CavalRe AMM

Smart Contract Security Assessment

October 16, 2023

ABSTRACT

Dedaub was commissioned to audit the CavalRe protocol implementation, located at the repository https://github.com/CavalRe/multiswap, commit hash 19dbcf2a3a11797bd7e7bf2f9bce2d0dd4bc6abe. The audit focuses on changes carried out since Dedaub's last audit of the protocol, dated August 2023, which can be found here CavalRe Audit - Aug 2023. Since the previous audit the `Users.sol` contract has been significantly simplified. The contract now only keeps track of whether a user is blocked, as well as the discount that the user can receive. The system of affiliates allowing multiple addresses to be connected to the same CavalRe account has been removed. The `LPToken.sol` contract has also been modified to reflect the changes in Users.sol. The concept of one CavalRe transaction per block has been done away with. A two-step design to bypass the need for approving funds to the contract has also been added to `Pool.sol`.

SETTING AND CAVEATS

The audit scope consists of the following files:

```
contracts/
├── ILPToken.sol
├── IPool.sol
├── IUsers.sol
├── LPToken.sol
├── Pool.sol
├── Users.sol
```

Two auditors worked on the code for 2 days.

The audit's main target is security threats, i.e., what the community understanding would likely call "hacking", rather than regular use of the protocol. Functional

correctness (i.e., issues in “regular use“) is a secondary consideration. Functional correctness of most aspects (e.g., relative to low-level calculations, including units, scaling, quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing. Importantly, thorough integration testing in the setting of final use is also an aspect that is not effectively covered by human auditing and remains the responsibility of the development team.

PROTOCOL-LEVEL CONSIDERATIONS:

ID	Description	STATUS
P1	The new 2-step design is susceptible to frontrunning attacks	**INFO**

In the previous version of the protocol, users had to give approvals to the Pool contract before they could execute certain actions. For example, to swap x A tokens for B tokens, the user would have needed to approve an allowance of at least x A tokens to the Pool contract, because during the execution of the swap function a `transferFrom` takes place.

In this new version, the user, instead of approving allowances, can directly send the tokens needed for the action to the contract in advance. The two steps (1. sending the tokens, 2. execute the action) can in principle be executed in different transactions. The problem is that the protocol only checks whether the amount needed for the action has been sent, but not who has sent it. Therefore, if the user does not use a multicall to be sure that the two steps will be executed in this exact order and no other action will take place between them, an attacker can frontrun the user and make use of his tokens.

The CavalRe team has acknowledged this issue and stated that it is mitigated by the fact that the CavalRe frontend will be using a multicall, making this a problem only in the case of a user directly calling the contract without a multicall.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues that affect the functionality of the contracts. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: -User or system funds can be lost when third party systems misbehave. -DoS, under specific conditions. -Part of the functionality becomes unusable due to programming error.
LOW	Examples: -Breaking important system invariants, but without apparent consequences. -Buggy functionality for trusted users where a workaround exists. -Security issues which may manifest when the system evolves.

Issue resolution includes “dismissed” or “acknowledged” but no action taken, by the client, or “resolved”, per the auditors.

CRITICAL SEVERITY:

ID	Description	STATUS
C1	Malicious user can drain the pool if contract already contains enough or more LPTokens to complete a transaction	**RESOLVED** **(2ac2ae)**

In the new version of CavalRe, users can directly transfer the tokens needed for an action to the contract (stake, unstake, add or remove liquidity or multiswap) and afterwards execute this action. The protocol checks whether the amount needed for the action has been sent and if not, transfers or burns the remaining amount from the user’s address (who should have given approval to the Pool contract in advance).

Consider the following section of the code, which deals with the situation where an action requires sending LP tokens to the contract.

Pool::_multiswap

```
for (uint256 i; i < payTokens.length; i++) {
            address payToken = payTokens[i];
            uint256 amount = amounts[i];
            if (payToken == address(this)) {
                uint256 contractBalance = balanceOf(address(this));
                if (contractBalance == 0) {
                    _burn(sender, amount);
                } else if (contractBalance < amount) {
                    _burn(sender, amount - contractBalance);
                    _burn(address(this), contractBalance);
                }//Dedaub: the case contractBalance >= amount, i.e. the
```

```
//user has sent more than needed for the multiswap LP tokens, is not
//covered. If this happens, the LP tokens sent to the Pool are not burnt!!
            } else {
                .. .
        }
```

In the above code, the protocol will check whether the user has sent it the required amount of LP tokens. If the user has sent x LP tokens, but x is less than the amount needed for the action, it burns amount-x tokens from the user address and x tokens from the Pool contract (therefore amount LP tokens are burnt in total). But if the user has sent enough or more LP tokens than required (contractBalance >= amount), the protocol does not burn these tokens! A malicious user could thus attack and drain the protocol using the following steps:

1. Stake tokens with the protocol and get an x amount of LP tokens.
2. Transfer the x LP tokens to the Pool contract.
3. Ask the protocol to unstake an amount of y LP tokens. Here y can be any amount which is less than or equal to x. The protocol finds that its balance (x) is >= y (the amount required for the unstake action). Thus the transaction will proceed, but the LP tokens are not burnt!
4. The user can then repeat step 3 as many times as he wishes. It will always succeed since the protocol will always hold the x LP tokens.

We suggest burning `amount` LP tokens (`_burn (address(this)), amount)`), when `contractBalance` is >= `amount`.

HIGH SEVERITY:

[NO HIGH SEVERITY ISSUES]

MEDIUM SEVERITY:

[NO MEDIUM SEVERITY ISSUES]

LOW SEVERITY:

ID	Description	STATUS
L1	LPToken approval should check whether spender is blocked	**RESOLVED (2ac2ae)**
In the LPToken::approve function, one should check that the spender is not blocked, since a blocked spender with an approval could potentially still control tokens.		
L2	LPToken transferFrom should check whether spender is blocked	**RESOLVED (2ac2ae)**
In LPToken::transferFrom: one should check that the spender is not blocked, so as to avoid a blocked user from spending someone else's LP tokens.		
L3	A blocked user should not get a discount	**RESOLVED (2ac2ae)**
The Users::setDiscount function should check whether a user has been blocked, before awarding that user a discount.		

CENTRALIZATION ISSUES:

It is often desirable for DeFi protocols to assume no trust in a central authority, including the protocol's owner. Even if the owner is reputable, users are more likely to engage with a protocol that guarantees no catastrophic failure even in the case the owner gets hacked/compromised. We list issues of this kind below. (These issues should be considered in the context of usage/deployment, as they are not uncommon. Several high-profile, high-value protocols have significant centralization threats.)

[NO CENTRALISATION ISSUES]

OTHER/ ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend considering them.

ID	Description	STATUS
A1	Redundant reset of _isBlocked array when setting Protocol Fee Recipient	**RESOLVED (2ac2ae)**
The LPToken::setProtocolFeeRecipient function sets _isBlocked[recipient] = false at the end. But if one calls setProtocolFeeRecipient, and _isBlocked[recipient] == true, this line will be unreachable, because the function will revert beforehand. Hence this line is redundant and can be removed.		
A2	Missing cleanup of discount when user is blocked	**RESOLVED (2ac2ae)**
The Pool::setIsAllowed function should set the discount to zero when a user is blocked so as to perform proper cleanup and give a correct view from the UI.		

A3	Compiler bugs	INFO
The code is compiled with Solidity `0.8.19`. Version `0.8.19`, in particular, has some known bugs, which we do not believe affect the correctness of the contracts.		

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring through Dedaub Watchdog.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent protocols in DeFi. The founders, as well as Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.