log clearing	`wevtutil cl "<log>"`
Process termination	`taskkill /F /IM <name>`
Service stop and disable	`sc stop "<name>"`, `sc config "<name>" start= disabled`
Host shutdown / reboot	`shutdown /s /f /t 0`, `shutdown /r /f /t 0`
Shell	`cmd.exe /C "`

The configuration's process and service lists are the **arguments** to `taskkill` and `sc` - they are not inert.

In this build all six flags are clear, so none of it runs. Three flags are set, enabling exactly: proceeding to encryption, writing the ransom note, and replacing the wallpaper.

> **Correction.** An earlier reading of this sample reported these capabilities as absent, on the basis that no *`vssadmin`*, *`bcdedit`*, *`powershell`* or *`taskkill`* string appears in the binary and no process- or service-control API is imported. Both observations are true and both conclusions were wrong: the commands are obfuscated, and *`WinExec`* needs no such API. Treat Umbra as a family that **does** destroy shadow copies, disable Defender, clear event logs, kill processes and disable services.

A seventh flag gates a **victim status record**, also clear here. Its template is fixed in the image:

```
{"bot_id":"{}","hostname":"{}","username":"{}","status":"encrypted","ext":"{}"}
```

`status` is a literal, not a substitution point - the record exists only for the encryption case. It identifies the host and names the extension applied: a check-in for campaign accounting.

The routine that builds it **also calls the `WinExec` wrapper**. With no networking API anywhere in the image, handing the record to an operator-supplied command is the only egress path this design has. So the correct reading of "no exfiltration capability" is: no network stack, and no command configured in this build - but a build that supplies one could carry this record off the host through a child process.

How to read this: the absence of `vssadmin`, `wbadmin`, `bcdedit`, `wevtutil`, `wmic` and `powershell` strings is a fact about **this configuration**, not about the family. A build with those flags set and command strings supplied would execute them. Detection should assume Umbra can run operator-chosen commands, and hunt for a short-lived child process from the payload, rather than concluding the family never does.

2.3 Canary-file avoidance - a usable mitigation

Before descending into a directory, the encryption walk calls a routine that scans that directory's filenames for six obfuscated substrings:

```
canary      _decoy      honeytoken      eicar      testfile      honey
```

If any filename contains one of them, **the entire directory is skipped** - not just the matching file. The set targets canary-file conventions and antivirus test artefacts, so the goal is to avoid tripping file-integrity canaries and to avoid detonating inside a sandbox.

This is directly usable as mitigation: a file named to contain any of these six substrings protects the directory it sits in. Two caveats - the markers are compiled into the binary, so a rebuild can change them, and the protection is per-directory, not recursive. Treat it as a cheap additional layer, never as a primary control.

---

3. Configuration Content Without Corresponding Code

The appended configuration is operator-supplied, but only part of it governs behaviour. Three fields were traced to actual use: the **encryption extension**, the **272-byte RSA key blob** and the **ransom-note template**.

The exclusion lists were not. The directory walk filters through a `switch` on candidate length with inlined constant comparisons - the shape of a match against a static array. Decoding those immediates yields a **hardcoded 12-directory set** (`boot` , `windows` , `perflogs` , `msocache` , `programdata` , `windows.old` , `$recycle.bin` , `$windows.~bt` , `$windows.~ws` , `program files` , `program files (x86)` , `system volume information`) matching a string table in `.rdata` - not the **20-entry list** carried in the configuration. The eight extra configured entries (`config.msi` , `tor browser` , `intel` , `x64dbg` , `public` , `all users` , `default` , `microsoft`) do not appear in the comparison code.

The same applies to extensions: eleven hardcoded values (`.exe .dll .sys .drv .lnk .msi .com .bat .cmd .ps1 .vbs`) each appear once as an inlined constant in `.text` , while extensions present only in the 49-entry configured list appear zero times.

Practical consequence: hunting or scoping built on the configured exclusion lists would be wrong. Directories such as `tor browser` and `x64dbg` are listed in the configuration but are not spared by this build.

3.1 Configuration format - tracking value

The parser reads a fixed 301-byte header (272-byte key blob, then 25 byte flags, a u16 and two bytes) followed by exactly ten length-prefixed strings. In this build three of those ten slots are empty, six of the 25 flags are set, and the first dword is not the `DEC1` magic that selects the parser's second configuration format.

Because the whole structure is operator-supplied and appended after build, these are the fields that discriminate one operator or campaign from another:

- the **256-byte RSA modulus** at key-blob offset 16 - the strongest per-operator discriminator;
- which of the **ten string slots** are populated, and their contents;
- which of the **25 flags** are set;
- presence of the `DEC1` magic, which marks the alternate format.

A build whose empty slots become populated, or which carries the `DEC1` magic, should be treated as a materially different configuration rather than a rebuild.

The appended configuration carries a process name list (`sql` , `oracle` , `ocssd` , `dbsnmp` , `synctime` , `agntsvc` , `isqlplussvc` , `xfssvccon` , `mydesktopservice` , `ocautoupds` , `encsvc` , `firefox` , `thunderbird` , `excel` , `outlook` , `word` , `notepad`) and a service name list (`vss` , `sql` , `svc$` , `memtas` , `mepocs` , `msexchange` , `sophos` , `veeam` , `backup` , `GxVss` , `GxBlr` , `GxFWD` , `GxCVD` , `GxCIMgr`).

The import table contains no process-termination and no service-control API, but these lists are **not** inert: they are the arguments to the obfuscated `taskkill /F /IM`, `sc stop` and `sc config ... start= disabled` commands (§2.2), executed through `WinExec`. They are dormant in this build only because the gating flags are clear.

---

4. Encryption and Recovery Outlook

Layer	Detail
Key wrapping	RSA-2048, public exponent 65537, OAEP padding; modulus supplied in the configuration trailer
Bulk cipher	ChaCha20-Poly1305, RFC 8439 construction, no associated data
Key material	per-file key and nonce from `BCryptGenRandom` / `SystemFunction036`
Scope	whole-file encryption; no intermittent mode
Size ceiling	files larger than **256 MiB** (`0x10000000`) are skipped
Footer	fixed 296 bytes, terminated by the magic `RBMU`

The size ceiling is a concrete recovery lever. Anything above 256 MiB is left untouched - typically virtual-machine disks, database files, archives, backup sets and large media. Inventory those before assuming total loss.

The ChaCha20 implementation is vendored with its 16-byte state constant replaced (`20E946D5 65B139F6 12FE4BD2 D9787022` in place of `"expand 32-byte k"`). The substitution neither strengthens nor weakens the cipher; it breaks interoperability with stock ChaCha20-Poly1305 implementations and provides a precise, recompilation-resistant signature.

The composition is sound. **There is no recovery path without the operator's private key.** The file-format details published here support identification, triage and impact scoping only.

---

5. Incident Response Guidance

Ordered by value, given the capability assessment in §2.

1. **Check shadow copies and backups - but verify, do not assume.** In this build the destruction flags are clear, so `vssadmin list shadows`, Windows File History and existing backups are worth checking first and may well be intact. Do not generalise: the family carries a full destruction set (§2.2), and a build with those flags set will have removed shadow copies, cleared event logs and disabled Defender before encrypting. Establish which build you are facing before relying on local recovery.
2. **Inventory files above 256 MiB.** They are skipped by the size ceiling (§4) and remain intact. On a typical estate this covers virtual-machine disks, database files and backup archives - often the highest-value data on the host.
3. **Look for skipped directories.** Any directory holding a file whose name contains `canary`, `_decoy`, `honeytoken`, `honey`, `testfile` or `eicar` was left entirely untouched (§2.3). If the estate already deploys canary files, those trees survived.
4. **Capture memory before rebooting.** The RSA modulus is in the file, but per-run symmetric material exists only in process memory.
5. **Preserve evidence.** Retain one encrypted file together with its original from backup, plus a dropped ransom note.

6. **Containment is not time-critical for spread.** The payload has no lateral movement or network capability. Do not let isolation delay evidence capture - but investigate initial access independently, since the delivery mechanism is not determined by this payload.
7. **Scope impact without decrypting.** The `u64` at `filesize - 12` of each encrypted file gives the exact original size, enabling inventory and data-loss quantification directly.
8. **No negotiation channel exists in this sample.** No address of any kind is published in the note or the configuration.

---

6. Detection Guidance

Highest-value signatures

Signature	Why
ChaCha state constant `D5 46 E9 20 F6 39 B1 65 D2 4B FE 12 22 70 78 D9`	survives recompilation and reconfiguration; strongest variant-hunting signature
Encrypted-file footer: last 4 bytes `52 42 4D 55` **and** `u64` at `-12` equal to `filesize - 296`	structural and extension-independent; identifies renamed victim files
Configuration footer `52 42 4D 55` + `u32` length + `21 44 4E 45` at end of file	identifies configured builds and carved configurations
Unsigned PE carrying the `srmscan.exe` / `Windows System Resource Monitor` / `Microsoft Corporation` version resource and the `Microsoft.Windows.Srmscan` manifest identity	pivot for related builds; may also surface unrelated abuse of the same cover identity
Rust source paths `payload/src/engine.rs`, `payload/src/config.rs`	present while the project layout is unchanged

Complete YARA rules are provided in the accompanying full analysis.

Behavioural detection

- Bulk `MoveFileExW` renames to a single new extension by one process, skipping `Windows`, `Program Files`, `Program Files (x86)`, `ProgramData` and `System Volume Information`. The selective skip pattern is itself the signal.
- A process reading its own image file shortly after start, then beginning mass file I/O - the configuration-trailer read.
- Desktop wallpaper replacement via `SystemParametersInfo` by a recently written, non-shell process.
- A short-lived batch file written under the temporary path and executed via `WinExec`, deleting the parent executable. This is the final action of the top-level routine, so it fires after encryption completes - an endpoint that captures short-lived child processes will see it even when the sample itself is gone.
- The repeating unit on disk is **encrypt a tree -> drop the note -> change the wallpaper**, once per operating mode. A wallpaper change immediately following bulk renames is a high-confidence sequence.

Network detection

The payload opens no connection itself, so there is no protocol to signature. Watch the **process tree** instead: the command-execution paths (§2.2) run through `WinExec`, so any egress attributable to Umbra appears as traffic from a short-lived **child** process, not from the payload. Correlate child-process creation from the payload with any outbound connection in the same window.

Traffic originating from the payload process itself indicates a different build or an additional tool, and should be triaged as such.

---

7. Indicators of Compromise

Hashes

```
SHA-256  976ea6c54c8eea1b7c3d1d5227c50fd16f301518fd659a9ee4a770568850f553
SHA-1    a4079e9a82504a22d36e36bd42a2140fc278bf09
MD5      ed511819294ce9f3e53784f75aa73a88
imphash  1b68727006ce19fb446ae13503cb9f32
```

Host

```
Ransom note               README_<...>.txt
Dropped wallpaper file    ~umbra_wp.bmp
Self-delete batch         ~umbra_del.bat
Command-line flags        -path  -decrypt  -safe  -safe-path
Event logs cleared        Application  System  Setup  Security
Encrypted extension       .umbra
Encrypted-file footer     296 bytes, last 4 bytes 52 42 4D 55 ("RBMU")
Configuration footer      52 42 4D 55 | u32 length | 21 44 4E 45 ("!DNE") at EOF
ChaCha state constant     D5 46 E9 20 F6 39 B1 65 D2 4B FE 12 22 70 78 D9
Version resource          srmscan.exe / Windows System Resource Monitor /
                          Microsoft Corporation / 14.0.23107.0, unsigned
Manifest identity         Microsoft.Windows.Srmscan
Ransom note markers       "~~~ UMBRA", ">>>> Your personal DECRYPTION ID:"
```

Network

None. No `.onion` host, URL, e-mail address, TOX ID or cryptocurrency address is present in the sample.

---

8. Attribution and Tracking Notes

No leak site, contact address or payment address is associated with this sample. No victim has been observed. Umbra is documented here as a **malware family**; nothing in the sample evidences an operational extortion infrastructure.

8.1 This build was very probably not deployed against a victim

Six independent observations point the same way:

- the ransom note names **no contact channel**, which makes extortion impossible by construction - there is no way for a victim to pay;

- the reporting URL list is **empty**, and three of the ten configuration string slots are unpopulated;
- **all six** destruction and command flags are clear; only encrypt, note and wallpaper are enabled;
- the image carries a **test path**, `C:\TestFiles`, twice;
- it carries dedicated mode banners - `=== Test Mode ===`, `=== Test Complete ===`, `=== Safe-Path Mode ===`, `=== Safe-Path Complete ===` - and the command-line flags `-safe` and `-safe-path`;
- operator-facing status lines (`ENC:`, `Drives:`, `WP: set`, `report sent`) are written by a binary marked GUI, which has no console.

Against that: the RSA-2048 modulus is genuine - 2048 bits, odd, no small factors, composite - so a real key pair exists and someone holds the private half. The cipher stack, the wallpaper artwork and the extension are all production-quality.

Reading: a production-grade engine with an unfilled campaign configuration. The evidence does not distinguish between a QA build, a demonstration build, a sample taken from a builder before configuration, and an operator who deliberately ran encryption only. It does argue against this specific file having been used in a real extortion.

Confidence: good. All 73 obfuscated strings were recovered and none carries a contact channel, so the "no way to pay" observation no longer rests on an incomplete decode.

Tracking priorities:

- Retro-hunt on the ChaCha state constant and the `RBMU` footer to identify additional builds.
- Compare configuration trailers between builds - the extension, exclusion lists and RSA modulus are per-build values and the modulus is a direct operator discriminator.
- Watch for a build that imports networking APIs; that would mark a substantive capability change rather than a reconfiguration.

Static analysis. Tools: IDA Pro with Hex-Rays, Ghidra, pefile, YARA. 2026-08-07