hashlock.

Security Audit

SatLayer (DeFi)

April 2025

info@hashlock.com.au

Hashlock Pty Ltd

Table of Contents

CAUTION

THIS DOCUMENT IS A SECURITY AUDIT REPORT AND MAY CONTAIN CONFIDENTIAL INFORMATION. THIS INCLUDES IDENTIFIED VULNERABILITIES AND MALICIOUS CODE WHICH COULD BE USED TO COMPROMISE THE PROJECT. THIS DOCUMENT SHOULD ONLY BE FOR INTERNAL USE UNTIL ISSUES ARE RESOLVED. ONCE VULNERABILITIES ARE REMEDIATED, THIS REPORT CAN BE MADE PUBLIC. THE CONTENT OF THIS REPORT IS OWNED BY HASHLOCK PTY LTD FOR USE OF THE CLIENT.

Executive Summary

The SatLayer team partnered with Hashlock to conduct a security audit of their smart contracts. Hashlock manually and proactively reviewed the code in order to ensure the project's team and community that the deployed contracts are secure.

Project Context

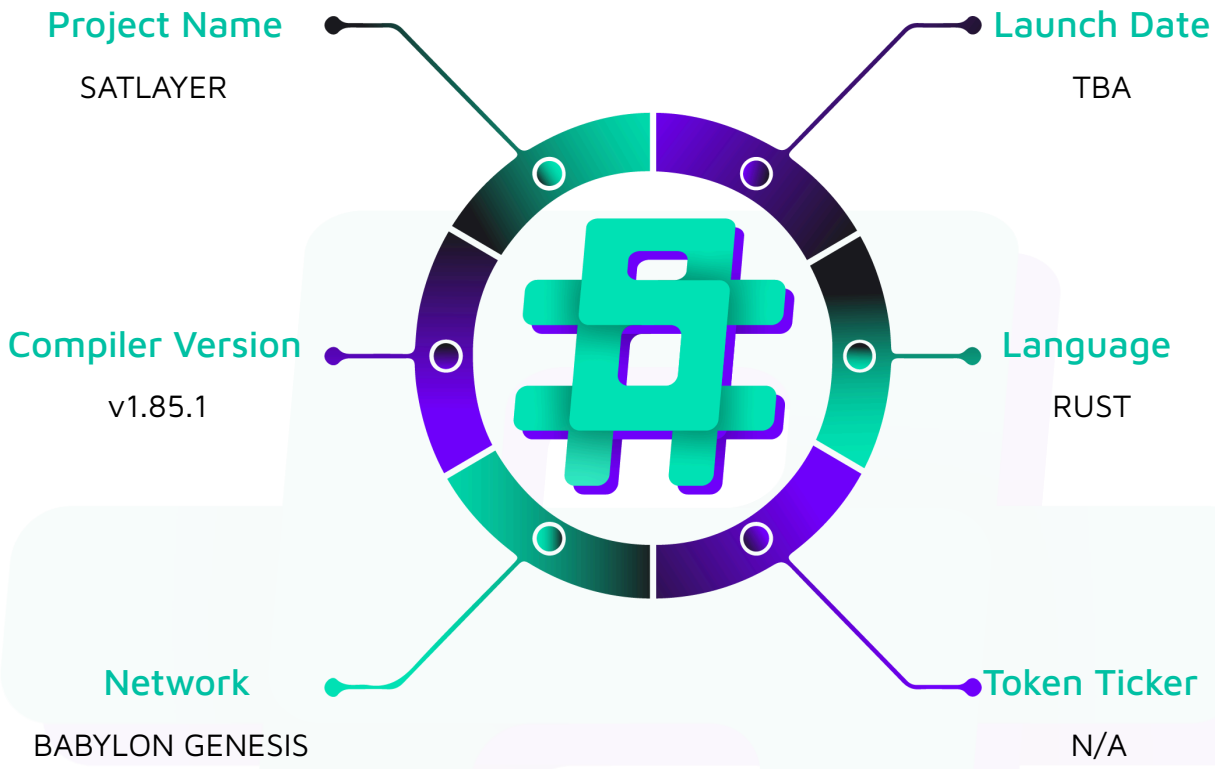
SatLayer is a shared security platform designed to leverage Bitcoin as the primary security collateral.

By deploying as a set of smart contracts on Babylon, SatLayer enables Bitcoin restakers to secure any type of decentralized application or protocol as a Bitcoin Validated Service (BVS). This means that Bitcoin, the largest and most widely trusted cryptocurrency, can now provide security to a broad range of applications, with full Turing-complete programmability while minimizing trust assumptions.

By expanding the ways Bitcoin can be used, SatLayer helps Bitcoin maximizes its potential as a reward-generating asset.

Project Name: SatLayer
Project Type: Defi
Compiler Version: v1.85.1
Website: https://satlayer.xyz/
Logo:

Visualised Context:



Project Visuals:

Audit scope

We at Hashlock audited the Rust code within the SatLayer project, the scope of work included a comprehensive review of the smart contracts listed below. We tested the smart contracts to check for their security and efficiency. These tests were undertaken primarily through manual line-by-line analysis and were supported by software-assisted testing.

Description	**SatLayer Protocol Smart Contracts**
Platform	**CosmWasm / Rust**
Audit Date	**April, 2025**
Github link	https://github.com/satlayer/satlayer-lst
Commit hash	73c8b49b0dd11f9a9f9325478c9c7ada145ca6ac
Component 1	lst_reward_dispatcher - contract.rs - lib.rs - state.rs
Component 2	lst_staking_hub - config.rs - constants.rs - contract.rs - error.rs - lib.rs - math.rs - msg.rs - query.rs - stake.rs - state.rs - unstake.rs
Component 3	lst_token - contract.rs - lib.rs - msg.rs - state.rs
Component 4	lst_validators_registry - contract.rs - lib.rs - state.rs

Component 5	lst_common - delegation.rs - errors.rs - hub.rs - lib.rs - msg.rs - rewards_msg.rs - signed_integer.rs - types.rs - validator.rs

Security Rating

After Hashlock's Audit, we found the smart contracts to be **"Secure"**. The contracts all follow simple logic, with correct and detailed ordering. They use a series of interfaces.

The 'Hashlocked' rating is reserved for projects that ensure ongoing security via bug bounty programs or on chain monitoring technology.

All issues uncovered during automated and manual analysis were meticulously reviewed and applicable vulnerabilities are presented in the Audit Findings section. The general security overview is presented in the Standardised Checks section and the project's contract functionality is presented in the Intended Smart Contract Functions section.

We initially identified some vulnerabilities that have since been addressed.

Hashlock found:

2 High severity vulnerabilities

2 Medium severity vulnerabilities

2 Low severity vulnerabilities

1 Gas Optimisations

2 QAs

Caution: *Hashlock's audits do not guarantee a project's success or ethics, and are not liable or responsible for security. Always conduct independent research about any project before interacting.*

Intended Smart Contract Functions

Claimed Behaviour	Actual Behaviour
lst_reward_dispatcher - Allows the hub contract to compute rewards and compound them. - Allows the owner to update the configuration.	**Contract achieves this functionality.**

lst_staking_hub
- Allows users to:
  - Stake tokens
  - Unstake tokens
  - Withdraw unstaked tokens
  - Perform slashing checks.
  - Update global indexes.
- Allows the owner to:
  - Update the parameters and configuration
  - Perform redelegations.
- Allows the rewards contract to compound rewards.
	Contract achieves this functionality.

lst_token - Allows users to perform general CW20 tokens interaction, such as sending tokens.	**Contract achieves this functionality.**

lst_token
- Allows the owner or hub contract to add validators.
- Allows the owner to:
  - Remove validators.
  - Update the configuration.
	Contract achieves this functionality.

Code Quality

This audit scope involves the smart contracts of the SatLayer project, as outlined in the Audit Scope section. All contracts, libraries, and interfaces mostly follow standard best practices and to help avoid unnecessary complexity that increases the likelihood of exploitation, however, some refactoring was required.

The code is very well commented on and closely follows best practice nat-spec styling. All comments are correctly aligned with code functionality.

Audit Resources

We were given the SatLayer project smart contract code in the form of Github access.

As mentioned above, code parts are well commented. The logic is straightforward, and therefore it is easy to quickly comprehend the programming flow as well as the complex code logic. The comments are helpful in providing an understanding of the protocol's overall architecture.

Dependencies

As per our observation, the libraries used in this smart contracts infrastructure are based on well-known industry standard open source projects.
Apart from libraries, its functions are used in external smart contract calls.

Severity Definitions

The severity levels assigned to findings represent a comprehensive evaluation of both their potential impact and the likelihood of occurrence within the system. These categorizations are established based on Hashlock's professional standards and expertise, incorporating both industry best practices and our discretion as security auditors. This ensures a tailored assessment that reflects the specific context and risk profile of each finding.

Significance	Description
High	High-severity vulnerabilities can result in loss of funds, asset loss, access denial, and other critical issues that will result in the direct loss of funds and control by the owners and community.
Medium	Medium-level difficulties should be solved before deployment, but won't result in loss of funds.
Low	Low-level vulnerabilities are areas that lack best practices that may cause small complications in the future.
Gas	Gas Optimisations, issues, and inefficiencies
QA	Quality Assurance (QA) findings are informational and don't impact functionality. Supports clients improve the clarity, maintainability, or overall structure of the code.

Status Definitions

Each identified security finding is assigned a status that reflects its current stage of remediation or acknowledgment. The status provides clarity on the handling of the issue and ensures transparency in the auditing process. The statuses are as follows:

Significance	Description
Resolved	The identified vulnerability has been fully mitigated either through the implementation of the recommended solution proposed by Hashlock or through an alternative client-provided solution that demonstrably addresses the issue
Acknowledged	The client has formally recognized the vulnerability but has chosen not to address it due to the high cost or complexity of remediation. This status is acceptable for medium and low-severity findings after internal review and agreement. However, all high-severity findings must be resolved without exception.
Unresolved	The finding remains neither remediated nor formally acknowledged by the client, leaving the vulnerability unaddressed.

Audit Findings

High

[H-01] lst_staking_hub#ExecuteMsg::Unstake - Users can steal available LST tokens from the contract

Description

The `execute_unstake` function in `contracts/lst_staking_hub/src/contract.rs:99` allows anyone to unstake LST from the contract. Unlike the `ExecuteMsg::Receive` handler, the `execute_unstake` function does not validate the caller to provide the LST tokens.

For example, if a user wants to unstake tokens in two transactions:

1. In the first transaction, the user sends the tokens to the contract.
2. In the second transaction, the user calls `ExecuteMsg::Unstake`.

In this case, the attacker can front-run the second transaction to unstake on behalf of the user, effectively stealing their funds.

Impact

Users' LST tokens will be stolen by attackers.

Recommendation

Consider removing the `ExecuteMsg::Unstake` handler and let users unstake via the `ExecuteMsg::Receive` entry point.

Status

Resolved

[H-02] lst_staking_hub#receive_cw20 - Users' LST tokens are lost if called from Cw20HookMsg::UnStake

Description

The `execute_unstake` function in `contracts/lst_staking_hub/src/contract.rs:272` is called with the `sender` parameter as `info.sender`. This is incorrect because the caller is the LST token contract instead of the actual caller (`cw20_msg.sender`).

Impact

Users' LST tokens will be stuck because they will be sent to the LST token contract, which cannot be withdrawn.

Recommendation

Consider updating the `sender` parameter to `cw20_msg.sender`.

Status

Resolved

Medium

[M-01] lst_staking_hub#execute_redelegate_proxy - Owner can redelegate funds to non-registered validators

Description

The `execute_redelegate_proxy` function allows the owner to redelegate funds to any validators by specifying it in the `redelegations` parameter. However, no validation exists to ensure the validators redelegated to are registered by the `VALIDATOR_REGISTRY` state in the `lst_validators_registry` contract.

Impact

Funds could be redelegated to non-registered validators, which will not be included in the `query_validators` function, causing inconsistencies in the `lst_validators_registry` contract.

Recommendation

Consider adding validation to ensure the destination validators are registered in the `VALIDATOR_REGISTRY` state before performing the redelegation.

Status

Resolved

[M-02] lst_validators_registry#add_validator - Hub contract cannot add validators

Description

The `add_validator` function allows the hub contract to add new validators to the `VALIDATOR_REGISTRY`. However, the hub contract itself does not implement any entry points to perform this action.

Impact

The hub contract cannot add validators as expected.

Recommendation

Consider implementing the required entry points in the hub contract or only allowing the owner to add validators.

Status

Resolved

Low

[L-01] Contracts - Two-step ownership transfer is not implemented

Description

In a few instances of the codebase, the owner can be updated in a transaction. This is problematic because if the ownership is set to an incorrect address, the ownership will be lost.

This issue affects the following codelines:

- `contracts/lst_reward_dispatcher/src/contract.rs:89`
- `contracts/lst_staking_hub/src/config.rs:30`
- `contracts/lst_validators_registry/src/contract.rs:178`

Recommendation

Consider implementing a two-step ownership transfer: the first transaction occurs by nominating an admin, and the second transaction requires the nominated admin to accept the ownership transfer.

Status

Acknowledged

Note

SatLayer's team informed Hashlock that the contract admin (a CosmWasm feature) has the omni-ability to override by migration, which makes the two-step ownership transfer mostly redundant. Ownership can be overridden using the CosmWasm migrate entry point. The Contract Admin is a feature of wasmd and is set during contract instantiation, not within the contract itself. It cannot be changed after that.
Refer to:
https://github.com/CosmWasm/cosmwasm/blob/dc70bf3554f63308f68109881985477 3355b60cb/packages/std/src/query/wasm.rs#L57

[L-02] lst_token#instantiate - Contract name and version will be overwritten

Description

During the instantiation phase, the `set_contract_version` is called to update the contract name to `crates.io:satlayer-lst` and version to `0.1.0`.

However, this will be overwritten in the `cw20_init` function to be `crates.io:cw20-base` and `2.0.0` in:

https://github.com/CosmWasm/cw-plus/blob/v2.0.0/contracts/cw20-base/src/contract.rs#L100

Impact

The registered contract version and contract name will be incorrect.

Recommendation

Consider calling the `set_contract_version` function after the `cw20_init` function.

Status

Resolved

Gas

[G-01] contracts/lst_common/src/delegation.rs - Iteration can be skipped when target_delegation equals val_current_delegation

Description

In line `34`, if the `target_delegation` equals `val_current_delegation`, the iteration continues to set `delegations[index]` to zero. This is unnecessary because it will eventually be skipped in `contracts/lst_staking_hub/src/stake.rs:102` and `contracts/lst_validators_registry/src/contract.rs:128`.

Recommendation

Consider skipping the iteration if `target_delegation` equals `val_current_delegation`:

```
if target_delegation <= val_current_delegation {

}
```

Status

Acknowledged

Note

SatLayer's team confirmed that the iteration can be skipped if the amount to distribute is zero. However, since one validator reaching the target delegation amount does not mean all validators have reached their respective amounts, the iteration should not be skipped if the distribution amount is above zero.

QA

[Q-01] contracts/lst_reward_dispatcher/src/contract.rs - Unneeded is_paused validation

Description

The execute_dispatch_rewards function returns an error if the hub is paused via the is_paused function. This validation is unnecessary because the hub can only call this function, which already implements the pause validation in contracts/lst_staking_hub/src/contract.rs:91.

Recommendation

Consider removing the `is_paused` function validation.

Status

Acknowledged

Note

SatLayer's team stated that this validation will be removed in future versions of the contracts.

[Q-02] contracts/lst_staking_hub/src/stake.rs - Unneeded unwrap_or_default() function

Description

The query_total_lst_token_issued function is called with the unwrap_or_default() function. If the function fails, the total_supply will be set to zero, which may cause unintended consequences.

Recommendation

Since this function is not supposed to fail, consider changing it to use unwrap() directly.

Status

Resolved

Centralisation

The SatLayer project values security and utility over decentralisation.

The owner executable functions within the protocol increase security and functionality but depend highly on internal team responsibility.

Conclusion

After Hashlock's analysis, the SatLayer project seems to have a sound and well-tested code base, now that our vulnerability findings have been resolved and acknowledged. Overall, most of the code is correctly ordered and follows industry best practices. The code is well commented on as well. To the best of our ability, Hashlock is not able to identify any further vulnerabilities.

Our Methodology

Hashlock strives to maintain a transparent working process and to make our audits a collaborative effort. The objective of our security audits is to improve the quality of systems and upcoming projects we review and to aim for sufficient remediation to help protect users and project leaders. Below is the methodology we use in our security audit process.

Manual Code Review:

In manually analysing all of the code, we seek to find any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behaviour when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our methodologies include manual code analysis, user interface interaction, and white box penetration testing. We consider the project's website, specifications, and whitepaper (if available) to attain a high-level understanding of what functionality the smart contract under review contains. We then communicate with the developers and founders to gain insight into their vision for the project. We install and deploy the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We undergo a robust, transparent process for analysing potential security vulnerabilities and seeing them through to successful remediation. When a potential issue is discovered, we immediately create an issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is vast because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyse the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the contract details are made public.

Disclaimers

Hashlock's Disclaimer

Hashlock's team has analysed these smart contracts in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in the smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Hashlock is not responsible for the safety of any funds and is not in any way liable for the security of the project.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to attacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

About Hashlock

Hashlock is an Australian-based company aiming to help facilitate the successful widespread adoption of distributed ledger technology. Our key services all have a focus on security, as well as projects that focus on streamlined adoption in the business sector.

Hashlock is excited to continue to grow its partnerships with developers and other web3-oriented companies to collaborate on secure innovation, helping businesses and decentralised entities alike.

Website: hashlock.com.au
Contact: info@hashlock.com.au