Smart Contract Audit Report

Prepared by: **Cybring**
Prepared for: **Nocturnal**

Version: 1.0 (final report)
Date: 15th September 2025
Commit/Hash: 18cf66afaccf644ef9e68d87151c9bd87a1354d2
Auditor: Anh Bui, Duc Cuong Nguyen

Contents

1 Introduction

This document presents the results of a security assessment of the `Nocturnal` Smart Contract. The engagement was commissioned by `Nocturnal` to obtain an independent evaluation of the on-chain logic that collects the trading fee from all trades on the `Nocturnal` system and initializes/updates the platform configuration.

1.1 Objective

This audit was conducted to assess the robustness, reliability, and security of the code base. The goal was to identify vulnerabilities, ensure compliance with best security practices, and provide mitigation measures. The initial audit report was provided by `CYBRING` on 11th September 2025.

After the initial audit report, a reassessment was conducted on 15th September 2025 to verify the status of the reported issues. While some issues were addressed, others were acknowledged and will be prioritized for future improvement.

1.2 Disclaimer

This security audit is not produced to replace any other type of assessment and does not aim to guarantee the discovery of all security issues within the scope of the assessment. Further, economic modeling, business-logic desirability, and market comparisons (e.g., target price ceilings relative to other launch platforms) were explicitly out of scope.

While this audit was carried out with good faith and technical proficiency, it's crucial to understand that no single audit can guarantee the absolute security of a smart contract. To minimize risks, `CYBRING` recommends a multi-faceted approach involving multiple independent assessments. Please note that this report is not intended to provide financial advice.

2 Scope of Audit

The audit was conducted on commit 18cf66afaccf644ef9e68d87151c9bd87a1354d2 from git repository `https://github.com/SL8-Pte-Ltd/nocturnal-smart-contract`. Further details regarding The `Nocturnal` audit scope are provided below:

- **Codebase details:**
  - Language: Rust
  - Frameworks: Anchor
  - Initial audit's commit hash: 18cf66afaccf644ef9e68d87151c9bd87a1354d2
  - Reassessment audit's commit hash: a917277cefe63052d28f1fb0cf9d3d6bb48f1fcf

3 Audit Summary

Our first pass identified 8 distinct issues across the `Nocturnal` program. The overview of the findings is presented in Table 1. There are no high or critical risks. Medium risks primarily relate to an insecure authority transfer mechanism, a potential griefing attack, and centralization risk tied to the admin key.

Severity	**Count**
Medium	2
Low	2
Informational	2
To be discussed	2

Table 1: Initial assessment summary

After code updates and design clarifications from the developers, we re-audited the patched build (c.f. Table 2).

Issue	**Severity**	**Status**	**Remediation Evidence**
5.1	Medium	Fixed	a917277
5.2	Medium	Accepted Risk (Operational)	
5.3	Low	Fixed	abd9471
5.4	Low	Design Choice	
5.5	Informational	Won't Fix	
5.6	Informational	Won't Fix	
5.7	Discussed (Informational)	Acknowledged	
5.8	Discussed (Informational)	Acknowledged	

Table 2: Final assessment summary

2 of 8 findings have been fixed. The rest with acknowledged statement have been confirmed to be intentional design.

4 Methodology

`CYBRING` conducts the following procedures to enhance security level of our client's smart contracts:

- **Pre-auditing:** Understanding the business logic of the smart contracts, investigating the deployment states of samples, and preparing for the audit.
- **Auditing:** Examining the smart contract by evaluating on multiple perspectives:
  - **Manual code review:** `CYBRING` auditors evaluate the static code analysis report to filter out false positive reports. Besides, inspect the smart contract logic, access controls, and data flows ensure the contract behaves as intended and is free from risky logic.
  - **Static code analysis:** By using advanced static analysis techniques combined with our customized detectors, we set out to identify potential vulnerabilities, optimize compute units, and ensure adherence to best practices in smart contract development.
  - **Fuzz testing:** `CYBRING` leverages fuzzing tools to stress-test the contract, ensuring it performs securely under unpredictable conditions.

- **First deliverable and consulting:** Presenting an initial report on the findings with recommendations for remediation and offering consultation services.
- **Reassessment:** Verifying the status of the issues and identifying any additional complications in the implemented fixes.
- **Final deliverable:** Delivering a comprehensive report detailing the status of each issue.

4.1 Risk Rating

The OWASP Risk Rating Methodology was applied to assess the severity of each issue based on the following criteria, arranged from the perspective of smart contract security.

- **Likelihood:** a measure of how likely this vulnerability is to be discovered and exploited by an attacker.
- **Impact:** a measure of the potential consequences or severity of a vulnerability if it is exploited by an attacker, including the extent of damage, data loss, or disruption of operations.

Table 3 shows the details of the security severity assessment for each issue.

Impact	*High*	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Informational	Low	Medium
		Low	*Medium*	*High*
			Likelihood	

Table 3: Overall Risk Severity

4.2 Audit Categories

- **Common vulnerabilities:** Smart contracts are analyzed following OWASP smart contract top 10.
- **Advanced vulnerabilities:** `CYBRING` simulates a certain types of attack scenarios to exploit the smart contracts. These scenarios were prioritized from high to low severity.
- **Security best practices:** The source code of the smart contract is analyzed from the development perspective, providing suggestions for improving the overall code quality.

4.3 Audit Items

Table 4 shows the details of the issues our auditors will conduct the audit upon.

Category	**Item**
Missing PDA validation	Missing verify user-supplied PDA
	Incorrect seeds used in derivation
	Missing bump seed validation
	Reusing same seed for multiple roles
Data Validation and Integrity	Integer overflow/underflow
	Misuse of *unwrap()* or *expect()*
	Unvalidated input (especially in CPI)
Exceeding compute budget and Denial of service (DOS)	Unbounded loops
	Exceeding compute unit limit with loops
	Exceeding compute unit limit with nested CPI
Access control & authority	Authority change without proper checks
	Upgrade authority mishandled
	Missing lock upgrade authority post-deployment
Private information and randomness	Store sensitive information on smart contracts
	Insecure randomness source
Timestamp dependence	Timestamp dependence on critical function
Best practices	Use of *checked_add*, *checked_sub*, etc.
	Separate validation logic from execution logic
	Follow standard style guide (naming conversion, program modularity, order of layout, etc.)

Table 4: Details of examined items

5 Detailed Findings

5.1 Insecure Authority Transfer Mechanism

- **Description:** The `transfer_authority()` lacks confirmation from the new authority, which can lead to loss of authority to any address.
- **Risk:** Medium (Impact: High, Likelihood: Low)
- **Affected code:**

```rust
    pub fn transfer_authority(ctx: Context<TransferAuthority>,
  new_admin: Pubkey) -> Result<()> {
        ctx.accounts.nocturnal_platform.admin = new_admin;
        Ok(())
    }

```

programs/nocturnal/src/lib.rs:25

- **Mitigation:** Change the authority transfer mechanism to include a pending state, with the transfer only completing upon confirmation from the new authority.
- **Status [15/09/2025]:** `Nocturnal` team has addressed this issue.

5.2 Potential Griefing Attack by Unrestricted Initialization

- **Description:** After deployment, the `initialize()` instruction can be called by any address. If the attacker claims admin authority by calling the instruction before the legitimate admin, it could lead to a loss of admin authority and a wasted smart contract deployment fee.
- **Risk:** Medium (Impact: High, Likelihood: Low)
- **CWE:** CWE-754: Improper Check for Unusual or Exceptional Conditions
- **Mitigation:** To prevent the issue, `Nocturnal` team should either pin the initial admin public key or use the program's upgrade authority as the initial admin key[1].
- **Status [15/09/2025]:** `Nocturnal` team has acknowledged this issue.

5.3 Potential Overflow on Computing Trading Fee

- **Description:** At the `charge_native_fee()` function, multiplying two high-value u64 numbers could lead to overflow. Although the overflow error is handled, it is still an issue when a very high transaction executes. For example, `trading_amount` = 1.8M SOL and `max_fee_rate` = 10000 (10%)
- **Risk:** Low (Impact: Medium, Likelihood: Low)

[1]https://github.com/solana-foundation/anchor/blob/master/lang/src/accounts/account.rs#L154

- **Affected code:**

```rust
    require!(trading_amount > 0, NocturnalError::ErrInvalidAmount);
    let fee_amount = trading_amount
        .checked_mul(ctx.accounts.nocturnal_platform.max_fee_rate)
        .ok_or(NocturnalError::ErrMathOverflow)?
        .checked_div(FEE_DECIMAL)
        .ok_or(NocturnalError::ErrMathOverflow)?;

```

programs/nocturnal/src/lib.rs:41

- **CWE:** CWE-190: Integer Overflow or Wraparound
- **Mitigation:** To resolve the issue, `trading_amount` and `max_fee_rate` should be cast to `u128` before multiplying. Then convert the result to `u64` by `try_into::<u64>()` with a bound check.
- **Status [15/09/2025]:** `Nocturnal` team has fixed this issue.

5.4 Missing Balance Check before Transfer

- **Description:** In the `charge_native_fee` instruction, if a user's lamport balance is insufficient to cover the transaction fee, the transaction will fail at the Solana runtime level. This results in a poor user experience, making it difficult to debug and preventing the protocol from providing a custom error message.
- **Risk:** Low (Impact: Medium, Likelihood: Low)
- **CWE:** CWE-754: Improper Check for Unusual or Exceptional Conditions
- **Mitigation:** Check user's balance before running `system_program::transfer()`.
- **Status [15/09/2025]:** `Nocturnal` team has acknowledged this issue.

5.5 Missing Event Emission

- **Description:** There is missing event emission on several instructions. It could prevent off-chain monitoring systems from accurately tracking critical state changes, potentially hindering timely detection of malicious activity or operational errors.
- **Risk:** Informational (Impact: Low, Likelihood: Low)
- **CWE:** CWE-778: Insufficient Logging
- **Mitigation:** Emit events for TransferAuthority, UpdateConfigs, and each ChargeNativeFee instruction.
- **Status [15/09/2025]:** `Nocturnal` team has acknowledged this issue.

5.6 Redundant Functions

- **Description:** All account data is already readable on-chain, making a separate function to get this information redundant and unnecessary.
- **Risk:** Informational (Impact: Low, Likelihood: Low)
- **CWE:** CWE-1164: Irrelevant Code
- **Mitigation:** Migrate `view_configs()` to off-chain systems.
- **Status [15/09/2025]:** `Nocturnal` team has acknowledged this issue.

5.7 Fee underpayment via untrusted amount

- **Description:** The function `charge_native_fee` calculates the fee using a caller-provided `trading_amount`. An adversary can pass a smaller amount than the actual trade (or a minimal lamport), causing the program to undercharge fees while completing the trade off-chain or via another CPI path. This is a trust-boundary mistake: the value that determines protocol revenue is not derived from an authoritative source.
- **Initial Risk:** To be discussed:
  - High if any external caller can invoke charge_native_fee and the fee is computed from a caller-supplied trading_amount that is not verified on-chain. Loss is unbounded over repeated calls and trivially automatable on Solana.
  - Informational only if the amount is cryptographically attested or derived on-chain from authoritative state (see Mitigations) and the current parameter cannot influence final fee.
- **Finalized Risk:** Informational.
- **CWE:** CWE-345: Insufficient Verification of Data Authenticity.
- **Mitigation:** Do not compute fees from caller-supplied amounts. Instead, ensure the fee basis comes from an authoritative source and that the fee is enforced atomically with the trade.
- **Status [15/09/2025]:** After the discussion, `Nocturnal` team has acknowledged this issue. Final risk is informational under the explicit assumption that fee enforcement is an off-chain policy of `Nocturnal`'s backend (non-trustless); users can avoid fees by bypassing the router, by design.

5.8 Centralization Risk

- **Description:** At the `Nocturnal` smart contract, the admin address holds the authority to perform critical, sensitive actions. If the admin's private key is compromised, or if the admin decides to act maliciously, the entire protocol and its user base are at risk.
- **Initial Risk:** To be discussed

  - Medium (Impact: High, Likelihood: Low) if there is no solution applied to prevent admin compromise.

- **Finalized Risk:** Informational
- **CWE:** CWE-440: Expected Behavior Violation
- **Mitigation:** Multi-sig wallet should be used for mitigating this issue as it requires multiple trusted parties to approve transactions, distributing control and making it much harder for a single point of failure to cause damage.
- **Status [15/09/2025]:** `Nocturnal` team has acknowledged this issue. Admin authority is held by a third-party multisig (e.g., Squads), with no governance logic inside the program. Security therefore depends on the multisig's threshold/policy and operators; we recommend publishing the admin/multisig policy and (optionally) enforcing a timelock for sensitive actions.