SALUS SECURITY

AUG 2024

CODE SECURITY ASSESSMENT

SATLAYER

Overview

Project Summary

- Name: Satlayer
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/satlayer/deposit-contract
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Satlayer
Version	v2
Type	Solidity
Dates	Aug 15 2024
Logs	Jul 26 2024; Aug 15 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	2
Total informational issues	1
Total	3

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction **4**
1.1 About SALUS 4
1.2 Audit Breakdown 4
1.3 Disclaimer 4
Findings **5**
2.1 Summary of Findings 5
2.2 Notable Findings 6
1. Not applicable for fee on transfer token 6
2. Centralization risk 7
2.3 Informational Findings 8
3. Redundant code 8
Appendix **9**
Appendix 1 - Files in Scope 9

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Not applicable for fee on transfer token	Low	Business Logic	Resolved
2	Centralization risk	Low	Business Logic	Mitigated
3	Redundant code	Informational	Redundancy	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Not applicable for fee on transfer token

Severity: Low	Category: Business Logic
Target: - src/SatlayerPool.sol	

Description

There are ERC20 tokens that charge a fee for each transfer() or transferFrom(), for example the PAXG token. As a result, when a token transfer occurs, the recipient's balance will be lower than expected.

src/SatlayerPool.sol:L55-L65

```
function depositFor(address _token, address _for, uint256 _amount) whenNotPaused
external {
    if (_amount == 0) revert DepositAmountCannotBeZero();
    if (_for == address(0)) revert CannotDepositForZeroAddress();
    if (!tokenAllowlist[_token]) revert TokenNotAllowedForStaking();
    if (capsEnabled && caps[_token] < getTokenTotalStaked(_token) + _amount) revert
CapReached();

    emit Deposit(++eventId, _for, _token, _amount);
    ReceiptToken(tokenMap[_token]).mint(_for, _amount);
    IERC20Metadata(_token).safeTransferFrom(msg.sender, address(this), _amount);
}
```

If the user enters the protocol with this type of token, the call to router in the deposit() function will revert because the contract has less than desc.amount of tokens at that point.

Recommendation

Consider calling balanceOf() to get the actual balances.

Status

The team has resolved this issue in commit 174f45e.

2. Centralization risk

Severity: Low	Category: Centralization
Target: - src/SatlayerPool.sol	

Description

In the `SatlayerPool.sol`, there is a privileged owner role. This role has the ability to:

- set caps enabled
- set migrator
- add token
- set token staking params
- pause/unpause the contract

If an attacker were to gain access to the private keys associated with this role, they could set a migrator address to steal users' tokens..

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been mitigated by the team. The team has stated that they will use a timelock governor as the owner.

2.3 Informational Findings

3. Redundant code

Severity: Informational	Category: Redundancy
Target: - src/SatlayerPool.sol	

Description

Unused code should be removed before deploying the contract to mainnet. We have identified the following variables are not being utilized:

src/SatlayerPool.sol:L20,L27

```
bytes32 private constant MIGRATE_TYPEHASH =
        keccak256("Migrate(address user,address migratorContract,address
destination,address[] tokens,uint256 signatureExpiry,uint256 nonce)");

mapping(address => address) public reverseTokenMap;
```

Recommendation

Consider removing the redundant codes.

Status

The team has resolved this issue in commit 174f45e.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 9f7c49a:

File	SHA-1 hash
ReceiptToken.sol	e84b0c49c087c1f2a00014a9d9bad66e93e298c8
SatlayerPool.sol	e42c8e5d30930fee634eb2f88762fb01ad4d4e89