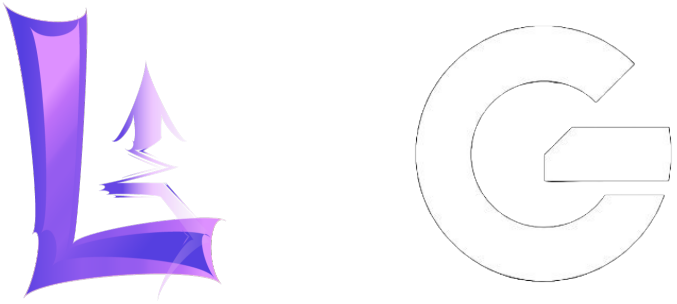
Gluon - Security Review



09.19.2025

Conducted by:
Kann, Lead Security Researcher
Radev_eth, Lead Security Researcher
Fitro, Lead Security Researcher

Table of Contents

1 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

2 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

2.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

3 Executive summary

Overview

Project Name	Gluon
Repository	Private
Commit hash	Private
Resolution	Fixed
Documentation	
Methods	Manual review

Scope

/src/Forwarder.sol

Issues Found

Critical risk	0
High risk	0
Medium risk	1
Low risk	0
Informational	1

4 Findings

4.1 Medium

4.1.1 Missing Validation of deposit to targetvault Return Value

Severity: *Medium risk*

Description: The _deployFunds function forwards funds to targetVault by calling:

targetVault.deposit(deployAmount, address(this));

The _deployFunds function assumes targetVault.deposit(…) will always give the strategy a non-zero amount of vault shares. In normal, well-behaved vaults that's true, but there are cases where this assumption can fail. For example, if a freshly launched/integrated vault has been pre-funded (someone transfers assets directly to the vault contract without calling transfer the vault's share-calculation can be skewed and a subsequent deposit call can result in 0 shares minted for the depositor. Other edge cases could also lead to the same outcome, leaving the strategy's assets effectively locked in the vault with no credited shares.

Recommendation Always validate that shares were actually received:

```
uint256 sharesReceived = targetVault.deposit(deployAmount, address(this));
require(sharesReceived > 0, "Deposit did not mint shares");
```

Resolution: Fixed

4.2 Informational

4.2.1 Redundant Balance Check and Math.min in deployFunds

Severity: *Informational*

Description: A user calls deposit() This function is defined in TokenizedStrategy and is executed via delegatecall from the Forwarder (which inherits BaseStrategy). As a result, storage updates happen in the Forwarder, msg.sender remains the user, and address(this) refers to the Forwarder.

Inside TokenizedStrategy.deposit: Assets are transferred from the user to the Forwarder (_asset.safeTransferFrom(msg.sender, address(this), assets)). then IBaseStrategy(address(this)).deployFunds(_asset.balanceOf(address(this))); We call deployFunds with the balance of the forwarder = BaseStrategy where then inside Forwarder it gets uint256 deployAmount = Math.min(_amount, balance); Which both values are same since _amount is address this balance of asset and balance is address(this) balance of the asset in the contract.

Math.min and getting the balance of the contract in _deployFunds is redudant.

Resolution: Fixed