Vesper Pools Security Audit

coinspect

Smart Contract Audit
Vesper Pools

Prepared for Bloq • December 2020

Third Review v201230

1. Executive Summary

In November 2020, Bloq engaged Coinspect to perform a third 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 **Dec 1st**.

The following issues were identified during the assessment:

High Risk	Medium Risk	Low Risk
0	1	1

A medium risk vulnerability (Unprotected Uniswap swaps vulnerable to price manipulation) was found during this audit related to how Vesper blindly trusts Uniswap spot price for its token swaps. Only one low risk finding (Duplicate contract names for ManagerInterface and VatInterface) related to coding best practices was found during this third engagement.

In December 2020, Coinspect updated the fix status for each finding with Vesper's feedback.

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 this audit were three new Vesper strategies that were added to the existing ones:

- Compound Strategy
- Yearn Strategy
- VSPR strategy and the related VVSPR pool and VSPR token

The audit started on November 30th and was conducted on the Git repository at https://github.com/bloqpriv/bpools. The last commit reviewed during this engagement was `95c03acba4647d81ec0105904865950d1bd88781` from **December 1st**:

```
commit 95c03acba4647d81ec0105904865950d1bd88781
Merge: 47ea32b 3060fcd
Author: patidarmanoj10 <32006767+patidarmanoj10@users.noreply.github.com>
Date:   Tue Dec 1 11:17:14 2020 -0800

   Merge pull request #300 from bloqpriv/yVault

   yearn strategy
```

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

```
223e40d50d93268b31aa1b1481505ff2c01815c24641b93e75fce6f6996ce11c  ./strategies/YearnEarnStrategyEthDai.sol
eeb88ee813845230a688169ba7b11c7741c41f7c8b3206dc6e5f173bdaa81101  ./strategies/YearnVaultStrategyEthDai.sol
eaae73510696841e52f1ea1ef3c006bb95049704af943803b536165aeac7ceb5  ./strategies/YearnMakerStrategy.sol
fa85b187dff8bc04015414313b887315a6d4686c16202c9cdfd6791ef1e374e7  ./strategies/CompoundStrategy.sol
60f9406eb9fc1953895cc1cc5587bf1acaa9235db9a02de8d0a12bbe1c141683  ./strategies/CompoundStrategyUSDC.sol
302c8e4d978d7818db00122b5fc671fbe65021edcfb0c1ef78fe501e22d4e19b  ./strategies/CompoundStrategyETH.sol
07a768e11ec827f754e282ddbd8dcf0e8cc224ea8f51df5f92ceb5da332782b7  ./strategies/VSPRStrategy.sol
fa4d7d6f8b24424bd78f0122c14b5aec3f1181a1cae47258c94bd157b9c699e8  ./token/VSPR.sol
5b8178ab49a45baa7c70d563b0af7785cd87161dafdb5404c631389fe3a7ba75  ./pools/VVSPR.sol
```

These smart contracts interact with multiple other contracts deployed by other projects which were not part of this review such as: MakerDAO, Uniswap, Compound and Yearn. 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 and strategies which handle different types of collateral deposits and invest them in different projects. 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:

- YearnMakerStrategy.sol
- YearnVaultsStrategyEthDai.sol
- YearnEarnStrategyEthDai.sol
- CompoundStrategy.sol
- CompoundStrategyUSDC.sol
- CompoundStrategyETH.sol
- VSPRStrategy.sol
- VSPR.sol
- VVSPR.sol

These new contracts already take into account comments from Coinspect previous reports, such as the improved handling of tokens with different than 18 decimals precision and utility conversion functions when needed.

The new **YearnMakerStrategy** strategy invests all the collateral tokens in the Yearn platform in order to profit from the interests generated. The collateral token is deposited in a Maker vault in order to borrow DAI. Most of its functions are accessible only from its Controller or its pool. All rebalance functions (`rebalance`, `rebalanceCollateral` and `rebalanceEarned`) are public in order to let any user trigger the rebalancing of funds. The new **YearnVaultsStrategyEthDai** and **YearnEarnStrategyEthDai** contracts are

YearnMakerStrategies that utilize the Wrapped ETH (WETH) token (contract address 0xC02aaa39b223fe8d0a0e5c4f27ead9083c756cc2) as collateral.
The **YearnEarnStrategyEthDai** deposits funds in the yDAI contract at 0x16de59092dae5ccf4a1e6439d611fd0653f0bd01.
The **YearnVaultsStrategyEthDai** deposits funds in the yVault contract at 0xACd43E627e64355f1861cEC6d3a6688B31a6F952.

The new **CompoundStrategy** strategy invests all the collateral tokens in the Compound platform in order to profit from the interests generated. The strategy is capable if claiming COMP tokens (via Compound's Unitroller contract at 0x3d9819210A31b4961b30EF54bE2aeD79B9c9Cd3B), which in turn get swapped for the collateral token. Again, most of these strategy functions are accessible only from its Controller or its pool. The rebalance function in this contract (`rebalance`) is public in order to let any user trigger the rebalancing of funds. As explained by the Vesper team, this strategy implementation of the `sweepErc20` function does not allow sweeping the strategy's reserved tokens because it holds tokens in contrast with the rest of the reviewed strategies. Another difference with the other strategies is that the `deposit` function can be accessed by anybody, not only the pool. The new **CompoundStrategyUSDC** is a CompoundStrategy with USDC token as collateral (0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48) and cUSDC token as cToken (0x39AA39c021dfbaE8faC545936693aC917d5E7563). The new **CompoundStrategyETH** is a CompoundStrategy with WETH token as collateral and cETH token (0x4ddc2d193948926d02f9b1fe9e1daa0718270ed5) as cToken.

The new **VSPRStrategy** strategy keeps a list of pools obtained from its controller. In this case, only the pool is allowed to trigger a rebalance. The rebalance function triggers earnings rebalance for each pool in the list, once each time the rebalance function is called, until the whole list is traversed, then it starts from the first one again. During the earnings rebalance, the founders fee is transferred to the controller's `founderVault`. The new **VVSPR** is a PoolShareToken used in conjunction with the VSPRStrategy. The new **VSPR** contract implements the VesperToken governance token. Its `mint` function only permits the owner minting a maximum of `INITIAL_MINT_LIMIT` (10000000 * (10**18)) tokens during the first year.

As observed in previous assessments, the **Controller** contract/account has the capability to affect most parameters in the new pools and strategies as well, 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 cToken `exchangeRateStored` function (instead of the `exchangeRateCurrent` function which accrues interests) is used to calculate fees and convert to collateral.

The contracts are compiled with Solidity version 0.6.12. A few warnings are reported by the compiler at build time, see Duplicate contract names for ManagerInterface and VatInterface.
The repository includes a set of tests for each contract, after a quick fix of a typo in `yearn-earn-weth.js`: 191 tests passed, 13 did not finish and 5 failed (82 tests more than previous review). From those failures, only one is related to the new functionality in scope for this audit, the full output is included in Appendix 1 - Failed test in yearn-yVault-weth.js.

Coinspect auditors investigated the test failure and determined it was caused by a mistake in the test itself and was not relevant security wise. The **test coverage** could not be evaluated because it was not working properly during this review. Coinspect strongly recommends this problem to be addressed in order to understand if all the functionality is being tested.

4. Summary of Findings

ID	Description	Risk	Fixed
VSP-014	Duplicate contract names for ManagerInterface and VatInterface	Low	✘
VSP-015	Unprotected Uniswap swaps vulnerable to price manipulation	Medium	✘

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-014	Duplicate contract names for ManagerInterface and VatInterface	Will fix
VSP-015	Unprotected Uniswap swaps vulnerable to price manipulation	Won't fix

More detailed status information can be found in each finding.

6. Findings

VSP-014	Duplicate contract names for ManagerInterface and VatInterface

Total Risk
Low

Fixed
✘

Impact
Low

Likelihood
Low

Location
YearnMakerStrategy.sol
AaveMakerStrategy.sol

Description

The new `YearnMakerStrategy` contract duplicates the contract names for `VatInterface` and `ManagerInterface` already found in the existing `AaveMakerStrategy` contract as reported by the compiler at build time:

```
> Duplicate contract names found for ManagerInterface.
> This can cause errors and unknown behavior. Please rename one of your contracts.

> Duplicate contract names found for VatInterface.
> This can cause errors and unknown behavior. Please rename one of your contracts.
```

Recommendation

Rename the duplicated interface contracts so they do not overlap.

Status

The Vesper team has stated they will be fixing this issue as soon as possible. This strategy is not part of the beta release.

VSP-015 Unprotected Uniswap swaps vulnerable to price manipulation

Total Risk	Impact	Location
Medium	High	YearnMakerStrategy.sol, CompoundStrategy.sol VSPRStrategy.sol, VVSPR.sol, VTokenBase.sol, AaveMakerStrategy.sol
Fixed ✘	Likelihood Medium	

Description

Vesper utilizes the `swapExactTokensForTokens` function from the Uniswap Router V2 contract when it needs to swap tokens.

Uniswap documentation states this function swaps an exact amount of input tokens for as many output tokens as possible, along the route determined by the path.

```
function swapExactTokensForTokens(
 uint amountIn,
 uint amountOutMin,
 address[] calldata path,
 address to,
 uint deadline
) external returns (uint[] memory amounts);
```

The second parameter, `amountOutMin` indicates the minimum amount of output tokens that must be received for the transaction not to revert. This parameter is intended for the caller to express what is the minimum exchange rate it is willing to accept from the AMM.

In Vesper, this parameter is always set to 1:

- `pools/VTokenBase.sol: uniswapRouter.swapExactTokensForTokens(amt, 1, path, address(this), now + 30);`
- `pools/VVSPR.sol: uniswapRouter.swapExactTokensForTokens(amt, 1, path, address(this), now + 30);`
- `strategies/CompoundStrategy.sol: uniswapRouter.swapExactTokensForTokens(amt, 1, path, address(this), now + 30);`
- `strategies/YearnMakerStrategy.sol: uniswapRouter.swapExactTokensForTokens(balance, 1, path, address(this), now + 30);`

- `strategies/YearnMakerStrategy.sol: uniswapRouter.swapExactTokensForTokens(tokenNeeded, 1, path, address(this), now + 30);`
- `strategies/VSPRStrategy.sol: uniswapRouter.swapExactTokensForTokens(`
- `strategies/AaveMakerStrategy.sol: uniswapRouter.swapExactTokensForTokens(balance, 1, path, address(this), now + 30);`
- `strategies/AaveMakerStrategy.sol: uniswapRouter.swapExactTokensForTokens(tokenNeeded, 1, path, address(this), now + 30);`

This means Vesper will accept any exchange rate provided from Uniswap as valid. This could allow attackers to manipulate the AMM exchange rate right before a Vesper transaction is executed and/or trigger themselves a Vesper transaction involving swaps after manipulating the token pair price.

Token swaps can be triggered by anybody, for example via the `rebalance` function in Vesper strategies.

Uniswap documentation clearly warns against this practice, for example in https://uniswap.org/docs/v2/smart-contract-integration/trading-from-a-smart-contract/ it states:

> **When trading from a smart contract, the most important thing to keep in mind is that access to an external price source is *required*. Without this, trades can be frontrun for considerable loss.**
>
> **First you must use an external price source to calculate the safety parameters for the function you'd like to call. This is either a minimum amount received when selling an exact input or the maximum amount you are willing to pay when a buying an exact output amount**
>
> *Safety Considerations*
>
> Because Ethereum transactions occur in an adversarial environment, smart contracts that do not perform safety checks *can be exploited for profit*. **If a smart contract assumes that the current price on Uniswap is a "fair" price without performing safety checks, *it is vulnerable to manipulation*. A bad actor could e.g. easily insert transactions before and after the swap (a "sandwich" attack) causing the smart contract to trade at a much worse price, profit from this at the trader's expense, and then return the contracts to their original state.** (One important caveat is that these types of attacks are mitigated by trading in extremely liquid pools, and/or at low values.)

Also, in https://uniswap.org/docs/v2/advanced-topics/pricing/#pricing-trades it states:

When swapping tokens on Uniswap, it's common to want to receive as many output tokens as possible for an *exact input amount*, or to pay as few input tokens as possible for an *exact output amount*. In order to calculate these amounts, a contract must look up the *current reserves* of a pair, in order to understand what the current price is. **However, it is *not safe to perform this lookup and rely on the results without access to an external price*.**

Say a smart contract naively wants to send 10 DAI to the DAI/WETH pair and receive as much WETH as it can get, given the current reserve ratio. If, when called, the naive smart contract simply looks up the current price and executes the trade, it is *vulnerable to front-running and will likely suffer an economic loss*. To see why, consider a malicious actor who sees this transaction before it is confirmed. They could execute a swap which dramatically changes the DAI/WETH price immediately before the naive swap goes through, wait for the naive swap to execute at a bad rate, and then swap to change the price back to what it was before the naive swap. **This attack is fairly cheap and low-risk, and can typically be performed for a profit.**

The price manipulation scenarios above described are currently being actively exploited by attackers and have successfully drained funds from many projects in the DeFi ecosystem.

High-volatility and/or low-volume trading pais are particularly easier to manipulate.

Recommendations

Potentials solutions include:

1. Integrate off-chain oracle solutions such as Chainlink in order to compare the exchange rate obtained on-chain from the AMM
2. Comparing the AMM exchange rate with previous trade rates in order to protect from abrupt changes
3. Build a time-weighted average price (TWAP) oracle based on the time-weighted average price accumulators `price0CumulativeLast` and `price1CumulativeLast` introduced in Uniswap V2 protocol.
4. Utilize more than one AMM and compare their exchange rates before a trade is performed.

References

1. https://uniswap.org/docs/v2/smart-contract-integration/trading-from-a-smart-contract/
2. https://uniswap.org/docs/v2/core-concepts/oracles/
3. https://uniswap.org/docs/v2/advanced-topics/pricing/#pricing-trades
4. https://uniswap.org/audit.html#orga1aca00
5. https://samczsun.com/taking-undercollateralized-loans-for-fun-and-for-profit/
6. https://decrypt.co/49758/after-100-million-lost-to-flash-loan-attacks-curve-pushes-chainlink

Status

The Vesper team analyzed this finding and determined it does not represent an immediate risk because of Uniswap and Vesper withdrawal fees. They stated they might fix the issue in a future version.

7. Conclusion

Overall, Coinspect did not find any high risk security vulnerabilities introduced by the new features added to Vesper pools and strategies since its previous audit that would result in stolen or lost user funds.

However, during this audit Coinspect observed the way tokens are swapped by Vesper is vulnerable to price manipulation attacks in the Uniswap AMM and this could lead to funds being drained from the strategies and pools.

The overall project security can be improved in few aspects as suggested in this report, including:

- Fix test coverage.
- 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.

It is important to note that the security of the whole platform 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
- Compound
- Yearn
- 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 test in yearn-yVault-weth.js

Coinspect investigated this failed test to understand if it was caused by a bug in the smart contracts. The reason the test fails is the variable `depositedAmount` is assigned `accounts[0]` balance, which is greater than the rest of the accounts because of the previous test. Simply using `accounts[2]` balance for all the deposits in this test is enough for it to properly work.

```
  Contract: VETH Pool using WETH
   Basic functionality tests
earning from yearn vPool 30002886620837369411
earning from aave vPool 30016876456165584565
     ✓ Should deposit, rebalance and withdraw all amount from vault (107213ms)
     1) Should earn interest fee on earned amount

   Events emitted during test:
   ---------------------------

   Ownable.OwnershipTransferred(
     previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
     newOwner: <indexed> 0x23878197A56BFD9088b196f4b3CeE8D21B621f08 (type: address)
   )

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!

   Warning: Could not decode event!
```

```
Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0x23878197A56BFD9088b196f4b3CeE8D21B621f08 (type: address),
  value: 60002886620837369411 (type: uint256)
)

PoolShareToken.Deposit(
  owner: <indexed> 0x23878197A56BFD9088b196f4b3CeE8D21B621f08 (type: address),
  shares: 60002886620837369411 (type: uint256),
  amount: 60002886620837369411 (type: uint256)
)

Warning: Could not decode event!

Warning: Could not decode event!
```

```
    IERC20.Transfer(
      from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
      to: <indexed> 0xA1CF5D4837e61C18134daA14d4970E2aa1195C0b (type: address),
      value: 60002886620837369411 (type: uint256)
    )

    PoolShareToken.Deposit(
      owner: <indexed> 0xA1CF5D4837e61C18134daA14d4970E2aa1195C0b (type: address),
      shares: 60002886620837369411 (type: uint256),
      amount: 60002886620837369411 (type: uint256)
    )

    Warning: Could not decode event!


    ---------------------------
      ✓ Should rebalance collateral (8277ms)
      ✓ Should rebalance when debt is below dust (6912ms)
pricePerShare 1000000000000000000
pricePerShare 1000020485154714446
pricePerShare 1000023661177021467
pricePerShare 1000024514679106258
pricePerShare 1000025368178869120
pricePerShare 1000045852116618026
      ✓ Should increase share prices every block (21855ms)
      ✓ Should withdraw small amount of WETH from the pool (8392ms)
      ✓ Should withdraw WETH and leave WETH dust in vault (9953ms)
      ✓ Should withdraw WETH before rebalance, leave dust (3373ms)
      ✓ Should withdraw all WETH after rebalance (9979ms)
      ✓ Should verify that low water decreases debt (9041ms)
      ✓ Should sweep erc20 for veth (8989ms)
      ✓ Should not be able sweep aDai, VETH and WETH (1636ms)
      ✓ Should collect fee on withdraw (10316ms)
      ✓ Should allow without fee withdraw by fee collector (14364ms)
      ✓ Should be able to rebalanceEarned (14915ms)
      ✓ Should be able to rebalanceEarned with 0 interestFee (11255ms)


  15 passing (11m)
  1 failing

  1) Contract: VETH Pool using WETH
       Basic functionality tests
         Should earn interest fee on earned amount:
     Error: Returned error: VM Exception while processing transaction: revert SafeERC20:
low-level call failed -- Reason given: SafeERC20: low-level call failed.
      at Context.<anonymous> (test/yearn-yVault-weth.js:106:26)
      at process._tickCallback (internal/process/next_tick.js:68:7)

npm ERR! Test failed.  See above for more details.
```