Vesper Pools Security Audit

coinspect

Smart Contract Audit
Vesper Pools

Prepared for Bloq • November 2020

Second Review v201230

1. Executive Summary

In November 2020, Bloq engaged Coinspect to perform a second source code review of their new Vesper platform. The objective of the audit was to continue evaluating the security of the smart contracts being developed.

The assessment was conducted on contracts from the Git repository at https://github.com/bloqpriv/bpools obtained on **October 21st**.

The following issues were identified during the assessment:

High Risk	Medium Risk	Low Risk
0	0	4

Only low risk findings related to coding best practices were found during this second engagement, and the ones previously reported have been properly addressed.

In December 2020, Coinspect reviewed the fixes for each finding and updated their status accordingly.

The following report details the tasks performed and the vulnerabilities found during this audit as well as several suggestions aimed at improving the overall code quality and warnings regarding potential issues.

2. Introduction

The focus of the audit was the current implementation of the Vesper Pools smart contracts including minor changes performed to the initial code reviewed in Coinspect's previous audit and the new features introduced. Additionally, the fixes to the previously reported vulnerabilities were verified to be correct.

The audit started on November 3rd and was conducted on the Git repository at https://github.com/bloqpriv/bpools. The last commit reviewed during this engagement was `d3eae096f0dd24b8103cf57012256bc4793775bc` from **October 21st**:

```
commit d3eae096f0dd24b8103cf57012256bc4793775bc
Merge: 819cbde b7d0729
Author: patidarmanoj10 <32006767+patidarmanoj10@users.noreply.github.com>
Date:   Wed Oct 21 09:42:59 2020 -0700

   Merge pull request #259 from bloqpriv/function-order

   Code restructure and reward test
```

The scope of the audit was limited to the following Solidity source files, shown here with their sha256sum hash:

```
b84afb00be0c4e6f9b837eae62b2de2f722c1875b198bc0407c4f7f6c8448171  ./Owned.sol
fe7aa619584c82a3663723c4ac10d6e410f34551ad3b62f7703d0313174b2238  ./Pausable.sol
6fe8acdba397fba7986ac073d20cc3c91dd66e90cf4f062a355afc9fe635a75f  ./interfaces/vesper/ICollateralManager.sol
5755f81b129d7ba147fad893f34808eb3e865a160b6885a6ac82fc96a73557e5  ./interfaces/vesper/IController.sol
769b19e6049dd8de1e1028a2a9bd32186a6d1d442ab03e26344390ef2233ba5d  ./interfaces/vesper/IVPool.sol
894f17446ad3ca7fa56c51e7c8391d8f2649b81aa4d205742f2e664475f87632  ./interfaces/vesper/IStrategy.sol
a7f9266f171e399d0a8fc6d51421cad313da7642d8d50cc2ff690383084772d7  ./interfaces/bloq/IAddressListExt.sol
acac3631ea6e43d2d51361673855c66e8b459d9c4661bc14d265b768f0e51968  ./interfaces/maker/IMakerDAO.sol
46124db513cedef14c7540ca196b931376ba3a5aae5b2e16cde9303275060b7b  ./interfaces/uniswap/IUniswapV2Router01.sol
9dfc160b08756faa53497fb457754d5c9c52dabb07f22f17d6971f8cc47b3a68  ./interfaces/uniswap/IUniswapV2Router02.sol
9b60b8301068b57ee43b241f62dfe8cf1f5d7ab567f22fb7ebc4a4577574be39  ./interfaces/token/IToken.sol
3ba84242cda8b8314dae39b603e55149685a3920d93a86e481b8a14fb6c0fc07  ./interfaces/aave/IAave.sol
3e7242d1b463376fecb1f492216aaccae28dcbb8c15b0d7fdf034c3aa76af7df  ./Timelock.sol
8e92fba7b79a4e1a301fe4db552167da0b2bd2ce23f35beb393ddcf580b6c196  ./token/VSPR.sol
a624f3e6f0d7726622a5c5d935b6250d3e6b1251607bcd01702628a1932eae40  ./pools/VETH.sol
3ffd95310c7260b5d772ce69374cd7095a5bbb6fac2f87711e5f5deb4b1df143  ./pools/VTokenBase.sol
7bb967d2513ee971754680c9e03bb27e3aae8f94f0248d67100b9404837c6cb6  ./pools/PoolShareToken.sol
8ee6501c6f081c9b5d0695d28b913e339258cbe9f2ec7ec7385aa20cfd0f6b61  ./pools/PoolRewards.sol
f5000ff139ed30fcccbf4be8f7228397e2eaeb2987cf3ef3185d80abf2f16ba3  ./Controller.sol
dec3566abf09d0c7b95aab13cb58192e969b2f64e9c24e272d059d843e31aa2a  ./strategies/AaveStrategy.sol
151b94e23509391af810cf762c7031d6602f0653070cffb49e4f5b8fe826ae42  ./strategies/AaveMakerStrategyETH.sol
6b0effde16705617fa47406fcec1e4a9824a1f35dd3723b79fa2cd2a8d8dbc9f  ./strategies/AaveMakerStrategy.sol
a6c30e1c2e3e5df5e86d7a905ebcc80f38c5c84b16debff02b4e3c3a525a3b75  ./strategies/CollateralManager.sol
b0045ed87fb560f0877530e5a9424b8fe76b9148d115fd5f04c37ad957cb2522  ./Migrations.sol
```

Even though the target repository included more smart contracts besides those listed above, they were not audited during this engagement as per Vesper's team request.

These smart contracts interact with several additional contracts deployed by other projects which were not part of this review such as: MakerDAO, Uniswap and AAVE. A full review

comprising these interactions should be performed in order to fully assess the security of the project.

3. Assessment

Vesper's end goal is to become an AI-driven resource management engine for the tokenized world. The code reviewed during this engagement implements crypto asset pools that enable users to generate earnings by supplying liquidity to existing 3rd party pools in the DeFi ecosystem.

The code reviewed is currently under active development and consists of several pools which handle different types of collateral deposits. These deposits are in turn deposited by the pool in MakerDAO to generate the DAI stable coin. One investment strategy was included in the repository, which generates earnings by supplying DAI to AAVE.

For example, this is how the ETH pool works:

1. User deposits ETH and receives VETH (ERC-20 tokens) representing his shares in the pool.
2. The ETH is locked in MakerDAO, and a stablecoin DAI loan is created.
3. This DAI is deposited in Aave, the ETH pool receives a DAI ERC-20 tokens in exchange.
4. A periodic rebalance (triggered off-chain) process is responsible for keeping the configured collateral ratio between MakerDAO and Aave. In case funds in Aave grow in excess of what is required, the rebalance mechanism takes this excess interest from Aave, swaps it for ETH with Uniswap and deposits it in MakerDAO.

The new features that were added in this release and were the focus of this audit are implemented in the following new contracts:

- PoolRewards.sol
- AaveStrategy.sol
- Timelock.sol
- AaveStrategyETH.sol

The remaining contracts that were reviewed included several refactoring modifications and minor functionality and code quality changes that were also reviewed.

The new **AaveStrategy** strategy invests all the collateral tokens in the Aave platform in order to profit from the interests generated. Most of its functions are accessible only from its Controller or its pool. The older AaveMakerStrategy has been refactored, in this case the functions rebalanceCollateral and rebalanceEarned are public in order to let any user to trigger the rebalancing of funds. This differs from the new AaveStrategy where only the pool can trigger a rebalance.

The new **AaveStrategyETH** contract is an AaveStrategy that utilizes the Wrapped ETH (WETH) token contract address [0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2](0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2) as collateral.

The new **PoolRewards** contract is responsible for distributing rewards to depositors based on the amount of shares in their possession. It is also managed by the Controller contract.

A new feature has been added to allow time-locked stage changes to the system before they are applied as a security mechanism. This is implemented in the **Timelock** smart contract, which is based on compound-protocol/Timelock.sol, and it is used by the GovernorAlpha.sol token contract, which was not reviewed during this audit. The Timelock is managed by its admin, which is the only address allowed to queue, cancel or execute actions. The actions must be queued first, and an ETA is enforced by the Timelock. There is a minimum delay of 2 days hardcoded in the contract, actions expire after a grace period of 14 days. The Timelock responsibility is to enforce the ETA parameter for the action performed by the admin.

The **fee collection mechanism** in the strategies takes place during the rebalancing of earnings. Fees are calculated as a percentage (configurable for each pool in the Controller) of pool earnings and are sent to a fee collector address (also configurable per pool in the Controller contract).

Additionally, two new functions were added to the Controller contract: `executeTransaction` and `executeTransactions` that allow proxying arbitrary calls through it. We suggest the `onlyOwner` modifier is added to the `executeTransactions` function as well, in order to catch the wrong caller earlier and just in case the function is modified in the future not to rely on `executeTransaction`. Because `executeTransaction` reverts if the call fails, if any of the transactions passed to `executeTransactions` fail, the whole transaction will revert costing gas to the caller. Depending on how transactions passed to the `executeTransactions` function are being constructed off-chain, and if they can be influenced by a third party, an attacker could force the Controller contract owner to waste gas.

It is worth noting the **Controller** contract/account has the capability to affect most parameters in the platform and access to user deposited funds and its security must be considered a priority. The team has already manifested its intention to use a multisig contract to handle this role. Coinspect recommends considering splitting this role into several roles with different responsibilities, for example, separating the management of pool rewards from the ability to withdraw funds.

The contracts are compiled with Solidity version 0.6.12. A few warnings are reported by the compiler and detailed in findings VSP-010 Unused parameter in hook functions in PoolShareToken and VSP-011 Function state mutability should be restricted to pure in PoolShareToken

The repository includes a set of tests for each contract: 113 tests pass and 14 fail; these failures are all related to UniswapV2 reverting (see Appendix 1 - Failed tests). These tests only check basic functionality and it is recommended these are further expanded to include more edge case scenarios. The **test coverage** could be improved, in particular for the contracts: Controller.sol and Timelock.sol. The test coverage output is provided in Appendix 2 - Tests coverage.

Several findings related to smart contracts coding best practices are included in this report, these can be easily fixed and will result in a better quality and easier to maintain source code.

Lastly, the constant 1e18 is hardcoded in several contracts, care must be taken when using tokens with different than **18 decimals precision** in the future.

4. Summary of Findings

ID	Description	Risk	Fixed
VSP-010	Timelock contract does not check for 0 address admin	Low	✔
VSP-011	PoolRewards contract does not check for 0 address controller	Low	✔
VSP-012	Function access control modifiers not used or used inconsistently	Low	✔
VSP-013	executeTransactions function missing return value	Low	✔

5. Remediations

During December 2020 Coinspect verified the findings that Vesper addressed had been correctly fixed.

The following table lists the findings that were fixed and the corresponding fix status and commit:

ID	Description	Status
VSP-010	Timelock contract does not check for 0 address admin	Fixed 9e7ab30c8eddf1a9a13076a524ed032140ebc3ac
VSP-011	PoolRewards contract does not check for 0 address controller	Fixed 9e7ab30c8eddf1a9a13076a524ed032140ebc3ac
VSP-012	Function access control modifiers not used or used inconsistently	Fixed cbc89c608d71ad98422176962f8d7a8da66dbb44
VSP-013	executeTransactions function missing return value	Fixed 9e7ab30c8eddf1a9a13076a524ed032140ebc3ac

6. Findings

VSP-010	Timelock contract does not check for 0 address admin

Total Risk	Impact	Location
Low	Low	Timelock.sol
Fixed	**Likelihood**	
✔	Low	

Description

The `Timelock` contract does not verify the provided parameter `admin` is not the address 0:

```
constructor(address admin_, uint256 delay_) public {
    require(delay_ >= MINIMUM_DELAY, "Timelock::constructor: Delay must exceed minimum delay.");
    require(delay_ <= MAXIMUM_DELAY, "Timelock::constructor: Delay must not exceed maximum delay.");
    admin = admin_;
    delay = delay_;
}
```

This check is present in the constructors of the other project's contracts reviewed.

Recommendation

Require the new admin to be different than address 0 to prevent deployment mistakes.

Status

This issue was fixed in commit 9e7ab30c8eddf1a9a13076a524ed032140ebc3ac.

VSP-011 PoolRewards contract does not check for 0 address controller

Total Risk	Impact	Location
Low	Low	PoolRewards.sol
Fixed	**Likelihood**	
✔	Low	

Description

The `PoolRewards` contract does not verify the provided parameter `_controller` is not the address 0:

```
constructor(
    string memory name,
    string memory symbol,
    address _controller
) public ERC20(name, symbol) {
    controller = IController(_controller);
}
```

This check is present in the constructors of the other project's contracts reviewed.

Recommendation

Require the new controller to be different than address 0 to prevent deployment mistakes.

Status

This issue was fixed in commit 9e7ab30c8eddf1a9a13076a524ed032140ebc3ac.

VSP-012 Function access control modifiers not used or used inconsistently

Total Risk	Impact	Location
Low	Low	AaveMakerStrategy.sol CollateralManager.sol
Fixed	**Likelihood**	
✔	Low	

Description

The `withDrawAll` function does not employ the existing `onlyController` modifier, the check is rewritten inside the function:

```
function withdrawAll() public override {
    require(_msgSender() == address(controller), "Not a controller");
    _moveDaiToMaker(cm.getVaultDebt(vaultNum));
    require(cm.getVaultDebt(vaultNum) == 0, "Debt should be 0");
    cm.withdrawCollateral(vaultNum, totalLocked());
    collateralToken.transfer(pool, collateralToken.balanceOf(address(this)));
}
```

A similar pattern is observed in the CollateralManager contract, modifiers are used in some cases, and requires statements in some others.

It is advised to maintain the same style throughout the project for consistency's sake.

Recommendation

Replace the require inside the function with the `onlyController` modifier to improve code base consistency.

Status

This issue was fixed in commit cbc89c608d71ad98422176962f8d7a8da66dbb44.

VSP-013 executeTransactions function missing return value

Total Risk	Impact	Location
Low	Low	Controller.sol
Fixed	**Likelihood**	
✔	Low	

Description

The executeTransactions function is expected to return an `uint256` value but its implementation does not return anything:

```
    function executeTransactions(
        address[] memory targets,
        uint256[] memory values,
        string[] memory signatures,
        bytes[] memory calldatas
    ) external returns (uint256) {
        require(targets.length != 0, "Must provide actions");
          require(targets.length == values.length && targets.length == signatures.length &&
targets.length == calldatas.length, "Transaction data mismatch");

        for (uint256 i = 0; i < targets.length; i++) {
            executeTransaction(targets[i], values[i], signatures[i], calldatas[i]);
        }
    }
```

This inconsistent behaviour might confuse the contract caller.

Recommendation

Return a value or remove it from the function declaration.

Status

This issue was fixed in commit 9e7ab30c8eddf1a9a13076a524ed032140ebc3ac.

7. Conclusion

Overall, Coinspect did not find any high risk security vulnerabilities introduced by the new features added to Vesper pools since its previous audit that would result in stolen or lost user funds. However, the reviewed code can be improved in several aspects as suggested in this report, including:

- A more throughout set of tests involving multiple users, several operations, edge cases and failures in transactions with external components.
- The Controller contract could be splitted into several roles with different access levels and responsibilities.
- Improved test coverage.
- Code quality best practices: unused parameters, constant token precision, etc.

It is important to note that the security of the pools will depend heavily on its interactions with the external DeFi smart contracts and platforms utilized by Vesper such as:

- 3rd party liquidity providers integrated in the future
- Aave
- Uniswap
- MakerDAO

The complexity added by these components implies an elevated risk, and for that reason Coinspect suggests a full assessment of the system is performed before deployment to mainnet. It is also recommended that the first releases are only allowed to manage a limited amount of funds.

8. Disclaimer

The information presented in this document is provided as is and without warranty. Vulnerability assessments are a “point in time” analysis and as such it is possible that something in the environment could have changed since the tests reflected in this report were run. This report should not be considered a perfect representation of the risks threatening the analysed system, networks, and applications.

Appendix 1 - Failed tests

```
113 passing (43m)
 14 failing

 1) Contract: VBTC Pool
      Basic functionality tests
        Should resurface before withdraw when underwater:
    Error: Returned error: VM Exception while processing transaction: revert UniswapV2:
LOCKED -- Reason given: UniswapV2: LOCKED.
     at Context.<anonymous> (test/vbtc.js:70:18)
     at process._tickCallback (internal/process/next_tick.js:68:7)

 2) Contract: VBTC Pool
      Basic functionality tests
        Should be able to rebalanceEarned:
    Error: Returned error: VM Exception while processing transaction: revert UniswapV2:
LOCKED -- Reason given: UniswapV2: LOCKED.
     at Context.<anonymous> (test/vbtc.js:337:22)
     at process._tickCallback (internal/process/next_tick.js:68:7)

 3) Contract: VBTC Pool
      Basic functionality tests
        Should resolve underwater scenario:
    Error: Returned error: VM Exception while processing transaction: revert UniswapV2:
LOCKED -- Reason given: UniswapV2: LOCKED.
     at Context.<anonymous> (test/vbtc.js:354:18)
     at process._tickCallback (internal/process/next_tick.js:68:7)

 4) Contract: VWETH Pool
      Basic functionality tests
        Should sweep erc20 for veth:
    Error: Returned error: VM Exception while processing transaction: revert UniswapV2:
LOCKED -- Reason given: UniswapV2: LOCKED.
     at Object.swapEthForToken (test/tokenSwapper.js:26:15)
     at process._tickCallback (internal/process/next_tick.js:68:7)

 5) Contract: VETH Pool using WETH
      Basic functionality tests
        Should sweep erc20 for veth:
    Error: Returned error: VM Exception while processing transaction: revert UniswapV2:
LOCKED -- Reason given: UniswapV2: LOCKED.
     at Object.swapEthForToken (test/tokenSwapper.js:26:15)
     at process._tickCallback (internal/process/next_tick.js:68:7)

 6) Contract: VETH Pool using WETH
      Basic functionality tests
        Should be able to rebalanceEarned:
    Error: Returned error: VM Exception while processing transaction: revert UniswapV2:
LOCKED -- Reason given: UniswapV2: LOCKED.
     at Object.swapEthForToken (test/tokenSwapper.js:26:15)
     at process._tickCallback (internal/process/next_tick.js:68:7)

 7) Contract: VETH Pool using WETH
```

```
      Basic functionality tests
        Should be able to rebalanceEarned with 0 interestFee:
    Error: Returned error: VM Exception while processing transaction: revert UniswapV2:
LOCKED -- Reason given: UniswapV2: LOCKED.
     at Object.swapEthForToken (test/tokenSwapper.js:26:15)
     at process._tickCallback (internal/process/next_tick.js:68:7)

 8) Contract: VETH Pool
      Basic function tests
        Should update CM and strategy and after that rebalance should work:
    Error: Returned error: VM Exception while processing transaction: revert Transaction
execution reverted. -- Reason given: Transaction execution reverted..
     at Context.<anonymous> (test/veth.js:216:24)
     at process._tickCallback (internal/process/next_tick.js:68:7)

 9) Contract: VETH Pool
      Basic function tests
        Should redeem full after earning some interest:
    Error: Returned error: VM Exception while processing transaction: revert UniswapV2:
LOCKED -- Reason given: UniswapV2: LOCKED.
     at Context.<anonymous> (test/veth.js:296:22)
     at process._tickCallback (internal/process/next_tick.js:68:7)

 10) Contract: VKNC Pool
      Basic functionality tests
        Should sweep erc20 for vknc:
    Error: Returned error: VM Exception while processing transaction: revert UniswapV2:
LOCKED -- Reason given: UniswapV2: LOCKED.
     at Object.swapEthForToken (test/tokenSwapper.js:26:15)
     at process._tickCallback (internal/process/next_tick.js:68:7)

 11) Contract: VMANA Pool
      Basic functionality tests
        Should sweep erc20 for vmana:
    Error: Returned error: VM Exception while processing transaction: revert UniswapV2:
LOCKED -- Reason given: UniswapV2: LOCKED.
     at Object.swapEthForToken (test/tokenSwapper.js:26:15)
     at process._tickCallback (internal/process/next_tick.js:68:7)

 12) Contract: VMANA Pool
      Basic functionality tests
        Should be able to rebalanceEarned:
    Error: Returned error: VM Exception while processing transaction: revert UniswapV2:
LOCKED -- Reason given: UniswapV2: LOCKED.
     at Object.swapEthForToken (test/tokenSwapper.js:26:15)
     at process._tickCallback (internal/process/next_tick.js:68:7)

 13) Contract: VMANA Pool
      Basic functionality tests
        Should resolve underwater scenario:
    Error: Returned error: VM Exception while processing transaction: revert UniswapV2:
LOCKED -- Reason given: UniswapV2: LOCKED.
     at Context.<anonymous> (test/vmana.js:314:19)
     at process._tickCallback (internal/process/next_tick.js:68:7)

 14) Contract: VUSDC Pool
      Basic functionality tests
```

```
        Should sweep erc20 for vusdc:
    Error: Returned error: VM Exception while processing transaction: revert UniswapV2:
LOCKED -- Reason given: UniswapV2: LOCKED.
     at Object.swapEthForToken (test/tokenSwapper.js:26:15)
     at process._tickCallback (internal/process/next_tick.js:68:7)
```

Appendix 2 - Tests coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	70.83	42.39	70	72.22	
Controller.sol	74	45.45	64.71	74.51	... 127,131,146
Owned.sol	68.75	37.5	71.43	70.59	62,63,71,72,73
Pausable.sol	100	75	100	100	
Timelock.sol	58.54	31.25	50	58.54	... ,99,101,123
contracts/interfaces/aave/	100	100	100	100	
IAave.sol	100	100	100	100	
contracts/interfaces/bloq/	100	100	100	100	
IAddressListExt.sol	100	100	100	100	
contracts/interfaces/maker/	100	100	100	100	
IMakerDAO.sol	100	100	100	100	
contracts/interfaces/token/	100	100	100	100	
IToken.sol	100	100	100	100	
contracts/interfaces/uniswap/	100	100	100	100	
IUniswapV2Router01.sol	100	100	100	100	
IUniswapV2Router02.sol	100	100	100	100	
contracts/interfaces/vesper/	100	100	100	100	
ICollateralManager.sol	100	100	100	100	
IController.sol	100	100	100	100	
IStrategy.sol	100	100	100	100	
IVPool.sol	100	100	100	100	
contracts/pools/	86.45	60.29	87.01	86.36	
PoolRewards.sol	96.97	87.5	90.91	97.06	55
PoolShareToken.sol	98.46	54.55	91.67	98.48	171
VBTC.sol	100	100	100	100	
VETH.sol	100	66.67	100	95.45	22
VKNC.sol	100	100	100	100	
VMANA.sol	100	100	100	100	
VTokenBase.sol	90.91	75	94.44	91.07	... 126,127,128
VUSDC.sol	100	100	100	100	
VVSPR.sol	40.54	16.67	53.85	42.11	... 6,97,98,100
contracts/strategies/	86.19	60.53	86.79	86.34	
AaveMakerStrategy.sol	81.94	61.54	89.47	82.07	... 300,301,302
AaveMakerStrategyBTC.sol	100	100	100	100	
AaveMakerStrategyETH.sol	100	100	100	100	
AaveMakerStrategyMana.sol	100	100	100	100	
AaveStrategy.sol	81.82	57.14	66.67	82.14	... 106,107,116
AaveStrategyBTC.sol	100	100	100	100	
AaveStrategyETH.sol	100	100	100	100	
AaveStrategyKNC.sol	100	100	100	100	
AaveStrategyUSDC.sol	100	100	100	100	
CollateralManager.sol	91.45	58.33	90.63	91.53	... 223,258,259
VSPRStrategy.sol	88.24	66.67	100	88.57	69,70,71,72
contracts/test/	100	100	100	100	
IUniswapFactoryTest.sol	100	100	100	100	
IUniswapRouterTest.sol	100	100	100	100	
contracts/token/	11.05	3.26	20.51	11.9	
GovernorAlpha.sol	0	0	0	0	... 448,449,452
VSPR.sol	100	50	100	100	
VSPRGovernanceToken.sol	5.36	3.57	30	7.14	... 214,218,219
All files	69.24	41.53	74.43	70.11	