SatLayer Smart Contract Audit

audited by Asymptotic

Summary

In this audit, we at Asymptotic identified several issues, including one high-severity issue related to missing treasury cap validation, three low-severity issues covering incorrect cooldown checks, withdrawal time validation, and insecure coin metadata handling, and additional advisory-level findings addressing areas such as parameter validation, storage optimization, event handling, and transaction contexts. All reported issues have been remediated. Our verification also confirmed the vault's balance consistency and the correct usage of explicit assertions for input validation.

Legend

Issue severity

- **Critical** — Vulnerabilities which allow account takeovers or stealing significant funds, along with being easy to exploit.
- **High** — Vulnerabilities which either can have significant impact but are hard to exploit, or are easy to exploit but have more limited impact.
- **Medium** — Moderate risks with notable but limited impact.
- **Low** — Minor issues with minimal security implications.
- **Advisory** — Informational findings for security/code improvements.

Legal Disclaimer

Terms and Conditions and Liability

This report is subject to the terms and conditions—including liability limitations—established between Asymptotic and the paying entity. By sharing code with Asymptotic, developers acknowledge and agree to these conditions.

Scope

This security audit report focuses specifically on reviewing the Move smart contract code. The audit does not analyze or make any claims about other components of the system, including but not limited to:

- Frontend applications and user interfaces
- Backend services and infrastructure
- Off-chain components and integrations
- Deployment procedures and operational security
- Third-party dependencies and external services

Limitations

The findings and recommendations in this report are limited to the Move code implementation and its immediate interactions within the Sui blockchain environment.

Creating proofs for specifications is on a ”best effort” basis. We manual audited the code in instances where formal proofs were not technically feasible.

H-1: Missing Treasury Cap Validation in Vault Initialization

Severity: High

Description

The initialize_vault function accepts a TreasuryCap¡K¿ parameter for minting receipt tokens when users deposit tokens of type T into the vault. However, there is no validation to ensure that this treasury cap hasn't been previously used to mint tokens. This creates a vulnerability where an attacker could:

1. Pre-mint receipt tokens using the same treasury cap before vault initialization
2. Wait for legitimate users to deposit their T tokens into the vault
3. Exchange their pre-minted receipt tokens for the deposited T tokens
4. Effectively drain the vault, leaving legitimate users unable to withdraw their deposits

Recommendation

Implement validation checks in the initialize_vault function to ensure the total supply of receipt tokens (K) is zero at the time of vault initialization:

```
assert!(coin::total_supply(&receipt_treasury_cap) == 0, ...);
```

Remediation

✓ **Commit:** 75fc5ff

L-1: Incorrect Cooldown Check in withdraw Function

Severity: Low

Description

The withdraw function uses an incorrect comparison operator when validating if the cooldown period has passed. The current implementation uses a strict greater than (¿) operator, which could lead to users being unable to withdraw their funds at the exact moment when the cooldown period expires.

Recommendation

Replace the strict greater than (¿) operator with a greater than or equal to (¿=) operator in the withdrawal validation check to allow withdrawals exactly when the cooldown period expires:

assert!(clock.timestamp_ms() ¿= *vault.withdrawal_requests.borrow(ctx.sender()), EWithdrawAttemptedTooEarly);

Remediation

✓ **Commit:** 75fc5ff

L-2: Missing Withdrawal Time Validation

Severity: Low

Description

The withdrawal_time parameter, which defines the cooldown period between a withdrawal request and the actual withdrawal execution, lacks proper validation in both initialize_vault and update_withdrawal_time functions. Without appropriate bounds checking, the cooldown period could potentially be set to an unreasonably high value, effectively preventing users from accessing their funds for extended periods.

Additionally, setting a zero cooldown period would make the two withdrawal functions call redundant, so consider to forbid zero cooldown as well.

Recommendation

Implement a constant to define the maximum allowable cooldown period and add validation checks in both initialize_vault and update_withdrawal_time functions to enforce this limit.

Additionally, consider prohibiting a zero value and renaming the parameter to "withdrawal_cooldown" to improve clarity.

Remediation

✓ **Commit:** 75fc5ff

L-3: Insecure Coin Metadata Object Handling

Severity: Low

Description

Coin metadata in the lbtc module is handled insecurely during initialization. The module uses transfer::public_share_object(meta) instead of transfer::public_freeze_object() when creating a new coin currency. This allows the coin metadata to remain mutable, which could lead to unauthorized modifications of the coin's properties.

Recommendation

Replace transfer::public_share_object() with transfer::public_freeze_object() to ensure coin metadata immutability. Additionally, consider providing essential coin metadata parameters (name, description, and icon_url) during initialization for better coin identification and user experience.

If the coin's metadata is intended to be modifiable after initialization, implement proper access control to ensure only the authorized authority can make changes.

Remediation

✓ **Commit:** 75fc5ff

A-1: Multi-sig for Admin and VAdmin Caps

Severity: **Advisory**

Description

The system implements an AdminCap capability that grants administrative control over a pool. While the access control implementation is appropriate, the administrator has broad powers including:

1. Modify the withdrawal cooldown period
2. Enable/disable and adjust the total deposit cap
3. Pause all deposits and withdrawals

Additionally, there is a VAdminCap which controls versioning.

Though these administrative functions are necessary and well-implemented, the centralization of such significant control presents a security risk if the admin capability were to be compromised.

Recommendation

Implement decentralization measures to reduce centralization risks:

- If you are not already, use multi-sig for the admin cap
- Consider reducing administrative functionality where feasible
- Consider adding time-locks for sensitive operations

Remediation

✓ **Status:** Acknowledged

A-2: Suboptimal Storage Design in Withdrawal Tables

Severity: Advisory

Description

The current implementation uses two separate tables for managing withdrawals: withdrawal_requests and withdraw_amount, where user addresses are separately mapped to timestamps and amounts. This design creates unnecessary storage overhead and complexity when accessing related withdrawal data.

Recommendation

Consolidate the two tables into a single table that maps user addresses to a struct containing both timestamp and amount fields.

Furthermore, this structure could be enhanced in the future to support multiple withdrawal entries per user by using vector of withdrawal structs per user address. This would allow implementing partial withdrawals with separate cooldown periods for each withdrawal, providing greater flexibility for users.

Remediation

✓ **Commit:** 75fc5ff
Notes: The withdraw function can be optimized by removing the withdraw_info entry from the table before checking the cooldown period, rather than borrowing it first

A-3: Missing Public Getters for Withdrawal Information

Severity: Advisory

Description

The contract lacks public-getter functions to retrieve withdrawal-related information. This makes it difficult for UI applications and third-party integrations to access user withdrawal data such as timestamps and amounts.

Recommendation

Add public view functions to retrieve withdrawal information:

- Implement separate get_withdrawal_timestamp and get_withdraw_amount functions that allow retrieving a user's withdrawal request timestamp and pending withdrawal amount
- Alternatively, create a combined get_withdraw_info function that returns both pieces of information in a single call

Remediation

✓ **Commit:** 75fc5ff

A-4: Missing Parameter Change Validation

Severity: Advisory

Description

The set_staking_cap and update_withdrawal_time functions lack validation to prevent redundant calls where the new value matches the current value. This could lead to unnecessary state updates and gas consumption when parameters are set to their existing values.

Recommendation

Add validation checks to ensure the new parameter values differ from the current values before proceeding with updates:

```
assert!(vault.staking_cap != new_cap, EParamsUnchanged);
assert!(vault.withdrawal_time != new_time, EParamsUnchanged);
```

Remediation

✓ **Commit:** 75fc5ff

A-5: Inefficient Token Amount Calculation and Event Fields Redundancy

Severity: Advisory

Description

The deposit_for function uses an unnecessarily complex method to calculate the actual deposit amount by comparing vault balances before and after token joining.

Additionally, the DepositEvent emits both deposit_amount and receipt_token_minted fields containing identical values, creating unnecessary redundancy. The same issue applies to the WithdrawalRequest event, which contains two matching amount fields.

Recommendation

Simplify the actual_amount calculation by directly using coin::value(deposit_amount) instead of calculating the difference in vault balances and consolidate the redundant events fields into a single one.

Or if these are intended for future functionality, add explaining documentation.

Remediation

✓ **Commit:** 75fc5ff

A-6: Missing User Address in Event Emissions

Severity: Advisory

Description

The current implementation of DepositEvent, WithdrawEvent, and WithdrawalRequest events does not include the user's address as an event field. This omission creates difficulties in tracking user activity and auditing transactions, as there is no way to directly connect events to the users who initiated them.

Recommendation

Enhance all relevant events by adding a sender or user_address field containing the address of the user who initiated the transaction.

Remediation

✓ **Commit:** 75fc5ff

A-7: Consider Events During Admin Vault Configuration

Severity: Advisory

Description

The current implementation does not emit events when an admin configures vault parameters such as deposit caps or withdrawal cooldown or when an admin toggles vault pause status. This makes it difficult to track administrative actions for auditing purposes.

Recommendation

Implement dedicated events for all administrative vault configuration changes.

Remediation

✓ **Commit:** 75fc5ff

A-8: Unnecessary Mutable Transaction Context

Severity: **Advisory**

Description

Several functions including admin operations and queue_withdrawal accept a mutable transaction context parameter (ctx: &mut TxContext) despite not modifying it. This violates the principle of least privilege and creates misleading code.

Recommendation

Remove the mut keyword from the TxContext parameter in all functions that don't perform mutations on it. Change signatures to use &TxContext instead.

Remediation

✓ **Commit:** b02981a

Verification

Balance Consistency

Verifies that the vault's balance always equals the sum of receipt token supply and pending withdrawals, ensuring funds are properly accounted for.

About conditions

We verified that ‘assert‘s properly validate inputs such that code will only abort with explicit `assert`s.

See patch with formal verification specs in the Notion report

About the Auditor

Asymptotic provides a white-glove, formal-verification-based auditing service for Sui smart contracts.

- Analyze customer codebases to create mathematical proofs of security
- Use AI to accelerate specification writing, proof construction, and adjusting specs as code changes
- Focus on real-world security properties that matter for production systems

Asymptotic Team

Andrei Stefanescu

- Led verification of AWS cryptographic algorithms using SAW at Galois
- Created first Ethereum smart contract verification tool
- Three-time International Math Olympiad silver medalist
- PhD in Computer Science from UIUC with David J. Kuck Outstanding Thesis Award
- ACM SIGPLAN Distinguished Paper Award at OOPSLA 2016
- Pioneered K Framework for programming language semantics and verification

Cosmin Radoi

- Created GritQL, a language for large-scale code migration (backed by Founders Fund, 8VC)
- PhD in Computer Science from UIUC focusing on programming languages
- Multiple ACM SIGSOFT Distinguished Paper Awards at top conferences (ICSE, OOPSLA, FSE)
- IBM PhD Fellowship recipient and NSF SBIR Award winner
- Key contributor to K Framework for language semantics and verification
- Research expertise in parallelism, race detection, and performance optimization