CavalRe AMM

Smart Contract Security Assessment

August 18, 2023

ABSTRACT

Dedaub was commissioned to audit the CavalRe protocol implementation, located at the repository https://github.com/CavalRe/multiswap, commit number 5314924. The audit focuses on changes carried out since Dedaub's last audit of the protocol, dated July 2023, which can be found here CavalRe audit, July 2023. Since the previous audit, the user management functionality was updated, the protocol's functions were made to resolve to multiswap, and various minor enhancements were made.

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.

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:

[NO CRITICAL SEVERITY ISSUES]

HIGH SEVERITY:

ID	Description	STATUS
H1	Division by `_ratio` to transform actual to virtual amounts is applied twice	**RESOLVED (8c905d) (cc1751)**
In `LPToken::transfer` the `amount` variable, which is the actual amount the user wants to transfer, is divided by `_ratio`. This new quantity is the virtual amount and this is passed as an argument to the internal `_transfer` function. But `_transfer` expects this argument to describe an actual amount and divides it again by _ratio. The result is that the actual amount is divided by `_ratio`**2. The same problem occurs in `LPToken::approve,_approve`.		

MEDIUM SEVERITY:

ID	Description	STATUS
M1	When a user is set as not-allowed to interact with the protocol, his associates are not forced to remove their liquidity	**RESOLVED (8c905d)**
The owner can call `Pool::setAllowed(user, false)` to block the `user` from interacting with the protocol. This function also forces the user to remove his deposited		

liquidity. Although not only the user but also his `associates` will not be able to interact with the protocol from now on, the associates are not forced to remove their liquidity.		
M2	Users can take advantage of the non-symmetric conditions of `transfer` and `transferFrom`	**RESOLVED** **(8c905d)**
In `LPToken::transfer` the `onlyAllowed` modifier is applied, which will revert the transaction if the owner has paused the trading. In `LPToken::transferFrom` this modifier is not applied. The user can therefore call `transferFrom`, with his address as the `from` argument, to transfer tokens even if `_tradingPaused == 1`.		
M3	Minting and increasing the total supply are not executed in the right order	**RESOLVED** **(53a0452)**
The function `LPToken::distributeFee` executes the following operations (in this exact order): • Splits the fee amount into two parts. One goes to the protocol (`protocolAmount`) and the other to the LPs (`lpAmount`). • Mints `protocolAmount` of LP tokens for the protocolFeeRecipint • Increases the `_totalSupply` • Computes the new `_ratio`. The `balanceOf(protocolFeeRecipient)` immediately after the execution of this function will not be equal to `protocolAmount`, as someone would expect, but larger (`new_ratio/old_rato*protocolAmount`). The reason is that the new `_ratio` is computed after the LP tokens for the `protocolFeeRecipient` have been minted, therefore the `lpAmount` does not go exclusively to the LP providers but also part of it goes to the `protocolFeeRecipient`.		

LOW SEVERITY:

[NO LOW SEVERITY ISSUES]

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	Tokens with more than 18 decimals are not supported	**RESOLVED (53a0452)**
The protocol uses 18 decimals for accounting. Whenever a computation involves an ERC20 token with n decimals, its amount is multiplied by 10**(18-n), implicitly assuming that n will be less than 18. If n >18 the transaction will fail. *Check was inserted to fail gracefully if n > 18.*		
A2	Potentially missing checks	**RESOLVED (53a0452)**

	• In `Pool::stake/unstake` one could check whether `payAmount !=0` and revert otherwise. • In Pool::multiswap there is a missing check on the minReceiveAmounts array to ensure that it is the same length as the receiveTokens array.	
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.