× PESSIMISTIC

SECURITY ANALYSIS

by Pessimistic

This report is public

December 28, 2024

ABSTRACT

In this report, we consider the security of smart contracts of Drizzle Finance project. Our task is to find and describe security issues in the smart contracts of the platform.

DISCLAIMER

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

SUMMARY

In this report, we considered the security of Drizzle Finance smart contracts. Several issues, ranging from critical to low severity, were identified, and are currently resolved or commented as of the final version of the audit. We described the audit process in the section below.

The audit showed one critical issue: No access check. The audit also revealed several issues of medium severity: No documentation, Excessive owner role, Rounding at cTokens conversion (M03). Moreover, several low-severity issues were found. All tests for the current scope passed.

After the initial audit, the developers provided us with a new version of the code. In that version, they fixed the critical issue No access check, and commented on the medium severity issue No documentation. The Excessive owner role issue was commented on and partially fixed. The developers also fixed all low-severity issues. The severity of the medium severity issue Rounding at cTokens conversion (M03) was changed from medium to note (N03) as the attack is unprofitable because of the precision of the calculations.

The overall code quality is good.

GENERAL RECOMMENDATIONS

We recommend writing the public documentation.

PROJECT OVERVIEW

Project description

For the audit, we were provided with Drizzle Finance project on a private GitHub repository, commit 5a8b53a8a1c7c79b58fc705f697e65853de5b186.

The scope of the audit included:

- **DrizzleFi.sol**;
- **Connectors/CompoundV3Connector.sol**;
- **Interface/** folder.

The project has no public documentation, only a brief description.

All 13 tests of the current scope pass successfully. The code coverage of the scope is 80,67%.

The total LOC of audited sources is 879.

Codebase update

After the audit, the developers provided us with a new commit: 9103490c625a71b79dc2314a9ad5e24452eaae56. In that update, they fixed or commented on all the found issues. One medium severity issue was moved to note.

The entire project's 14 tests passed successfully. The code coverage of the current scope was 79,72%.

AUDIT PROCESS

We started the audit on December 18 and finished on December 24, 2024.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- Whether the token conversions work correctly;
- Integration with Compound v3;
- Owner influence on the price of cTokens, on rewards, and on the whole project;
- Standard Solidity checks;
- Whether any user can withdraw tokens of other users.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts. We also sent the results to the developers in the text file.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

We made the recheck on December 27, 2024. The developers fixed or commented on all of the issues from our report. The M03 issue was moved to N03.

We reviewed the updated codebase, reran tests and tools, and updated the report accordingly.

MANUAL ANALYSIS

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

C01. No access check (fixed)

The `CompoundV3Connector.withdrawProtocolRewards` function does not have the `onlyOwner` modifier. This allows anyone to claim protocol rewards since there is no access control.

> The issue has been fixed and is not present in the latest version of the code.

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. No documentation (commented)

The project has no public documentation, only a brief description. The documentation is a critical part that helps to improve security and reduce risks. It should explicitly explain the purpose and behavior of the contracts, their interactions, and key design choices. It is also essential for any further integrations.

> `Comment from the developers:`
>
> As v1, Drizzle contracts are not expected to be integrated by third parties due to the current product focus. Documentation for the end users and for code transparency will be finalized and published after launch.

M02. **Excessive owner role** (partially fixed)

In the current implementation, the system depends on the owner role. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if admin's private keys become compromised. The owner of the **DrizzleFi** contract can:

- **(fixed)** Withdraw any remaining `COMP` tokens that could remain after calling `CompoundV3Connector.withdrawProtocolRewards`. It can happen since the malicious owner will get approval by calling the `DrizzleFi.allowToken` function with `_token == COMP` and `_LPToken == owner address` parameters.
- **(fixed)** Change the allowed tokens.
- Change the fee numerator to increase the fee for the collector and front-run the `withdrawInterest` function.

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> The developers limited some of the owner's rights: they can no longer change the status of the connector and switch the allowance of LP tokens.

> `Comment from the developers:`
>
> We recognize the considerable power the Owner role holds, namely, to change fees (up to a hard-coded limit of 20%, DrizzleFi, line 48), to change the fee recipient, to add new connectors and enable additional tokens within existing connectors.
>
> Following the auditor's recommendation, we limited the power of the role to explicitly disallow affecting any of the already deployed connectors. As a consequence, once the initial setup is deployed, it can be validated once and cannot be further affected by the holders of the Owner role.
>
> Eventually, the role will be migrated to a multisignature wallet, given to DAO contracts, or renounced, following the product direction of the protocol.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Storage variable -> constant (fixed)

Consider declaring the `feeDenominator` storage variable as constant to decrease gas consumption at line 43 of the **DrizzleFi** contract.

> The issue has been fixed and is not present in the latest version of the code.

L02. Using custom errors (fixed)

Consider using custom errors instead of `require` with `string` to reduce gas consumption in the entire project.

> The issues have been fixed and are not present in the latest version of the code.

L03. Gas consumption (fixed)

The `withdrawInterest` and the `_withdrawInterestInternal` functions of the **DrizzleFi** contract have the same check `!feeExempt[msg.sender] && (feeNumerator > 0)`. The `withdrawInterest` function calls the `_withdrawInterestInternal` function internally.

Consider replacing the `!feeExempt[msg.sender] && (feeNumerator > 0)` with `protocolFeeFromInterest != 0` at line 270 of the **DrizzleFi** contract to avoid reading from the storage twice.

> The issue has been fixed and is not present in the latest version of the code.

L04. Gas consumption (fixed)

The rewards could be updated only in the `else` branch at lines 412-417 of the `DrizzleFi._addPrincipalInternal` function and at lines 484-489 of the `DrizzleFi._withdrawPrincipalInternal` function to reduce gas consumption.

> The issues have been fixed and are not present in the latest version of the code.

L05. Implicit visibility (fixed)

Storage variables are considered internal by default. Consider declaring visibility explicitly of all internal variables to improve code readability in the **DrizzleFi** and **CompoundV3Connector** contracts.

> The issues have been fixed and are not present in the latest version of the code.

L06. Multiple reads from the storage (fixed)

The `markets[_baseToken]` storage value is read multiple times in the following functions of the **CompoundV3Connector** contract: `withdrawPrincipal`, `addPrincipal`, `_updateTotalTokenRewards`, `withdrawProtocolRewards`.

> The issues have been fixed and are not present in the latest version of the code.

L07. Deviation from the ERC-20 Standard (fixed)

Consider replacing the following functions to check the returned values, for more safe approval and to correspond to the ERC-20 Standard:

- The `approve` function with `forceApprove`;
- The `transfer` function with `safeTransfer`;
- The `transferFrom` function with `safeTransferFrom`.

> The issues have been fixed and are not present in the latest version of the code.

Notes

N01. Re-used signature (commented)

Consider adding `chainId` field to the signature in the `DrizzleFi.claimBeneficiaryPositionsFrom` function to avoid re-use if contracts are deployed to other chains.

> `Comment from the developers:`
>
> The functions in question, `DrizzleFi.claimBeneficiaryPositions` and `DrizzleFi.claimBeneficiaryPositionsFrom`, are only intended for the narrow use case of transferring a position from ephemeral keys (one-time frontend address, "claim address") to the permanent wallet of the same beneficiary. Its role in the product is to allow patrons to send drizzles without knowing the beneficiary's address beforehand — and even to beneficiaries who don't yet own a blockchain wallet. This functionality is not intended to move positions between arbitrary beneficiaries. As such, it does not fall under the scope of replay attacks: even if a beneficiary has the same claim address on multiple chains (which is unlikely), the worst an attacked can do by replaying a signed message on another chain is to pay the beneficiary's gas for the transfer from the claim address to the beneficiary's address on the other chain.

N02. Rewards addresses might differ (fixed)

The address of the rewards contract is hardcoded at line 19 of the **CompoundV3Connector** contract. It can differ if the project is deployed to other chains.

> The issue has been fixed and is not present in the latest version of the code.

N03. **Rounding at cTokens conversion** (addressed)

Conversion from cToken to base token and vice versa is always rounded down in the `viewInterestStored`, `withdrawInterest`, `principalToLPTokens`, `addPrincipal`, `withdrawPrincipal` functions of the **CompoundV3Connector** contract. There are several examples of how it can be exploited:

- Rounding down the conversion in the `withdrawInterest` funciton results in less cTokens being burned than the amount in the base token. This means that the remainder of the cTokens will remain at the beneficiary's position and they will be able to withdraw interest again by encroaching on other users' tokens.
- During `withdrawPrincipal`, the value of cTokens in the base token will be less than `principal`. Thus, the beneficiary may receive more interest than they should have.

However, if the `withdrawInterest` function has rounded up conversion, a beneficiary may output a larger value in base tokens than corresponds to cTokens. This will result in, for example, someone not being able to withdraw their principal completely.

Consider adding precision so that rounding does not affect the calculation too much, or a second option:

For instance, the `withdrawPrincipal` function. If the number of cTokens is less than the corresponding `principal` value, consider withdrawing the part of the `principal` corresponding cTokens value (not the entire `principal`) and then withdrawing the remainder when rounding is unnecessary.

> The computational error is less than `1e-6`, which is not significant. Potential attacks will not be beneficial because the transaction cost will exceed the profit.

This analysis was performed by Pessimistic:

Oleg Bobrov, Security Engineer
Daria Korepanova, Senior Security Engineer
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

December 28, 2024