BLOCKSEC

Security Audit
Report for glp

Date: August 15, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Radiant Capital
Target	glp

Version History

Version	Date	Description
1.0	August 15, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-08-15 11:15:40+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository [1] of glp of Radiant Capital.

The glp is a core component of the Radiant Guardian architecture. It enables capital-efficient protection and utilization of protocol reserves through tokenized shares , a modular strategy layer, and DAO-controlled asset management via a multisig. As an ERC4626-compliant vault, it accepts WETH deposits from users, minting shares that represent a proportional claim on a diverse basket of assets, which are managed by a strategy. The vault's net asset value is precisely tracked using a dedicated OracleRouter for accurate, WETH-denominated pricing, with all critical governance actions and asset management securely controlled by the DAO's multisig for enhanced security and flexibility.

Note this audit only focuses on the smart contracts in the following directories/files:

- src/*

Other files are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
glp	`Version 1`	`addfbad2b8f00f4dd8efe2e0152a6341259b2866`
	`Version 2`	`5173223e0f66c6620b455cf84784231d80572f21`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

[1] https://github.com/radiant-capital/glp-main

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness
* Oracle security
* Business logic correctness
* Semantic and functional consistency
* Emergency mechanism
* Economic and incentive impact

1.3.2 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we found **three** potential security issues. Besides, we have **three** recommendations and **four** notes.

- Medium Risk: 1
- Low Risk: 2
- Recommendation: 3
- Note: 4

ID	Severity	Description	Category	Status
1	Medium	Potential sandwich attacks to the function `burnDaoShares()`	Security Issue	Fixed
2	Low	Potential DoS via donating to the `GUARDIAN_MULTISIG`	Security Issue	Confirmed
3	Low	Incorrect asset comparison in the function `setStrategy()`	Security Issue	Fixed
4	-	Insecure price handling	Recommendation	Fixed
5	-	Add non-zero check on minted shares	Recommendation	Fixed
6	-	Incorrect comment of the function `getAssetPrice()`	Recommendation	Fixed
7	-	Future use of functions `getRoundData()` and `latestRoundData()` requires validation	Note	-
8	-	Potential centralization risks	Note	-
9	-	Ensure invoking the function `bootstrap()` in the same transaction with the `GLPVault` deployment	Note	-
10	-	Potential incompatibility with the tokens with decimals greater than 18	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Potential sandwich attacks to the function `burnDaoShares()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `GLPVault`, the function `burnDaoShares()` allows the owner to burn shares from the `DAO_TREASURY`, which reduces the total supply of shares without affecting the total assets of the vault. This action can lead to a sudden increase in the share price, as the same amount of assets is now divided among fewer shares. An attacker could exploit this price change with a sandwich attack, profiting from the temporary spike in the share price before and after the burn.

```
167   function burnDaoShares(uint256 shares) external onlyOwner whenBootstrapped {
168       if (shares == 0) revert Errors.InsufficientShares();
169       if (shares > balanceOf(DAO_TREASURY)) revert Errors.InsufficientShares();
170       _burn(DAO_TREASURY, shares); // internal ERC20 burn, no assets move
171       emit DAOSharesBurned(DAO_TREASURY, shares);
172   }
```

Listing 2.1: src/GLPVault.sol

Impact An attacker can profit from a temporary increase in the share price by executing a sandwich attack around the `burnDaoShares()` transaction.

Suggestion Revise the logic accordingly.

2.1.2 Potential DoS via donating to the `GUARDIAN_MULTISIG`

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The functions `addBasketToken()` and `removeBasketToken()` both contain checks that will revert the transaction if the `GUARDIAN_MULTISIG` holds a balance of the token being added or removed. This design allows any external user to intentionally send a small amount of a specific token to the `GUARDIAN_MULTISIG` address. By doing this, an external attacker can block the protocol's owner from adding or removing that token from the basket.

```
70    function addBasketToken(address token) external onlyOwner {
71        if (token == address(0)) revert Errors.NullAddress();
72        // Revert if there is balance in the multisig for this token
73        if (IERC20(token).balanceOf(GUARDIAN_MULTISIG) > 1 && token != RDNT_ARBITRUM) revert Errors
              .PositiveBalance();
74        // Do not allow WETH to be added as a basket token, as it's already the base asset
75        if (token == address(weth)) revert Errors.TokenAlreadyExists();
76        // Make sure token wasn't already added
77        for (uint256 i = 0; i < basketTokens.length; i++) {
78            if (basketTokens[i] == token) {
79                revert Errors.TokenAlreadyExists();
80            }
81        }
82        // If feed is not set, revert
83        (uint256 price,) = oracleRouter.getAssetPrice(token);
84        if (price == 0) revert Errors.InvalidPrice();
85        basketTokens.push(token);
86        emit BasketTokenAdded(token);
87    }
88
89    /// @notice Remove a basket token by index, replacing it with the last element.
90    /// @param index The index of the token to remove.
91    function removeBasketToken(uint256 index) external onlyOwner {
92        if (index >= basketTokens.length) revert Errors.IndexOutOfBounds();
93        address token = basketTokens[index];
```

```
94      // Do not allow to remove the token if there is any balance of it in the multisig(1 wei
            alloowed)
95      if (IERC20(token).balanceOf(GUARDIAN_MULTISIG) > 1) revert Errors.TokenHasBalance();
96      basketTokens[index] = basketTokens[basketTokens.length - 1];
97      basketTokens.pop();
98      emit BasketTokenRemoved(token);
99  }
```

Listing 2.2: src/GLPStrategy.sol

Impact An external attacker can block the protocol's owner from adding or removing that token from the basket.

Suggestion Revise the logic accordingly.

Feedback from the project The project states that if they receive some donation, they will send it elsewhere from the `GUARDIAN_MULTISIG` before adding or removing tokens.

2.1.3 Incorrect asset comparison in the function `setStrategy()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description To prevent drastic changes in total asset value, the function `setStrategy()` compares the new strategy's total assets against the vault's total assets which include the old strategy's total assets and assets in the vault. However, if the vault holds a balance, the comparison may potentially cause the transaction to fail and create opportunities for arbitrage. The correct logic should compare the total assets before the switch (assets in the old strategy + assets in the vault) with the total assets after the switch (assets in the new strategy + assets in the vault).

```
51  function setStrategy(IGlpStrategy _strategy) external onlyOwner {
52      if (address(_strategy) == address(0)) revert Errors.NullAddress();
53      if (address(_strategy).code.length == 0) revert Errors.NotAContract();
54      // If old strategy exists, revoke its approval
55      if (address(strategy) != address(0)) {
56          IERC20(asset()).approve(address(strategy), 0); // revoke approval
57          // Do not allow more than 2% asset increase or decrease when setting a new strategy.
                This will prevent
58          // arbitrage opportunities.
59          if (
60              _strategy.totalAssets() < (totalAssets() * (BIPS - maxAllowedNAVChange)) / BIPS
61                  || _strategy.totalAssets() > (totalAssets() * (BIPS + maxAllowedNAVChange)) /
                        BIPS
62          ) {
63              revert Errors.InvalidStrategy();
64          }
65      }
66      strategy = _strategy;
67      // Approve the strategy to pull WETH from this vault
68      IERC20(asset()).approve(address(_strategy), type(uint256).max);
69      emit GLPStrategySet(address(_strategy));
```

```
70    }
```

Listing 2.3: src/GLPVault.sol

```
113   function totalAssets() public view override returns (uint256) {
114       return IERC20(asset()).balanceOf(address(this)) + strategy.totalAssets();
115   }
```

Listing 2.4: src/GLPVault.sol

Impact Potential arbitrage opportunities or failures of the execution of the function `setStrategy()` due to the incorrect asset comparison.

Suggestion Revise the logic accordingly.

2.2 Recommendation

2.2.1 Insecure price handling

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `GLPStrategy` calculates the vault's total asset value by aggregating the values of all tokens in its basket, with prices fetched from an oracle. A security risk exists because the code uses a continue statement when a token's price is zero, rather than reverting the transaction.

While the current `oracleRouter` is designed to revert when the price is zero, this check is not robust. If the `oracleRouter` were to be replaced with a less secure version , the contract `GLPStrategy` would be vulnerable. This may allow a high-value asset to be completely ignored in the total asset calculation, when an incorrect price is returned by the oracle.

```
104   function totalAssets() public view override returns (uint256) {
105       // If no basket tokens are set, return only WETH balance
106       uint256 total = weth.balanceOf(GUARDIAN_MULTISIG);
107       if (basketTokens.length == 0) return total;
108       for (uint256 i = 0; i < basketTokens.length; i++) {
109           address token = basketTokens[i];
110           uint256 tokenBalance = IERC20(token).balanceOf(GUARDIAN_MULTISIG);
111           // Scale token balance based on decimals to 18 decimals
112           tokenBalance = tokenBalance * (10 ** (18 - ERC20(token).decimals()));
113           if (tokenBalance == 0) continue;
114           (uint256 price, uint8 decimals) = oracleRouter.getAssetPrice(token);
115           // If price is 0, skip this token
116           if (price == 0) continue;
117           total += tokenBalance.mulDiv(price, 10 ** decimals);
118       }
119       return total;
120   }
```

Listing 2.5: src/GLPStrategy.sol

Suggestion Revert the transaction when a token's price is zero.

2.2.2 Add non-zero check on minted shares

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `_deposit()` lacks a non-zero check on shares to be minted. If a user deposits a minimal amount of assets, this may result in zero shares being minted due to rounding.

```
178   function _deposit(address caller, address receiver, uint256 assets, uint256 shares)
179       internal
180       override
181       nonReentrant
182   {
183       super._deposit(caller, receiver, assets, shares);
184
185       uint256 liquid = IERC20(asset()).balanceOf(address(this));
186       if (liquid > 0) {
187           strategy.invest(liquid);
188       }
189   }
```

Listing 2.6: src/GLPVault.sol

Suggestion Add non-zero check on minted shares

2.2.3 Incorrect comment of the function `getAssetPrice()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The comment for the function `getAssetPrice()` incorrectly states that the price is returned in USD. This is a contradiction, as the function `getAssetPrice()` actually returns the price of the asset in WETH. The discrepancy between the code's behavior and the comment's description can lead to a misunderstanding of the pricing unit.

```
27   /// @notice Get the price of an asset
28   /// @param asset The address of the asset
29   /// @return price The price of the asset in USD, scaled to 18 decimals
30   /// @return decimals The number of decimals the price is scaled to
31   function getAssetPrice(address asset) external view override returns (uint256, uint8) {
32       IAggregatorV2V3 feed = assetToFeed[asset];
33       // If feed is not set, return 0 price
34       if (address(feed) == address(0)) revert Errors.FeedNotSet();
35
36       int256 price = feed.latestAnswer();
37       if (price <= 0) revert Errors.InvalidPrice();
38       return (uint256(price), feed.decimals());
39   }
```

Listing 2.7: src/oracles/OracleRouter.sol

Suggestion Revise the comment accordingly.

2.3 Note

2.3.1 Future use of functions `getRoundData()` and `latestRoundData()` requires validation

Introduced by `Version 1`

Description The functions `getRoundData()` and `latestRoundData()` are designed to fetch price data from a `Chainlink` oracle in the contract `BaseChainlinkAdapter`. These functions currently do not verify if the returned `answer` is a positive value, if the `updatedAt` timestamp is recent. If these functions are called by other contracts in the future. While these functions are not currently in use, any future contract that calls them must implement these checks to prevent the use of stale or incorrect price data.

```
74   function getRoundData(uint80 _roundId)
75       external
76       view
77       returns (uint80 roundId, int256 answer, uint256 startedAt, uint256 updatedAt, uint80
             answeredInRound)
78   {
79       (roundId, answer, startedAt, updatedAt, answeredInRound) = chainlinkFeed.getRoundData(
             _roundId);
80   }
81
82   /**
83    * @notice Returns data of latest round
84    * @return roundId is the round ID from the aggregator for which the data was retrieved.
85    * @return answer is the answer for the given round
86    * @return startedAt is the timestamp when the round was started.
87    * @return updatedAt is the timestamp when the round last was updated.
88    * @return answeredInRound is the round ID of the round in which the answer was computed.
89    */
90   function latestRoundData()
91       public
92       view
93       returns (uint80 roundId, int256 answer, uint256 startedAt, uint256 updatedAt, uint80
             answeredInRound)
94   {
95       (roundId, answer, startedAt, updatedAt, answeredInRound) = chainlinkFeed.latestRoundData();
96   }
```

Listing 2.8: src/oracles/adapters/BaseChainlinkAdapter.sol

2.3.2 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `owner`) can conduct sensitive operations, which introduces potential centralization risks. For example, the `owner` can set the oracle feed source of an asset based on the protocol. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.3.3 Ensure invoking the function `bootstrap()` in the same transaction with the `GLPVault` deployment

Introduced by `Version 1`

Description The project should invoke the function `bootstrap()` in the same transaction with the `GLPVault` deployment. Moreover, the project should ensure that the value of `totalAssets()` is greater than a minimum threshold when invoking the function `bootstrap()`, to ensure that the amount of the minted `shares` is large enough. This is to prevent a donation attack.

```
89   function bootstrap() external {
90       if (bootstrapped) revert Errors.AlreadyBootstrapped();
91       if (address(strategy) == address(0)) revert Errors.StrategyNotSet();
92       uint256 nav = totalAssets(); // e.g. 500 WETH $1 M
93       uint256 shares = nav; // 1:1 initial price (or pick any ratio)
94       _mint(DAO_TREASURY, shares); // internal ERC20 mint, no assets move
95       // Make sure balance of DAO treasury is reflected in shares
96       if (shares == 0) revert Errors.InsufficientShares();
97       bootstrapped = true;
98       emit Bootstrapped(DAO_TREASURY, shares);
99   }
```

Listing 2.9: src/GLPVault.sol

```
113  function totalAssets() public view override returns (uint256) {
114      return IERC20(asset()).balanceOf(address(this)) + strategy.totalAssets();
115  }
```

Listing 2.10: src/GLPVault.sol

```
167  function burnDaoShares(uint256 shares) external onlyOwner whenBootstrapped {
168      if (shares == 0) revert Errors.InsufficientShares();
169      if (shares > balanceOf(DAO_TREASURY)) revert Errors.InsufficientShares();
170      _burn(DAO_TREASURY, shares); // internal ERC20 burn, no assets move
171      emit DAOSharesBurned(DAO_TREASURY, shares);
172  }
```

Listing 2.11: src/GLPVault.sol

2.3.4 Potential incompatibility with the tokens with decimals greater than 18

Introduced by `Version 1`

Description The protocol does not support tokens with decimals greater than 18. The project should only add basket tokens with decimals less or equal than 18 when invoking the function `addBasketToken()` to avoid DoS risk.

```
104  function totalAssets() public view override returns (uint256) {
105      // If no basket tokens are set, return only WETH balance
106      uint256 total = weth.balanceOf(GUARDIAN_MULTISIG);
107      if (basketTokens.length == 0) return total;
108      for (uint256 i = 0; i < basketTokens.length; i++) {
109          address token = basketTokens[i];
```

```
110             uint256 tokenBalance = IERC20(token).balanceOf(GUARDIAN_MULTISIG);
111             // Scale token balance based on decimals to 18 decimals
112             tokenBalance = tokenBalance * (10 ** (18 - ERC20(token).decimals()));
113             if (tokenBalance == 0) continue;
114             (uint256 price, uint8 decimals) = oracleRouter.getAssetPrice(token);
115             // If price is 0, skip this token
116             if (price == 0) continue;
117             total += tokenBalance.mulDiv(price, 10 ** decimals);
118         }
119         return total;
120     }
```

Listing 2.12: src/GLPStrategy.sol

```
70  function addBasketToken(address token) external onlyOwner {
71      if (token == address(0)) revert Errors.NullAddress();
72      // Revert if there is balance in the multisig for this token
73      if (IERC20(token).balanceOf(GUARDIAN_MULTISIG) > 1 && token != RDNT_ARBITRUM) revert Errors
            .PositiveBalance();
74      // Do not allow WETH to be added as a basket token, as it's already the base asset
75      if (token == address(weth)) revert Errors.TokenAlreadyExists();
76      // Make sure token wasn't already added
77      for (uint256 i = 0; i < basketTokens.length; i++) {
78          if (basketTokens[i] == token) {
79              revert Errors.TokenAlreadyExists();
80          }
81      }
82      // If feed is not set, revert
83      (uint256 price,) = oracleRouter.getAssetPrice(token);
84      if (price == 0) revert Errors.InvalidPrice();
85      basketTokens.push(token);
86      emit BasketTokenAdded(token);
87  }
```

Listing 2.13: src/GLPStrategy.sol

Feedback from the project The project states that they will not use any assets with the decimals greater than 18.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS