Vesper Protocol audit

Smart Contract Security Assessment

Nov 10 2021

ABSTRACT

Dedaub was commissioned to perform a security audit of several smart contract modules of the Vesper.finance protocol.

The audit was performed on the contracts at (repo commit or archive title shared):

- https://github.com/bloqpriv/vesper-pools-v3, commit 71edb604135c444b7fd46fc0adf558be0b860067, continuing our previous audit, which had considered changes up to commit 57f4e471e7d5afc64e96d2dce717aaecb2d53515.
- https://github.com/bloqpriv/vesper-pools, from commit 8db2c976ebf340ae069c83ad2de6574037b2032f to 1060ffb53864208d3d89fc7c6cd03d0fed0c4740.
- https://github.com/bloqpriv/pf-payment-stream/, from commit e9d96fe6014b990ea1490aed51a926bfeb36c155 to 6a5ad4f70bba433899e0dcc49b5da42ca2f951f8.

We have previously audited the Vesper v2 and v3 pools and several strategy contracts. The current audit focuses mostly on verifying newly introduced changes to existing pools and strategies. Specifically, the audit scope includes:

- The PaymentStream and PaymentStreamFactory contracts
- The newly added Aave and dYdX flash loan functionality
- The PoolRewards and PoolRewardsUpgrader contracts
- The VFR pools and strategies
- The Earn pool and strategies
- The CompoundXY, CompoundLeverage and other Compound strategies
- The Maker strategies
- The Alpha strategies
- The Rari Fuse strategies

Two auditors worked on the task over the course of three days. We reviewed the code in significant depth, assessed the economics of the protocol and processed it through automated tools. We also decompiled the code and analyzed it, using our static analysis (incl. symbolic execution) tools, to detect possible issues.

Setting and Caveats

The audited code consists of some-800 LoC of diffs in the Vesper pools (primarily v3) and some-200 LoC in the payment stream module. Auditing diffs may miss issues that require a full understanding of context. Given the project's detailed test suite and our past audits, we believe that such issues are mitigated but cannot be precluded.

We refrain from commenting on issues that have been well-identified in the past (mainly unprotected swaps on AMMs) and mitigated by environmental conditions (i.e., performing swaps periodically before an attack becomes profitable).

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. Typically it can only be covered if we are provided with unambiguous (i.e., full-detail) specifications of what is the expected, correct behavior. In terms of functional correctness, we often trusted the code's calculations and interactions, in the absence of any other specification. Functional correctness 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.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues that affect the functionality of the contract. 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", by the client, or "resolved", per the auditors.

CRITICAL SEVERITY:

[No critical severity issues]

HIGH SEVERITY:

[No high severity issues]

MEDIUM SEVERITY:

ID	Description	STATUS
M1	PaymentStreamFactory could be reading stale data from oracle	**OPEN**

The `_getQuote()` method of the PaymentStreamFactory is used internally by the `usdToTokenAmount()` method which is queried by the different PaymentStreams to send the appropriate amount of different assets. The freshness of the data `_getQuote()` fetches using Chainlink's `FeedRegistry::latestRoundData()` is never checked.

The "best practice" we have observed regarding staleness of Chainlink oracle information (in the Chainlink code itself) is to declare a tolerance for staleness and disable the functionality when the information is more stale. The tolerance can be set to zero as a special value indicating "accept all stale information".

LOW SEVERITY:

[No low severity 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	ConvexStrategy::claimableRewardsCVX() replicates implementation-specific computation.	**OPEN**
The claimableRewardsCVX() method of the ConvexStrategy performs its computation by replicating the internal computation performed, but not externalized, by the convex Cvx contract. It should be noted that it can be invalidated by a future update to Cvx.sol.		
A2	Compiler bugs	**OPEN**
The contracts were compiled with the Solidity compiler v0.8.3 which, at the time of writing, has a known minor issue. More specifically the known compiler bug associated with Solidity compiler v0.8.3: • Memory layout corruption can happen when using abi.decode for the deserialization of two-dimensional arrays. We have reviewed the issue and do not believe it to affect 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, as well as a public bug bounty program.

ABOUT DEDAUB

Dedaub offers technology and auditing services for smart contract security. The founders, Neville Grech and Yannis Smaragdakis, are top researchers in program analysis. Dedaub's smart contract technology is demonstrated in the contract-library.com service, which decompiles and performs security analyses on the full Ethereum blockchain.