Vesper Synth, Pools, VUSD deltas for March ‘22

Smart Contract Security Assessment

Mar 9, 2022

ABSTRACT

Dedaub was commissioned to perform a security audit of several smart contract modules of three independent Vesper protocols.

The scope of the audit consisted of the deltas in the repos of 3 Vesper projects, while the report only lists the issues for the latter two, with the issues of the Vesper Synth protocol populating a separate report:

1. Recent changes (on multiple files) in the Vesper Synth protocol, in the repository https://github.com/bloqpriv/vesper-synth, up to commit 0e21e9796c56595d6c2c032556acf4a0d794f9dc (from commit 9a49b060051fa34f323eb699515a254e85f3c664).
2. Recent changes (on multiple files), of the Vesper Pools V3 protocol, in the repository https://github.com/bloqpriv/vesper-pools-v3, up to commit 3f3161b037f727572a107076620e6b26ed3fb3eb (from commit fe609958b54b97463d958d14e5e8e3a8bdfdbe16).
3. Recent changes (on multiple files), of the VUSD protocol, in the repository https://github.com/bloqpriv/vusd, up to commit 33692897131211f0ab51a7f95fc3103a6374578f (from commit 0f49e6d4c7a4fd81a8f4ccd3c1701a0b8288a92e).

We have audited all 3 projects previously, up to the listed earlier commits in the case of Vesper Synth and Vesper Pools V3, and up to similar levels (without direct access to the repository) for VUSD.

Setting and Caveats

Our earlier audits describe the setting and caveats for Vesper Synth and Vesper Pools V3. As a general warning, we note that an audit of small changes in a large protocol is necessarily out-of-context. We made a best-effort attempt to understand the changed lines of code and assess whether these changes are reasonable and do not introduce vulnerabilities. The audit, however, was restricted to the modified lines, and their interaction with the rest of the protocol is not always easy to assess.

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.

We list issues of Vesper Pools v3 without annotation, but explicitly flag issues of VUSD with [VUSD].

CRITICAL SEVERITY:

[No critical severity issues]

HIGH SEVERITY:

H1	Wrong logic for decimals of tokens in Convex4PoolStrategyMUSDPool	**RESOLVED**

The initialization function in Convex4PoolStrategyMUSDPool has wrong logic, which produces the wrong number of decimals for the tokens involved. Specifically:

```
function _init(
      address _crvPool,
      uint256 /* _n */
) internal virtual override {
      coins.push(IStableSwap(_crvPool).coins(0));
      coinDecimals.push(IERC20Metadata(coins[0]).decimals());
      for (uint256 i = 0; i < 3; i++) {
            coins.push(IStableSwap(THREEPOOL).coins(i));
            coinDecimals.push(IERC20Metadata(coins[i]).decimals());
      }
}
```

That is, coin 0 of the `_crvPool` is pushed first into the `coins` array. Then the three tokens of the `THREEPOOL` are pushed, but their decimals are read from the tokens in positions `0-2` of the `coins` array, i.e., they inadvertently read a different token's decimals.

It would seem to us that this would immediately result in wrong decimals if the tokens are MUSD, DAI, USDC, USDT, as expected of this pool. (I.e., decimals are 18, 18, 6, 6, respectively.)

It is also not clear if this `_init` function needs to exist in such generality in the first place. Both its parameters (i.e., not just `n`, but also `_crvPool`) seem to be known statically. The function has this form possibly only to satisfy an overriding requirement?

MEDIUM SEVERITY:

ID	Description	STATUS
M1	Migration of Fuse Pool in Rari strategies does not update all relevant state variables	**RESOLVED**

In the `migrateFusePool()` method of the `RariFuseStrategy` and the `RariFuseStrategyETH` contracts, the rewardToken and rewardDistributor are not updated.
This will leave the Strategy attempting to claim the rewards of the old pool, and the new rewards unclaimed.

ID	Description	STATUS
M2	Misconfiguration can allow Rari fuse strategies to be deployed in an unusable state	**OPEN**

In the constructor of the RariFusePool:

```
constructor(address _pool, address _swapManager, uint256 _fusePoolId,
       IFusePoolDirectory _fusePoolDirectory, string memory _name)
       CompoundStrategy(...)
{
  fusePoolId = _fusePoolId;
  fusePoolDirectory = _fusePoolDirectory;
  // Find and set the rewardToken from the fuse pool data
  (rewardDistributor, rewardToken) =
    _fusePoolDirectory.getRewardToken(_fusePoolId);
}
```

The call to `RariCore::getRewardToken()` setting the rewardToken and rewardDistributor fields can return two values of `address(0)`:

```
function getRewardToken(IFusePoolDirectory _fusePoolDir, uint256 _fusePoolId)
internal view
  returns (address _rewardDistributor, address _rewardToken)
{
  uint256 _success;
  address _comptroller = getComptroller(_fusePoolDir, _fusePoolId);
  bytes4 _selector = IComptroller(_comptroller).rewardsDistributors.selector;

  // Low level static call to prevent revert in case the Comptroller doesn't have
  // rewardsDistributors function exposed
  // which may happen to older Fuse Pools

  assembly {
    let x := mload(0x40) // Find empty storage location using "free memory
pointer"
    mstore(x, _selector) // Place signature at beginning of empty storage
    mstore(add(x, 0x04), 0) // Place first argument directly next to signature

    _success := staticcall(
          30000, // 30k gas
          _comptroller, // To addr
          x, // Inputs are stored at location x
          0x24, // Inputs are 36 bytes long
          x, // Store output over input (saves space)
          0x20
    ) // Outputs are 32 bytes long

    // Dedaub: This can return 0
    _rewardDistributor := mload(x) // Load the result
  }
```

```
    if (_rewardDistributor != address(0)) {
      _rewardToken = IRariRewardDistributor(_rewardDistributor).rewardToken();
    }
}
```

If this happens the Strategy will be left in a broken state and will need a migration in order to work properly.

M3	Swaps can be sandwiched	OPEN

There is a long-standing warning regarding swaps of profits of strategies in the Vesper Pools codebase. The threat is partly mitigated by the small amounts being swapped and by the use of two separate exchanges for the swap, so that an attacker would need to manipulate both. We reiterate the warning here in the face of the following code in CompoundLegerageAvancheStrategy (although similar code can be found elsewhere):

```
function _safeSwap(
      address _tokenIn,
      address _tokenOut,
      uint256 _amountIn
) internal override {
      // Removed UniV3 Oracle slippage check on Avalanche
      _safeSwap(_tokenIn, _tokenOut, _amountIn, 1);
}
```

The comment refers to weakening the protection that would exist in the Ethereum mainnet, by consulting an oracle (Uniswap v3 TWAP) before the swap. We note that Avalanche is also susceptible to sandwich attacks (e.g., https://www.coinfirm.com/blog/vee-finance-hack/).

Other apparently unsafe swaps in newly-audited code include instances in VesperCompoundXYStrategy (`_rebalanceBorrow`, `_claimRewardsAndConvertTo`).

LOW SEVERITY:

ID	Description	STATUS
L1	Strategies can leave funds behind upon migrating	**DISMISSED**

In two cases, strategies can leave funds behind when migrating to a new strategy.

The first case concerns the`_beforeMigration()` function of the CompoundStrategy, which doesn't claim any accrued rewards.

The second case is in the VesperCompoundXYStrategy contract where, upon migrating, it doesn't make sure to withdraw all the funds deposited in the VPool, withdrawing just enough to repay its Compound loan.

In both cases the remaining sums should be small.

We communicated these issues to the team, who dismissed them, motivated by the need to keep migrations as simple as possible, in order not to fail in case of emergency.

L2	Wrong token index in Convex4PoolStrategyMUSDPool::_depositToCurve	**RESOLVED**

The function Convex4PoolStrategyMUSDPool::_depositToCurve uses the same index into a 2-element and a 4-element array. If there is an underlying assumption that the deposit token can only be token[0] or token[1] of the two arrays, this assumption needs to be stated and checked in the code. Specifically:

```
function _depositToCurve(uint256 _amt) internal virtual override
returns (bool) {
      if (_amt != 0) {
            uint256[2] memory _depositAmounts;
            _depositAmounts[collIdx] = _amt;
            uint256[4] memory _depositAmountsZap;
            _depositAmountsZap[collIdx] = _amt;
  …
```

L3	Accounting for loss seems wrong in CrvPoolStrategyBase	**RESOLVED/ DISMISSED (left as-is in the cited method, fix applied to subclass)**

In `CrvPoolStrategyBase::rebalance` the code has the logic:

```
(uint256 _profit, uint256 _loss, uint256 _payback) = _generateReport();
IVesperPool(pool).reportEarning(_profit, _loss, _payback);
_reinvest();
if (!depositError) {
    uint256 _depositLoss =
      _realizeLoss(IVesperPool(pool).totalDebtOf(address(this)));
    IVesperPool(pool).reportLoss(_depositLoss);
}
```

However, the last line was originally:

```
    if (depositLoss > _loss)
      IVesperPool(pool).reportLoss(depositLoss - _loss);
```

The original logic seems correct and the same logic (`depositLoss - loss`) can be found elsewhere in the codebase (e.g., the EarnCrvSBTCPoolStrategy).

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	Low-level external call does not check that it calls a contract	**OPEN**

The getRewardToken() method of the RariCore contract contains a staticcall in inline assembly. The target of this call is not checked to be a contract, typically done using extcodesize. Although the target is taken from the fuse pool directory and will be a contract the check should either take place or at least be acknowledged in code as an optimization.

```
function getRewardToken(IFusePoolDirectory _fusePoolDir, uint256
_fusePoolId)  internal view
  returns (address _rewardDistributor, address _rewardToken)
{
  ...
  assembly {
    ...
    _success := staticcall(
          30000, // 30k gas
          _comptroller, // To addr
          x, // Inputs are stored at location x
          0x24, // Inputs are 36 bytes long
          x, // Store output over input (saves space)
          0x20
    ) // Outputs are 32 bytes long
    ...
  }
  ...
}
```

A2	Revert message in VesperCompoundXYStrategy seems copy-paste error	OPEN

In the VesperCompoundXYStrategy constructor, the following statement can be found:

```
require(address(IVesperPool(_vPool).token()) == borrowToken,
        "not-a-valid-dai-pool");
```

This seems like a copy-paste message and not one that applies here.

A3	[VUSD] Minor inconsistency for `type(uint256).max`	OPEN

In the Minter contract, the maximum 256-bit unsigned int value is referred to as both `type(uint256).max` and `MAX_UINT_VALUE`.

A4	Compiler bugs	INFO

Vesper Pools contracts were compiled with the Solidity compiler v0.8.3 which, at the time of writing, has a known minor issue. We have reviewed the issue and do not believe it to affect the contracts. More specifically the known compiler bug associated with Solidity compiler v0.8.3 is:

- Memory layout corruption can happen when using abi.decode for the deserialization of two-dimensional arrays.

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.