Zawoo

Technical Analysis — Linux

REVERSE-ENGINEERED REPORT

RansomLook · ransomlook.io
File last modified: 2026-08-29

Sample SHA-256: `33d3afddaa5710cdcc4e93a7bae8be010c19747fb53d85fcd54ef32056a1eb0b`

ZAWOOO Ransomware — Full Analysis

1. Sample Identification

Field	Value
Family	ZAWOOO (self-designated in the ransom note)
SHA-256	`33d3afddaa5710cdcc4e93a7bae8be010c19747fb53d85fcd54ef32056a1eb0b`
SHA-1	`9d371d77700dca0950249b8e8a1da128dbbdc719`
MD5	`d8957e860cb421e9c06e0fd05010cd4b`
Type	ELF 64-bit LSB PIE, x86-64, **static-pie**, **stripped**
Size	478 128 bytes (467 KB)
Language	C (mbedTLS + static libc statically linked)
Toolchain	`GCC: (GNU) 11.2.1 20211120` (`.comment`)
Image base	`0x0` (PIE)
Entry point	`0x7CBD`
Sections	25 — `.text` `0x7A10–0x4C635` , `.rodata` `0x4C640–0x60760` , `.data.rel.ro` `0x26B9E0–0x272E30` , `.data` `0x273020–0x274488` , `.bss` `0x2744A0–0x287520`
Functions	960 total — **47 malware-specific**, the rest is mbedTLS + libc
Dynamic symbols	1 (empty) — no imported API at all

ZAWOOO is a **Linux NAS encryptor built specifically for Synology DiskStation**. It is not a generic Linux locker: the directory-exclusion list is made exclusively of Synology internal share/metadata directories (`@SynoFinder-log` , `@eaDir` , `@synoconfd` , …), the scan root defaults to the glob `/volume*` , and the whole "fast mode" is built around parsing Synology's own filesystem index under `/etc/f-index` .

There is **no packing, no obfuscation, no anti-debug, no anti-VM and no string encryption**. On the contrary the binary carries a complete `printf` -style logging subsystem with source file names (`src/enc/thread.c` , `src/enc/work.c`), line numbers and human-readable messages — this is a debug/verbose build, and it is what makes the analysis unusually precise. The only meaningful analysis friction is that the binary is statically linked and stripped (T1027.008), so there is no import table to anchor on.

The code quality is that of a competent but not elite developer: correct pthread producer/consumer plumbing, correct XTS usage, a genuine anti-double-encryption design — but also typos baked into paths and messages (`slient` for silent, `alredy` , `invlid` , `[-]rsa nncryption failed`) and two mutually inconsistent copies of the exclusion list left in the binary.

Imports

None. The binary is `static-pie` with a single empty `.dynsym` entry and no `DT_NEEDED` . All libc and crypto code is linked in. Syscalls are issued directly through the embedded libc stubs.

2. Infrastructure

Field	Value
Onion (DLS)	`fyenuhkq3pfhnbpidj5jm2fl2lryxip4byhg6eozynrnlomu4szf2nyd.onion`
E-mail	`zawoooorecover@onionmail.org`
Session ID	`05b0794c896a54271d3d39c3b395d86aee2d5c05f196c59f3781bac1b1ac1a1a12`
Victim ID (hardcoded)	`<VICTIM_ID>` — 16 uppercase hex, compiled into the note template (redacted)
Note filename	`How To Restore Your Files.txt`
Encrypted filename	`<16 uppercase hex>.fHiQJervS` (original name destroyed)
Single-instance lock	`/var/run/enc-nas.lock` (`flock` `LOCK_EX|LOCK_NB`)
Payment	Bitcoin, negotiated over Session / e-mail. No wallet in the binary.

The victim ID is compiled into the note template rather than derived at runtime, which means **one build per victim**. Combined with the per-build encryption suffix (`fHiQJervS`) and the per-build RSA key, each intrusion gets its own binary.

---

3. Ransom Note

Filename

`How To Restore Your Files.txt` — pointer at `.data:0x273030` → `.rodata:0x4C882` .

Dropped by `zw_drop_ransom_note` @ `0xBA70` , called from `zw_encrypt_file_full` @ `0x9F37` **once per encrypted file**, into that file's own directory. The function builds `dirname(path) + note_name` , calls `access(path, F_OK)` and only writes if the note is not already present (`open(O_WRONLY|O_CREAT, 0777)` @ `0xBB07`), so each directory ends up with exactly one note. Silent mode (`-s`) drops **no** note at all.

Content (template at `.data:0x273080`, 3 618 bytes, NUL-terminated, SHA-256 `7cee3cec1812fb94de70e1a968e1508328030f20214ddf9671b2f82fa16de45e`)

```
Please contact with your ID : <VICTIM_ID>
~~~ You have been attacked by ZAWOOO - a ransomware that prioritizes reputation. ~~~~

>>>>> The file will have three different suffixes, don't worry, this is a normal phenomenon and
can be decrypted.

>>>>> Your data will be published on the Dark Web, download Onion Browser to visit.
    https://fyenuhkq3pfhnbpidj5jm2fl2lryxip4byhg6eozynrnlomu4szf2nyd.onion

>>>>> You must pay us.
    I know you may have other file backups, but there is no backup of the customer's privacy and
trust.
    We will disclose your data and send the download link to your customers and the GDPR (Data
Protection Authorities).
    Instead of paying huge data breach fines and losing customers, it's better to choose to
cooperate with us

>>>>> What is the guarantee that we won't scam you?
    We are not a politically motivated group and want nothing but financial rewards for our work.
    If we defraud even one client, other clients will not pay us.

>>>>>If you pay the ransom, we will fulfill all the terms we agreed during the negotiation
process.
     Otherwise, we may consider sending your files, chat history, mailbox content, etc. to all
your customers by email.

>>>>> You can think of this paid decryption as a security test.
      We will tell you the entire intrusion process, give you security advice, and help you
protect your system.
      This amount may be cheaper than finding a security company for testing.

>>>>> Warning! Do not delete or modify encrypted files, it will lead to irreversible problems
with decryption of files!

>>>>> Don't go to the police or the FBI for help and don't tell anyone that we attacked you.
      They will forbid you from paying the ransom and will not help you in any way, you will be
left with encrypted files and your business will die.

>>>>> When buying bitcoin, do not tell anyone the true purpose of the purchase.
      Some brokers, do not allow you to buy bitcoin to pay ransom. Communicate any other reason
for the purchase,
      such as: personal investment in cryptocurrency, bitcoin as a gift, paying to buy assets for
your business using bitcoin,
      cryptocurrency payment for consulting services, cryptocurrency payment for any other
services, cryptocurrency donations,
      buying bitcoin to participate in ICO and buy other cryptocurrencies, buying
cryptocurrencies to leave an inheritance for your children,
      or any other purpose for buying cryptocurrency. Also you can use adequate cryptocurrency
brokers who do not ask questions for what you buy cryptocurrency.

>>>>> After buying cryptocurrency from a broker, store the cryptocurrency on a cold wallet,
      such as https://electrum.org/ or any other cold cryptocurrency wallet,
      more details on https://bitcoin.org By paying the ransom from your personal cold
cryptocurrency wallet,
      you will avoid any problems from regulators, police and brokers.

>>>>> Don't be afraid of any legal consequences, you were very scared, that's why you followed
```

```
all our instructions,
      it's not your fault if you are very scared. Not a single company that paid us has had
issues.
      Any excuses are just for insurance company to not pay on their obligation.

>>>>> You can contact us in the following ways:
1. Download and install  the session: https://getsession.org/,
   our session ID: 05b0794c896a54271d3d39c3b395d86aee2d5c05f196c59f3781bac1b1ac1a1a12
2. Send email: zawooorecover@onionmail.org
In order to ensure that you can receive the message, please send the message in two ways at the
same time
```

Note vs. binary — one contradiction

The note announces "three different suffixes". **This build only ever appends one**: `.fHiQJervS` (`zw_encrypt_file_full` @ `0x9CC4` and `0x9E72`). The three-suffix pattern is visible only in the self-exclusion entries left over from sibling builds — `.MXvF0` , `.MXvF0_F` , `.c2w` in one list and `SpXv2` , `SpXv2_F` , `ctw` in the other — which follow a `<base>` / `<base>_F` / `c2w|ctw` scheme. Either the note template is shared across builds without being updated, or a companion binary (not this one) produces the `_F` and `c2w` variants.

---

4. Execution Flow (`main` @ `0x7A70`)

```
1.  getopt(argc, argv, "p:sa")                                      @ 0x7AB6
      -p <path>   target path      (mandatory)
      -s          "slient" mode    (stealth encrypt, no rename, no note)
      -a          auto mode        (drive from the /etc/f-index index)
2.  For -a only: open("/var/run/enc-nas.lock", O_CREAT|O_RDWR, 0644)   @ 0x7B2F
      flock(fd, LOCK_EX|LOCK_NB)                                        @ 0x7B47
      EWOULDBLOCK -> "alredy run(-x ), exit"                            @ 0x7C7C
3.  No -p            -> "target path is null", exit -1                @ 0x7C93
4.  zw_load_baselines()                                                @ 0x86C0
      T_slient = zw_load_T_slient()   from /etc/slient-time/baseline.stamp
      T_findex = zw_load_T_findex()   from /etc/f-index/baseline.stamp
5.  Dispatch:
      -s present    -> zw_run_mode_walk(path, 1)                       @ 0x7BCA
      -a and T_findex != 0
                    -> zw_run_mode_findex(path)                        @ 0x7BA4
      -a and T_findex == 0
                    -> "no f-index find, please use -p", exit -1       @ 0x7CAE
      default       -> zw_run_mode_walk(path, 0)                       @ 0x7BB1
```

Thread architecture

Both run modes are a producer/consumer pipeline over three intrusive linked-list queues (`dir_queue` , `file_queue` , `new_queue`) guarded by three mutexes and driven by counting semaphores. Handle layout observed at the call sites:

Offset	Content
`+0x00`	`dir_queue` head/tail (walk mode)
`+0x08`	`file_queue` head/tail
`+0x10`	`new_queue` head/tail (findex mode)
`+0x18` / `+0x20` / `+0x28`	mutex for each queue
`+0x30`	semaphore "work available"
`+0x38`	semaphore used as bounded-queue throttle (findex mode)
`+0x40`	target path (scan root)

`zw_run_mode_walk` @ `0xAFF0` — classic recursive walk:

Thread	Function	Count
Producer	`zw_dirwalk_thread` @ `0xB300`	1
Worker (default)	`zw_worker_thread_default` @ `0xA040`	**3**
Worker (`-s`)	`zw_worker_thread_silent` @ `0xA0A0`	**1**

After joining the producer, `main` 's helper posts the semaphore 3× (default) or 1× (`-s`) to release the workers, then joins them (`0xB15B` – `0xB18F`).

`zw_run_mode_findex` @ `0xADE0` — index-driven, the fast path:

Thread	Function	Count
Worker	`zw_worker_thread_findex` @ `0x9FA0`	**3** (spawned 1 s apart, `0xAF08`)
Producer A	`zw_walk_thread` @ `0x9530`	1
Producer B	`zw_newer_thread` @ `0xA6B0`	1

Both producers are joined first, `zw_g_producers_done` is set (`0xAF5B`), the semaphore is posted 3× and the workers drain and exit.

The `/etc/f-index` index (`zw_walk_thread` @ `0x9530` , `zw_emit_bucket` @ `0xBCC0`)

Rather than walking the filesystem, `-a` mode reads a pre-built index laid out as month buckets:

```
/etc/f-index/_other/index.dat        files with year > 2026 or < 2000
/etc/f-index/<YYYY>/<MM>/index.dat   one bucket per month, 2026/12 down to 2000/01
/etc/f-index/baseline.stamp          T_findex (epoch, "%lld")
```

`zw_walk_thread` parses the buckets in an explicit **newest-first** order (logged verbatim by the binary):

```
phase 0 = _other (year>2026, newest)          @ 0x95E5   filter = 1
phase 1 = month buckets 2026/12 .. 2000/01    @ 0x9656   filter = -1
phase 2 = _other (year<2000, oldest)          @ 0x9697   filter = 0
```

`zw_emit_bucket` streams each bucket in 32 MB chunks (`0xBD03`) and parses fixed records:

```
offset 0x00  u64  mtime
offset 0x08  u64  size
offset 0x10  u16  name length N
offset 0x12  N    path
```

The `filter` argument selects records by an mtime test (`0xBE95` – `0xBECF`), which is how the "newest first" prioritisation is implemented. Each record becomes a `file_queue` node carrying `(path, size)`.

This index is **not created by this binary**. It is either Synology's own indexer output repurposed, or the product of a separate staging tool the operator runs first. Either way, its presence on a NAS is a strong pre-encryption indicator.

`zw_newer_thread` @ `0xA6B0` runs in parallel and covers what the index cannot know about: it `glob()` s `/volume*` (`0xA73E`), recurses with `zw_newer_walk` @ `0xA350` , and queues every regular file whose `mtime > T_findex` — i.e. everything created or modified since the index was built.

---

5. Encryption System

Key exchange

Parameter	Value
Algorithm	RSA, PKCS#1 v1.5 encryption (mbedTLS default for a parsed RSA key)
Key size	2048 bits, `e = 65537`
Implementation	mbedTLS, statically linked
Public key form	PEM text, `.rodata:0x4C8B0` , pointer at `.data:0x273040`
SPKI SHA-256	`66e37055247446e1f23efe62cf011b8cf77aa8aa0a937469319a5d8ce9053004`
RNG	`mbedtls_ctr_drbg` seeded from `mbedtls_entropy` , personalisation string `"IEvDBAMINBgkIAqhQ"` (`0x809D`)

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAoL/4FVBq7cKXb0r9bVxl
5pR7ugvMZ/c6Ew63kN1T/BnQqywtPrmgn9Bbvi2SrAUEBLwjmyXY2/jILo6AyUZH
lUxNFRa+uGhJvMOg8p9dtuZiXtptmcXLKBB+8Gw0f6vPxxM969dH1QxC7M2pEECb
ffq18tI/oYPHib7+nEVV0IHDXGSy1SzNg4Rjkasqq6eAhTvB3dKKFEGHygwCsBKC
ITUT0C+mZoLL13hZDqwJELKFb+Q5QTu8y004a5mqbFY+/roMaz4qPJir15i2IVMT
ZmUSOE5P/+PO/tlCzUN4DYwhCpHk9kZdZTdYf4wP+DyqBGORNiP9koS+DNiyLSBe
vwIDAQAB
-----END PUBLIC KEY-----
```

Modulus (hex):

```
a0bff815506aedc2976f4afd6d5c65e6947bba0bcc67f73a130eb790dd53fc19
d0ab2c2d3eb9a09fd05bbe2d92ac050404bc239b25d8dbf8c82e8e80c9464795
4c4d1516beb86849bcc3a0f29f5db6e6625eda6d99c5cb28107ef06c347fabcf
c7133debd747d50c42eccda910409b7dfab5f2d23fa183c789befe9c4555d081
c35c64b2d52ccd83846391ab2aaba780853bc1ddd28a144187ca0c02b0128221
3513d02fa66682cbd778590eac0910b2856fe439413bbccb4d386b99aa6c563e
feba0c6b3e2a3c98abd798b6215313666512384e4fffe3cefed942cd43780d8c
210a91e4f6465d6537587f8c0ff83caa0463913623fd9284be0cd8b22d205ebf
```

`zw_rsa_encrypt_filekey` @ `0x8030` does `pk_init` → `entropy_init` → `ctr_drbg_init` → `ctr_drbg_seed` → `pk_parse_public_key(PEM)` → `pk_can_do(RSA)` → `pk_encrypt(32-byte key)`. Output must be exactly 256 bytes or the file is skipped with `[-]rsa crypt failed` (`0x825F`).

Symmetric cipher

Parameter	Value
Algorithm	AES
Mode	**XTS** (`mbedtls_aes_crypt_xts`)
Key material	32 bytes from `/dev/urandom` , **per file**
Effective cipher	AES-128-XTS — `mbedtls_aes_xts_setkey_enc(ctx, key, 256)` @ `0xCA90` splits the 256 bits into a 128-bit data key and a 128-bit tweak key
Data unit / tweak	file content: `LE64(block_index)` zero-padded to 16 bytes, index starts at **1**; filename: fixed 16-byte constant
Filename tweak constant	`DF B3 07 AC 09 27 21 54 B1 E1 3B 58 7F 52 13 DE` (`.rodata:0x4D430`)

`zw_read_urandom` @ `0xB980` opens `/dev/urandom` and requires a full 32-byte read; on short read it aborts the file with `Failed to read enough random bytes` . There is **no fallback to a weak PRNG** — key generation is sound.

`mbedtls_aes_crypt_xts` @ `0xD600` was verified to be a textbook XTS implementation: it rejects any length outside `[16, 0x1000000]` (`0xD621`), encrypts the data unit with the second AES context at `ctx+288` , then per block does `P ⊕ T` → AES → `⊕ T` with a GF(2^128) multiply-by-α between blocks (`0xD716`) and ciphertext stealing for a partial tail. It takes the AES-NI path when `mbedtls_aesni_has_support` succeeds (`0xD644`). `mbedtls_aes_xts_init` @ `0xC4F0` zeroes a 576-byte structure — two 288-byte `mbedtls_aes_context` — confirming the two-key XTS layout, and `mbedtls_aes_xts_setkey_enc` @ `0xCA90` accepts only 256 or 512 bits and splits the buffer at `key + keybits/16` .

Because the smallest possible chunk is 16 bytes (a 64-byte file yields `(64>>2) & ~0xF` = 16) and the largest is 512 KB, every call sits inside the accepted XTS range — the cipher never fails on a size boundary.

`zw_set_xts_data_unit` @ `0xBB40` zeroes the 16-byte tweak buffer and writes the block index little-endian into the first 8 bytes. Note this **overwrites** the constant loaded at function entry, so the fixed constant is used for the filename only.

File encryption process (`zw_encrypt_file_full` @ `0x9A30`)

```
 1.  size <= 63 bytes                      -> file skipped entirely          @ 0x9AA8
 2.  fd = open(path, O_RDWR)                                                 @ 0x9AE2
 3.  if T_slient > 0 and mtime(fd) >= T_slient:                              @ 0x9B06
        zw_recover_name_from_footer(fd)  -> if the ZAWOOO footer is present,
        the file was already encrypted by a previous "-s" pass: rename it only
        and return (no re-encryption)                                        @ 0x9E54
 4.  key[32] = read(/dev/urandom)                                            @ 0x9B16
 5.  mbedtls_aes_xts_init(ctx); mbedtls_aes_xts_setkey_enc(ctx, key, 256)    @ 0x9B77
 6.  name = basename(path)                                                   @ 0x9BAD
 7.  enc_name = AES-XTS(name padded to 16, tweak = constant)                 @ 0x9BF7
 8.  zw_write_footer(fd, key, enc_name, len)   -> appended at end of file    @ 0x9C52
 9.  new = HEX(enc_name[0:8]) + "." + "fHiQJervS" ; rename(path, new)         @ 0x9CAB
10.  (chunk, stride) = zw_calc_chunk_and_stride(size)                        @ 0x9CEF
11.  for i in 1 .. size/stride:                                              @ 0x9DC7
         set tweak = LE64(i)
         lseek(off) ; read(chunk) ; AES-XTS encrypt ; lseek(off) ; write(chunk)
         off += stride
12.  zw_drop_ransom_note(dir(path), "How To Restore Your Files.txt")         @ 0x9F37
```

Ordering matters: the footer is appended **before** the content passes, and the chunk schedule is computed from the original size captured at scan time, so encryption never overlaps the footer.

`zw_encrypt_file_silent` @ `0x86F0` is the same routine with steps 7–9 reduced and step 12 removed: it writes the footer, encrypts the chunks, and **leaves the filename untouched and drops no note**. Its already-encrypted guard is cheaper — it reads the last 8 bytes and compares them to the magic (`0x8A2B` – `0x8A5D`) instead of parsing the whole footer.

Intermittent encryption (`zw_calc_chunk_and_stride` @ `0x7F70`)

The function returns **two** values — `rax` = bytes encrypted per block, `rdx` = stride between blocks. The decompiler hides the second return; it is only visible in the disassembly (`mov rcx, rdx` at `0x88E9` , then `div rcx`).

File size	Chunk (`rax`)	Stride (`rdx`)	Blocks	Effective coverage
≤ 63 B	0	0	—	file skipped
64 B – 256 KB−1	`(size >> 2) & ~0xF`	`size & ~0xF`	1	~25 % (head only)
256 KB – 1 MB−1	`(size / 10) & ~0xF`	`size & ~0xF`	1	~10 % (head only)
1 MB – 8 MB−1	`((size>>1 & ~0xF) / 10) & ~0xF`	`(size >> 1) & ~0xF`	2	~10 % (2 × 5 %)
8 MB – 64 MB−1	`0x80000` (512 KB)	`0x400000` (4 MB)	size / 4 MB	~12.5 %
≥ 64 MB	`0x80000` (512 KB)	`0x800000` (8 MB)	size / 8 MB	~6.25 %

Because `chunk < stride` in every band, the last block always ends inside the original data. A 1 GB file has only 64 MB encrypted, spread as 512 KB every 8 MB — fast enough to tear through a multi-terabyte NAS, and destructive enough that virtually every container format is unusable.

Encrypted file format

```
[ORIGINAL DATA, selectively overwritten in place]
  block i (i = 1..n) at offset (i-1)*stride, length chunk
      = AES-128-XTS(data, key, data_unit = LE64(i))

[FOOTER, appended, total length L + 268]
  +0x000   256 B   RSA-2048 PKCS#1 v1.5 ( 32-byte AES-XTS key )
  +0x100   L       AES-128-XTS( basename padded to 16, tweak = 0x4D430 constant )
  +0x100+L   4 B   u32  total footer length  (= L + 268)
  +0x104+L   8 B   u64  magic 0x54212709AC07B3DF
                        on disk: DF B3 07 AC 09 27 21 54
```

The magic doubles as the first half of the filename tweak constant — the same 8 bytes appear at `.rodata:0x4D430` and at `.data:0x273028`. That is a reliable, cheap file-carving signature.

The footer is written with `lseek(fd, 0, SEEK_END)` followed by a single `write`, then `fdatasync` (syscall 75, `0x44FAC`) before the content pass begins.

`zw_recover_name_from_footer` @ `0x9940` is the reader for this structure and documents it exactly: `lseek(-8, SEEK_END)` → magic check, `lseek(-12, SEEK_END)` → total length, `lseek(-(total-256), SEEK_END)` → first 8 bytes of the encrypted name → hex → new filename.

The two-stage operation

This is the design decision that matters most operationally. `T_slient` is written by the `-s` worker into `/etc/slient-time/baseline.stamp` when silent mode starts, and it is read back by `zw_load_baselines` on every subsequent run.

1. **Stage 1 —** `-s`: files are encrypted in place. Names unchanged, no notes, no extension. From a user's point of view the share still "looks" normal; only file contents are already destroyed. The victim keeps working, keeps backing up the already-encrypted data, and the operator keeps a low profile.
2. **Stage 2 — default or** `-a`: `T_slient` is now set. For every file whose `mtime` is at or after it, the footer magic is checked; if present, the file is **renamed only** and the note is dropped. The whole estate flips to a visibly ransomed state in the time it takes to `rename()`, with no second encryption pass.

Any file created or modified between the two stages has no footer, fails the magic check, and is encrypted normally in stage 2 — so the coverage is complete either way.

---

6. File Targeting

Targeted files

All regular files larger than 63 bytes under the target path, except those excluded below. There is no extension allow-list.

Scan roots

Mode	Root
default, `-s`	the `-p` argument, recursive
`-a`	records listed in `/etc/f-index/**/index.dat`, plus a `glob("/volume*")` sweep for anything newer than `T_findex`

`/volume*` is the Synology data-volume naming convention (`/volume1`, `/volume2`, …).

Excluded directories (16) — `zw_is_excluded_syno_dir` @ `0x8560`

Matched with `strcmp` (**case-sensitive**), only for names starting with `@`, against `.data.rel.ro:0x26B9E0`:

Directory	Purpose on a Synology NAS
`@appconf` `@apphome` `@appstore` `@appdata` `@appshare` `@apptemp`	package/application state
`@database`	internal databases (PostgreSQL, …)
`@synoconfd` `@synoldap` `@userpreference`	DSM configuration
`@SynoFinder-log` `@SynoFinder-etc-volume`	file-indexing service
`@eaDir`	thumbnails / extended attributes
`@sssd_cache`	SSSD credential cache
`@S2S` `@tmp`	sync / temporary

The intent is unambiguous: keep DSM itself bootable and the web UI reachable so the victim can read the note and negotiate. Also, `zw_dirwalk_thread` skips every directory whose name starts with `.` (`0xB3C1`), and `zw_newer_walk` skips `.` and `..` (`0xA4E8`).

Excluded extensions (54) — two inconsistent implementations

Both compare **case-insensitively** (`strcasecmp`, `libc_strcasecmp` @ `0x42AF0`).

List A — `.data.rel.ro:0x26BA60`, entries carry a leading dot, used by `zw_is_excluded_ext_dotted` @ `0x85D0` on the `-a` / `newer_walk` path:

```
.
386 .adv .ani .bat .bin .cab .cmd .com .cp .cur .deskthemepack .diagcab .diagcfg .diagpkg .d .dr
v .exe .hp .ic .icns .ico .ics .idx .df .nk .mod .mpa .msc .msp .msstyes .msu .ns .nomedia .ocx
.prf .ps1 .rom .rtp .scr .shs .sp .sys .theme .themepack .wpx .ock .hta .msi .pdb .search-
ms .ini .MXvF0 .MXvF0_F .c2w
```

List B — `.data:0x274080`, same entries without the dot, used by `zw_should_encrypt_file` @ `0xB230` on the default / `-s` walk path, with the last three replaced:

```
… ini SpXv2 SpXv2_F ctw
```

`zw_should_encrypt_file` additionally rejects the current build's own suffix (`fHiQJervS`, `0xB27A`) and the note filename (`0xB28B`) before consulting list B.

Two observations:

- The base 51 extensions are the **LockBit Windows exclusion list**, copied verbatim (`.386`, `.deskthemepack`, `.diagcab`, `.msstyes` — with LockBit's own typos preserved). On a

Synology NAS almost none of these exist. It is dead weight that betrays where the builder's configuration came from.

- The last three entries of each list are **self-exclusions from other builds**. List A protects files ending in `.MXvF0` / `.MXvF0_F` / `.c2w`; list B protects `SpXv2` / `SpXv2_F` / `ctw`. Neither list protects `fHiQJervS`. Only the hardcoded check at `0xB27A` does — and only on the walk path. On the `-a` path a file already renamed to `.fHiQJervS` is **not** filtered out by extension; it is caught instead by the footer magic check.

Excluded files

File	Mechanism
`How To Restore Your Files.txt`	name compare, `zw_should_encrypt_file` @ `0xB28B`
any file already carrying the ZAWOOO footer	magic check @ `0x999B` / `0x8A5D`, gated on `mtime >= T`

---

7. Recovery Inhibition

None. There is no shadow-copy equivalent, no snapshot deletion, no `synosnapshot` / `btrfs subvolume delete` / `synoshare` call, no log wiping, no backup-service tampering and no `unlink` of anything.

Data destruction is achieved purely by in-place overwrite: the plaintext blocks are read, encrypted and written back to the same offsets, so the original bytes are gone from the filesystem with no intermediate copy. Combined with the destroyed filenames this is effective enough that the operator apparently saw no need for anything else.

Synology snapshot/Btrfs snapshots, if configured and stored on an unaffected volume or replicated off-box, remain a viable recovery path — the binary does nothing to touch them.

---

8. Targeted Services

None. No service is stopped, disabled or killed. `synoservice`, `systemctl` and equivalents are absent from the binary. Files held open by DSM services are simply encrypted underneath them (the encryptor opens `O_RDWR`, and NAS shares rarely hold mandatory locks).

---

9. Targeted Processes

None. There is no process enumeration and no `kill` / `SIGTERM` logic anywhere in the malware-specific code.

---

10. Persistence & Evasion

Persistence

None. No cron entry, no `/etc/rc*`, no systemd unit, no DSM task scheduler entry, no copy of itself anywhere. The binary is a one-shot payload; persistence, if any, belongs to the intrusion tooling that dropped it.

Execution guardrail — the clock window (`zw_worker_thread_silent` @ `0xA0A0`)

Before silent mode does anything it must establish a trustworthy "now". It tries six independent clock sources in order, and the first value that lands inside a hardcoded window wins:

Order	Source	Function
1	`clock_gettime`	`zw_time_via_clock_gettime` @ `0x9150`
2	legacy `SYS_clock_gettime`	`zw_time_via_legacy_syscall` @ `0x91C0`
3	`time()`	`zw_time_via_time` @ `0x8DC0`
4	`gettimeofday`	`zw_time_via_gettimeofday` @ `0x8E30`
5	`/proc/stat` `btime` + `/proc/uptime`	`zw_time_via_proc` @ `0x8FB0`
6	`popen("date +%s")`	`zw_time_via_date_cmd` @ `0x8EA0`

Accepted window (`0xA20C` , expressed as `(t - TS_MIN) <= 0x11A2100`):

```
TS_MIN   1780272000  = 2026-06-01T00:00:00Z
TS_MAX   1798761600  = 2027-01-01T00:00:00Z
```

If all six fail, a hardcoded fallback is used: `1784990000` = **2026-07-25T14:33:20Z**. That constant is the closest thing this binary has to a build timestamp, and it sits inside the accepted window.

If `/etc/slient-time/baseline.stamp` already exists but holds a value outside the window, the process **exits immediately** (`0xA342` , `slient_time_file: exist %s but invlid`). Same outcome if the stamp cannot be created (`0xA2F6`).

The effect is a genuine execution guardrail (T1480): the payload refuses to run outside a seven-month window, and an analysis machine with a skewed or reset clock will see it exit without touching a single file. Reading `btime` + `uptime` from `/proc` and shelling out to `date` are specifically there to defeat a naive `LD_PRELOAD` or `faketime` hook on `time()` / `gettimeofday()` .

Note that this guard exists **only in silent mode**. Default mode reads whatever value the stamp holds and never validates it.

Debug logging

`zw_log_thread` @ `0x8AE0` and `zw_log_work` @ `0xA930` implement a levelled logger on stderr. The level comes from the environment variable **`FTI_LOG`** (`0x8CB7`), accepting `error|warn|info|debug|trace` or `0` – `4` , defaulting to `info` (2). Level names are at `.data.rel.ro:0x26BC20` = `ERR WRN INF DBG TRC` . Format:

```
%H:%M:%S.%03ld [LVL] tid=%ld src/enc/thread.c:387: <message>
```

`zw_log_work` @ `0xA930` is a byte-for-byte duplicate of `zw_log_thread` with its own cached level and its own level-name table at `.data.rel.ro:0x26BC60` — the logger is a `static` helper instantiated once per translation unit, which is what exposes the two source filenames.

Setting `FTI_LOG=trace` turns the sample into its own instrumentation. This is a significant analyst advantage and a significant operator liability — a NAS running this under a captured stderr leaves a full inventory of what was encrypted.

Anti-analysis summary

Technique	ID	Address	Notes
Stripped, statically linked payload	T1027.008	whole binary	no symbols, no imports, no anchor points
Execution guardrail on wall-clock	T1480	`0xA20C`	seven-month window, six clock sources
Single-instance guard	—	`0x7B2F`	`flock` on `/var/run/enc-nas.lock`, not an evasion

`.init_array` @ `0x26B9D0` holds a single entry, `0x7F30`, which tail-calls `0x7E90` and returns 0 — the stock `frame_dummy` / `register_tm_clones` pair. `.fini_array` @ `0x26B9D8` holds `0x7ED0`, the stock `__do_global_dtors_aux`. **No malicious constructor or destructor.**

Dead code — an earlier variant left in the binary

Nine malware functions are compiled in with **zero cross-references**, verified against the call graph rather than assumed. Two of them are not leftovers of debugging but of an **earlier encrypted-file format**, and they describe what previous ZAWOOO victims' files look like:

Address	Name	What it does
`0x83E0`	`zw_write_footer_v1_dead`	**v1 footer writer.** RSA-encrypts the 32-byte key and appends **only the 256-byte RSA blob** at EOF — no encrypted filename, no length field, **no magic**. Logs `[-]rsa nncryption failed` (note the typo, distinct from the live path's `[-]rsa crypt failed`).
`0xB740`	`zw_rename_append_suffix_v1_dead`	**v1 rename.** `rename(path, path + suffix)` — **appends** the extension to the original filename instead of replacing it. The original name is preserved.
`0x9700`	`zw_hexdump_dbg`	`%02x` dump helper
`0x9780`	`zw_file_time_str`	`mtime` → `%Y-%m-%d %H:%M:%S`
`0x8550`	`zw_align_down_dead`	`x & -n`
`0x86B0`	`zw_perror_mutex_dead`	`perror("mutex")`
`0xB1F0`	`zw_stat_or_perror_dead`	`stat()` + `perror("stat")`
`0xBA30`	`zw_strlwr_dead`	in-place lowercase
`0x9F80`	`zw_silent_body_wrapper_dead`	`if (path) zw_encrypt_file_silent(...)` — inlined by the live worker

The v1 pair matters operationally. Files encrypted by that earlier build have:

- the **original filename intact**, with the campaign suffix simply appended (`report.xlsx.<suffix>` rather than `<16 hex>.<suffix>`);
- a **256-byte tail with no magic and no length field**, so the `ZAWOOO_Encrypted_File` rule below will **not** match them.

Hunting for older infections therefore needs a different signature: a file whose size grew by exactly 256 bytes with a high-entropy 256-byte tail, carrying a double extension. The functions being present but unreferenced also means this sample's builder still ships the v1 code path — a future build could re-enable it.

No anti-debug, no `ptrace` self-attach, no timing checks, no VM/sandbox fingerprinting, no API hashing, no string encryption, no packing, no self-deletion. For a 2026 ransomware payload this is remarkably bare — the operator is relying on the target being an appliance nobody is watching, not on evading an EDR.

---

11. Command-Line Arguments

Argument	Description
`-p <path>`	Target path. Mandatory in every mode; without it, `target path is null` and exit `-1`.
`-s`	"slient" (silent) mode. One worker thread. Encrypts in place, **no rename, no ransom note**. Creates and validates `/etc/slient-time/baseline.stamp`.
`-a`	Auto mode. Takes the `flock` on `/var/run/enc-nas.lock`, then drives encryption from `/etc/f-index`. Requires `/etc/f-index/baseline.stamp`, otherwise `no f-index find, please use -p`. Three worker threads.
(none)	Default mode. Recursive walk of `-p`, three worker threads, rename + ransom note.

Usage string as printed by the binary (`.rodata:0x4CAE0`):

```
usage: %s -p path
  -p target_path
  -s slient
  -a auto
```

The `-x` mentioned in the error message `alredy run(-x ), exit` is **not implemented** in this build — another sign of a shared codebase between variants.

---

12. Static Imports Summary

No dynamic imports. Capabilities are provided by the statically linked libc and mbedTLS:

Category	Routines used
Crypto	`mbedtls_aes_xts_init` `0xC4F0`, `mbedtls_aes_xts_setkey_enc` `0xCA90`, `mbedtls_aes_crypt_xts` `0xD600`, `mbedtls_pk_parse_public_key`, `mbedtls_pk_encrypt`, `mbedtls_ctr_drbg_seed`, `mbedtls_entropy_func`
File I/O	`open` `0x3C742`, `read` `0x4508C`, `write` `0x450E7`, `lseek` `0x44FDE`, `close` `0x44F74`, `rename` `0x4031C`, `access` `0x44F5C`, `stat` `0x3F017`, `lstat` `0x3EFEE`, `mkdir` `0x3F003`, `flock` `0x3C803`
Directory	`opendir` `0x3C3B1`, `readdir` `0x3C3F2`, `closedir` `0x3C38B`, `glob` `0x3EACF`, `globfree` `0x3EDF9`
Threading	`pthread_create` `0x43986`, `pthread_join` `0x43F25`, `pthread_mutex_{init,lock,unlock}` `0x43F3F` / `0x43F57` / `0x4431E`, `sem_{init,wait,post}` `0x444C7` / `0x44568` / `0x444F5`
Process	`popen` / `/bin/sh` (only for `date +%s`), `getopt` `0x3E036`, `getenv` `0x3C66A`
Network	**none** — no `connect`, no `sendto`, no C2

The single `socket` syscall present in `.text` belongs to libc's NSS client (`/var/run/nscd/socket`), not to the malware.

---

13. IDA Analysis — Renamed Functions

Address	Name	Size	Description
0x7A70	main	0x24D	argv parsing, lock, mode dispatch
0x7F70	zw_calc_chunk_and_stride	0xBA	intermittent-encryption schedule, returns rax =chunk rdx =stride
0x8030	zw_rsa_encrypt_filekey	0x10D	mbedTLS RSA-2048 wrap of the 32-byte file key
0x8140	zw_write_footer	0x295	builds and appends the 268+L byte footer
0x8540	zw_align_up	0xC	round length up to a multiple of 16
0x8560	zw_is_excluded_syno_dir	0x62	@… directory exclusion, strcmp
0x85D0	zw_is_excluded_ext_dotted	0x7B	extension exclusion (list A), strcasecmp
0x8650	zw_get_file_mtime	0x58	fstat → st_mtime
0x86C0	zw_load_baselines	0x2A	loads T_slient and T_findex
0x86F0	zw_encrypt_file_silent	0x3E7	encrypt in place, no rename, no note
0x8AE0	zw_log_thread	0x2D4	levelled logger (thread.c call sites)
0x8DC0	zw_time_via_time	0x67	clock source 3
0x8E30	zw_time_via_gettimeofday	0x63	clock source 4
0x8EA0	zw_time_via_date_cmd	0x10A	clock source 6, popen("date +%s")
0x8FB0	zw_time_via_proc	0x195	clock source 5, /proc/stat btime + /proc/uptime
0x9150	zw_time_via_clock_gettime	0x63	clock source 1
0x91C0	zw_time_via_legacy_syscall	0x77	clock source 2
0x9240	zw_load_T_slient	0x2D6	parse + range-validate /etc/slient-time/baseline.stamp
0x9530	zw_walk_thread	0x1CB	f-index bucket producer, newest-first
0x9700	zw_hexdump_dbg	0x74	debug hex dump helper (unreferenced in the encrypt path)
0x9780	zw_file_time_str	0x1B2	mtime → %Y-%m-%d %H:%M:%S
0x9940	zw_recover_name_from_footer	0xE8	footer parser, yields the renamed filename
0x9A30	zw_encrypt_file_full	0x54F	**core**: key, footer, rename, chunk loop, note
0x9FA0	zw_worker_thread_findex	0x9C	consumer for -a (drains new_queue then file_queue)
0xA040	zw_worker_thread_default	0x55	consumer for default mode
0xA0A0	zw_worker_thread_silent	0x2A7	consumer for -s , plus the clock guardrail
0xA350	zw_newer_walk	0x358	recursive scan for mtime > T_findex
0xA6B0	zw_newer_thread	0x27B	glob("/volume*") driver for zw_newer_walk
0xA930	zw_log_work	0x2D4	levelled logger (work.c call sites)
0xAC10	zw_xmalloc	0x13	allocation wrapper
0xAC30	zw_queue_node_new	0x5A	queue node + path copy
0xAC90	zw_queue_push	0x51	tail insert under mutex

Address	Name	Size	Description
`0xACF0`	`zw_queue_node_free`	0x1A	node release
`0xAD10`	`zw_queue_pop`	0x61	head removal under mutex
`0xAD80`	`zw_load_T_findex`	0x60	parse `/etc/f-index/baseline.stamp`
`0xADE0`	`zw_run_mode_findex`	0x20C	3 workers + walk_thread + newer_thread
`0xAFF0`	`zw_run_mode_walk`	0x1FC	1 producer + 3 (or 1) workers
`0xB230`	`zw_should_encrypt_file`	0xC1	extension exclusion (list B) + note-name guard
`0xB300`	`zw_dirwalk_thread`	0x43F	recursive producer, pushes dirs and files
`0xB7D0`	`zw_to_hex`	0x40	uppercase hex encoder
`0xB810`	`zw_encrypt_filename_xts`	0xAF	pad + AES-XTS the basename
`0xB8C0`	`zw_basename_r`	0xB1	basename into caller buffer
`0xB980`	`zw_read_urandom`	0xAA	`/dev/urandom` key material
`0xBA70`	`zw_drop_ransom_note`	0xC5	per-directory note drop
`0xBB40`	`zw_set_xts_data_unit`	0x2B	tweak = `LE64(index)`
`0xBB70`	`zw_rename_encrypted`	0x149	`rename()` in the same directory
`0xBCC0`	`zw_emit_bucket`	0x44E	32 MB chunked parser for `index.dat`
`0x83E0`	`zw_write_footer_v1_dead`	0x15E	**dead** — v1 footer: 256-byte RSA blob only
`0xB740`	`zw_rename_append_suffix_v1_dead`	0x8A	**dead** — v1 rename: appends suffix, keeps original name
`0x8550`	`zw_align_down_dead`	0xA	**dead** — `x & -n`
`0x86B0`	`zw_perror_mutex_dead`	0xC	**dead** — `perror("mutex")`
`0xB1F0`	`zw_stat_or_perror_dead`	0x3A	**dead** — `stat()` + `perror("stat")`
`0xBA30`	`zw_strlwr_dead`	0x38	**dead** — in-place lowercase
`0x9F80`	`zw_silent_body_wrapper_dead`	0x11	**dead** — inlined by the live silent worker

Renamed globals

Address	Name	Value / role
`0x273028`	`zw_g_footer_magic`	`0x54212709AC07B3DF`
`0x273030`	`zw_g_note_name`	→ `How To Restore Your Files.txt`
`0x273038`	(suffix pointer)	→ `fHiQJervS`
`0x273040`	(pubkey pointer)	→ RSA PEM
`0x274080`	`zw_g_excl_ext_nodot`	exclusion list B, 54 entries
`0x274500`	`zw_g_T_slient`	silent-mode baseline epoch
`0x274508`	`zw_g_T_findex`	f-index baseline epoch
`0x274514`	`zw_g_producers_done`	producer-finished flag
`0x274518`	`zw_g_newer_pushed`	counter for `newer_thread`
`0x274520`	`zw_g_scan_path_buf`	64 KB path buffer for `newer_walk`
`0x26B9E0`	(syno dir list)	exclusion list, 16 entries
`0x26BA60`	`zw_g_excl_ext_dotted`	exclusion list A, 54 entries
`0x26BC20`	`zw_g_log_level_names`	`ERR WRN INF DBG TRC`
`0x4D430`	`zw_g_xts_name_tweak`	16-byte filename tweak constant

14. Indicators of Compromise (IOCs)

Hashes

Type	Value
SHA-256	`33d3afddaa5710cdcc4e93a7bae8be010c19747fb53d85fcd54ef32056a1eb0b`
SHA-1	`9d371d77700dca0950249b8e8a1da128dbbdc719`
MD5	`d8957e860cb421e9c06e0fd05010cd4b`
RSA SPKI SHA-256	`66e37055247446e1f23efe62cf011b8cf77aa8aa0a937469319a5d8ce9053004`

Network

Type	Value
Onion (DLS)	`fyenuhkq3pfhnbpidj5jm2fl2lryxip4byhg6eozynrnlomu4szf2nyd.onion`
E-mail	`zawoooorecover@onionmail.org`
Session ID	`05b0794c896a54271d3d39c3b395d86aee2d5c05f196c59f3781bac1b1ac1a1a12`

The binary itself generates **no network traffic**.

Files and paths

Indicator	Value
Ransom note	`How To Restore Your Files.txt` (one per directory, mode 0777)
Encrypted filename	`<16 uppercase hex>.fHiQJervS`
Lock file	`/var/run/enc-nas.lock`
Silent baseline	`/etc/slient-time/baseline.stamp` (directory created mode 0755)
Index baseline	`/etc/f-index/baseline.stamp`
Index buckets	`/etc/f-index/<YYYY>/<MM>/index.dat`, `/etc/f-index/_other/index.dat`
Victim ID	`<VICTIM_ID>` (16 uppercase hex, per build — redacted)

File-carving signature

Last 12 bytes of every encrypted file:

```
[u32 total_footer_len][DF B3 07 AC 09 27 21 54]
```

with `total_footer_len == 268 + padded_basename_len` and `total_footer_len % 16 == 12`.

YARA

```
rule ZAWOOO_Linux_NAS_Encryptor
{
    meta:
        description = "ZAWOOO Synology NAS ransomware (ELF64 static-pie)"
        reference   = "33d3afddaa5710cdcc4e93a7bae8be010c19747fb53d85fcd54ef32056a1eb0b"

    strings:
        $magic   = { DF B3 07 AC 09 27 21 54 B1 E1 3B 58 7F 52 13 DE }
        $lock    = "/var/run/enc-nas.lock"      ascii
        $slient  = "/etc/slient-time/baseline.stamp" ascii
        $findex  = "/etc/f-index/baseline.stamp"     ascii
        $note    = "How To Restore Your Files.txt"   ascii
        $log1    = "src/enc/thread.c"           ascii
        $log2    = "src/enc/work.c"             ascii
        $usage   = "usage: %s -p path"          ascii
        $typo1   = "alredy run(-x ), exit"      ascii
        $typo2   = "xts encrypt fail: -0x%04X"  ascii
        $syno    = "@SynoFinder-etc-volume"     ascii

    condition:
        uint32(0) == 0x464C457F and filesize < 4MB and 3 of them
}

rule ZAWOOO_Encrypted_File
{
    meta:
        description = "File encrypted by ZAWOOO - footer magic"

    condition:
        filesize > 268 and
        uint32be(filesize - 8) == 0xDFB307AC and
        uint32be(filesize - 4) == 0x09272154
}
```

Behavioural detections

- Creation of `/etc/slient-time/` or `/etc/f-index/` on a DSM appliance — neither is a Synology path.
- A process holding an exclusive `flock` on `/var/run/enc-nas.lock`.
- Mass `rename()` to a `^[0-9A-F]{16}\.[A-Za-z0-9]{5,12}$` pattern across `/volume*`.
- `How To Restore Your Files.txt` appearing in many directories at once.
- A single process issuing sustained `open(O_RDWR)` + `pread` / `pwrite` at a fixed stride (4 MB or 8 MB) across large files.
- An unexpected `sh -c "date +%s"` child of a long-running binary on a NAS.

Distinctive strings

- `"~~~ You have been attacked by ZAWOOO - a ransomware that prioritizes reputation. ~~~~"`
- `"alredy run(-x ), exit"`
- `"no f-index find, please use -p"`
- `"walk_thread: phase 0 = _other (year>2026, newest)"`
- `"load_T_slient: value %lld is BELOW TS_MIN(%lld), diff=%lld seconds"`

- `"emit_bucket: chunked read %s (CHUNK=%uMB) filter=%d"`
- `"[-]rsa nncryption failed"`
- `"IEvDBAMINBgkIAqhQ"` (CTR-DRBG personalisation)
- `"fHiQJervS"`, `"SpXv2"`, `"SpXv2_F"`, `"ctw"`, `".MXvF0"`, `".MXvF0_F"`, `".c2w"`

---

15. MITRE ATT&CK Mapping

ID	Technique	Implementation
T1486	Data Encrypted for Impact	AES-128-XTS intermittent in-place encryption, RSA-2048 key wrap, `zw_encrypt_file_full` @ `0x9A30`
T1485	Data Destruction	plaintext blocks overwritten in place; original filenames destroyed and only recoverable with the private key
T1083	File and Directory Discovery	recursive `opendir` / `readdir` walk @ `0xB300`, `glob("/volume*")` @ `0xA73E`
T1005	Data from Local System	`/etc/f-index` bucket parsing @ `0xBCC0` gives a full pre-built inventory of the estate
T1480	Execution Guardrails	wall-clock window `2026-06-01 .. 2027-01-01` validated across six clock sources @ `0xA20C`; refuses to run outside it
T1027.008	Obfuscated Files or Information: Stripped Payloads	stripped `static-pie` ELF, empty `.dynsym`, no import table
T1059.004	Command and Scripting Interpreter: Unix Shell	`popen("date +%s")` as clock source of last resort @ `0x8EA0`
T1657	Financial Theft	double-extortion ransom note, Session + OnionMail negotiation, GDPR-fine framing
T1082	System Information Discovery	`/proc/stat`, `/proc/uptime` reads @ `0x8FB0`

Deliberately **not** claimed: no T1490 (Inhibit System Recovery), no T1489 (Service Stop), no T1562 (Impair Defenses), no T1070 (Indicator Removal), no T1071 / T1041 (C2 or exfiltration) — none of these exist in the binary. The note's exfiltration threat is not backed by any code in this sample; any data theft happened through separate tooling.

---

16. Summary

ZAWOOO is a purpose-built Synology NAS encryptor: a 467 KB stripped `static-pie` ELF64 carrying mbedTLS, 47 functions of actual malware logic, and nothing else. Its cryptography is correct — a fresh 32-byte key per file from `/dev/urandom`, AES-128-XTS with a proper per-block data unit, the key wrapped under a hardcoded RSA-2048 public key and appended in a self-describing footer. There is no key reuse, no PRNG weakness, no plaintext key left on disk. Recovery without the operator's private key is not possible from this sample.

What distinguishes it is not the crypto but the **operational design**. The `-s` "silent" stage encrypts an entire NAS in place while leaving every filename intact and dropping no note, so the estate looks untouched — and keeps getting backed up in its already-encrypted state. A second run then flips everything visible in the time it takes to `rename()`, because the footer magic lets it recognise its own prior work and skip re-encryption. The `-a` mode makes the first stage fast by consuming a pre-built `/etc/f-index` inventory and processing it newest-file-first, so the data the victim actually cares about is

destroyed before anyone notices. The intermittent schedule — 512 KB every 8 MB on large files — means a multi-terabyte volume can be ruined in a fraction of the time a full-encryption locker would need.

Against that, the payload is technically unsophisticated in every other respect. It has no packing, no obfuscation, no anti-debug, no anti-VM, no C2 and no persistence. It ships with a full debug logger complete with source filenames, line numbers and an `FTI_LOG` environment switch, which hands an analyst a narrated execution trace. It carries LockBit's Windows extension exclusion list verbatim — `.deskthemepack`, `.diagcab`, `.msstyes` — on a platform where none of those files exist, and it carries two mutually inconsistent copies of that list with self-exclusion suffixes belonging to other builds. The ransom note promises three file extensions where the code produces one. Typos are compiled into filesystem paths (`/etc/slient-time`).

The picture is of an operator with a working, victim-specific builder and a well-thought-out NAS attack playbook, running on top of a codebase that is stitched together from borrowed configuration and never cleaned up. The one real defensive lever the binary offers is its clock guardrail: outside `2026-06-01` – `2027-01-01`, silent mode refuses to run at all. The practical detection levers are the two directories it must create — `/etc/slient-time` and `/etc/f-index` — the `/var/run/enc-nas.lock` file, and the 8-byte footer magic that makes every encrypted file trivially identifiable.

---

17. Cross-Verification (Ghidra)

The binary was independently re-analysed in Ghidra (`x86:LE:64:default`, image base `0x100000`; subtract `0x100000` from Ghidra addresses to obtain the addresses used above). All addresses in this report are the ELF virtual addresses as loaded by IDA.

Confirmed independently:

- **Footer layout.** Ghidra's decompilation of `zw_write_footer` shows `malloc(namelen + 0x10C)`, the 256-byte RSA blob at offset 0, `memcpy(buf + 0x100, enc_name, namelen)`, `*(int *)(buf + namelen + 0x100) = total`, `*(u64 *)(buf + namelen + 0x104) = DAT_00373028`, then `lseek(fd, 0, SEEK_END)` → `write` → `fdatasync`. Identical to the layout given in §5.
- **Footer reader.** `zw_recover_name_from_footer` reads `-8` (magic), `-12` (u32 length), `-(length - 0x100)` (8 bytes of encrypted name) — identical.
- **XTS key split.** `mbedtls_aes_xts_setkey_enc` accepts only 256/512 bits, keys the crypt context with `keybits >> 1` and the tweak context at `ctx + 0x120` with `key + keybits/16` — confirming **AES-128-XTS** from a 32-byte key.
- **Function boundaries.** Ghidra found 1 038 functions to IDA's 960. In the malware range (`0x7A00` – `0xC200`) the two agree on every function start; Ghidra additionally recovered three small ones IDA had not created (`0x8550`, `0x86B0`, `0x9F80`). No function start disagreed, so the boundary-shift failure mode does not apply here.

Where Ghidra did **not** help: its decompiler renders `zw_calc_chunk_and_stride` @ `0x7F70` as returning a single `uint`, exactly like Hex-Rays. **Both decompilers drop the second return value in `rdx`.** The dual return is only visible in the disassembly — `mov rdx, r8` before every `retn` in the callee, and `mov rcx, rdx` / `div rcx` at `0x88E9` in the caller — and it is what defines the intermittent-encryption stride. Any analysis of this family that trusts either decompiler on that function will get the encryption coverage wrong.