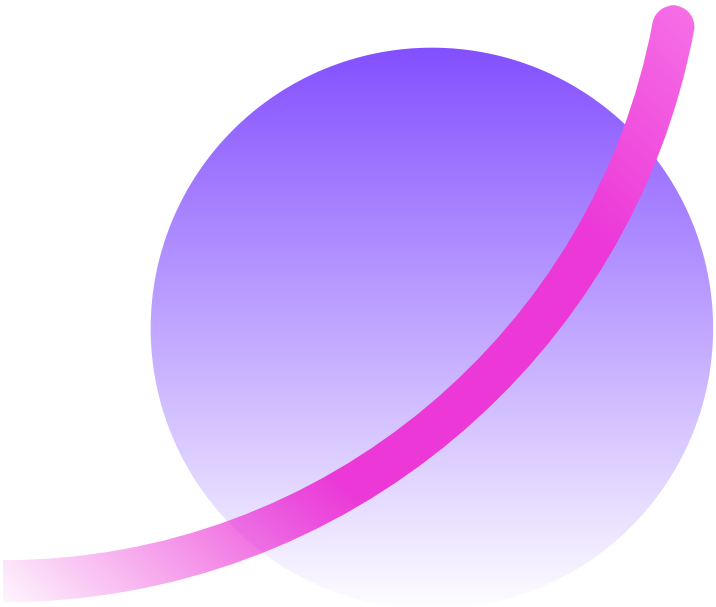

Mellow Protocol HStrategy Security Analysis

by Pessimistic

This report is public

November 14, 2022

Abstract

In this report, we consider the security of smart contracts of Mellow Protocol: HStrategy 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 Mellow Protocol: HStrategy smart contracts. We described the audit process in the section below.

The audit showed a few low-severity issues. The overall code quality is good.

After the audit, the developers provided comments regarding all the found issues. The comments can be found in the Manual analysis section.

The project has detailed documentation. The codebase is covered with tests.

General recommendations

We recommend fixing the mentioned issues.

Project overview

Project description

For the audit, we were provided with Mellow Protocol: HStrategy project on a public GitHub repository, commit 01e69067d8bdd98f4dbfc6b65d10716c01c41c88.

The scope of the audit included only the following files:

- **contracts/strategies/HStrategy.sol**,
- **contracts/utils/HStrategyHelper.sol**,
- **contracts/utils/UniV3Helper.sol**,
- **contracts/utils/DefaultAccessControlLateInit.sol**,
- dependencies.

The documentation for the project included the following:

- **README.md** file in the repository,
- **HStrategy spec.pdf** file, sha1sum `82c72dbd86772e2bdc67fd9bc150ce0d7f8967b9`,
- Research paper link,
- Vaults design article link,
- Protocol documentation link.

All 32 tests pass successfully. The code coverage of the scope is 94.78%.

The total LOC of audited sources is 1171 without dependencies.

Audit process

We started the audit on October, 25 and finished on November, 3, 2022.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project. After a discussion, we performed preliminary research and specified those parts of the code and logic that require additional attention during an audit.

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:

- The emergency function `manualPull` works correctly.
- The rebalance process corresponds to the documentation.
- The implementation of the u_1, u_2, and u_3 formulas corresponds to the documentation.

We scanned the project with the static analyzer Slither with our own set of rules and then manually verified all the occurrences found by the tool.

We ran tests and calculated the code coverage.

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

After the audit, the developers commented the mentioned issues.

Finally, we updated the report.

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.

The audit showed no critical issues.

Medium severity issues

Medium 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.

The audit showed no issues of medium severity.

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. Excessive arguments (commented)

In **HStrategy** contract, the `_pullExtraTokens` function accepts the `hStrategyHelper_` value as an argument at line 563. However, this value always contains an address from an `immutable` variable. Thus, there is no need to pass it as an argument.

> *Comment from the developers: We agree that it is possible not to pass `hStrategyHelper_` as a parameter to the `_pullExtraTokens` function, and this will reduce gas consumption somewhat. But since the impact on the amount of gas consumed is insignificant, we will leave the current implementation of the function.*

L02. Duplicated NFT check (out of scope, commented)

In **IntegrationVault** contract, `pull` and `externalCall` functions verify that `_nft != 0` at lines 95 and 188, respectively. However, both of these functions perform `_isApprovedOrOwner` call that also verifies non-zero `_nft` value. Consider removing excessive check.

> *Comment from the developers: Here we aim to maintain uniformity between the `reclaimTokens`, `externalCall`, `push`, and `pull` functions by making a perhaps redundant but explicit check that the system is initialized. The `_isApprovedOrOwner` function also has a check for `nft == 0`, but here we do it with the goal not to call the registry contract functions twice for this case.*

L03. Check before swap (commented)

In **HStrategy** contract, the `_swapTokensOnERC20Vault` checks for swap restrictions at lines 746–754. Consider performing the checks at lines 746–749 and 754 before the swap since they do not depend on swap results.

> *Comment from the developers: Moving the checks higher will reduce gas consumption if the parameters in the rebalance limits are set incorrectly. But since this function is called only on behalf of our account, users will not incur any losses.*

L04. Revert instead of an assert (commented)

Since the **HStrategy** contract calls `_swapTokens` function only after checking its parameters with `swapNeeded` function, the `else` condition should never occur if the contract operates correctly. Thus, it should use `assert` instead of `revert`.

> *Comment from the developers: Usually we do not use the `assert` function in the code of the main contracts, also passing an errorCode instead of an explicit error looks less consistent for our system.*

Notes

N01. Overpowered role (commented)

In **HStrategy** contract, the admin role can:

- Transfer liquidity between vaults without any restrictions using the `manualPull` function.
- Manipulate strategy parameters.

These actions can significantly affect strategies operation. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if the admin's private keys become compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> *Comment from the developers: The `manualPull` function allows an admin user to transfer tokens from one vault to another within the same vaults system. Therefore, even in the case of compromise of the private key, users will be able to withdraw their funds from vaults.*
>
> *Also, to call operations on behalf of the administrator of the strategy, we use a multi-signature with a gnosis safe, which, according to our logic, is a sufficient level of security for an account with this role.*

This analysis was performed by Pessimistic:

Vladimir Tarasov, Security Engineer
Vladimir Pomogalov, Security Engineer
Nikita Kirillov, Junior Security Engineer
Yhtyyar Sahatov, Junior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

November 14, 2022