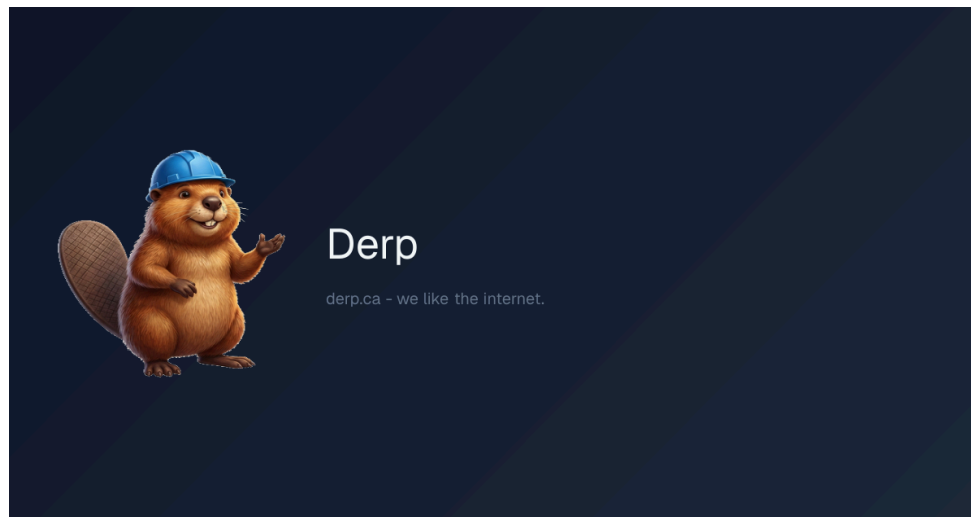


Qilin: inside the Windows ransomware build behind 1,000+ victims

 derp.ca/research/qilin-ransomware-teardown

Kirk

April 9, 2026



malware ransomware qilin reverse-engineering byovd encryption

► On this page

Qilin (aka Agenda, AgendaCrypt) is the ransomware group that took down NHS blood testing at Synnovis in June 2024, cancelled roughly 1,100 surgeries and 2,000 outpatient appointments, and demanded a \$50 million ransom for 400 GB of data tied to 300 million patient interactions (BBC, June 2024). The same group later claimed Japan's Asahi Group Holdings, Nissan Creative Box, Lee Enterprises, Malaysia Airports (with a \$10 million demand and 2 TB of data), and over a thousand other organisations across more than 60 countries. In 2025 Qilin ran 1,022 tracked attacks, the highest count of any ransomware operator that year (Ransomfeed, 2025). In April 2026 Qilin was still the market leader, accounting for roughly 14% of all tracked ransomware activity in a 30-day window (Check Point Research, April 2026).

Qilin is a Russian-speaking ransomware-as-a-service (RaaS) operation first observed in July 2022 under the "Agenda" name and rebranded as Qilin in September 2022. It runs a double-extortion model with a Tor negotiation portal and leak blog, pays affiliates 80-85% of ransoms collected, and excludes CIS (Commonwealth of Independent States) countries from targeting. After RansomHub went dark on April 1 2025, many of its affiliates moved to Qilin and disclosures on Qilin's data leak site roughly doubled between February 2025 and mid-2025. Microsoft Threat Intelligence confirmed in July 2024 that Octo Tempest (aka Scattered Spider) is a Qilin affiliate, alongside a roster that also reportedly includes groups like Devman and Arkana.

Cisco Talos [published the Qilin EDR-killer infection chain](#) in April 2026, showing a legitimate PDF reader sideloads a malicious `msimg32.dll` which eventually unloads EDR kernel callbacks via two signed vulnerable drivers. The Talos writeup had the loader and the in-memory EDR killer PE. What it did not have, because the samples were never publicly shared, is the Rust ransomware binary the whole kill chain exists to protect. We recovered that binary from Triage on 2026-03-27 and broke it apart end to end over the following two weeks. This post is that teardown.

A few things worth knowing about the build:

- Rust, x86, statically linked, 4.9 MB. The two Triage samples (260326 and 260327) are the same build.
- The file cipher is either AES-256 or ChaCha20, chosen per-file by a flag in the encryption-options struct. Inside each cipher path the binary dispatches a second time on CPU features: AES has both an AES-NI hardware path (AES-256, 14 rounds) and a software bitsliced `fixslice32` fallback; ChaCha20 has both an AVX2 path and a baseline path. Two lazy-init bytes in `.data` cache the CPU-feature decisions.
- Partial encryption is tuned by file size. Files under ~100 MiB are processed without chunking. Files in 100 MiB - 1 GiB use 10 chunks at 10% each. 1 - 1.77 GiB use 20 chunks at 5% each. Anything larger uses 50 chunks at 1% each. As file size grows the binary keeps total bytes encrypted roughly constant rather than scaling linearly with the file, the standard intermittent-encryption pattern.
- The file footer is 550 bytes. Layout is a 512-byte RSA-OAEP-wrapped symmetric key, an 8-byte length slot, and a 30-byte `-----END CIPHERTEXT BLOCK-----` ASCII marker.
- The EDR-killing behaviour is split across three drivers. Two signed BYOVD (Bring Your Own Vulnerable Driver) drivers provide hardware-level kernel access (RWEverything's `RwDrv.sys` and Quanta's `ControlCenter.sys`, both shipped in toolkit ZIPs from October 2025 onward). A third 9 KB custom driver, `hlpdrv.sys`, does the actual process termination with three IOCTLs.

- **CheckQilin.exe** contains a SHA-256 password comparison that is bypassed in-binary by a one-byte unconditional jump. Any password is accepted at runtime. Verified empirically with internal tooling.
- The clearweb leak domain in the ransom note, **wikileaks2.site**, is a fake WikiLeaks clone hosted behind Cloudflare. It went into registry **clientHold** on 2026-03-02, fifteen days before this build was compiled.

If you operate a threat intelligence platform with API access and can provide a researcher account, please reach out to . Additional data sources directly increase the quality and coverage of the threat intel published here.

Samples

Sample	SHA256	Type	Size	Compiled
CheckQilin.exe	4eac69e5cb9e4101d1eb76ec1e8cea128d3e10d577e28d7e954f72ba477214b8	PE32 x86 (Rust)	4.9 MB	2026-03-17
qilin_exe_260326	0894c497a0e03c08e9d5a2fa39e1dc8de543fae61061f7976e5c0265dd7df0dd	PE32 x86 (Rust)	4.9 MB	2026-03-17
Qilin DLL	dcce9e235d247c1c368a85edc1480d8a5d8ba07413a222161bfc52632d6f0762	DLL x86 (Rust)	4.9 MB	2024-11-22
nwdrv.sys (RWEverything)	16f83f056177c4ec24c7e99d01ca9d9d6713bd0497eedb777a3ffefa99c97f0	Kernel driver (signed)	50,216 B	2020-10-06
nwdrv.sys (Quanta)	0b12eb25db68d8714ba52583597ed20e5fab2f6e82dcd0bcb23161acb4a9a126	Kernel driver (signed)	11,816 B	2017-01-17
hlpdrv.sys	bd1f381e5a3db22e88776b7873d4d2835e9a1ec620571d2b1da0c58f81c84a56	Kernel driver (custom)	9 KB	2025-03-03

The two EXE samples (260326 and 260327) share everything that matters:

- Identical embedded JSON config
- Identical per-build RSA-4096 public key (modulus SHA-256 **f150d19c57a910d7...**)
- Identical **QEz6CwqBK1** company ID
- Identical vCenter PowerShell at the same **.rdata** offset
- Identical Sysinternals PsExec pair at the same **.rdata** offsets
- Identical compile timestamp

The only differences are the PE checksum field and the SHA-256 of the whole file.

The DLL (**dcce9e**) is a different campaign with a different RSA key (modulus SHA-256 **e68d17f78f19af07...**), different company ID (**0c-IyC4m1G**), and different embedded leak URL (**https://31.41.244.100** instead of **https://wikileaks2.site/**). Same Rust codebase, different deployment.

Two samples referenced in Cisco Talos's April 2026 writeup are not available to us: the **msimg32.dll** sideloading loader (SHA-256 **7787da25...**) and the EDR killer PE that executes in memory (**12fcde06...**). Neither is on MalwareBazaar or Triage. Everything in the rest of this post is derived from bytes on disk we could verify ourselves.

EDR kill chain: three drivers, one goal

None of the three drivers is novel individually. What matters is how they fit together.

rwdrv.sys variant 1: RWEverything RwDrv.sys

A 50,216-byte driver with PDB path `Driver.pdb`, compiled 2020-10-06, signed. Version resource identifies it as RWEverything "Low-Level Driver" v3.0.0.0. This is a well-known LOLDriver (living off the land driver) catalogued on loldrivers.io. It exposes direct physical memory reads and writes, I/O port access, MSR (model-specific register) access, PCI configuration space access, CPUID calls, and performance counter reads. No process manipulation, no filesystem I/O, no registry access, no driver loading. It is a pure hardware-access primitive.

Qilin uses it purely to stage physical-memory manipulations against the kernel.

rwdrv.sys variant 2: Quanta ControlCenter.sys

An 11,816-byte driver with PDB path `f:\_prog_prj\1610_n15_led\js970_driver\jspostestercmd\sys\output\amd64\ControlCenter.pdb`, compiled 2017-01-17, signed by Quanta (the Taiwanese OEM). Version resource identifies it as the Quanta Control Center Driver. It is also a LOLDriver, though a much smaller one, and also catalogued on loldrivers.io.

This variant has 17 IOCTLS fully decoded: five I/O port access (read DWORD/WORD, write WORD/BYTE, HLT), two MSR access (RDMSR/WRMSR), three physical memory read/write (DWORD, QWORD, via `MmMapIoSpace`), one performance counter (RDPMC), six PCI configuration read/write (via `HalGetBusDataByOffset` and `HalSetBusDataByOffset`), and one state flag.

The Quanta driver entered the Qilin toolkit in October 2025. Across six submission dates between August 2025 and December 2025, every `.sys` file in a toolkit ZIP matches one of three known hashes (the RWEverything driver, the Quanta driver, or `hlpdrv.sys`). No new driver variants appeared in six months. One of the October submissions (251007) is a password-encrypted ZIP shipping the RWEverything variant. Qilin ships both drivers side by side once the Quanta option is available.

Neither of these drivers has any Qilin-specific IOCTLS. They are stock signed vulnerable drivers being reused for their hardware-access capabilities. The thing Qilin actually writes itself is the third driver.

hlpdrv.sys: the custom EDR killer

A 9 KB custom driver with no PDB, compiled 2025-03-03, not signed. This is the one where the actual process killing happens. It exposes three IOCTLS:

IOCTL	Function	Purpose
0x0222000	GetDriverState	Returns a driver-state word (used as a handshake)
0x0222004	UntrustFile	Strips the DACL on a target file by writing a NULL DACL
0x0222008	TerminateProcessByPID	Opens a process by PID and calls <code>ZwTerminateProcess</code> on it

The EDR kill flow is: user-mode code sends a PID to `0x0222008`, the driver calls `PsLookupProcessByProcessId` to resolve it to an `EPROCESS`, walks the image path with `GetProcessImagePath`, calls `UntrustFile` on the image to strip the file ACL, then calls `ZwTerminateProcess`. All 11 functions in the driver are decompiled and all 21 `DbgPrint` strings are extracted. This driver is custom-built for the Qilin campaign and is the only one in the chain that touches processes. The two drivers are loaded in tandem via the `msimg32.dll` sideloader documented by Cisco Talos.

Encryption scheme

The cipher choice is layered. A top-level per-file flag in the encryption-options struct picks AES-256 or ChaCha20. Inside each branch, a separate CPU-feature dispatcher picks a hardware-accelerated implementation or a portable fallback.

Top-level cipher selection (per-file flag)

The per-file finalizer `fcn.00433e70` takes the cipher selector as its fifth argument. The branch at the top, in r2ghidra's decompile:

```

if (arg_18h == '\0') {
    // ChaCha20 setup
} else {
    // AES setup
}

```

Trace the argument back up the call chain and you get `fcn.0045e820` -> `fcn.00433b60` -> `fcn.00433e70`. `fcn.00433b60` calls the finalizer with `arg_14h & 0xff` as the selector. `fcn.0045e820` calls `fcn.00433b60` with a one-byte local (`uStack_2c`) that originates from the encryption-options struct passed in by `fcn.0043bf40`. There is no CPUID check at this level. The top-level cipher choice is whatever the upstream orchestrator wrote into the struct field.

AES-NI vs bitsliced AES (CPU-feature sub-dispatcher)

Inside the AES branch, the binary uses a one-byte cache at `.data 0x6c3008` (loaded via constant pointer at `.rdata 0x856ee8`). The sentinel value `0xff` means "not yet initialised". The first AES call into `fcn.00433e70` reads the byte and hits an inline CPUID block at `0x4345e3`:

```

push 1
call fcn.004044c0      ; CPUID.1, output -> [esp+0x440]
push 0
push 7
call fcn.004044f0      ; CPUID.7, output -> [esp+0x1454]
mov  ebx, [esp+0x448]  ; CPUID.1.ECX
not  eax
test eax, 0xc0000000   ; XSAVE+OSXSAVE in ECX[26:27]
jne  no_aesni
call fcn.004303a0      ; xgetbv -> XCR0
and  eax, 2            ; XCR0[1] (XMM)
shr  ebx, 0x19         ; CPUID.1.ECX[25] (AES-NI) -> bit 0
shr  eax, 1
and  ebx, eax
mov  byte [esi], bl    ; cache the AES-NI flag at 0x6c3008

```

If the result is 1, the AES path calls `fcn.00404520` to run an AES-256 key schedule built around eight `aeskeygenassist` thunks at `fcn.00402130` through `fcn.004021a0` (one per round constant). The block encrypt is `fcn.004306f0`. It loads a 64-bit counter from `[edx+0x10] / [edx+0x14]`, increments it with carry propagation, builds the counter block in `xmm0`, XORs in the first round key, runs thirteen `aesenc` and one `aesenclast`, then writes the AES-256 keystream block to `[eax]`. Fourteen rounds is AES-256, and the counter increment plus XOR-out structure is textbook CTR mode.

If the result is 0, the AES path calls `fcn.004021b0` for the software key expansion and `fcn.00402e50` for the round function. `fcn.00402e50` is the `aes-0.8.3` crate's `fixslice32` bitsliced software AES, identifiable by the canonical delta-swap pattern with masks `0x55555555` and `0x33333333` in its first basic block. The bitsliced function is called three times from `fcn.00433e70` at offsets `+0x240a`, `+0x24fc`, and `+0x261c`.

Byte searches for `66 0f 38 dc` (`aesenc`), `66 0f 38 dd` (`aesenclast`), and `66 0f 3a df` (`aeskeygenassist`) all hit inside `.text` code reachable from the per-file finalizer. There is also a separate startup logger `fcn.00469470` that prints `[INFO] AESNI support detected! Using AES-CTR` (or the WARNING variant) on first run. Its return value is stored at `[esp+0x49d]` in `fcn.0044b7b0` and never read again. The logger is informational only and does not feed any dispatcher, but the `Using AES-CTR` substring is the cleanest empirical confirmation of the AES mode the binary itself provides.

AVX2 vs baseline ChaCha20 (CPU-feature sub-dispatcher)

The ChaCha20 branch has its own equivalent. The cache byte is at `.data 0x6c300c` (constant pointer at `.rdata 0x856ef8`). Lazy init at `0x43458b` calls a different pair of CPUID wrappers (`fcn.0041cc90` for leaf 1, `fcn.0041ccc0` for leaf 7), checks the same XSAVE+OSXSAVE bits, queries `xgetbv`, and writes `(CPUID.1.ECX[28] AVX) AND (CPUID.7.EBX[5] AVX2) AND (XCR0[1:2] XMM+YMM)` to the byte. If the byte is set, the ChaCha20 implementation uses AVX2; if clear, it falls back to the baseline path.

ChaCha20 state setup begins at `0x4346d1`. The first instruction is `mov dword [arg_9ch], 0x6b206574`. The immediate `0x6b206574` is 'te k', the last 4 bytes of "expand 32-byte k". The runtime setup writes the ChaCha20 init constants as inlined u32 immediates into the state buffer rather than loading them from a 16-byte memory constant. The literal "expand 32-byte k" string does also exist in `.rdata` at four addresses.

Partial encryption tuned by file size

Before the per-file finalizer is even called, `fcn.0045e820` picks the chunked-encryption parameters from the file size and feeds them into `fcn.00437180`. The constants live at `.rdata 0x6d3490`, `0x6d34a0`, and `0x6d34b0` as 4-tuples (`count`, `_`, `ratio`, `_`):

File size threshold	Chunk count	Per-chunk ratio	Source constant
0 bytes (empty)	0	0	inline zeros
< ~100 MiB (< 0x6400001)	0	0	inline zeros
< ~1 GiB (< 0x3e800001)	10	10	0x6d34b0
< ~1.77 GiB (< 0x71000001)	20	5	0x6d34a0
>= ~1.77 GiB	50	1	0x6d3490

As file size grows, the binary uses more chunks at smaller per-chunk ratios. The total bytes encrypted stay roughly constant rather than scaling linearly with the file. This is the standard intermittent-encryption pattern modern RaaS families use to maximise files-per-second on large victim shares. LockBit and BlackCat published the same pattern years earlier.

Per-file flow

The per-file encryption flow is:

1. Generate a symmetric key (per-file random)
2. Encrypt the file body in place with AES-CTR or ChaCha20 keystream XOR, up to 512-byte chunks
3. RSA-OAEP-wrap the symmetric key with the embedded campaign RSA-4096 public key
4. Append a 550-byte footer

The 550-byte footer

The 512-byte RSA-OAEP-wrapped symmetric key sits at the front, followed by an 8-byte length slot and a 30-byte ASCII marker. We recovered the layout from the offset arithmetic in `fcn.00433e70`'s write paths.

The function is a state-machine dispatcher with multiple write call sites scattered across its body. The two relevant ones for the footer layout are:

- An 8-byte length-slot write at `0x00435457`. Just before the call, the code does `add ebx, 0x200`, so it is writing 8 bytes at `base + 0x200` (offset 512 from the footer base).
- A 30-byte marker write at `0x00435512`. Just before the call, the code does `add eax, 0x208` and pushes the `.rdata` pointer to `-----END CIPHERTEXT BLOCK-----` at `0x6d2980`. It is writing 30 bytes at `base + 0x208` (offset 520).

The body encryption loop lives at `0x00436656`. The chunk size is capped at 512 bytes per write via `mov eax, 0x200; cmp ebx, 0x200; cmovae ebx, eax`, the buffer at `[esp+0x1450]` is the in-place AES-CTR / ChaCha20 keystream-XOR output, and the loop iterates over the file body. This is the body cipher write, not the footer.

The footer base must therefore have a 512-byte block at `base + 0x000`, an 8-byte slot at `base + 0x200`, and a 30-byte marker at `base + 0x208`, totalling 550 bytes. The 512-byte block is the RSA-4096 OAEP-wrapped symmetric key. We did not isolate the specific call site that writes that 512 bytes inside `fcn.00433e70`; the function dispatches through a jump table and contains multiple `fcn.005770e0` and `fcn.004370d0` write calls beyond the three above. What we can state from offset evidence:

Offset from EOF	Size	Content
-550	512 bytes	RSA-4096 OAEP-wrapped symmetric key
-38	8 bytes	Length or magic slot
-30	30 bytes	-----END CIPHERTEXT BLOCK----- ASCII marker

There is no start marker. A decryptor scans backward from end-of-file looking for the ASCII marker, walks back 8 bytes to find the length prefix, then reads the 512 bytes before that as the RSA-OAEP ciphertext. The OAEP random padding uses `BCryptGenRandom(NULL, buf, 0x20, 2)` with `SystemFunction036` (`RtlGenRandom`) as a fallback, routed through `fc.0049de60` and `fc.0049d220` padding helpers.

Embedded artifacts in .rdata

CheckQilin.exe is a Rust binary that ships several large embedded blobs in its `.rdata` section via the Rust `include_bytes!` pattern: a vCenter PowerShell spreader, the Sysinternals PsExec pair, two wallpaper JPEGs, and the per-build RSA-4096 campaign public key. None of them live in PE resources. All have been carved out.

vCenter PowerShell spreader

A ~20,800-byte PowerShell script is embedded in `.rdata` in all three samples. The simplest anchor is the `write-Host "[DEBUG|POWERSHELL] Script execution started."` line, which sits at file offset `0x0044a893` in both EXE samples; the closing `write-Host "[DEBUG|POWERSHELL] Script execution completed."` marker is roughly 20,760 bytes later. The script blob itself starts a few hundred bytes before the start marker; we did not pin the exact first byte. SHA-256: `3a24cd31c8287f7ee7336936a95f82b5d71a3746d210b4240869f3e3f5b34208`.

The script targets VMware vCenter. It installs PowerCLI, connects to vCenter, enumerates ESXi hosts, optionally reboots into single-user mode, copies the Linux Qilin encryptor via datastore upload, and launches it on each host. The only embedded URL is the legitimate Microsoft .NET 4.7.2 installer, pulled as a PowerCLI dependency.

All three samples ship the bit-identical script. The two EXE samples embed it at the same file offset; the DLL has a different `.rdata` layout and embeds it at a different offset.

Sysinternals PsExec

Two PE files carve out of `.rdata`: `PsExec.exe` (716,176 bytes) at file offset `0x0038e24c` and `psxessvc.exe` (193,984 bytes) at `0x004082cc`. Both start with the canonical `MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\xff\xff` DOS header followed by the `This program cannot be run in DOS mode` Sysinternals stub. Both are unmodified Sysinternals PsExec v2.43, compiled 2023-04-06, signed by Microsoft. The PDB paths are `D:\a\1\s\psexec\...` (GitHub Actions layout), meaning they came directly from Microsoft's official PsExec build pipeline.

This matters for detection: an EDR that allowlists signed Sysinternals utilities will not block a dropped PsExec. Detection logic has to target the wrapper that drops it, not the embedded binary.

Wallpaper

Two JPEG images carve out at file offsets `0x002d60e4` (1920x1080, 388,563 bytes) and `0x002d626e` (160x90, 4,617 bytes). The resource string in the binary refers to them as `wallpaper_3840x2160_simple`, but the rendered dimensions are 1920x1080 (the EXIF metadata keeps the 3840x2160 TIFF header, suggesting the image was downsized from a 4K source during development). The wallpaper module at `shared/windows/src/wallpaper/mod.rs` writes the JPEG to a temp file and calls `SystemParametersInfof(SPI_SETDESKWALLPAPER)`.

Per-build campaign RSA-4096 pubkey

The RSA-4096 public key is a Rust string literal in `.rdata` with `\n` escapes. The `-----BEGIN PUBLIC KEY-----` marker sits at file offset `0x002d344a` in both EXE samples; the surrounding JSON config blob `{ "public_rsa_pem": ... }` starts at `0x002d3438`. Decoded modulus SHA-256:

Sample	Modulus SHA-256
CheckQilin.exe / 260327	<code>f150d19c57a910d714ef773a470bbb8ad88185f4b4713852fce706a1e7482b59</code>
qilin_exe_260326	<code>f150d19c57a910d714ef773a470bbb8ad88185f4b4713852fce706a1e7482b59</code>
Qilin DLL (dcce9e)	<code>e68d17f78f19af07405d78c8e4f48c93d2a278b036392437b05d9bdcef359085</code>

The matching modulus across EXE_260327 and EXE_260326 is the proof that they are the same victim build. Triage's Maco extractor confirmed the EXE key independently from a process memory dump of PID 2580, which is the only place the key was re-observed at runtime.

The eb 26 bypass

The password-validation path lives inside the 15,458-byte main function `fcn.0044b7b0` at file offset range `0x44e1c9` to `0x44e21f`. The full pipeline is short:

1. Log "Checking password validity"
2. `slice::to_vec` on the runtime password buffer
3. Call `fcn.00437610` (SHA-256) to produce a 32-byte digest
4. `memcmp` the digest against the config's `password_hash` field
5. `sete b1` to capture the comparison result
6. `eb 26` unconditional jump to `0x44e21f`
7. Log "Password is correct" and continue

The bytes at offset `0x44e1f7` are `eb 26`, not `74 26` (`je`) or `75 26` (`jne`). That is a one-byte difference from a conditional branch. The `sete b1` at `0x44e1f4` dutifully writes the `memcmp` result into `b1`, and a `grep` of the entire function's disassembly for `test b1, b1` (84 `db`) returns two hits: `0x44e217` and `0x44e30e`. Both of them sit inside length-mismatch failure cleanup, where `ebx` has just been zeroed with `xor ebx, ebx` immediately above. Neither reads the real comparison result.

Which means the "[FATAL] Password is not correct." print at `0x44e316` is unreachable in practice. It sits on the length-mismatch failure path and that path can only fire if the two hashes are different sizes. Both hashes are SHA-256 outputs. Both are 32 bytes. The length check at `0x44e1de` always succeeds, so the FATAL branch is never entered.

We verified the behaviour dynamically with internal tooling. Running with `--password a`, the SHA-256 of "a" is `ca9781122a5828....`. The config's `password_hash` is `81b62806....`. `memcmp` returns 1 (non-equal). The very next log line in the trace is `[INFO] Password is correct..`. The trace captured the full register state before and after the jump site: `b1 = 0` (mismatch captured), `eb 26` taken, execution lands at `0x44e21f`, and the success message is printed.

We hashed every plausible candidate for the config's `81b62806....`: "a", "QEz6CWqBK1", "0c-IyC4m1G", "ejy1aq", "*ejy1aq", "kCfxtUsh", "qilin", the `--password a --no-admin` autostart string, the 16-byte runtime-derived canary we recovered from memory dumps, and more. None of them match. The target hash has no known preimage we can derive from on-disk artifacts.

The bypass has a few plausible origins (an intentional RaaS backdoor patched into compiled binaries, a build-time `#[cfg(...)]` toggle that omits the conditional, a compiler dead-code-elimination artifact that left the SHA-256 call but cut the branch) and we do not have enough evidence to pick one. What we can state as fact: the check is present in the code, the check is skipped unconditionally at runtime, any password is accepted, and the `--password` flag itself is required only because an empty string fails an earlier length check.

The DLL (`dcce9e`) takes a different approach. It does not even bother with the comparison. Its password validation only checks `len(password) > 0`. The DLL has exactly one SHA-256 call in its `.text` section, not near any password strings, and the password hash in the DLL's config (`bf1d5469....`) is a dead string with zero code xrefs. The two builds handle the same config field in two different ways: the EXE includes the check but patches around it, and the DLL does not include the check at all.

Sandbox behavioural summary

Triage sandbox run 260327 (44.2 MB behavioural archive, Windows 10 build) captured:

- Process tree: legitimate PDF reader sideloads `msimg32.dll`, spawns `CheckQilin.exe` with `--password a --no-admin`, creates the `*ejy1aq` mutex, acquires `SeDebugPrivilege` / `SeImpersonatePrivilege` / `SeIncreaseBasePriorityPrivilege`, re-enables and then disables the VSS service around shadow copy deletion
- Configuration pulled at runtime matches the static config (same mutex `*ejy1aq`, same autostart Run key value `CheckQilin.exe --password a --no-admin`, same file extension `.QEz6CWqBK1`)
- Encryption working extension marker: `.kCfxtUsh` (a per-run marker used during the rename-encrypt-rename cycle)
- 75 files dropped: 50 legitimate Microsoft DISM components, 6 PowerShell startup artifacts, 1 DISM session log, and 18 Qilin artifacts (17 progressive QLOG snapshots plus a ransom note). No encrypted victim files in the dropped set (Triage's MFT-based extraction does not follow renames)
- 44.2 MB / 50,296 frame PCAP with 11 TLS CLIENT_RANDOM keys. All traffic is Microsoft telemetry, Defender definition updates, Akamai-for-Microsoft, Google Fonts, and DigiCert OCSP. Zero Qilin markers in any of the 73 decrypted HTTP objects
- Event logs cleared by Qilin during execution. `Application.evtx` has 31 records with only one inside the Qilin run window. `Security.evtx` is a corrupted 332 KB file where `python-evtx` parses zero records cleanly; raw record-header carving recovers 238 fragments (167 inside the run window), but all bodies are BinXML template substitutions referencing per-chunk string tables that no longer exist. The audit content is gone

- 50 process memory dumps across 8 image dumps (the loaded PE of CheckQilin) and 42 PowerShell .NET CLR heaps. None of the dumps captured the encryption-thread heap, so runtime AES keys, ChaCha20 keys, and per-file nonces are not recoverable from this run

The binary makes no network calls of its own. Static analysis finds no `.onion` strings, no `wikileaks2.site` strings, and no HTTP client initialisation in `.text` or `.rdata`. Any leak-site interaction is operator-driven through an out-of-band Tor browser workflow.

wikileaks2.site: infrastructure enrichment

The clearweb leak URL embedded in the CheckQilin ransom note points to `https://wikileaks2.site/`. The short version: it is a fake WikiLeaks clone that ran behind Cloudflare for roughly a year. The registration was not renewed at the end of its first term and the domain entered the registrar's auto-renew grace period on 2026-03-02. There is no sign in WHOIS, DNS, or CT data of a takedown or sinkhole; the parking nameservers and `expired@globaldomaingroup.com` placeholder contact are the registrar's standard handling of an unrenewed domain.

WHOIS

Registered 2025-03-02 via Global Domain Group LLC (reseller LinkHost, LLC). Status 2026-04-09: `clientHold` plus `clientTransferProhibited` plus `autoRenewPeriod`. The grace period runs to 2027-03-02. Parked nameservers: `ns1/ns2.globaldomaingroup.com`.

DNS history (SecurityTrails)

Three distinct A-record ranges across the lifetime of the domain. All IPs are Cloudflare AS13335 anycast. The origin IP was never exposed in DNS.

Range	IPs
2025-03-03 to 2025-05-02	104.21.2.4, 172.67.128.115
2025-05-03 to 2025-09-18	104.21.{16,32,48,64,80,96,112}.1
2025-09-18 to 2026-04-09	104.21.2.4, 172.67.128.115

The Cloudflare nameserver pair changed five times in 13 months:

Range	NS pair
2025-03-03 to 2025-03-18	arturo.ns + marlowe.ns
2025-03-18 to 2025-04-20	kallie.ns + plato.ns
2025-04-21 to 2025-05-02	elisabeth.ns + sonny.ns
2025-05-03 to 2025-10-02	frida.ns + morgan.ns
2025-10-03 to 2026-04-09	elisabeth.ns + sonny.ns (reused)

Cloudflare assigns a deterministic NS pair per zone. A different pair means the site was removed and re-added. Five re-adds in 13 months is high churn for a single domain.

Certificate Transparency

28 certificates in `crt.sh` across four issuers: Sectigo (GB) for the first two on registration day, then Google Trust Services WE1, Let's Encrypt E7, and Cloudflare TLS Issuer rotating across 90-day cycles. First cert on 2025-03-02 (the day of registration). The last cert was issued 2025-12-06, with a not-after of 2026-03-06. Cert renewal stopped roughly three months before the domain expired.

48 public captures between 2025-05-19 and 2025-12-15. Nothing afterwards. The page title is **WikiLeaks V2 - New Version of the Legendary WikiLeaks Project**, rendered by a Nuxt.js (Vue 3) single-page app. Loaded assets include the `_nuxt/entry.*` bundle pattern, Google Fonts, and a Yandex Metrika tag at `mc.yandex.ru/metrika/tag.js` with counter ID `97200228`. An urlscan search for other sites referencing the same counter returned zero hits. The verdict on the latest scan is **malicious: false** from both the urlscan ML model and community votes: it looked like a news-style leak site, not a phishing page, and Cloudflare was fronting valid TLS certs.

The most recent screenshot (2025-12-15) shows a WikiLeaksV2-branded leak portal with a top nav reading **LEAKS NEWS / INVESTIGATIONS / LIST OF LEAKS / DONATE / SUBMIT** and two named victims visible on the front page: a featured **ANCO** entry with a red LEAKED stamp dated 2025-12-15, and a **Cinvestav** hot-leak banner at the top. We make no attribution claim about these names. These are the strings the operator chose to display.

Internet Archive

Zero captures. The Wayback Machine never snapshotted the domain.

Summary of the timing anomaly

The build was compiled fifteen days after the leak URL went into **clientHold**. Two plausible explanations: the builder template was stale, or the affiliate who produced this build had moved on to other leak infrastructure and did not update the URL. Either way, the dead URL did not break execution because the binary makes no network calls of its own.

IOC summary

Hashes

File	SHA256
CheckQilin.exe	4eac69e5cb9e4101d1eb76ec1e8cea128d3e10d577e28d7e954f72ba477214b8
qilin_exe_260326	0894c497a0e03c08e9d5a2fa39e1dc8de543fae61061f7976e5c0265dd7df0dd
Qilin DLL	dcce9e235d247c1c368a85edc1480d8a5d8ba07413a222161bfc52632d6f0762
rwdrv.sys (RWEverything)	16f83f056177c4ec24c7e99d01ca9d9d6713bd0497eedb777a3ffefa99c97f0
rwdrv.sys (Quanta)	0b12eb25db68d8714ba52583597ed20e5fab2f6e82dcd0bcb23161acb4a9a126
hlpdrv.sys	bd1f381e5a3db22e88776b7873d4d2835e9a1ec620571d2b1da0c58f81c84a56
Embedded vCenter PS	3a24cd31c8287f7ee7336936a95f82b5d71a3746d210b4240869f3e3f5b34208
Embedded PsExec.exe	078163d5c16f64caa5a14784323fd51451b8c831c73396b967b4e35e6879937b
Embedded psexesvc.exe	cc14df781475ef0f3f2c441d03a622ea67cd86967526f8758ead6f45174db78e

Network

Type	Value	Context
Clearweb	https://wikileaks2.site/	CheckQilin leak page (clientHold since 2026-03-02)
Clearweb	https://31.41.244.100	DLL campaign leak page (AS57678 Red Bytes LLC, Saint Petersburg RU)

Type	Value	Context
Onion	me7xeyqcybqsz2uag745dcxicvfyermmvelcb6cqwl4qtp3sailg2xid.onion	CheckQilin negotiation portal
Onion	fsthixb5w4uq3nge4xrgmvmxnufnd77vvic7mxvrgkjycrpyy6fw5qad.onion	DLL campaign negotiation portal
Onion	kbsqoivihgdmwczmxkbovk7ss2dcynitwhhfu5yw725dboqo5kthfaad.onion	Leak blog
Onion	ijzn3sicrcy7guixkzjkib4ukbiilwc3xhnmb4mcbccnsd7j2rekvqd.onion	Leak blog
Yandex Metrika	counter 97200228	Embedded on wikileaks2.site

Host

Indicator	Context
Mutex	*ejy1aq
Company ID	QEz6CWqBK1
Extension	.QEz6CWqBK1
Working extension	.kCfxtUsh
Ransom note	README-RECOVER-QEz6CWqBK1.txt
Run key name	*ejy1aq
Run key value	CheckQilin.exe --password a --no-admin
Config password hash	81b62806... (decorative, bypassed)
Negotiation login	hPijim5-P-CDtKvs0MZbrQAn1WHpNpEd
Negotiation password	a
Footer marker	-----END CIPHERTEXT BLOCK----- (last 30 bytes of an encrypted file)

Behavioural

Indicator	Context
RWEverything RwDrv.sys + Quanta ControlCenter.sys loaded alongside hlpdrv.sys	BYOVD stack
DeviceIoControl to hlpdrv.sys IOCTL 0x0222008 with a PID	EDR process termination
vssadmin.exe shell (cmd.exe parent) to delete shadows	Shadow copy deletion
HKLM Run key value *ejy1aq	Autostart persistence

Indicator	Context
Security.evtx cleared via PowerShell <code>Clear-EventLog</code> during run	Event log wipe
Per-file rename <code>file.ext</code> to <code>file.ext.QEz6CwqBK1.kCfxtUsh</code> to <code>file.ext.QEz6CwqBK1</code>	Working-extension encrypt-in-place pattern
Trailing 550-byte footer with ASCII marker <code>-----END CIPHERTEXT BLOCK-----</code>	Encrypted file layout

Operational impact

The password-validation bypass means any operator-supplied argument unlocks the build, so detection has to target the `msimg32.dll` sideload chain and the `hlpdrv.sys` IOCTLs rather than the password wrapper.

See also: [InterLock tooling teardown](#), [Pay2Key encryptor analysis](#), [Payload ransomware \(Babuk derivative\)](#).

Kirk

I like the internet. Want to get in touch? ·