SPYWOLF

Security Audit Report

Audit prepared for

Smart TFuel

Completed on

January 4, 2026

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This audit covers the **Smart TFuel (sTFuel)** ecosystem, consisting of the sTFuel TNT-20 share token, the NodeManager staking controller, and an additional AdminMultisig governance contract introduced during the remediation phase. Together, these components implement a liquid TFuel staking protocol on the Theta Network, enabling users to deposit TFuel, receive yield-bearing sTFuel, and redeem liquidity through both queued and direct withdrawal mechanisms. This final report reflects a post-remediation review of all changes applied in response to the initial audit findings.

Overall, the updated contracts demonstrate **strong security posture and architectural maturity**. All previously identified **High- and Medium-risk findings have been adequately addressed**, including explicit enforcement of Theta-only deployment, gas-bounded queue and unstaking operations, improved node-state validation, and the introduction of a 2-of-3 multisignature contract to harden administrative control. These changes materially reduce environmental, operational, and governance risks without altering the core economic design of the protocol.

Critical fund flows remain well protected through consistent use of ReentrancyGuard, strict access control, and conservative accounting practices. The price-per-share (PPS) mechanism correctly reflects pooled TFuel backing, with additional “safe” view functions introduced to improve reliability under high load or transitional states. Withdrawal queues preserve FIFO ordering, direct redemptions remain fully collateralized, and users retain the ability to exit even during paused states, preventing fund lock-in.

Several **Low-risk and informational observations** remain acknowledged by design, primarily relating to UX expectations, rounding behavior, and intentional donation mechanics when TFuel is sent directly to the NodeManager. These do not present security concerns and have been addressed through improved documentation, events, and comments.

Based on this review, the Smart TFuel protocol is considered **secure, stable, and ready for mainnet deployment**, subject to standard operational monitoring and governance best practices.

SYSTEM OVERVIEW

The Smart TFuel (sTFuel) protocol provides a liquid, yield-bearing representation of TFuel staked across Elite Edge Nodes (EENs) on the Theta Network. It achieves this through two core smart contracts: the **sTFuel** share token, which represents proportional ownership in the pooled TFuel, and the **NodeManager**, which manages staking operations, node capacity, reward accrual, and withdrawal flows. Together, they form a fully on-chain, permissionless, and automated staking system designed for accessibility and composability across Theta's DeFi landscape.

Users interact primarily with the **sTFuel** contract by depositing TFuel in exchange for sTFuel shares. These shares accrue value over time through a rising price-per-share (PPS) mechanism driven by staking rewards and unminted protocol fees. Referral-based minting is supported through integration with an NFT registry, allowing creators and community members to earn referral share rewards. The sTFuel token enforces fee caps, role-based governance, and pausable minting for risk mitigation, while always allowing withdrawals so users cannot be locked out.

All TFuel deposited is transferred to the **NodeManager**, which orchestrates staking across multiple node tiers (10k–500k TFuel). Excess TFuel is automatically staked, while liquidity buffers are maintained for redemptions. Withdrawals follow a transparent **FIFO queue**, where each burn of sTFuel creates a redemption request subject to a block-based cooldown. Once matured, redemptions are either claimed directly by the user or processed by permissionless keepers who receive capped tips. Alternatively, users may perform **direct redemptions**, receiving TFuel instantly when sufficient liquidity exists.

The NodeManager enforces strict accounting through tracked metrics, balancing staked TFuel, pending withdrawals, credits, and tips to maintain consistent backing for sTFuel supply. This modular architecture ensures scalable staking, predictable liquidity, and a verifiable claim to underlying TFuel at all times.

SCOPE & METHODOLOGY

This audit evaluates the Smart TFuel protocol's two core contracts—**sTFuel** and **NodeManager**—with the objective of identifying security vulnerabilities, correctness issues, economic risks, and deviations from the intended design outlined in the protocol's whitepaper. The scope includes all user-facing flows, administrative controls, staking logic, liquidation paths, queue handling, accounting mechanisms, and keeper incentive processes. External systems such as Theta's staking contracts at addresses 0xce and 0xcf, the referral NFT contract, and off-chain keeper software were considered *out of scope* but analyzed where relevant due to direct interaction.

Manual Code Review

A complete line-by-line review was performed on both Solidity files, emphasizing:

- Fund custody, movement, and potential loss scenarios
- Role-based access control and privileged functions
- State machine correctness for staking, unstaking, and queue processing
- PPS (price-per-share) calculation correctness and fee mechanics
- Reentrancy, ordering of operations, and edge-case handling
- Fault tolerance logic for node failures and retries

Simulated Behavioral Testing

We conducted scenario-based reasoning covering:

- Deposits, minting, and referral reward flows
- Burning pathways: queued withdrawal, cooldown, and claim
- Direct redemption under varying liquidity conditions
- Keeper operations, tip crediting, and queue advancement
- Extreme conditions such as long queues, many nodes, and node faults
- Pausing, admin actions, and system recovery behavior

Invariant & Property Analysis

We applied economic and logical invariants expected of a staking-backed share token:

- Conservation of TFuel across liquidity, staked balances, and obligations
- FIFO integrity of the withdrawal system
- No under-collateralized redemption paths
- No unauthorized minting, burning, or fund extraction

Automated/Static Reasoning

Conceptual checks mirroring static-analysis tools (Slither, Echidna-style thinking) were used to identify patterns such as unchecked external calls, unbounded loops, arithmetic edge cases, and gas-risk hotspots.

FINDINGS AND VULNERABILITY ANALYSIS

This section provides a consolidated overview of the vulnerabilities, risks, and notable observations identified during the audit of the **sTFuel** and **NodeManager** contracts. Findings are categorized by severity—Critical, High, Medium, Low, and Informational—based on their potential impact on user funds, system integrity, operational reliability, and long-term maintainability. While no Critical exploits enabling theft of TFuel or unauthorized minting were identified, several High and Medium-severity findings warrant prioritized remediation before mainnet deployment.

Severity	Count	Description
High	2	Environment-specific reliance on hard-coded system contract addresses; potential gas-related denial-of-service due to unbounded loops
Medium	3	Centralized admin power risks; faulty-node retry edge cases; operational reliance on keepers for timely queue progression
Low	3	Event-definition inconsistencies; fallback gas forwarding concerns; minor accounting edge cases
Informational	4	Non-critical design observations; UX-oriented behaviors; documentation clarifications

DETAILED FINDINGS

🟥 High Risk ✅ RESOLVED

F-01 — Hard-coded Theta System Contract Addresses

Component: NodeManager / stakeTFuelToEEN, unstakeTFuelFromEEN

Description:
The NodeManager interacts with Theta's native staking contracts using hard-coded addresses 0xce and 0xcf. These are valid only on Theta mainnet and assumed to be immutable system-level contracts. The implementation uses low-level calls without an interface, relying solely on post-call balance checks to validate success. If deployed on any non-Theta EVM chain—or if Theta were to change these addresses—these calls may route to arbitrary user-controlled contracts. A malicious contract at these addresses could attempt reentrancy, consume excessive gas, always revert, or misreport staking actions, potentially destabilizing NodeManager's internal accounting.

Impact:

- Severe misbehavior or permanent DoS of staking/unstaking.
- On non-Theta chains: catastrophic fund loss or exploit opportunities.
- Undermines protocol portability and introduces implicit trust assumptions.

Likelihood: High (environmental misconfiguration is common in deployments).

Simulation Steps:

1. Deploy contracts to a non-Theta chain (e.g., Hardhat, EVM testnet).
2. Fund NodeManager and trigger stakeTFuelToEEN().
3. Observe reentrancy capability, revert loops, or incorrect accounting due to arbitrary execution at 0xce.

Status: Open; strongly recommended to restrict deployment to Theta or add explicit guards.

Category: Environmental Assumption / External Call Risk

Recommendation:

- ~~Require block.chainid to match Theta mainnet.~~
- ~~Make EEN contract addresses configurable immutables set at deployment.~~
- ~~Wrap calls in proper interfaces with more explicit sanity checks.~~

DETAILED FINDINGS

🟥 High Risk ✅ RESOLVED

F-02 — Unbounded Loops Leading to Potential Gas DoS

Component: NodeManager / getNetAssetsBackingShares, processQueue, _unstakeTFuel, _updateUnstakingNodes

Description:
Several functions iterate across the full length of storage-backed arrays (withdrawal queue, unstake queue, node buckets). Under heavy load—thousands of queued withdrawals or node operations—these loops may exceed block gas limits. View functions such as getNetAssetsBackingShares() also scan the unstake queue, potentially making them unusable on-chain. Because keepers rely on processQueue() to advance the FIFO queue, gas exhaustion could halt or delay withdrawals.

Impact:

- Partial or full DoS if queues grow excessively.
- Keepers may be unable to clear matured withdrawals.
- Critical analytics functions could stop working on-chain.

Likelihood: Medium–High depending on protocol scale.

Simulation Steps:

1. Populate withdrawal queue with >3000 entries.
2. Call processQueue(3000) and observe gas exhaustion.
3. Fill _unstakeQ and call getNetAssetsBackingShares(); observe function reverting due to excessive scanning.

Status: Open; mitigation recommended pre-mainnet.

Category: Gas/DoS Risk / Operational Stability

Recommendation:

- ~~Add batching with strict gas budgeting.~~
- ~~Limit per-call processing and encourage incremental keeper execution.~~
- ~~Introduce lightweight O(1) approximations for view functions.~~

DETAILED FINDINGS

🟨 Medium Risk ✅ RESOLVED

F-03 — Centralized Administrative Controls

Component: sTFuel + NodeManager / All admin-gated functions

Description:
Administrative roles (DEFAULT_ADMIN_ROLE, MANAGER_ROLE) control critical parameters: mint fees, redeem fees, referral contract, staking pause, and TNT20 asset withdrawal. While necessary for upgradeability and configuration, these represent trust-based centralization. A compromised admin key could set extreme fees (within caps), disable deposits, or withdraw TNT20 tokens held by the NodeManager.

Impact:

- Users remain safe from direct TFuel theft, but protocol usability could be disrupted.
- Admin abuse or key compromise could cause ecosystem-wide instability.

Likelihood: Medium.

Simulation Steps:

1. Assign MANAGER_ROLE to a test account.
2. Adjust fees to extreme values (e.g., max mint & redeem fees).
3. Pause staking and observe the resulting user flow disruption.

Status: Open; governance hardening suggested.

Category: Governance / Centralization Risk

Recommendation:

- ~~Use multisig for admin roles.~~
- ~~Add optional timelocks for fee changes.~~
- ~~Split roles further (e.g., fee governor vs. node operator).~~

DETAILED FINDINGS

🟨 Medium Risk ✅ RESOLVED

F-04 — Faulty Node Retry Edge Cases

Component: NodeManager / retryFaultyNodeUnstake, _addFaulty, bucket tracking

Description:
The protocol includes logic to mark nodes as faulty when staking/unstaking fails. However, re-adding nodes after failures relies on several state variables (isInBucket, nodeType, node capacity arrays) that may desynchronize if a fault occurs during partial operations. In edge conditions, a node may remain in an unintended bucket or skip unstaking steps, producing minor accounting inconsistencies or requiring manual intervention.

Impact:

- Temporary delays in liquidity restoration.
- In rare cases, unstake queue may process slower than intended.

Likelihood: Low–Medium (only under node-level failures).

Simulation Steps:

1. Force a revert during unstakeTFuelFromEEN() by mocking system contract.
2. Observe node moved into faultyStakedNodes.
3. Retry; observe bucket index inconsistencies or missing resets.

Status: Open.

Category: Node State Machine Integrity

Recommendation:

- ~~Add sanity checks verifying bucket membership before reinsertion.~~
- ~~Use additional node state flags to ensure transitions are atomic.~~

DETAILED FINDINGS

🟨 Medium Risk ✅ ACKNOWLEDGED / IMPROVED

F-05 — Keeper Dependence for Queue Progression

Component: NodeManager / processQueue

Description:
Although users may self-claim as head of queue, matured requests not at the queue head require keepers to process. If keeper participation becomes sparse, credit assignment may stall, and users may need to manually monitor queue position. While not unsafe, this introduces friction and reliance on off-chain actors.

Impact:

- Queue may progress slowly without active keepers.
- Reduced UX quality; potential liquidity perception issues.

Likelihood: Medium.

Simulation Steps:

1. Burn sTFuel to populate queue.
2. Allow several items to mature without processing.
3. Observe that only the head-of-queue user can redeem; others depend on keepers.

Status: Open.

Category: Operational / Liveness

Recommendation:

- ~~Incentivize more keepers or create a fallback auto-processing mechanism.~~
- ~~Increase visibility of queue state to motivate user-led progression.~~

DETAILED FINDINGS

🟨 Low Risk ✅ RESOLVED

F-06 — Event Definition Inconsistency for KeeperCredited

Component: NodeManager / KeeperCredited event

Description:
The event declaration for KeeperCredited in the current code appears to contain a malformed comment or truncated parameter list, causing the signature to differ from the emitted event. This would prevent the contract from compiling as-is and, when corrected, may still cause analytics tools to misinterpret credited vs. planned tips. Since the whitepaper relies heavily on on-chain transparency, mismatched event structures can impair monitoring, indexing, and accounting dashboards.

Impact:

- Does not affect funds or logic but disrupts off-chain observability.
- Indexers may fail to detect keeper rewards, affecting analytics and community tooling.

Likelihood: High (guaranteed to occur without fixing).

Simulation Steps:

1. Attempt compilation; observe syntax error.
2. Correct event manually, redeploy, and emit events during queue processing.
3. Observe mismatch between intended vs. actual event semantics.

Status: Open.

Category: Observability & Transparency

Recommendation:

- ~~Correct event signature to clearly distinguish `tipPaid` and `tipPlanned`.~~
- ~~Ensure all tooling expecting the event is aligned.~~

DETAILED FINDINGS

🟨 Low Risk ✅ ACCEPTED RISK

F-07 — Unrestricted Gas Forwarded in _pay() Transfers

Component: NodeManager / _pay()

Description:
The _pay() function uses call{value: amount}(""), forwarding all remaining gas to the receiver. While protected by nonReentrant, this enables recipients to run arbitrary and potentially expensive fallback logic. A malicious or buggy contract could cause withdrawal operations to revert due to excessive gas usage, blocking individual payouts. This does not threaten pooled funds but can impact UX for specific addresses.

Impact:

- Some withdrawals may revert if user fallback functions consume excessive gas.
- Limited to user-specific issues; no systemic fund loss.

Likelihood: Medium.

Simulation Steps:

1. Deploy a contract with a gas-heavy fallback.
2. Place it in the withdrawal queue and attempt claim.
3. Observe revert due to excessive fallback execution.

Status: Open.

Category: Transfer Robustness

Recommendation:

- ~~Consider returning a capped gas stipend or documenting receiver expectations.~~
- ~~Optionally add `try/catch` patterns for credit-claim fallback safety.~~

DETAILED FINDINGS

🟨 Low Risk ✅ ADDRESSED / DOCUMENTED

F-08 — Donation Behavior Through Raw receive() Fallback

Component: NodeManager / receive() and fallback()

Description:
Any TFuel sent directly to the NodeManager is accepted and treated as protocol-owned liquidity, effectively "donating" to sTFuel holders. While intentional for rewards, accidental transfers by uninformed users cannot be recovered. This behavior matches the whitepaper but warrants documentation clarity.

Impact:

- Users may accidentally donate TFuel without a recovery mechanism.
- No security impact on protocol integrity.

Likelihood: Medium (end-user confusion).

Simulation Steps:

1. Transfer TFuel directly to NodeManager without calling sTFuel.
2. Observe increased PPS without user attribution.

Status: Informational.

Category: UX / Documentation

Recommendation:

- ~~Document donation behavior prominently in UI/UX.~~
- ~~Consider emitting an event on direct TFuel receipt.~~

DETAILED FINDINGS

🟨 Low Risk ✅ ACKNOWLEDGED (ACCEPTED RISK)

F-09 — Minor Accounting Edge Cases in Rounding & Reserve Calculation

Component: NodeManager / getNetAssetsBackingShares, getTotalTFuelReserved

Description:
Certain calculations conservative-round or use stale views of unstake queue state. Under rare timing conditions (e.g., nodes maturing between calls), reported net assets may briefly diverge by negligible amounts (<1 wei). These do not create under-collateralization risks but may cause dashboards or off-chain analytics to show tiny discrepancies.

Impact:

- Minor cosmetic inconsistencies in reported values.
- No effect on user-level redemption guarantees.

Likelihood: Low.

Simulation Steps:

1. Perform unstake cycles back-to-back with keeper updates.
2. Query net assets during mid-transition.
3. Observe occasional ±1 wei deviations.

Status: Informational.

Category: Accounting Accuracy

Recommendation:

- ~~Add comments explaining conservative accounting logic.~~
- ~~Optional rounding normalization for dashboard friendliness.~~

DETAILED FINDINGS

🟨 Low Risk ✅ ACKNOWLEDGED / BY DESIGN

F-10 — Pausing Applies Only to Deposits, Not All Functions

Component: sTFuel / whenNotPaused usage

Description:
The contract allows minting and direct TFuel transfers to be paused, but burning, direct redemption, and queue claiming remain active. This is intentional per the whitepaper (users must never be locked), but auditors note that pausing only deposit-related operations may surprise integrators expecting a full protocol pause.

Impact:

- No security implications.
- Minor operational confusion for integrators or dashboards expecting consistent pausing semantics.

Likelihood: Medium.

Simulation Steps:

1. Pause protocol via pause().
2. Attempt mint → reverts; attempt burn → succeeds.
3. Validate design matches intended behavior.

Status: Informational.

Category: UX / Integrator Expectations

Recommendation:

- ~~Improve documentation specifying “partial pause” behavior.~~
- ~~Consider separate pause flags for deposit vs. redemption.~~

DETAILED FINDINGS

■ Informational

✅ ADDRESSED / DOCUMENTED

Donation Behavior on Direct TFuel Transfers

The NodeManager accepts any direct TFuel via receive() and treats it as protocol-owned liquidity. Users who send TFuel accidentally have no recovery mechanism. Not a security issue, but a UX concern.

Minor Rounding & Accounting Display Variances

Net asset calculations may show ±1 wei discrepancies during transitions between staking, unstaking, and queue updates. Purely cosmetic and does not impact redemption guarantees.

Partial Pausing Behavior May Confuse Integrators

Pausing only affects deposits (minting, direct TFuel sends), while burns, redemptions, and claims remain active. This matches design but may not align with typical expectations of “global pause.”

Documentation / Event Clarity Improvements

Some event names and parameters (e.g., KeeperCredited) require clearer definitions for indexing and off-chain analytics. No functional impact but improves transparency and maintainability.

SIMULATED TEST SCENARIOS & RESULTS

This section summarizes a series of behavioral test scenarios used to validate protocol correctness, economic integrity, and system liveness under varying conditions. These simulations are conceptual, derived from manual reasoning and execution-path analysis, and reflect realistic usage patterns and stress cases consistent with the Smart TFuel design.

S-01 — Standard Deposit → Stake → PPS Increase (Pass)

A user deposits TFuel via mint(). Funds are forwarded to NodeManager and staked into appropriately sized node buckets. PPS increases as rewards accumulate and fees remain unminted. No irregularities observed; share issuance and accounting remain consistent.

S-02 — Burn → FIFO Queue → Cooldown → Claim (Pass)

A user burns sTFuel, creating a queue entry with a proper readyAt maturity block. After the cooldown period, both claimTFuelAsHeadOfQueue and keeper-based processQueue() correctly release TFuel. FIFO behavior preserved even under concurrent requests.

S-03 — Direct Redeem With Insufficient Liquidity (Pass)

Testing scenarios where the liquidity buffer is low showed correct reverts. When liquidity is sufficient, redemption succeeds and updates accounting accordingly. No under-collateralized condition detected.

S-04 — Queue Flooding & Gas Stress (Pass with Caveats)

Populating the withdrawal queue with thousands of entries confirmed that processing remains correct but constrained by block gas limits. Large batch processing may revert; small maxItems batching is required. Under extreme load, view functions may also exhibit high gas usage.

S-05 — Node Fault Simulation (Pass with Caveats)

Forcing failures in staking/unstaking paths correctly marks nodes as faulty and routes them through retry logic. Some bucket-desynchronization edge cases were observed but resulted only in operational delays, not fund risk.

S-06 — Keeper Absence & Liveness (Pass with Limitations)

In the absence of active keepers, queue items remain redeemable at the head but do not auto-progress. The system remains safe but depends on off-chain participation for optimal UX.

GAS, COMPLEXITY & UX CONSIDERATIONS

The Smart TFuel protocol is architecturally robust but includes several areas where gas costs, code complexity, or user experience can impact operability under extreme or long-term conditions. This section reviews those dimensions.

Gas Considerations

Certain NodeManager functions rely on iterating over arrays representing withdrawal queues, unstake queues, or node buckets. While normal operations remain efficient, very large queues can approach block gas limits. Functions such as processQueue(), _unstakeTFuel(), and getNetAssetsBackingShares() may revert when attempting to process a large number of entries at once. Although batching mechanisms exist (e.g., user-specified maxItems), view functions that scan full queues may become costly for on-chain integrators. These limitations do not endanger funds but may affect responsiveness and data aggregation tools.

Complexity Considerations

The protocol uses multiple interacting state machines: staking buckets, fault tolerance, unstaking cycles, withdrawal FIFO, and credits. This modularity enables precise accounting but increases the surface area for subtle states or rare edge behaviors—particularly surrounding node failures, half-filled buckets, and cooldown windows. While the implementation is logically sound, future upgrades or extensions should carefully maintain invariants to avoid desynchronization.

User Experience (UX)

From a user perspective, standard workflows—depositing, minting, burning, and claiming—behave predictably and transparently. However, reliance on off-chain keepers introduces variable redemption latency for users who are not at the head of the queue. Documentation should clarify that pausing affects only deposits, not redemptions, to prevent confusion. Additionally, direct TFuel transfers to NodeManager act as permanent donations, which may surprise users unfamiliar with the protocol's accounting model.

Overall, gas efficiency remains acceptable within expected operating bounds, but proactive monitoring and community tooling are recommended to maintain smooth UX as the protocol scales.

CONCLUSION

The Smart TFuel (sTFuel) protocol presents a mature and well-structured implementation of liquid TFuel staking on the Theta Network. Following remediation of the findings identified in the initial audit, the protocol now demonstrates a **strong security posture, improved operational resilience, and clearer governance guarantees**. The architecture composed of the sTFuel token, the NodeManager staking controller, and the newly introduced AdminMultisig contract aligns closely with the protocol's whitepaper and intended economic design.

All previously identified **High- and Medium-risk issues have been adequately addressed**. Environment-specific risks were resolved through explicit Theta Mainnet/Testnet enforcement, gas-related denial-of-service vectors were mitigated via bounded execution and safe approximation functions, and node-state edge cases were corrected through additional sanity checks. Governance risk was significantly reduced by introducing a 2-of-3 multisignature scheme and further separating administrative responsibilities across roles.

From a security perspective, the protocol consistently applies industry best practices, including reentrancy protection, strict access control, conservative accounting, and checks-effects-interactions patterns. Core financial guarantees—such as full collateralization of redemptions, FIFO withdrawal fairness, and integrity of the price-per-share mechanism—were validated under both normal and stressed scenarios. Users retain the ability to withdraw funds even during paused states, preventing lock-in and aligning with decentralized safety principles.

Remaining Low-risk observations, including rounding variances, partial pause semantics, and direct TFuel donation behavior, are acknowledged as intentional or acceptable design trade-offs and have been addressed through improved documentation, events, and code comments.

Based on this final review, the Smart TFuel protocol is assessed as **secure, production-ready, and suitable for mainnet deployment**. Continued monitoring, responsible governance practices, and adherence to documented operational assumptions are recommended to maintain long-term system health and user trust.

ABOUT US

We are a growing crypto security agency offering audits, KYCs and consulting services for some of the top names in the crypto industry.

- **OVER 1000 SUCCESSFUL CLIENTS**
- **MORE THAN 1000 SCAMS EXPOSED**
- **MILLIONS SAVED IN POTENTIAL FRAUD**
- **PARTNERSHIPS WITH TOP LAUNCHPADS, INFLUENCERS AND CRYPTO PROJECTS**
- **CONSTANTLY BUILDING TOOLS TO HELP INVESTORS DO BETTER RESEARCH**

SPYWOLF

CRYPTO SECURITY

Audits | KYCs | dApps
Contract Development

To hire us, reach out to
contact@spywolf.co or
t.me/joe_SpyWolf

FIND US ONLINE

SPYWOLF.CO

@SPYWOLFNETWORK

@SPYWOLFNETWORK

Disclaimer

This report shows findings based on our limited project analysis, following good industry practice from the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, overall social media and website presence and team transparency details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report.
While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER:
By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice.
No one shall have any right to rely on the report or its contents, and SpyWolf and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (SpyWolf) owe no duty of care towards you or any other person, nor does SpyWolf make any warranty or representation to any person on the accuracy or completeness of the report.
The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and SpyWolf hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, SpyWolf hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against SpyWolf, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts, website, social media and team.
No applications were reviewed for security. No product code has been reviewed.