Security Assessment Report

Glow LRT

December 2024

Prepared for Blueprint-Finance

certora Glow Finance

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Glow LRT	https://github.com/Glow-Finance/glow-lrt	e38d8bc	Solana

Project Overview

This document describes the findings of the manual review of **Glow LRT**. The work was undertaken from Dec 2 to Dec 6, 2024

The following contract list is included in our scope:

- `programs/glow-lrt/src/contexts/deposit.rs`
- `programs/glow-lrt/src/contexts/initiate_withdraw.rs`
- `programs/glow-lrt/src/contexts/execute_withdraw.rs`
- `programs/glow-lrt/src/contexts/calculate_sol.rs`
- `programs/glow-lrt/src/contexts/calculate_ssol.rs`
- `programs/glow-lrt/src/contexts/initialize.rs`
- `programs/glow-lrt/src/lib.rs`
- `programs/glow-lrt/src/contexts/set_multi_sign_wallet.rs`
- `programs/glow-lrt/src/err.rs`
- `programs/glow-lrt/src/contexts/pause.rs`
- `programs/glow-lrt/src/event.rs`
- `programs/glow-lrt/src/contexts/transfer_admin.rs`
- `programs/glow-lrt/src/contexts/update_cap.rs`
- `programs/glow-lrt/src/contexts/mod.rs`
- `programs/glow-lrt/src/utils.rs`
- `programs/glow-lrt/src/state/lrt_pool.rs`
- `programs/glow-lrt/src/state/lrt_withdraw_request.rs`
- `programs/glow-lrt/src/calcualtor_trait.rs`
- `programs/glow-lrt/src/enums.rs`
- `programs/glow-lrt/src/state/mod.rs`

Protocol Overview

This protocol allows users to deposit SOL or staked SOL (SSOL) and receive a liquid restaking token (gSOL) representing their share of the staked assets. The protocol manages input token pools (SOL/SSOL) and output token pools (GSOL). Deposits mint LRT tokens, while withdrawals burn LRT tokens to redeem the equivalent SOL. A multisig wallet receives the deposited assets, stakes them to generate yield, and redistributes funds back to the pools to fulfill withdrawal requests.

Findings Summary

The table below summarizes the review's findings, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	1	1	1
Medium	2	2	2
Low	1	1	1
Informational	3	3	3
Total	7	7	7

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

Detailed Findings

ID	Title	Severity	Status
H-01	Potential for First Depositor (Share Inflation) Attack	High	Fixed
M-01	Frontrunning Vulnerability on Pool Initialization	Medium	Fixed
M-02	Free Stake Pool Functionality	Medium	Fixed
L-01	Validation Function Disabled	Low	Fixed
I-01	Signed account aren't tracked for changes, even though they might be owned by the margin program	Info	Fixed
I-02	Liquidator can DoS liquidation by repeatedly registering themselves as the liquidator and not doing anything	Info	Fixed
I-03	Exchange rate might be zero if only uncollected fees remain in the pool	Info	Fixed

High Severity Issues

H-01 Potential for First Depositor (Share Inflation) Attack

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: calculate_sol.rs	Status: Fixed	

Description:

If the first depositor initializes the `pool` with minimal shares, they can inflate the value of shares by directly transferring SOL/SSOL into the `pool`/`multisig`. This reduces the shares granted to subsequent depositors, allowing the attacker to benefit disproportionately when redeeming shares for assets.

Exploit Scenario:

When the initial supply is 0: the shares amount returned is equivalent to an asset:

```
File: calculate_sol.rs
82:     fn calculate_shares(&self, assets: u64) -> Result<u64> {
83:         let (sol_assets, supply) = self.assets_and_supply()?;
84:         if supply == 0 {
85:             return Ok(assets);
86:         }
```

Here, any arbitrary user can be the first depositor.
An attacker can deposit 1 lamport of assets to set shares to 1.
In the `assets_and_supply` function, the SOL balance is directly used to compute the amount of assets, therefore the attacker can directly transfer (donate) sol to the `pool` or the multisig:

```
File: calculate_sol.rs
59:         // sol assets
60:         let mut sol_assets = self.pool.get_lamports();
61:         sol_assets += self.multi_sign_wallet.get_lamports();
```

In the subsequent transactions, this will make `sol_assets` much bigger than `supply` (== 1). Given that the share's calculation is divided by the SOL assets (balance of native SOL), a sandwiched user can have their shares rounded down to 0 (the program allows for 0 shares being minted):

```
File: calculate_sol.rs
88:         // glow sol amount = (assets * supply) / sol_assets
89:         let shares = (assets as u128)
90:             .checked_mul(supply as u128)
91:             .and_then(|x| x.checked_div(sol_assets as u128))
92:             .ok_or(LRTPoolError::Overflow)?;
```

Afterward, the attacker can redeem their 1 share worth of an inflated amount of assets (amounting to their first deposit + the deposit of the victim who minted 0 shares for their assets).

Recommendations:
Several fixes exist on the subject with their pros and cons. Here are some good resources on the subject:

- https://www.euler.finance/blog/exchange-rate-manipulation-in-erc4626-vaults
- https://blog.openzeppelin.com/a-novel-defense-against-erc4626-inflation-attacks
- https://docs.openzeppelin.com/contracts/4.x/erc4626#inflation-attack

Blueprint Finance's response: We will set the first depositor to be our admin and prevent zeros for assets and shares on deposits and withdrawals. Fixed in 8110ea8.

Fix Review: It's important to note that while this does mitigate the attack to some extent:

- It is still somehow feasible for the second depositor if the first deposit is too small.
- If the deposit is too big, this creates an imbalance in the protocol that can be perceived as unfair. The admin would hold a disproportionate amount of shares that can be redeemed

for a large portion of assets later. In the case of dead shares, no one can claim these assets.

But again, the solution does make the attack more difficult and comes with its own pros and cons.

Blueprint Finance's response 2: Indeed, it seems there is no perfect solution to it, but setting the first depositor to admin would always make it better.

Fix Review: Acknowledged.

Medium Severity Issues

M-01 Frontrunning Vulnerability on Pool Initialization

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: initialize.rs	Status: Fixed	

Description: Attackers can precompute the `pool` PDA address based on predictable seeds and initialize it with themselves as `signer`, making them the pool's admin. The `init` keyword ensures subsequent transactions will revert, locking out legitimate initializers.

Recommendations: Ensure that the `pool` is initialized on deployment.

Blueprint Finance's response: We will always combine the instructions (glowSOL token creation, set its mint authority, pool initialization) in one tx, so there would be no problem with the frontrun.

Fix Review: Trusted and managed by the team.

M-02 Free Stake Pool Functionality		
Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: initialize.rs initiate_withdraw.rs execute_withdraw.rs	Status: Fixed	

Description:
Since `withdraw_waiting_period` is a configurable parameter rather than a fixed constant, it could be set to a very low value or even be set to zero, allowing withdrawals to occur faster than the standard unstaking period for SSOL (around 2 days). This enables the protocol to function in an unintended way as a Stake Pool for withdrawals of unstaked SOL without any fees, provided the pool's SOL balance is sufficient to fulfill the withdrawal requests.

Blueprint Finance's response: Our withdrawal waiting period is 7 days. During this window, Glow's fund manager can decide how much SOL or SSOL to redeem from Solayer. Additionally, as a last-minute decision, we're removing the withdrawal of SSOL.

Fix Review: `withdraw_waiting_period` trusted as being set to 7 days by the team.

Low Severity Issues

L-01 Validation Function Disabled

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: lib.rs	Status: Fixed	

Description: The `validate_stake_pool_accounts` function in the deposit and withdraw workflows is currently commented out. This function is intended to ensure the integrity of the stake pool accounts used in the protocol.

Recommendation:
Uncomment the `validate_stake_pool_accounts` function to ensure proper validation of stake pool accounts in the respective workflows.

Blueprint Finance's response: We commented on it for testing. The code is for solayer pool address validation. Fixed in bb93442.

Informational Severity Issues

I-01. init_if_needed on Pool Input Vault During Withdrawals

Description: The `init_if_needed` directive for `pool_input_token_vault` in the `InitiateWithdraw` context is redundant. The vault is already created during deposits, and initiating its creation here is unnecessary and can lead to inefficiencies or confusion.

Blueprint Finance's response: It's initialized during deposit but we added the double checking when withdrawing. Fixed in 8110ea8.

I-02. Redundant has_one Constraints

Description: The `has_one` constraints on `withdraw_request` and `pool` are redundant because the attributes of the `seeds` already enforce the same relationships. These constraints can be safely removed without impacting security or functionality.

Blueprint Finance's response: Fixed in 8110ea8.

I-03. Typo in File Name

Description: The file `calcualtor_trait.rs` contains a typo in its filename. For consistency and clarity, it should be renamed `calculator_trait.rs`.

Blueprint Finance's response: Fixed in 8110ea8.

Appendix 1: Late-Stage Adjustment Evaluation

Some modifications were introduced to the codebase in the commit hash fa463a4f:

1. Making the withdrawal period configurable
2. Adding a restriction to the minimum SOL balance in the pool to avoid contract closing due to not enough SOL to cover the rent

Below is Certora's review.

All of the fixes from the Blueprint Finance team were also reviewed by Certora and can be found at the following commit hash: b1ed919a

R-01 : execute_withdraw.rs#L126

Certora: We understand that this change intends to force only new and migrated withdrawal requests to pass. About this, we have 2 hardening suggestions:

1. Consider enforcing `withdraw_waiting_period > 0` inside `initialize.rs:initialize()`.
2. Consider adding a fallback calculation to actually look at the `pool.withdraw_waiting_period` if `withdraw_request.withdraw_waiting_period == 0` so as not to block users. Of course, this would be temporary code (migration-phase code).

Blueprint Finance's response: We will optimize this.

R-02 : execute_withdraw.rs#L139

Certora: We understand that users can always call `initiate_withdraw` with 0.1 SOL less, but if all users withdraw all of their funds, the latest users in the list will be unable to withdraw all of their deposit (0.1 SOL may be worth $100 soon, so it does seem relatively high).

Blueprint Finance's response: The Glow team will deposit this 0.1 sol, which will prevent the contract from closing due to the low rent fee.

R-03 – migrate_withdraw_request.rs#L37

Certora: `system_program: Program<'info, System>` seems unnecessary.

Blueprint Finance's response: We will remove this.

R-04 – deposit.rs#L139-L145

Certora: The code went from directly using the PubKey to calling .key() for the PubKey. It seems redundant. The implementation is as such:

```
impl Key for Pubkey {
    fn key(&self) -> Pubkey {
        *self
    }
}
```

Blueprint Finance's response: We will optimize this.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.