Bavacai

Technical Analysis — Windows

REVERSE-ENGINEERED REPORT

RansomLook · ransomlook.io
File last modified: 2026-05-06

Sample SHA-256: `86b4d075d5bd0c49cbb21fd43935789b6612a2165273cc158dd0607b68941d04`

BAVACAI Ransomware — Full Analysis

1. Sample Identification

Field	Value
Family	BAVACAI (self-identified via `.BAVACAI` extension; internal project name `locker_win_x64_encrypter`)
SHA-256	`86b4d075d5bd0c49cbb21fd43935789b6612a2165273cc158dd0607b68941d04`
MD5	`7ae9c8b779ce3114199f4fd6335f681d`
Type	PE32+ x86-64 console subsystem, Windows
Size	752 640 bytes (735 KB)
Language	C++17 (MSVC) — STL streams, `<filesystem>`, `nlohmann::json` v3.11.3, MFC fragments (`afxGlobalData`), Microsoft `osyncstream`
Compile timestamp	not relied upon for attribution
PDB path	`J:\edu\niggerProject\bin\Publish\locker_win_x64_encrypter.pdb`
Image base	`0x140000000`
Sections	5 — `.text` (`0x70000`), `.idata` (`0x680`), `.rdata` (`0x3E980`), `.data` (`0x8000`), `.pdata` (`0x6000`)
Functions	2 253 total (284 named after Lumina pull + analysis renames; 931 library detected)
Entry point	`0x1400426B8` (`start` — MSVC CRT init) → `main` at `0x14001F890`

The binary is a **single-stage ransomware encryptor** built with verbose debug logging left in. Strings are not obfuscated — the only encrypted artefact is the embedded JSON configuration (PE resource id 101). The PDB path includes the developer's project tree, an internal slur as folder name, and the build subfolder — strong amateur-development signal. There is no anti-debug, no anti-VM, no anti-sandbox, no string obfuscation, no API hashing, no code injection, no packer; the whole pipeline is plainly visible to a casual reverser. The binary nonetheless ships with curated kill lists, multi-vector recovery inhibition, network-share enumeration, multi-threaded scanning, and a per-victim PKI hierarchy.

Imports (10 DLLs)

DLL	Count	Purpose
KERNEL32	139	Threading (`CreateThread` , SRW locks), file I/O (`CreateFileW` , `FindFirstFileW` , `FindFirstVolumeW` , `WriteFile`), drives (`GetLogicalDrives` , `GetDriveTypeW` , `SetVolumeMountPointW` , `QueryDosDeviceW` , `GetVolumePathNamesForVolumeNameW`), processes (`CreateProcessW` , `OpenProcess` , `TerminateProcess` , `CreatePipe` , `WaitForSingleObject`), resources (`FindResourceW` , `LoadResource` , `LockResource` , `SizeofResource`), CRT support
ADVAPI32	17	Legacy CryptoAPI (`CryptAcquireContextW` , `CryptGenKey` , `CryptImportKey` , `CryptExportKey` , `CryptDuplicateKey` , `CryptDestroyKey` , `CryptEncrypt` , `CryptGetKeyParam` , `CryptGenRandom` , `CryptReleaseContext`), token query (`OpenProcessToken` , `GetTokenInformation`), registry (`RegCreateKeyExW` , `RegSetValueExW` , `RegGetValueA/W` , `RegCloseKey`)
GDI32	13	Bitmap composition for the wallpaper option (`CreateCompatibleDC` , `CreateCompatibleBitmap` , `CreateFontW` , `TextOutA` , `GetBitmapBits` , `SelectObject` , `SetTextColor` , `SetBkMode` , `CreateSolidBrush` , `FillRect` , `GetTextExtentPoint32A` , `DeleteDC` , `DeleteObject` , `GetObjectW`)
USER32	10	Console window control (`ShowWindow` , `GetConsoleWindow` , `GetForegroundWindow` , `IsWindowVisible` , `EnumWindows` , `GetWindowThreadProcessId` , `GetShellWindow`), wallpaper application (`SystemParametersInfoW` , `FillRect` , `GetDC`), hotkey thread (`GetAsyncKeyState`)
SHLWAPI	6	Path predicates (`PathFileExistsW` , `PathFindExtensionW` , `PathIsDirectoryW` , `PathIsNetworkPathW` , `PathIsUNCW` , `StrStrIW`) — last one is the skip-path matcher
RstrtMgr	5	Restart Manager (`RmStartSession` , `RmRegisterResources` , `RmGetList` , `RmShutdown` , `RmEndSession`) — release handles on locked files
CRYPT32	4	Base64 codec (`CryptBinaryToStringA/W` , `CryptStringToBinaryA/W`) for blob encoding in note + registry
SHELL32	2	`SHEmptyRecycleBinW` , `SHGetFolderPathW`
MPR	1	`WNetGetConnectionW` — enumerate mapped network drives (mode `-network`)
urlmon	1	`URLDownloadToFileW` — fetch external IP from `api.ipify.org`

Notably absent: - No `WinHTTP` / `WinINet` / `Ws2_32` /socket APIs → **no exfiltration in this binary**, no C2 callback (the only outbound HTTP is `api.ipify.org` for victim IP fingerprinting) - No `OpenSCManager` / `EnumServicesStatus` → service stops are shell-spawned via `net stop` , not in-process - No `CreateToolhelp32Snapshot` → process kills are shell-spawned via `taskkill` , not in-process - No `IsDebuggerPresent` / `NtQueryInformationProcess` / `CheckRemoteDebuggerPresent` reachable from non-CRT code paths - No `BCryptOpenAlgorithmProvider` / CNG — relies entirely on **legacy CryptoAPI** (Microsoft Base Cryptographic Provider, `MS_DEF_PROV`) - No mutex API call (`CreateMutexA/W` / `OpenMutexW`) — no single-instance enforcement

---

2. Infrastructure

Field	Value
Onion (file server)	`http://t33zoj4qwv455fog7qnb2azi5xcdxkixughmmduzbw2rtdgryqfbh6id.onion`
Email	`nhuvgh@outlook.com`
TOX ID (qtox)	`7C564920870C0D33535D2012ECDDE389FE25BAF7AF427DD584EE39C04AF8CF024F8BFA93D8DB`
Note filename	`WHATS_HAPPEND.txt` (typo — "HAPPEND" instead of "HAPPENED")
Encrypted file extension	`.BAVACAI` (uppercase, appended after original extension)
Mutex	None
Payment	Free-text Tor / email negotiation (no wallet, no portal URL)
Master public key	RSA-2048, e=65537 — embedded in JSON config as base64 MS PUBLICKEYBLOB
Victim ID	`base64(RSA(masterPubkey, PCInfo_blob))` substituted for `[IDENTIFIER]` placeholder in note

Infrastructure surface is thin and operator-grade: a single Tor file server (claimed to host victim data), a clearnet outlook.com address, and a Tox ID. No leak site URL is embedded; no chat-portal UUID; no per-build campaign identifier. The note text is entirely static — there is no per-victim cohort key or affiliate flag.

---

3. Ransom Note

Filename and deployment

`WHATS_HAPPEND.txt` (note the misspelling) — deployed by `CreateNote` after encryption finishes. Function `CreateNote` writes the note in:

- The root of every encrypted drive (per drive iterator `Drive_EnumLogical` at `0x140030A80` and `Drive_EnumCipher` at `0x140039F30`)
- Every directory the scanner enters (per `DirWalker` at `0x14001AA20`)
- Path-skip exclusions apply: directories matching any `skipPathes` substring receive no note

If `openRequirementsOnFinish=true` (default), the note is `ShellExecute`-opened on the operator's console at the end of the run.

Content (decrypted JSON field `requirementsFileDataUTF8`, 723 bytes)

```
DON'T PANIC!!! YOUR FILES ARE PERFECT AND SAFE!
We've found flaws in your security system and gained access to your internal corporate network.
Your files were encrypted, and we can help you decrypt them and fix any existing security flaws.

We've also retrieved files from your servers, which will be published in 72 hours if you don't
contact us.

Our contact information: qtox -
7C564920870C0D33535D2012ECDDE389FE25BAF7AF427DD584EE39C04AF8CF024F8BFA93D8DB
e-mail: nhuvgh@outlook.com
our tor fileserver with your files - http://
t33zoj4qwv455fog7qnb2azi5xcdxkixughmmduzbw2rtdgryqfbh6id.onion

Your ID: [IDENTIFIER]

Please do not use file recovery services. They are either scammers or middlemen. In both cases,
you will simply pay more.
```

The `[IDENTIFIER]` placeholder is replaced at runtime by `Context__InitializePCInfo` at `0x1400070E0`:

1. Build a plaintext UTF-8 PCInfo blob via `Util_GetPCInfo` at `0x140033040`. The exact line-by-line format is:

   `IP: <api.ipify.org response> PC-Name: <GetComputerNameA> Domain: <GetComputerNameExA(ComputerNameDnsDomain)> CPU: <RegGetValueA HKLM\HARDWARE\DESCRIPTION\System\CentralProcessor\0\ProcessorNameString> RAM: <GlobalMemoryStatusEx.ullTotalPhys / 1048576> MB Disks: \\.\PhysicalDrive0 \\.\PhysicalDrive1 ...`

   Each section is omitted when the underlying API call fails. Disks are enumerated by opening `\\.\PhysicalDrive<N>` for increasing N until `CreateFileA` fails.
2. RSA-encrypt with master public key (multi-block PKCS#1 v1.5 → 5 × 256 = 1280 B output for the ~1 KB plaintext)
3. Base64-encode via `CryptBinaryToStringA(CRYPT_STRING_BASE64 | CRYPT_STRING_NOCRLF)`
4. Single string-replace of `[IDENTIFIER]` substring with the resulting base64 ID

Because the plaintext is a deterministic function of `(hostname, DNS domain, CPU model, RAM, physical-disk count, external IP)`, two runs from the same machine produce different ciphertexts (RSA-PKCS#1 padding randomises) but the **underlying plaintext digest is reproducible** — useful for attacker-side de-duplication of victims contacting from different IPs over time.

The 72-hour exfil claim is not backed by code in this build — see §10 (Exfiltration). The ransom note hints at exfiltration the binary cannot perform.

---

4. Execution Flow

`main` at `0x14001F890`

```
1.  setmode(stdout, _O_U16TEXT) — wide console output
2.  Argv parsing loop (i=1..argc):
      -help        → print usage, exit
      -network     → set flag: only network/UNC drives encrypted
      -skip_misc   → set flag: skip preRunCommands, recycle, wallpaper
      <unknown>    → "Unknown argument: <arg>" warning
3.  App_Run (0x1400081D0):
      a. SHGetFolderPathW for CSIDL 26 (APPDATA), 28 (LOCAL_APPDATA),
         46 (COMMON_APPDATA), 35 (WINDOWS), 36 (SYSTEM) — all appended
         to runtime skipPaths
      b. Context::LoadSettings → decrypt resource SETTINGS, parse JSON,
         hydrate global settings (extension, public key, etc.)
      c. Context::InitializeKeys → load HKLM\SOFTWARE\PAIDMEMES or
         generate fresh RSA-2048 victim keypair
      d. Crypto::RSA::ctor (master pubkey context)
      e. Context::InitializePCInfo → fetch external IP, build victim ID,
         substitute [IDENTIFIER] in note template
      f. Crypto::RSA::Cleanup
4.  hideConsole=true → ShowWindow(GetConsoleWindow(), SW_HIDE)
5.  Spawn Thread_HotkeyMonitor (0x14001EB90) — VK_INSERT/VK_HOME watcher
6.  Run preRunCommands (unless -skip_misc): 30 shell commands via
    Util_RunProcessElevated (no PPID spoof; sysnative WoW64 hop)
7.  FindFirstVolumeW / FindNextVolumeW iteration over all volumes
    including non-lettered \\?\Volume{GUID}\ entries; for each:
      - QueryDosDeviceW validates the NT mapping
      - Drive_EnumLogical (GetVolumePathNamesForVolumeNameW +
        SetVolumeMountPointW) mounts unmounted volumes at temp letters
8.  Drive_EnumCipher (GetLogicalDrives + GetDriveTypeW filter to
    REMOVABLE+FIXED+RAMDISK) — populates the local-drives vector
9.  Drive_EnumNetworkConnections (WNetGetConnectionW) — populates the
    mapped-network-drives vector
10. Privilege check: OpenProcessToken(TOKEN_QUERY) +
    GetTokenInformation(TokenElevation) → r15 = TokenIsElevated
11. Branch on elevation (master gate at 0x140020e8e):
      - elevated   → encrypt local drives only (the result of step 8)
                     + Util_RunProcess(<own_path> -network -skip_misc)
                       with PPID spoofed to explorer.exe to fork a
                       sibling process that handles the network drives
                       in parallel (see §10 PPID Spoofing)
      - not elev.  → encrypt mapped network drives only (step 9)
12. For each drive selected:
      a. CreateNote at root
      b. DirWalker recursion
         - StrStrIW skip-path check at every directory
         - For each file: extension skip check, then EncryptFile
         - threadPool=true → enqueue file path on thread pool
           (threadPoolMaxThreads=32, priority queue pulls
            threadPoolPriorityExtensions first)
13. Wait for all encryptor threads to drain
14. byte_1400B7B3C = 1 → signal Thread_HotkeyMonitor to exit
15. removeRecycle=true → SHEmptyRecycleBinW(NULL) per drive
16. backgroundImage=true → CreateAndSetBackgroundImage (renders the
    note text into a 1280×1024 BMP and sets it as wallpaper via
    SystemParametersInfoW(SPI_SETDESKWALLPAPER)) — disabled by this
    build's config (backgroundImage=false)
17. postRunCommands: 0 entries in this build — no-op
18. openRequirementsOnFinish=true → open the note in default handler
```

Privilege-asymmetric encryption strategy

Steps 8–11 implement a deliberate **privilege-aware target split**:

Token state at runtime	Target	Drive enumerator	Driven by
Elevated (admin)	Local drives (REMOVABLE + FIXED + RAMDISK)	`Drive_EnumCipher` (`0x140039F30`)	direct in-process loop
Not elevated	Already-mapped network drives	`Drive_EnumNetworkConnections` (`0x14003A160`)	direct in-process loop, OR self-relaunched sibling

When admin, the binary additionally spawns a child copy of itself with the command line `<own_path> -network -skip_misc`, using the PPID-spoofing primitive described in §10 to make the child appear as a descendant of `explorer.exe`. The child then runs as the "non-elevated" branch and processes network drives in parallel with the admin parent's local-drive pass. The `-skip_misc` flag suppresses re-execution of `preRunCommands`, recycle-bin emptying, and wallpaper setting in the child — those side effects only run once, in the parent.

When not admin, the binary processes only what it has access to via existing user mappings, and does not attempt elevation, escalation, or new SMB authentication.

Volume coverage beyond drive letters

Step 7 (`FindFirstVolumeW` loop) is **independent of drive letters**. It runs before the elevation check and reaches every volume the OS exposes to the active user, including hidden recovery partitions, EFI System Partitions, and BitLocker-suspended volumes that have no letter assignment. For each such volume it calls `SetVolumeMountPointW` to mount it at a temporary letter, after which the standard drive-letter pipeline picks it up. The hard-coded `skipPathes` entries `B:\Boot`, `A:\Boot`, `B:\EFI`, `A:\EFI`, and `:\Boot` are expressly there to guard against re-encryption of these mount targets.

Thread architecture

When `threadPool=true` (default):

- A worker pool of up to `threadPoolMaxThreads` (default 32) consumes a producer-consumer queue fed by `DirWalker`
- The queue prioritises extensions in `threadPoolPriorityExtensions = [".sql", ".bak", ".VHDX"]` — ensures high-value VM/database files encrypt early so kills happen before volumes are unmounted by service stops
- One auxiliary thread (`Thread_HotkeyMonitor`) runs in parallel watching `GetAsyncKeyState` polled at ticks (no blocking hook)

If `threadPool=false`, `EncryptFile` is called inline from `DirWalker` for each file.

---

5. Encryption System

Key Hierarchy (3 layers)

```
                [Master RSA-2048 keypair]
                 (private kept by operator,
                  public embedded in JSON config)
                           |
                           | wraps
                           ▼
              [Victim RSA-2048 keypair]
              (generated/persisted in HKLM\SOFTWARE\PAIDMEMES;
               PRIVATEKEYBLOB stored under PRIVATE,
               wrapped with master pub via RSA-PKCS1)
                           |
                           | wraps
                           ▼
        [Per-file ChaCha20 32-byte key + 8-byte nonce]
        (32+8=40 bytes random per file via CryptGenRandom,
         wrapped with victim pub key, footer-appended)
```

This is a textbook **operator-master / victim / per-file** three-tier key hierarchy. Recovery requires only the master private key — every encrypted file carries its own self-sufficient key envelope.

Asymmetric: RSA-2048 (legacy CryptoAPI)

Parameter	Value
Algorithm	RSA (Microsoft Base Cryptographic Provider, `PROV_RSA_FULL`)
Key size	2048 bits
AlgID	`CALG_RSA_KEYX` (`0x0000A400`)
Padding	PKCS#1 v1.5 (default for `CryptEncrypt` with `CALG_RSA_KEYX`, no `CRYPT_OAEP` flag set)
Public exponent	65537 (0x10001)
Master public key (config)	base64 MS PUBLICKEYBLOB, see appendix
Master modulus (BE, hex first 64 B)	`d1db59…` (full PEM extracted)

`Crypto::RSA::Encrypt` at `0x14003D210` chunks input into ≤245-byte blocks (RSA-2048 PKCS#1 max plaintext) and concatenates the 256-byte ciphertexts. It is used for: - Per-file 40-byte key+nonce wrap (1 block × 256 B) - Victim PRIVATEKEYBLOB wrap during key generation (5 blocks × 256 B = 1280 B) - PCInfo wrap for victim ID generation (5 blocks × 256 B = 1280 B, base64-encoded)

Symmetric: ChaCha20 (DJB original variant, no MAC)

Parameter	Value
Algorithm	ChaCha20 (custom implementation, not BCrypt/CryptoAPI)
Sigma constant	`expand 32-byte k` — DJB classic 16-byte constant
State layout	64 B init state + 8 B keystream byte counter at offset 0x80
Key	32 bytes (per-file from `CryptGenRandom`)
Nonce	8 bytes (per-file from `CryptGenRandom`) — **DJB original**, not IETF 12-byte
Block counter	8 bytes (initial = 0), at state[12..13]
Authentication	**None** — no Poly1305, no HMAC, no MAC of any kind
Per-file rekey	Yes (32+8=40 fresh bytes per file)

`ChaCha20__InitState` at `0x140002E90` performs the canonical state setup:

```
state[0..15]  = "expand 32-byte k"     // sigma
state[16..47] = key[0..31]              // 32-byte key
state[48..55] = 0                       // block counter (low+high = 0)
state[56..63] = nonce[0..7]             // 8-byte nonce
state[128]    = 64                      // keystream byte index, set in EncryptFile
                                        // ("flush" forces ChaCha block on first XOR)
```

The block function at `sub_140003390` performs the standard 20-round ChaCha20 block (10 double-rounds: column + diagonal). Output is XORed against plaintext byte-by-byte from `state[64..127]` (the keystream block).

File Encryption Process — `EncryptFile` at `0x14001A2A0`

```
1.  CreateFileW(GENERIC_RW, share=0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL)
2.  If failed with ERROR_SHARING_VIOLATION (32) or ERROR_LOCK_VIOLATION (33),
    AND path is not network → Util_RestartManagerUnlock:
       RmStartSession → RmRegisterResources → RmShutdown(RmForceShutdown)
       to force-close handles held by other processes (Office, SQL, …)
       → reopen
3.  Crypto::GenerateBytes(40) → buf[0..31]=key, buf[32..39]=nonce
4.  ChaCha20::InitState(state, key, nonce); state[128]=64; state[12..13]=0
5.  Crypto::RSA::Encrypt(victim_pubkey, buf40) → wrap = 256 B
6.  if (sizeof(wrap) != 256) throw "Bad password size: …"
7.  Crypto::CryptFile(hFile, state, ?, fileSize)  ── see §5 below
8.  Build footer (1544 B): victim_priv_blob_wrapped (1280) ‖ wrap (256) ‖
    qword(0x6) (8 bytes constant — sizeof(L".crypt") in chars)
9.  WriteFile(hFile, footer, 1544, offset=fileSize)
10. CloseHandle, MoveFileW(orig → orig + ".BAVACAI")
```

Intermittent Encryption — `Crypto::CryptFile` at `0x14003DDE0`

Two parameters from the JSON config drive the pattern (`bytesForEncrypt` is loaded into global `::Size` via `LODWORD` so its 32-bit value is what is consulted in the file-size comparison):

- `bytesForEncrypt` (BFE) = **630 057 793 bytes** (≈ 601 MiB) — Range A maximum buffer size
- `bytesCryptAndSkip` (BCS) = **352 055 393 bytes** (≈ 336 MiB) — chunk size of the inner encrypt-N / skip-N pattern

Algorithm:

```
size_A = min(fileSize, BFE)
read [0, size_A] into buffer of size_A bytes
v12 = 0
while v12 < size_A:
    encrypt min(BCS, size_A - v12) bytes at offset v12 (XOR ChaCha20 keystream)
    v12 += 2 * BCS                                          # skip BCS bytes
write buffer back at file offset 0

if fileSize > 2 * BFE:
    offset_B = fileSize - BFE
    read [offset_B, fileSize] into buffer of BFE bytes
    apply the same encrypt-BCS / skip-BCS inner loop on this buffer
    write buffer back at offset_B
```

Effective coverage (with the in-build constants BFE = 601 MiB, BCS = 336 MiB):

File size	Range A encrypts	Range B encrypts	Total encrypted	Coverage
≤ 336 MiB	Whole file (one pass, no skip triggered)	—	full file	**100 %**
336 MiB – 601 MiB	`[0, 336 MiB]`	—	336 MiB	56 % – 100 %
601 MiB – 1.2 GiB	`[0, 336 MiB]` (within the 601 MiB Range A buffer)	—	336 MiB	28 % – 56 %
> 1.2 GiB	`[0, 336 MiB]`	`[fileSize − 601 MiB, fileSize − 265 MiB]` (the first 336 MiB inside the last-601 MiB window)	672 MiB	6 % – 56 % (decreases as file grows)

Two consequences of this exact wording of the inner loop:

1. **Files larger than 1.2 GiB always have an unencrypted 265 MiB tail at the very end.** Range B encrypts only the first 336 MiB of the last-601 MiB window, not the last 336 MiB of the file. The bytes in `[fileSize − 265 MiB, fileSize]` are never touched.
2. **Range A coverage stops at offset 336 MiB even for files between 600 MiB and 1.2 GiB.** The Range A buffer is 601 MiB but the inner loop encrypts only the first 336 MiB inside it, then the `v12 += 2*BCS` jump goes past the buffer end and the loop exits. The bytes `[336 MiB, 601 MiB]` of the Range A buffer are read into memory and immediately written back unmodified.

Because BCS itself is 336 MiB (larger than virtually any user document), the "skip" is **never observed** within a single encrypted span — the `encrypt N, skip N` pattern degenerates into "encrypt one contiguous chunk per Range, then exit". The intermittent-encryption knob is effectively dormant in this build; what acts is the per-Range cap. For typical files (≤ 336 MiB), the binary performs **full file encryption**.

The configured constants make the encryption strategy practically equivalent to: "encrypt up to 336 MiB at the start of every file, plus 336 MiB ~600 MiB before the end of files larger than 1.2 GiB". The strategy is tuned to corrupt VHDX/database/backup files large enough to make full-encryption time-prohibitive while still touching enough of the file structure to render it unusable.

Encrypted File Format

```
[ original file content, partially XORed with ChaCha20 keystream      ]
[ size: original size (unchanged) — encryption is in-place            ]

----- Footer: 1544 bytes appended at end -----
[ 0      ..  1279 ] : RSA(masterPubkey, victim_PRIVATEKEYBLOB)
                      — 5 × 256-byte PKCS#1 blocks
                      — recoverable only with master private key

[ 1280   ..  1535 ] : RSA(victim_pubkey, ChaCha20_key32 ‖ ChaCha20_nonce8)
                      — 1 × 256-byte PKCS#1 block
                      — recoverable with victim private key

[ 1536   ..  1543 ] : 0x06 0x00 0x00 0x00 0x00 0x00 0x00 0x00
                      — fixed 8-byte trailer = sizeof(L".crypt") = 6
                      — usable as YARA tail magic
```

Recovery flow (operator-side):

1. Operator has master private RSA-2048 key
2. Decrypt footer[0..1279] → victim PRIVATEKEYBLOB
3. Import victim private key into CryptoAPI
4. Decrypt footer[1280..1535] → 32 B key + 8 B nonce
5. Re-init ChaCha20 state, XOR over Range A (and Range B if applicable) → plaintext
6. Truncate file to (size – 1544), strip `.BAVACAI` extension

Recovery feasibility for the victim: zero. Without the master private RSA key, neither inline footer nor registry blob is reachable. Both `bavacai_rsa_pubkey.pem` (2048-bit n+e) and the registry-resident victim public key are public information; the corresponding private keys are not in the binary.

---

6. File Targeting

Targeted extensions

All files except those matched by `skipExtensions` — there is no allow-list. Extension matching uses `PathFindExtensionW` followed by case-insensitive comparison against the skip list.

Excluded directories — substring match via `StrStrIW` on each directory path

JSON config `skipPathes` (typo) — 13 entries:

Path	Matches
`C:\perflogs`	OEM diag
`C:\Intel`	Intel drivers
`C:\HP`	HP drivers
`C:\AMD`	AMD drivers
`C:\Dell`	Dell drivers
`C:\Drivers`	generic drivers
`C:\inetpub`	IIS web root
`B:\Boot` , `A:\Boot`	floppy boot
`B:\EFI` , `A:\EFI`	EFI partitions on letter B/A
`C:\ProgramData\Anydesk`	RMM (probably the operator's own remote-access tool)
`:\Boot`	any drive's boot dir

Runtime additions (appended in `App_Run`):

CSIDL	Resolves to	Purpose
26 (APPDATA)	`%AppData%\…`	Roaming user app data
28 (LOCAL_APPDATA)	`%LocalAppData%\…`	Local user app data
35 (WINDOWS)	`%windir%`	Windows itself (avoid bricking)
36 (SYSTEM)	`%windir%\System32`	System binaries
46 (COMMON_APPDATA)	`%ProgramData%\…`	Machine-wide app data

The hard-coded path list also contains hard-coded JSON entry `\AppData\` (substring), which means **any directory path containing the substring "\AppData\" is skipped** — a wider exclusion than just `%AppData%`. Likewise `\Google\Chrome\` excludes Chrome user data.

Excluded extensions

Ext	Notes
`.dll`	exe stability
`.sys`	kernel drivers
`.readtext*`	non-existent extension — unclear intent (possibly placeholder for `.readme.*` family the dev forgot to finalise)
`.readtext95`	same as above

The `.readtext*` glob is **never** used — comparison is exact-string, not glob. So in practice only `.dll` and `.sys` are skipped.

Excluded files

None hard-coded. The ransom note `WHATS_HAPPEND.txt` is not in any excluded list — it would be re-encrypted by a second pass if the binary were re-run, but the encrypted file would have extension `.BAVACAI`, breaking the substring check on note name. The 4 bytes of effort it would have taken to skip the note are absent.

Linux/NAS leftovers — `startPathes` (sic)

```
"/volume/", "/vmfs/", "/share/", "/volumeusb/",
"/volume0/", "/volume1", "/volume2", "/volumeUSB1",
"/volume3", "/volume4", "/volume5", "/volume6", "/shares"
```

These 13 entries are **completely dead in this Windows binary**. The string `"startPathes"` is absent from `.rdata` — neither IDA's strings analysis nor `strings(1)` over the raw image returns a match. The Windows binary therefore cannot read this JSON field; the entries are pure config bloat inherited from a Linux/ESXi/Synology variant (notable `/vmfs/` for ESXi, `/volumeusb/` and `/volume[0-6]/` for Synology DSM, `/shares` for QNAP). The presence of these paths in the JSON template strongly suggests an existing or planned non-Windows counterpart in the same family.

---

7. Recovery Inhibition

All recovery-inhibition is shell-spawned via `cmd.exe /c <cmd>` (not in-process). The 7 commands embedded in `preRunCommands` (config-defined, not hard-coded in `.text`):

Command	Purpose	Notes
`vssadmin.exe Delete Shadows /All /Quiet`	Wipe all shadow copies	Standard vector
`wbadmin delete backup -keepVersion:0 -quiet`	Delete Windows Backup catalog	
`wbadmin DELETE SYSTEMSTATEBACKUP`	Delete system state backup	
`wbadmin DELETE SYSTEMSTABACKUP -deleteOldest`	**Typo** (`SYSTEMSTABACKUP` ≠ `SYSTEMSTATEBACKUP`) — silent failure	Inert command
`wmic.exe SHADOWCOPY /nointeractive`	Implicit (with `delete` action absent) → does nothing destructive in this exact form	Likely intended `wmic.exe shadowcopy delete /nointeractive` — **typo, inert**
`bcdedit.exe /set {default} recoverynabled No`	**Typo** (`recoverynabled` instead of `recoveryenabled`) — bcdedit error, no change applied	Inert command
`bcdedit.exe /set {default} bootstatuspolicy ignoreallfailures`	Disable Windows Recovery Environment auto-launch	Effective

Two of the seven recovery-inhibition commands silently fail due to typos (`SYSTEMSTABACKUP` , `recoverynabled`); a third (`wmic.exe SHADOWCOPY /nointeractive` without `delete`) is essentially a no-op listing. Net effect: VSS deletion and bcdedit boot-status policy do take effect; system-state backup deletion and bcdedit recovery-enable do not.

---

8. Targeted Services (10 explicit `net stop`)

```
MSSQLServerADHelper100   MSSQL$ISARS         MSSQL$MSFW
SQLAgent$ISARS           SQLAgent$MSFW       SQLBrowser
REportServer$ISARS       SQLWriter
```

Plus `taskkill -f -im` of equivalent service binaries. Service stops cover the **MSSQL Server** product family (Forefront/ISA-Reporting "ISARS" and "MSFW" are the historical instance names of Microsoft Forefront Threat Management Gateway / ISA Server) — i.e. the malware targets very specific legacy MSSQL deployments.

There is no `OpenSCManager` / `EnumServicesStatus` / `StopService` in-process — all stops go via `cmd.exe /c net stop <svc>`. There is no service-name pattern matching; the list is fixed.

---

9. Targeted Processes (13 `taskkill -f -im`)

```
sqlbrowser.exe          sql writer.exe (typo - should be sqlwriter.exe)
sqlserv.exe             msmdsrv.exe
MsDtsSrvr.exe           sqlceip.exe
fdlauncher.exe          Ssms.exe
SQLAGENT.EXE            fdhost.exe
ReportingServicesService.exe
msftesql.exe            pg_ctl.exe
"taskkill -f -impostgres.exe"  (missing space — never executes)
```

Two entries are broken:

- `sql writer.exe` — process name with embedded space; without quotes `taskkill` parses this as two arguments, fails
- `taskkill -f -impostgres.exe` — `-im` joined with `postgres.exe`; `taskkill` rejects the malformed switch

So PostgreSQL (`postgres.exe`) and SQLWriter (`sqlwriter.exe`) effectively survive `preRunCommands`. The intent was clearly to cover both Microsoft SQL Server, SSIS/SSRS/SSMS, MS Search, and PostgreSQL — but **the kill list is buggy in this build**.

---

10. Persistence & Evasion

Operator-side persistence (registry — victim side)

Hive	Subkey	Value	Type	Content
HKLM	`SOFTWARE\PAIDMEMES`	`PUBLIC`	REG_BINARY	base64(victim PUBLICKEYBLOB)
HKLM	`SOFTWARE\PAIDMEMES`	`PRIVATE`	REG_BINARY	base64(RSA(masterPubkey, victim PRIVATEKEYBLOB)) — 1280 B

Only these **two** values are written. There is no `SETTINGS` registry value — the wide string `"SETTINGS"` at `0x14009C6E0` is referenced exactly once, at `0x1400050CF`, where it is passed as the `lpType` parameter to `FindResourceW(101, "SETTINGS")` to load the ChaCha20-encrypted JSON config from the PE resource directory. Runtime configuration comes exclusively from the PE resource; no registry-based override path exists.

`Util_LoadKeyEntry_PUBLIC_PRIVATE` (`0x140004D30`) reads PUBLIC and PRIVATE on every run; if both decode and RSA-import successfully, fresh keys are not generated. With `regenerateKeysAlways=false` (default), this means **a single victim machine is encrypted with the same victim keypair across multiple runs of the binary**.

The `PAIDMEMES` registry key name is a deliberate edgy/troll handle — not an MSSQL vendor or product key.

Parent PID Spoofing (T1134.004)

`Util_RunProcess` at `0x14003AD60` is the malware's only PPID-spoofing primitive. The sequence:

1. `GetShellWindow()` → handle to the desktop's shell window (typically owned by `explorer.exe`)
2. `GetWindowThreadProcessId(shellWnd, &pid)` → resolve the explorer PID
3. `OpenProcess(PROCESS_CREATE_PROCESS = 0x80, FALSE, pid)` → handle on `explorer.exe` (no read/query rights — only the right needed for `UpdateProcThreadAttribute`)
4. `InitializeProcThreadAttributeList(NULL, 1, 0, &size)` to size the attribute list, then a real-allocated init pass
5. `UpdateProcThreadAttribute(attrList, 0, PROC_THREAD_ATTRIBUTE_PARENT_PROCESS = 0x20000, &hExplorer, sizeof(HANDLE), NULL, NULL)` — declares `explorer.exe` as the parent of the to-be-spawned process
6. `CreateProcessW(NULL, lpCommandLine, ..., dwCreationFlags = 0x80010 (EXTENDED_STARTUPINFO_PRESENT | CREATE_DEFAULT_ERROR_MODE | CREATE_NO_WINDOW), ..., &startupInfoEx, &pi)` with the extended `STARTUPINFOEX.lpAttributeList`

The single call site in `main` is at `0x140020D5D`, and the command line built immediately before is `<own_path> -network -skip_misc`. The admin process therefore self-relaunches with `explorer.exe` recorded as parent so that the network-drive child is not visibly a descendant of the admin malware in EDR/Sysmon process trees.

WoW64 redirection hop in `preRunCommands` execution

`Util_RunProcessElevated` at `0x140030530` (which runs every entry of `preRunCommands` and `postRunCommands` and also the hotkey-thread's explorer restart) wraps each command line as:

```
<windir>\SysWOW64\cmd.exe /c %windir%\sysnative\cmd.exe /c <command>
```

This is a **64 → 32 → 64 process hop** via the WoW64 redirection alias `sysnative`. The 32-bit `cmd.exe` (in `SysWOW64\`) starts as a WoW64 process; from a WoW64 process, the path `%windir%\sysnative\` resolves to `%windir%\System32\` (i.e. the 64-bit binaries) and re-spawns the native 64-bit `cmd.exe` to execute the actual command. Net effect for monitoring: the parent-child chain becomes `bavacai.exe → SysWOW64\cmd.exe (WoW64) → System32\cmd.exe → <recovery_command>`, breaking detection rules that match `<malware> → cmd.exe → vssadmin/taskkill` directly. This is not PPID spoofing — it adds extra real cmd parents into the chain rather than rewriting the recorded parent.

Console hide

`hideConsole=true` (default in this config) → after `App_Run` returns, `ShowWindow(GetConsoleWindow(), SW_HIDE)` removes the window from the user. This is not an analysis-evasion technique — Console-Mode hide does not prevent process or kernel-mode introspection — but it serves the operator UX (silent execution).

Hotkey monitor — `Thread_HotkeyMonitor` at `0x14001EB90`

A background thread polls (no hook) every iteration:

- **VK_INSERT (0x2D)** pressed → toggle visibility of every window owned by the current PID via `EnumWindows` + `ShowWindow(SW_HIDE/SW_SHOWNORMAL)`. Used by the operator to peek at the encryption progress on a hidden console.
- **VK_HOME (0x24)** pressed **AND** console window is foreground → spawn `taskkill /f /im explorer.exe` then `start explorer.exe` (via `Util_RunProcessElevated`, no PPID spoof). Forces a desktop refresh — used to make a wallpaper change visible immediately. Only triggers when the operator is sitting at the console.

These are **operator UX features, not anti-analysis**. They do not disable when a debugger is attached.

Anti-debug / anti-VM / anti-sandbox

None. The only `IsDebuggerPresent` calls are in MSVC CRT crash handlers (`__scrt_fastfail` , `__acrt_call_reportfault`) and serve the standard purpose of routing fatal-error reporting; they do not gate the encryption logic. There is no `RDTSC` timing check, no `CPUID` hypervisor-bit check, no `NtQueryInformationProcess(ProcessDebugPort)` , no `NtQuerySystemInformation(SystemKernelDebuggerInformation)` , no `GetTickCount` -delta check, no `WMI Win32_BIOS` / `Win32_ComputerSystem` VM-vendor lookup, no MAC-prefix VM check, no installed-program enumeration.

String obfuscation

None. All strings are stored either:

- In plaintext in `.rdata` (the implementation strings: error messages, JSON parse keys, registry value names, the markers `TRUMPTRUMP…` and `PUTLERPUTLER` themselves)
- In the ChaCha20-encrypted PE resource id 101 of type `SETTINGS` (the IOC-bearing strings: onion URL, email, Tox ID, ransom note text, master public key)

The encryption of the resource is functionally string obfuscation but trivially reversed — the ChaCha20 key (`TRUMPTRUMP…`) and nonce (`PUTLERPUTLER`) are themselves **plaintext strings** in `.rdata` , passed by reference to `ChaCha20::Init` at `0x14000511D` .

API hashing / dynamic resolution

None. Imports are static; no `LoadLibrary` / `GetProcAddress` walks; no `PEB.Ldr` traversal; no FNV/CRC32/SDBM hash table.

Anti-Analysis Summary

Technique	Unprotect / MITRE	Address	Description
File-content encryption	U0301 / T1486	`0x14001A2A0`	`EncryptFile` orchestration
Cryptography (RSA + ChaCha20)	U0703 / E1027.m04	`0x14003D210`, `0x140002E90`	RSA-2048 wrap + ChaCha20 stream
Encrypted PE resource (config)	T1027.013	`0x14003AA40`, `0x140005080`	`Utils::GetResource` + ChaCha20 over the SETTINGS resource
VSS Deletion	U0305 / T1070.004	`preRunCommands` shell-out	`vssadmin Delete Shadows /All /Quiet`
Kill Process	U0403 / T1057	`preRunCommands` shell-out	13 `taskkill -f -im`
Restart Manager file unlock	T1490	`0x140030F70`	`RmStartSession` + `RmShutdown`
Parent PID Spoofing	T1134.004	`0x14003AD60` (`Util_RunProcess`)	`UpdateProcThreadAttribute(PROC_THREAD_ATTRIBUTE_PARENT_PROCESS)` with `explorer.exe` for the network-drive self-relaunch
WoW64 sysnative hop	(no MITRE entry)	`0x140030530` (`Util_RunProcessElevated`)	64→32→64 cmd chain to break parent-child detection rules
Hidden Window	T1564.003	`App_Run` post-init	`ShowWindow(GetConsoleWindow(), SW_HIDE)` when `hideConsole=true`

There is no string obfuscation, no API hashing, no runtime anti-debug, no anti-VM, no sandbox evasion. The only proper obfuscation is the single ChaCha20 layer over the PE config resource, defeated by the plaintext key two `.rdata` entries away.

---

11. Command-Line Arguments

Argument	Effect
`-help`	Print usage block and exit (lists `-skip_misc` and `-network`)
`-network`	Encrypt only mapped network drives (via `WNetGetConnectionW`) — skips local volumes
`-skip_misc`	Skip `preRunCommands`, `postRunCommands`, recycle-bin emptying and wallpaper change

Anything else triggers `Unknown argument: <arg>` log line then a normal run.

---

12. Static Imports Summary

Category	Key APIs
Crypto (ADVAPI32)	`CryptAcquireContextW`, `CryptGenKey`, `CryptImportKey`, `CryptExportKey`, `CryptDuplicateKey`, `CryptDestroyKey`, `CryptEncrypt`, `CryptGetKeyParam`, `CryptGenRandom`, `CryptReleaseContext`
Crypto (CRYPT32)	`CryptBinaryToStringA/W`, `CryptStringToBinaryA/W` (base64)
File I/O	`CreateFileA/W`, `WriteFile`, `ReadFile`, `MoveFileW`, `DeleteFileW`, `SetFileAttributesW`, `SetEndOfFile`, `SetFilePointerEx`, `GetFileSizeEx`, `FindFirstFileW`, `FindNextFileW`, `FindClose`
Drive enum	`GetLogicalDrives`, `GetDriveTypeW`, `FindFirstVolumeW`, `FindNextVolumeW`, `GetVolumePathNamesForVolumeNameW`, `QueryDosDeviceW`, `SetVolumeMountPointW`, `WNetGetConnectionW`
Process	`CreateProcessW`, `OpenProcess`, `TerminateProcess`, `CreatePipe`, `WaitForSingleObject`, `WaitForSingleObjectEx`, `GetExitCodeProcess`, `InitializeProcThreadAttributeList`, `UpdateProcThreadAttribute`, `AssignProcessToJobObject`
Token / privilege	`OpenProcessToken`, `GetTokenInformation`
Threading	`CreateThread`, `ExitThread`, `AcquireSRWLockExclusive` / `Release` / `Try`, `SleepConditionVariableSRW`, `WakeAllConditionVariable`, `InitializeCriticalSectionEx`, `Tls*`
Registry	`RegCreateKeyExW`, `RegSetValueExW`, `RegGetValueA/W`, `RegCloseKey`
Network (HTTP only)	`URLDownloadToFileW` (urlmon) — `api.ipify.org` only
Restart Manager	`RmStartSession`, `RmRegisterResources`, `RmGetList`, `RmShutdown`, `RmEndSession`
GDI / wallpaper	`CreateCompatibleDC`, `CreateCompatibleBitmap`, `CreateFontW`, `TextOutA`, `GetBitmapBits`, `SystemParametersInfoW`
Console / window	`ShowWindow`, `GetConsoleWindow`, `GetForegroundWindow`, `EnumWindows`, `IsWindowVisible`, `GetAsyncKeyState`
Recycle bin	`SHEmptyRecycleBinW`
Resources	`FindResourceW`, `LoadResource`, `LockResource`, `SizeofResource`, `GetModuleHandleW`
Path	`PathFileExistsW`, `PathFindExtensionW`, `PathIsDirectoryW`, `PathIsNetworkPathW`, `PathIsUNCW`, `StrStrIW`

13. IDA Analysis — Renamed Functions

Address	Name	Size	Description
0x14001F890	main	0x2C44	Argv parse, App_Run, drive iter, encryption orchestration
0x1400081D0	App_Run	0x830	Runtime skipPath augmentation, settings load, key init, PCInfo
0x140005080	Context__LoadSettings	0x204C	Decrypt SETTINGS resource, parse JSON, populate globals
0x140007940	Context__InitializeKeys	0x88E	Load registry blobs or generate fresh victim RSA keypair
0x1400070E0	Context__InitializePCInfo	0x800	Build victim ID, substitute [IDENTIFIER] in note template
0x14003AA40	Utils__GetResource	0x290	FindResourceW (101) → LoadResource → vector
0x140002E90	ChaCha20__InitState	0x190	DJB-layout state init (key 32B, nonce 8B, sigma)
0x1400035A0	ChaCha20__Init	0x30	Wrapper setting state[16]=64 + state[6]=0
0x140003390	(block-fn)	inferred	20-round ChaCha20 block computation (column+diag rounds × 10)
0x14003C7B0	Crypto__GenerateBytes	0xD0	CryptAcquireContextW + CryptGenRandom(40)
0x14003C890	Crypto__RSA__GenerateKeys	0x410	CryptGenKey(CALG_RSA_KEYX,2048) + CryptExportKey ×2
0x14003CCA0	Crypto__RSA__ctor	0xA0	RSA context constructor
0x14003CD40	Crypto__RSA__InitializeKeys	0x2D0	CryptImportKey priv & pub from blobs
0x14003D010	Crypto__RSA__Duplicate	0x140	CryptDuplicateKey for key copy
0x14003D150	Crypto__RSA__Cleanup	0xC0	CryptDestroyKey ×2 + CryptReleaseContext
0x14003D210	Crypto__RSA__Encrypt	0xBD0	Multi-block RSA-2048 PKCS#1 encrypt
0x14001A2A0	EncryptFile	0x773	Per-file orchestration
0x14003DDE0	Crypto__CryptFile	0x220	Range A + (optional) Range B selection
0x14003DC60	ChaCha20__XorBlocks_Skip	0x180	Encrypt-N / skip-N inner loop
0x14001AA20	DirWalker	0x773	StrStrIW skip + FindFirstFileW recursion
0x140030A80	Drive_EnumLogical	0x410	GetLogicalDrives + mount unmounted volumes

Address	Name	Size	Description
`0x140039F30`	`Drive_EnumCipher`	0x230	Drive iter for `chiperDrives=true`
`0x14003A160`	`Drive_EnumNetworkConnections`	0x9C0	`WNetGetConnectionW` mapped drives (`-network`)
`0x140030F70`	`Util_RestartManagerUnlock`	0x2C0	RmStartSession+RmRegisterResources+RmShutdown
`0x140030530`	`Util_RunProcessElevated`	0x360	Spawn elevated cmd.exe child
`0x14003AD60`	`Util_RunProcess`	0x3A0	Generic CreateProcessW wrapper
`0x14003B440`	`Util_RunProcess_Inner`	0x190	Inner CreateProcessW
`0x140030890`	`Util_EmptyRecycleBin`	0x1F0	`SHEmptyRecycleBinW` per drive
`0x140039C50`	`Util_SetWallpaper`	0x2E0	`SystemParametersInfoW(SPI_SETDESKWALLPAPER)`
`0x140033040`	`Util_GetPCInfo`	0x150	Compose hostname + IP for victim ID
`0x140032060`	`Util_GetExternalIP_ipify`	0x5B0	URLDownloadToFileW api.ipify.org
`0x140004A00`	`Util_SaveKeysToRegistry`	0x320	RegSetValueExW PAIDMEMES.{PUBLIC,PRIVATE}
`0x140003E80`	`Util_LoadKeysFromRegistry`	0xA0	RegGetValueW
`0x140004D30`	`Util_LoadKeyEntry_PUBLIC_PRIVATE`	0x2D0	Helper that fetches both PUBLIC and PRIVATE values
`0x14001EB90`	`Thread_HotkeyMonitor`	0xC50	VK_INSERT/VK_HOME polling loop

14. Indicators of Compromise (IOCs)

Hashes

Type	Value
SHA-256	`86b4d075d5bd0c49cbb21fd43935789b6612a2165273cc158dd0607b68941d04`
MD5	`7ae9c8b779ce3114199f4fd6335f681d`

Network

Type	Value
Tor file server	`t33zoj4qwv455fog7qnb2azi5xcdxkixughmmduzbw2rtdgryqfbh6id.onion`
Email	`nhuvgh@outlook.com`
Tox (qtox) ID	`7C564920870C0D33535D2012ECDDE389FE25BAF7AF427DD584EE39C04AF8CF024F8BFA93D8DB`
Outbound HTTP	`https://api.ipify.org` (victim external IP discovery)

Files

Indicator	Value
Ransom note filename	`WHATS_HAPPEND.txt` (typo "HAPPEND")
Encrypted extension	`.BAVACAI` (uppercase, appended)
Footer signature (last 8 bytes)	`06 00 00 00 00 00 00 00` (constant per file)
Footer total size	1544 bytes (1280 + 256 + 8)
Wallpaper artefact	none in this build (`backgroundImage=false`)
External-IP scratch file	`xxxxxxxxx.crypt` (literal 11 chars + `.crypt` — 11-char placeholder, see below)

Registry

Key	Description
`HKLM\SOFTWARE\PAIDMEMES`	Root for victim key persistence
`HKLM\SOFTWARE\PAIDMEMES\PUBLIC`	Victim PUBLICKEYBLOB (REG_BINARY base64)
`HKLM\SOFTWARE\PAIDMEMES\PRIVATE`	RSA(masterPubkey, victim PRIVATEKEYBLOB) — 1280 B base64

Behavioural

- Spawns 30 child `cmd.exe` processes within seconds of launch (all from the static `preRunCommands` list)
- 13× `taskkill -f -im` of MSSQL/MSDTS/SSMS/PostgreSQL processes
- 7× `net stop` of MSSQL service variants (instances `ISARS` , `MSFW`)
- 1× `vssadmin Delete Shadows /All /Quiet` , 3× `wbadmin delete` , 1× `wmic.exe SHADOWCOPY` , 2× `bcdedit.exe /set {default}` — note 3 of these silently fail (typos)
- `RegCreateKeyExW HKLM\SOFTWARE\PAIDMEMES` , then `RegSetValueExW` of PUBLIC and PRIVATE blobs
- File handles released via `RmStartSession` for any locked target file
- `SHEmptyRecycleBinW` on every drive at end of run
- In-place encryption only — no shadow copy of original; `MoveFileW` rename to `<orig>.BAVACAI`
- 32 simultaneous encryption threads (`threadPoolMaxThreads` default)
- Outbound HTTPS to `api.ipify.org` exactly once per run

Distinctive Strings (`.rdata` , plaintext)

- `"TRUMPTRUMPTRUMPTRUMPTRUMPTRUMPTRUMP"` (35 chars — ChaCha20 key string)
- `"PUTLERPUTLER"` (12 chars — ChaCha20 nonce string)

- `"SOFTWARE\\PAIDMEMES"` (registry root)
- `"[IDENTIFIER]"` (note placeholder)
- `"--------------------------------"` (32 dashes — log separator)
- `"J:\\edu\\niggerProject\\bin\\Publish\\locker_win_x64_encrypter.pdb"` (PDB path)
- `"Crypto::RSA::GenerateKeys"`, `"Crypto::RSA::Encrypt"`, `"Context::InitializeKeys"`, `"EncryptFile"`, `"CryptFile"` (verbose log function names)
- `"chiperDrives"` (typo "cipher")
- `"skipPathes"` (typo "Paths")

Distinctive Strings (encrypted JSON config — `requirementsFileDataUTF8`)

- `"DON'T PANIC!!! YOUR FILES ARE PERFECT AND SAFE!"` (note opening line)
- `"WHATS_HAPPEND.txt"` (note filename)
- `"qtox - 7C564920870C0D33535D2012ECDDE389FE25BAF7AF427DD584EE39C04AF8CF024F8BFA93D8DB"` (full Tox ID)

---

15. MITRE ATT&CK Mapping

ID	Technique	Implementation
T1486	Data Encrypted for Impact	RSA-2048 wrap + ChaCha20 per-file in `EncryptFile`
T1490	Inhibit System Recovery	`vssadmin Delete Shadows`, `wbadmin delete backup`, `wmic.exe shadowcopy`, `bcdedit /set {default} bootstatuspolicy ignoreallfailures`, Restart Manager file unlock, recycle-bin emptying
T1489	Service Stop	`net stop` of 7 MSSQL service variants from `preRunCommands`
T1057	Process Discovery	implicit via `taskkill -f -im` (no enumeration code, fixed list)
T1083	File and Directory Discovery	`FindFirstFileW` / `FindNextFileW` recursion in `DirWalker`
T1135	Network Share Discovery	`WNetGetConnectionW` for mapped drives in non-elevated branch
T1134.004	Access Token Manipulation: Parent PID Spoofing	`Util_RunProcess` (`0x14003AD60`) sets `explorer.exe` as parent for the self-relaunched `<own_path> -network -skip_misc` child
T1082	System Information Discovery	`Util_GetPCInfo`: hostname, DNS domain, CPU model, total RAM, physical disk count
T1016	System Network Configuration Discovery	`URLDownloadToFileW("https://api.ipify.org")` for external IPv4
T1071.001	Application Layer Protocol: Web Protocols	single GET to `api.ipify.org` (no C2 channel)
T1112	Modify Registry	`RegCreateKeyExW HKLM\SOFTWARE\PAIDMEMES` + `RegSetValueExW` for victim key persistence
T1480.001	Execution Guardrails: Environmental Keying	hostname, domain and external IP enter the victim-ID derivation
T1027.013	Encrypted/Encoded Files: Encrypted Resources	ChaCha20 over PE resource id 101 (type `SETTINGS`) holding the JSON config
T1485	Data Destruction	`SHEmptyRecycleBinW` per drive at end of run
T1564.003	Hide Artifacts: Hidden Window	`ShowWindow(GetConsoleWindow(), SW_HIDE)` when `hideConsole=true`
T1059.003	Command and Scripting Interpreter: Windows Cmd	All recovery-inhibition + kills go through `cmd.exe /c <cmd>` (with the WoW64 sysnative hop in `Util_RunProcessElevated`)

16. Summary

BAVACAI is a **single-stage Windows x64 ransomware** distributed as a self-contained PE32+ console executable (~735 KB). Encryption combines **RSA-2048** (Microsoft legacy CryptoAPI, PKCS#1 v1.5, e=65537) for key wrapping and a **custom ChaCha20** implementation (DJB original 8-byte nonce variant, no Poly1305) for bulk file content. The cipher key for **every encrypted file** is freshly

generated from `CryptGenRandom`, wrapped with the victim's per-machine RSA-2048 public key, and appended along with the master-pub-wrapped victim PRIVATEKEYBLOB into a 1544-byte footer — yielding a **self-contained recovery envelope per file** that requires only the master private key.

The binary's **operator surface** is rich: 30 pre-run shell commands, 13 process kills + 7 service stops, multi-vector recovery inhibition (vssadmin / wbadmin / wmic / bcdedit), Restart Manager file-handle release, recycle-bin emptying, optional wallpaper change, optional console hide, mount-unmounted-volume support, mapped network drive enumeration, IPv4 external IP fingerprinting via `api.ipify.org`, and a 32-thread producer-consumer pool prioritising `.sql` / `.bak` / `.VHDX`. Configuration is read from a **single ChaCha20-encrypted PE resource** keyed by the plaintext strings `"TRUMPTRUMPTRUMPTRUMPTRUMPTRUMPTRUMP"` / `"PUTLERPUTLER"` — trivially decoded.

Cryptographic engineering is **mostly correct but unauthenticated** (no MAC over file content, vulnerable to bit-flip on the encrypted stream, but in practice unrecoverable for the victim because the per-file key is unrecoverable without the master). The engineering quality is otherwise **amateur**: the PDB path includes a slur as folder name, JSON keys contain typos (`skipPathes`, `chiperDrives`), the recovery-inhibition shell list contains 2 typos that silently fail, the kill list contains 2 malformed entries that never run, and the verbose error logs leak full class::method names (`Crypto::RSA::Encrypt`, `Context::InitializeKeys`, ...) that map directly back to the source layout.

Operational sophistication is **moderate**: the kill-list is curated for MSSQL/MSDTS/SSMS environments specifically (instance names `ISARS`, `MSFW` are historical Forefront/ISA Server), the `threadPoolPriorityExtensions` includes `.VHDX` (Hyper-V) and `.bak` (SQL Server backups), and Restart Manager is correctly used to recover locked database files — these are deliberate choices targeting Windows Server / Hyper-V / SQL Server estates. Network-share encryption is opt-in via `-network`; lateral movement, AD enumeration, PowerShell invocation, RDP enumeration, scheduled-task creation, service install — **none are present**. There is no exfiltration component, despite the "72 hours if you don't contact us" leak claim in the note.

The infrastructure is **single-handle** (one Tor file server, one outlook.com mailbox, one Tox ID), the per-victim ID is reproducible-by-hostname (no random salt), and the binary contains no anti-debug, no anti-VM, no anti-sandbox, no string obfuscation, no API hashing, no packing — making this an **inventory-friendly low-effort variant** suitable for IR triage but with **no in-place recovery option for the victim**: every file's key is wrapped exclusively against the master key.

Appendix A — Master RSA-2048 Public Key (PEM)

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0dtZGZGXTx1FWagCvFRC
Hr616k+7WrBT9CYCbyUVGSvuJ2Oe1pgW/U2N4LTmWxdYlZVFXEdRdKChfBC9biu5
meLxps2zpdsvV/9IAKUYIfSvSE3UBVaw7EUaYAS1ykj6o+FLTwEp+jqro001P7vn
GoiMIADie/v1oibQk1AQjKByEMmaZqGjD7wfpVeV/wdcLhPt+r6TGz7IldayOG6o
3IM52DFtjihWBL3g0iJYHPZW0CHb23fTz+EEh8JHshuH1G4lEvmZBTfaQjpFpCHo
2Rm2meNODI968tHKJ6ZMzGx0SUaVvXmuKckbCUg7Ve3TjvdMT/3NEFEeDJ9WBbn4
yQIDAQAB
-----END PUBLIC KEY-----
```

Modulus (BE): `d1db59 19 91 97 4f 1d 45 59 a8 02 bc 54 42 1e be b5 ea 4f bb 5a b0 53 f4 26 02 6f 25 15 19 2b ee 27 …` (full 256 B in `bavacai_rsa_pubkey.der`).

Appendix B — Wallpaper rendering (build-disabled)

Although `backgroundImage=false` in this build, the code path at `0x140039C50` (`Util_SetWallpaper`) and `0x140039530` (rendering helper) is fully present and would activate on a build that flips the flag. The output is a fixed **1280×1024 BMP**:

Property	Value
Background	Solid black (`CreateSolidBrush(0)` + `FillRect`)
Font	Tahoma (`L"Tahoma"`), height 50, weight 700 (bold), `OUT_TT_PRECIS=4`
Text colour	White (`SetTextColor(0xFFFFFF)`)
Text background	Transparent (`SetBkMode(TRANSPARENT=1)`)
Text source	`requirementsFileDataUTF8` from JSON config, split by `'\n'` into lines
Layout	Each line centered horizontally `(1280 − text_width) / 2`, vertical block centered with a 60-pixel line height
Output format	BMP (BITMAPFILEHEADER 14 B + BITMAPINFOHEADER 40 B + 32-bit RGB raster from `GetBitmapBits`)
Application	`SystemParametersInfoW(SPI_SETDESKWALLPAPER=0x14, 0, <bmp_path>, SPIF_UPDATEINIFILE | SPIF_SENDWININICHANGE=3)`

The output BMP path is a global initialised in `App_Run` (`pvParam` / `xmmword_1400B0DE0`).