Smart Contract Security Audit Report

Symbiosis Depository Audit

1. Contents

2. General Information

This report contains information about the results of the security audit of the Symbiosis (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 2025-10-23 to 2025-10-27.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repository: symbiosis-finance/depository/. Initial review was done for the commit fa98a8116a6aa373c41ead7a7685bd29fe6694f4 and the re-testing was done for the commit 5c5305b0381c3753385dcb18344ce3fbfc7ae77a.

The following contracts have been tested:

- contracts/Depository.sol
- contracts/DepositUnlockers.sol
- contracts/Router.sol

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User,

- Protocol owner,
- Liquidity Token owner/contract.

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have found three medium security issues which has been fixed and re-tested in the course of the work.

The other suggestions included fixing the low-risk issues and some best practices.

The team has given the feedback for the suggested changes and explanation for the underlying code.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of October 31, 2025.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Contracts are vulnerable to fee-on-transfer token related accounting issues	contracts/Depository.sol	Medium	Acknowledged
forceApprove should be used	contracts/Router.sol	Medium	Fixed
Hardhat Console Imports	contracts/DepositUnlockers.sol	Low	Acknowledged
Use Ownable2Step's transfer function rather than Ownable's for transfers of ownership	contracts/Depository.sol	Low	Fixed
Missing zero address check in setRouter	contracts/Depository.sol	Info	Acknowledged
No Emergency Unlock Mechanism for Depositor	contracts/Depository.sol	Info	Acknowledged

Issue	Contract	Risk Level	Status
Consider disabling renounceOwnership	contracts/Depository.sol	**Info**	Acknowledged
Unused error	contracts/Router.sol	**Info**	Fixed

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Contracts are vulnerable to fee-on-transfer token related accounting issues

Risk Level: Medium

Status: Acknowledged

Contracts:

- contracts/Depository.sol

Description:

Without measuring the balance before and after the transfer, there's no way to ensure that enough tokens were transferred, in the cases where the token has a fee-on-transfer mechanic. If there are latent funds in the contract, subsequent transfers will succeed.

```
File: contracts/Depository.sol
 59: deposit.token.safeTransferFrom(
60: msg.sender,
61: address(this),
62: deposit.amount
63: );
 79: deposit.token.safeTransfer(address(router), deposit.amount);
```

Remediation:

Measure token balances before and after transfers to handle fee-on-transfer tokens correctly:

```
uint256 balanceBefore = deposit.token.balanceOf(address(this));
deposit.token.safeTransferFrom(msg.sender, address(this),
deposit.amount);
uint256 actualReceived = deposit.token.balanceOf(address(this)) -
balanceBefore;
// Use actualReceived for further calculations
```

5.2. forceApprove should be used

Risk Level: Medium

Status: Fixed in the commit.

Contracts:

- contracts/Router.sol

Location:

- `deposit()`

Description:

The router's `_lazyApprove` assumes standard ERC-20 behavior and attempts to set an unlimited allowance if the current allowance is below the required amount:

```
    function _lazyApprove(IERC20 token, address to, uint256 amount)
internal {
        if (token.allowance(address(this), to) < amount) {
            token.approve(to, type(uint256).max);
        }
    }
```

This fails on non-standard tokens (e.g., USDT) that require setting the allowance to zero before setting it to a new non-zero value. If the current allowance is non-zero, approve(max) may revert. Additionally, because your system exposes raw external calls (via `_externalCall()`), a caller could set `approve(to, 1)` first; afterward all attempts by `_lazyApprove` to raise the allowance would fail on such tokens.

Remediation:

Consider using `forceApprove` function from OpenZeppelin's SafeERC20 library which performs a safe zero-reset when needed.

References:

- https://github.com/OpenZeppelin/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol

5.3. Hardhat Console Imports

Risk Level: Low

Status: Acknowledged

Contracts:

- contracts/DepositUnlockers.sol

Description:

The contract imports the file `hardhat/console.sol` and uses it in DepositUnlockers.sol.

Remediation:

Remove it to maintain clean and efficient code.

5.4. Use Ownable2Step's transfer function rather than Ownable's for transfers of ownership

Risk Level: Low

Status: Fixed in the commit.

Contracts:

- contracts/Depository.sol

Description:

Ownable2Step and Ownable2StepUpgradeable prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner permissions actively accept via a contract call of its own.

```
File: contracts/Depository.sol
 17: contract Depository is IDepository, UUPSUpgradeable,
OwnableUpgradeable {
```

Remediation:

Replace `OwnableUpgradeable` with `Ownable2StepUpgradeable` to implement two-step ownership transfer for enhanced security.

5.5. Missing zero address check in setRouter

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/Depository.sol

Location:

- `setRouter`

Description:

The `setRouter()` function does not include a check to prevent setting `router` to the zero address. Because all unlockers are calling the `router.externalCall()` the call the unlock will revert.

Remediation:

To mitigate this risk, add a check to ensure that router is not the zero address before assigning it.

5.6. No Emergency Unlock Mechanism for Depositor

Risk Level: Info

Status: Emergency Unlock was originally part of the Depository contract, but during development developer team decided to replace it with an additional branch using the SwapUnlocker, where outToken matches deposit.token and outMinAmount matches deposit.amount.

Contracts:

- contracts/Depository.sol

Description:

The Depository contract lacks an emergency unlock mechanism that allows the original depositor to reclaim their tokens when the unlock condition becomes invalid or unfulfillable. Currently, the unlock() function can only be called by anyone who can provide a valid solution to the condition, but there's no fallback mechanism for the depositor.

This creates several problematic scenarios:

1. Invalid conditions: If a depositor accidentally sets an impossible or invalid condition, their tokens become permanently locked.

2. Condition changes: If external conditions change making the original condition unfulfillable, depositors have no recourse.

Remediation:

Consider implementing an emergency unlock mechanism that allows the original depositor to reclaim their tokens under specific conditions.

5.7. Consider disabling renounceOwnership

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/Depository.sol

Description:

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities. The OpenZeppelin's Ownable used in this project contract implements renounceOwnership. This can represent a certain risk if the ownership is renounced for any other reason than by design. Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner.

```
File: contracts/Depository.sol
 17: contract Depository is IDepository, UUPSUpgradeable,
OwnableUpgradeable {
```

Remediation:

Override the `renounceOwnership()` function to disable it and prevent accidental loss of ownership:

```
function renounceOwnership() public virtual override onlyOwner {
    revert("Ownership cannot be renounced");
}
```

5.8. Unused error

Risk Level: Info

Status: Fixed in the commit.

Contracts:

- contracts/Router.sol

Description:

The is an unused error:`error CallToEOA(address target);`

Remediation:

Consider removing redundant code.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.