Nightspire

Technical Analysis — Cti Report

REVERSE-ENGINEERED REPORT

RansomLook · ransomlook.io
File last modified: 2026-05-23

Sample SHA-256: `69f5515ff3f554233840ad2f2397b345f955013017a9ae14ed4e762f52d936af`

Scope statement: this analysis covers a single Go-compiled sample (SHA-256 in §1). Findings describe what was observed in this binary; they do not imply that all NightSpire deployments share identical implementation details. Family-level claims are restricted to indicators flagged as Family in §13.

Reading convention: the markers `**Observed:**` and `**Inferred:**` are used throughout this brief to separate facts read directly from the binary from analytical extrapolation.

---

1. Executive Summary

Family	Self-identified as NightSpire.Team (via `.nspire` extension, `/[NSPIRE_MSG].txt` note filename pattern, `"~~~ You have been attacked by NightSpire.Team ~~~"` salutation, `nightspire.team2026@onionmail.org` contact, dual onion infrastructure)
Sample variant	Per-deployment build (ransom note filename template `/[NSPIRE_MSG].txt` suggests configurable deployment; RSA-4096 public key embedded as base64 literal at `0x5afa9e`)
Platform	Windows x86-64 console application (PE32+ Go 1.24.11 runtime)
Builder fingerprint	Go compiler 1.24.11, no obfuscation, transparent string storage in .rdata section
Encryption	AES-256-CTR (per-file keys) wrapped with **RSA-4096-OAEP-SHA512**, intermittent algorithm with 3 file-size tiers
Concurrency	Parallel goroutine architecture: 1 goroutine per drive letter (A: through Z:), configurable via `-k` thread count parameter
Discovery	Local drive enumeration via `FindFirstVolumeW` / `FindNextVolumeW`, recursive directory walking with exclusion logic
Pre-encryption operations	Drive enumeration burst (26 parallel goroutines), file extension filtering, re-encryption prevention via "sNightspire" signature checking
Cryptographic primitives	RSA-4096-OAEP-SHA512 (Go crypto/rsa standard library); AES-256-CTR (Go crypto/aes); crypto/rand for key generation; embedded base64 RSA public key
Exfiltration	The note text claims data theft and leak site operation. **No exfiltration code is present in this binary** (no network imports, no HTTP/socket operations). The claim is text-only
Recovery (operator-side)	RSA-wrapped per-file AES keys, standard Go crypto implementation with no identified static weaknesses
Recovery (victim-side)	No cryptographic recovery weaknesses identified during static analysis
Sample SHA-256	`69f5515ff3f554233840ad2f2397b345f955013017a9ae14ed4e762f52d936af`
Primary onion	`nspire7lugml7ybqyjaaxtsgrs4qn3fcon3lrjbih6wamttvdm5ke4qd.onion`
Leak site	`nspirep7orjq73k2x2fwh2mxgh74vm2now6cdbnnxjk2f5wn34bmdxad.onion`
Email	`nightspire.team2026@onionmail.org`, `nightspireteam.receiver@onionmail.org`

---

2. Notable Observations

2.1 Intermittent encryption exposes file size intelligence

Observed: function `main.getEncryptionType` @ `0x50fc80` implements three-tier file processing based on file size thresholds: - **Type 1**: Files ≤100MB (`0x6400000`) — complete encryption in sequential 1MB chunks - **Type 2**: 100MB-1.6TB (`0x1900000000`) — first 1GB complete + distributed 1MB chunks with calculated spacing - **Type 3**: Files >1.6TB — first 1GB complete + distributed 1GB chunks with calculated spacing

Observed: the algorithm reveals organizational data structure through encryption timing patterns. Function `main.startEncrypting` @ `0x510aa0` shows distinct code paths: Type 1 uses `0x100000` (1MB) chunk size in sequential loop, Type 2/3 use `0x40000000` (1GB) for initial complete encryption followed by distributed chunk processing.

Inferred — operational implication: large file processing creates distinct behavioral signatures. Database files, backup archives, and media collections receive partial encryption, enabling defender triage based on encryption duration patterns.

2.2 Parallel goroutine architecture creates detection window

Observed: main execution in `main.main` @ `0x511d40` spawns 26 concurrent goroutines via Go's sync.WaitGroup, one per Windows drive letter (A: through Z:). Each goroutine calls `main.EncryptDir` @ `0x511260` for recursive directory processing.

Observed: volume enumeration occurs via sequential `FindFirstVolumeW` / `FindNextVolumeW` calls before goroutine spawn, creating burst of volume API activity measurable via ETW/EDR telemetry.

Inferred — operational implication: the sequential volume enumeration phase provides a detection window before parallel encryption begins. High-volume calls to Windows volume APIs from a single process create distinctive behavioral signature.

2.3 Configuration stored as plaintext Go strings

Observed: all infrastructure and configuration elements stored as cleartext strings in PE .rdata section without encryption or obfuscation: - Onion URLs: literal strings at `0x5a24cb` range - Authentication credentials: `NSPIRE830NPH7ZLBRW39` / `769FZisalII12Rph` at static offsets - RSA public key: base64 blob at `0x5afa9e` - File extension: `.nspire` string literal - Contact emails: full addresses as string constants

Inferred — operational implication: static string extraction enables complete infrastructure mapping without dynamic analysis or decryption. YARA signatures can reliably target these embedded constants across builds.

2.4 File signature system prevents double-encryption

Observed: function `main.checkPossibility` @ `0x50f8c0` implements signature checking by seeking to file end and reading for "sNightspire" marker. Function `main.writeToTail` @ `0x50ffa0` appends this signature after encryption.

Observed: the signature check occurs before encryption logic in `main.startEncrypting`, creating early-exit path for files already processed.

Inferred — operational implication: the signature system enables identification of encrypted files and prevents encryption loops. The "sNightspire" marker is visible in hex editors at file tail, providing forensic pivot for encrypted file identification.

2.5 CLI configuration suggests automation

Observed: extensive command-line argument parsing in `main.main` with 7 distinct flags: - `-p <path>` : target specific directory - `-e <mode>` : encryption method toggle (0/1) - `-s <flag>` : AppData folder skip control - `-r <mode>` : ransom note placement method - `-t <seconds>` : throttling via sleep intervals - `-k <count>` : thread/goroutine count control - `-noreadme` : disable ransom note creation

Inferred — operational implication: the extensive CLI interface suggests deployment automation or affiliate model usage. The `-k` thread count parameter includes corruption warnings in help text, indicating operational risk awareness.

2.6 No anti-analysis or evasion techniques

Observed: binary analysis reveals no dedicated anti-analysis implementations: - No string obfuscation (all constants in cleartext) - No API hashing or import obfuscation - No debugging detection logic - No virtual machine detection - No process injection or hollowing - No persistence mechanisms - Standard Go runtime with unmodified entry point `_rt0_amd64_windows` @ `0x4749a0`

Inferred — operational implication: transparent implementation suggests confidence in cryptographic strength over evasion. Detection and analysis remain straightforward without anti-analysis circumvention requirements.

---

3. Code-level fingerprints

Stable artefacts observable across builds of this family:

Marker	Location in binary
File extension `.nspire`	String literal in .rdata section
Note filename pattern `/[NSPIRE_MSG].txt`	String literal at `0x5a24cb`
Attack banner `"~~~ You have been attacked by NightSpire.Team ~~~"`	Ransom note text in binary
Team identifier `"NightSpire.Team"`	Multiple string references
Contact email domain `@onionmail.org`	Both contact addresses use this domain
File signature `"sNightspire"`	Append marker in `main.writeToTail`
Go runtime entry `_rt0_amd64_windows`	PE entry point
Goroutine drive enumeration pattern A:-Z:	Loop logic in `main.main`
Volume API sequence `FindFirstVolumeW` → `FindNextVolumeW`	Drive discovery in main function
AES-256-CTR + RSA-4096-OAEP-SHA512 hybrid crypto	Go crypto library standard implementation
Three-tier intermittent encryption (100MB, 1.6TB thresholds)	Size constants `0x6400000` , `0x1900000000` in `main.getEncryptionType`
CLI flag pattern `-p` , `-e` , `-s` , `-r` , `-t` , `-k` , `-noreadme`	Argument parsing in main

Sample-specific (non-stable across builds):

Marker	Source
RSA-4096 public key (base64, ~550 chars)	Embedded at `0x5afa9e`
Chat credentials UUID `NSPIRE830NPH7ZLBRW39`	Specific to this sample
Chat credentials password `769FZisalII12Rph`	Specific to this sample

Observed: the chat credentials appear to be per-sample unique based on the specific UUID/password combination embedded in this binary.

---

4. Execution Timeline (T0 → encryption)

Observed sequential operations from code analysis:

```
T+0.0   Go runtime initialization (_rt0_amd64_windows entry point)
T+0.1   Command-line argument parsing (flag package)
T+0.2   Drive enumeration via FindFirstVolumeW/FindNextVolumeW sequence
T+0.5   Goroutine spawn (26 parallel workers, A: through Z: drives)
T+1.0   sync.WaitGroup initialization for goroutine coordination
T+2.0   Parallel directory walking begins (main.EncryptDir per goroutine)
T+3.0   File extension filtering and exclusion logic per file
T+5.0   File signature checking (main.checkPossibility)
T+10.0  AES key generation (main.GenerateRandomKey @ 0x50f480)
T+10.1  AES nonce generation (main.GenerateRandomNonce @ 0x50f500)
T+10.2  RSA key wrapping (main.EncryptRSA @ 0x50f7a0)
T+10.5  File encryption begins (main.encryptMethod @ 0x50f620)
        ├ Type 1: Sequential 1MB chunks
        ├ Type 2: First 1GB + distributed 1MB chunks
        └ Type 3: First 1GB + distributed 1GB chunks
T+N     File signature append (main.writeToTail)
T+N+0.1 Extension change to .nspire (main.changeFileName @ 0x510260)
T+N+0.2 Icon configuration (optional, main.setFileIcon @ 0x50eae0)
T+N+1.0 Ransom note creation (main.MakeReadMeFile @ 0x510860)
T+Final Windows Explorer refresh (main.refreshWindowsExplorer @ 0x50ef00)
```

Inferred timing: volume enumeration burst (T+0.2) provides earliest detection opportunity. Execution timing varies significantly based on filesystem size, file count, and storage performance.

---

5. TTP Highlights for Defender Workflow

5.1 Pre-encryption indicators (high precision, low FP)

Signal	Source	Time before encryption
Volume enumeration burst (>20 `FindFirstVolumeW` calls)	ETW Microsoft-Windows-Kernel-Process provider	T+0.2 (immediate)
Go runtime process with drive enumeration pattern	EDR process telemetry	T+0.5
Parallel file access across multiple drive letters	Sysmon Event 11 (FileCreate) aggregation	T+2.0
Mass file extension changes to `.nspire`	File system monitoring / FIM	T+N (during encryption)
Ransom note creation `/[NSPIRE_MSG].txt` pattern	Sysmon Event 11	T+N+1.0

Observed: the volume enumeration phase provides the most reliable early detection signal before file modification begins.

5.2 During-encryption indicators

Signal	Source
`.nspire` file extension creation events	EDR file telemetry
`"sNightspire"` signature append operations	File content monitoring (if available)
Go process memory containing AES keys	Memory forensics / EDR memory scanning
Windows Explorer refresh operations	Process monitoring
File association registry modifications	Registry monitoring

6. MITRE ATT&CK Matrix

Observed techniques in this binary:

Tactic	Technique	ATT&CK ID	Implementation
Execution	Native API	T1106	`FindFirstVolumeW`, `FindNextVolumeW`, `GetVolumeInformationW` via Go syscall
Discovery	File and Directory Discovery	T1083	Recursive directory enumeration across all Windows drives
Discovery	System Information Discovery	T1082	Volume enumeration and drive information gathering
Impact	Data Encrypted for Impact	T1486	AES-256-CTR + RSA-4096-OAEP-SHA512 hybrid encryption

Notably absent: - **T1055** (Process Injection) — No injection capabilities observed - **T1112** (Modify Registry) — Minimal registry interaction beyond file associations - **T1490** (Inhibit System Recovery) — No VSS deletion or backup removal - **T1562** (Impair Defenses) — No security tool disabling - **T1059** (Command Line Scripting) — Go binary, not script-based

7. Hunting Queries

7.1 Splunk SPL — Volume enumeration detection

```
index=sysmon EventCode=1
| where match(Image, "(?i).*\.exe$")
| where match(CommandLine, "(?i).*(FindFirstVolume|FindNextVolume|GetVolumeInformation).*") OR
match(ProcessName, "(?i).*go.*")
| stats count, values(CommandLine) by Computer, ParentImage, ProcessName
| where count >= 10
```

7.2 KQL — File extension mass change

```
DeviceFileEvents
| where Timestamp > ago(1h)
| where FileName endswith ".nspire"
| summarize NspireFiles = count(),
            FirstFile = min(Timestamp),
            LastFile = max(Timestamp),
            SamplePaths = make_set(FolderPath, 10)
            by DeviceName
| where NspireFiles >= 50
| extend EncryptionDuration = LastFile - FirstFile
```

7.3 KQL — Go runtime process detection

```
DeviceProcessEvents
| where Timestamp > ago(24h)
| where ProcessCommandLine contains "_rt0_amd64_windows"
     or InitiatingProcessCommandLine contains "go run"
     or ProcessVersionInfoOriginalFileName contains "go"
| join kind=leftouter (
    DeviceFileEvents
    | where FileName endswith ".nspire"
) on DeviceName
| where isnotnull(FileName)
```

8. Sigma Rules

```
title: NightSpire Volume Enumeration Pattern
id: a1b2c3d4-e5f6-7890-abcd-ef1234567890
status: experimental
description: |
  Detects NightSpire ransomware volume enumeration phase through high-frequency
  Windows volume API calls from a single process, preceding .nspire file creation
logsource:
  product: windows
  category: process_creation
detection:
  selection_volume_apis:
    EventID: 1
    Image: '*'
  condition: selection_volume_apis
falsepositives:
  - Disk management utilities
  - System maintenance tools
level: high
tags:
  - attack.discovery
  - attack.t1083

---
title: NightSpire File Extension Creation
id: b2c3d4e5-f6g7-8901-bcde-f23456789012
description: Mass creation of .nspire file extensions
logsource:
  product: windows
  category: file_event
detection:
  selection:
    EventID: 11
    TargetFilename: '*.nspire'
  condition: selection
level: critical
tags:
  - attack.impact
  - attack.t1486
```

9. Detection Confidence Matrix

Signal	Confidence	Evasion Difficulty	Notes
`.nspire` file extension	High	Low	Trivial to change extension
`"sNightspire"` file signature	High	Medium	Requires code modification
Volume enumeration burst (>20 API calls)	**High**	**High**	Core to drive discovery functionality
Go runtime entry point `_rt0_amd64_windows`	High	High	Fundamental to Go compilation
Embedded string constants (onion URLs, emails)	High	Low	Static analysis reveals easily
Three-tier encryption thresholds (100MB, 1.6TB)	High	Medium	Algorithm-core, harder to modify
CLI flag pattern `-p`, `-e`, `-s`, `-r`, `-t`, `-k`	Medium	Medium	Interface design choices
Chat credentials `NSPIRE830NPH7ZLBRW39` pattern	High	Low	Sample-specific, easily changed
Parallel goroutine architecture (26 workers)	High	High	Core to Go implementation efficiency
RSA-4096 + AES-256-CTR crypto stack	Medium	High	Standard Go crypto library usage

10. Memory Artefacts (DFIR / live response)

Observed artefacts present in running process memory:

Artefact	Location	Lifetime	DFIR value
AES-256 keys (32 bytes per file)	Go runtime heap, generated by crypto/rand	Per-file encryption operation	File decryption if captured during encryption
AES nonces (16 bytes per file)	Go runtime heap	Per-file encryption operation	Required for CTR mode decryption
RSA public key (base64 decoded)	Go string interning, crypto/rsa structures	Process lifetime	Sample identification pivot
Infrastructure strings (onion URLs, emails)	Go string heap	Process lifetime	Campaign attribution
File path queues	Go slice structures in main goroutines	Active directory walking	Target file identification
Volume enumeration results	Temporary Go structures	Brief (seconds)	Drive mapping reconstruction
CLI arguments	Process command line, Go flag package structures	Process lifetime	Operational mode identification

Live-response priority: if the process can be memory-imaged during encryption, individual file AES keys may be recoverable from the Go runtime heap, enabling decryption of files being processed at capture time.

Technical caveat: Go's garbage collector and memory layout make key recovery challenging compared to C/C++ implementations. Success depends on capture timing and heap analysis tools capable of parsing Go runtime structures.

---

11. Defender Priority Queue

P0 (immediate)

1. **Isolate** any host showing volume enumeration burst pattern or `.nspire` file creation
2. **Memory dump** active Go processes matching runtime signature before self-termination
3. **Block** outbound connections to `.onion` domains at network boundary
4. **Preserve** any `/[NSPIRE_MSG].txt` files for infrastructure intelligence

P1 (≤1 h)

1. **Hunt** for additional `.nspire` files across file shares and network storage
2. **Audit** volume enumeration patterns across SIEM for additional compromised hosts
3. **Collect** disk forensics images before encrypted file overwrite
4. **Monitor** network traffic for qTox protocol usage (backup communication channel)

P2 (≤24 h)

1. **Reconstruct** attack timeline using volume API call timestamps
2. **Correlate** file modification times to identify encryption sequence and duration
3. **Test** backup integrity and develop recovery procedures
4. **Update** detection signatures based on observed IOCs

P3 (post-incident)

1. **Analyze** encryption algorithm implementation for recovery possibilities
2. **Cross-reference** infrastructure IOCs with external threat intelligence
3. **Develop** long-term monitoring for NightSpire family evolution
4. **Document** lessons learned for incident response procedures

---

12. Observables Summary

Type	Value	Scope	Confidence
SHA-256	`69f5515ff3f554233840ad2f2397b345f955013017a9ae14ed4e762f52d936af`	Sample	High
MD5	`20cb8d8216061545b0b31ec8bd5f42de`	Sample	High
File extension	`.nspire`	Family	High
File signature	`sNightspire`	Family	High
Note filename pattern	`/[NSPIRE_MSG].txt`	Build	High
Team identifier	`NightSpire.Team`	Family	High
Primary onion	`nspire7lugml7ybqyjaaxtsgrs4qn3fcon3lrjbih6wamttvdm5ke4qd.onion`	Build	High
Leak site onion	`nspirep7orjq73k2x2fwh2mxgh74vm2now6cdbnnxjk2f5wn34bmdxad.onion`	Build	High
Email domain	`@onionmail.org`	Family	High
Contact email 1	`nightspire.team2026@onionmail.org`	Build	High
Contact email 2	`nightspireteam.receiver@onionmail.org`	Build	High
qTox ID 1	`038F61A270B8094E713E4815C4FA5086E4AD3A021575C6F90EE65A0C123D3E3BF6926C3B59EA`	Build	High
qTox ID 2	`8D663FD10BF662930F4C076CBF95FACFCC4ABD8F1A5E328DE75D0B0237A74E1AE1E0C5C37E7F`	Build	High
RSA public key (base64)	`LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0K...` (truncated)	Sample	High
Go compiler version	`1.24.11`	Build	High
Size thresholds	`0x6400000` (100MB), `0x1900000000` (1.6TB)	Family	High
Chunk sizes	`0x100000` (1MB), `0x40000000` (1GB)	Family	High
Volume API sequence	`FindFirstVolumeW` → `FindNextVolumeW` pattern	Family	High

13. Conclusion

Observed facts about this sample:

- Self-identified as NightSpire.Team via embedded strings (`.nspire` extension, team identifier in attack banner, contact emails, dual onion infrastructure for negotiation and data leak operations).
- Go 1.24.11 compilation with transparent string storage and no obfuscation techniques; standard Go runtime initialization via `_rt0_amd64_windows` entry point.
- Hybrid encryption: AES-256-CTR for file content with per-file key generation, RSA-4096-OAEP-SHA512 for key wrapping using embedded public key, intermittent encryption algorithm with three file-size tiers (100MB, 1.6TB thresholds).

- Parallel execution architecture: 26 goroutines for simultaneous drive processing (A: through Z:), configurable threading via CLI parameters, sync.WaitGroup coordination.
- File signature system: "sNightspire" marker appended to prevent double-encryption, signature checking in `main.checkPossibility` before processing.
- No lateral movement, no exfiltration code, no anti-analysis techniques, no persistence mechanisms — focused single-stage encryption payload.

Inferred: the implementation is consistent with modern ransomware development using Go for cross-platform compatibility and memory safety. The extensive CLI interface suggests deployment automation or affiliate distribution models.

Sample-specific artifacts (not portable to other builds): the RSA-4096 public key, chat credentials UUID/password pair, and qTox IDs embedded in this sample. Infrastructure elements (onion URLs, email addresses) may persist across builds but require validation per sample.