Vesper Synth

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 2nd, 2022 - March 2nd, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/17/2022	Juned Ansari
0.2	Document Update	02/27/2022	Juned Ansari
0.3	Document Update	03/01/2022	Juned Ansari
0.4	Draft Review	03/02/2022	Gabi Urrutia
1.0	Remediation Plan	03/30/2021	Juned Ansari
1.1	Remediation Plan Review	03/30/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Juned Ansari	Halborn	Juned.Ansari@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Vesper Finance engaged Halborn to conduct a security assessment on their Vesper Synth smart contracts beginning on February 2nd, 2022 and ending March 2nd, 2022. This security assessment was scoped to the Vesper Synth smart contracts code in Solidity.

1.2 AUDIT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that Vesper Synth Contract function are intended.
- Identify potential security issues with the assets in scope.

In summary, Halborn identified some security risks that were mostly addressed by the Vesper Finance team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the Vesper Synth contract solidity code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE : Vesper Synth Contract develop-branch
IN-SCOPE Commit-Id: 5e2ad925ae4f86518f12b4fae619c58036f9a446
Fixed Commit-Id:
- e6104311cda89a4b651a4e33b90f150962646e42
- 45a89f1da452ecef982230a8aab2b0fe079d1e80
- 4bb94cfa40aa3b003beed92eae0319137eecd3a5
- 40afc1fef76bdaf31dfb7d68b393b6b32cd6002f
- fe6d5d76d53f9c5ca548f55cbd6bfd41b8ba0f21

OUT-OF-SCOPE : External libraries, mock, dependencies, upgraders and economics attacks

Listing 1: Vesper synth - Develop branch

```
contracts/DebtToken.sol
contracts/Controller.sol
contracts/storage/ControllerStorage.sol
contracts/storage/TreasuryStorage.sol
contracts/storage/SyntheticAssetStorage.sol
contracts/storage/DebtTokenStorage.sol
contracts/storage/DepositTokenStorage.sol
contracts/Pausable.sol
contracts/SyntheticAsset.sol
contracts/oracle/ib/CTokenOracle.sol
contracts/oracle/ib/ATokenOracle.sol
contracts/oracle/ib/InterestBearingOracle.sol
contracts/oracle/MasterOracle.sol
contracts/oracle/UniswapV2PriceProvider.sol
contracts/oracle/DefaultOracle.sol
contracts/oracle/UniswapV3PriceProvider.sol
contracts/oracle/ChainlinkPriceProvider.sol
contracts/WETHGateway.sol
contracts/DepositToken.sol
contracts/Treasury.sol
contracts/access/Governable.sol
contracts/access/Manageable.sol
contracts/interface/IWETHGateway.sol
contracts/interface/IDepositToken.sol
contracts/interface/oracle/IMasterOracle.sol
contracts/interface/oracle/IPriceProvider.sol
contracts/interface/oracle/IOracle.sol
```

EXECUTIVE OVERVIEW

```
28 contracts/interface/oracle/IUniswapV3CrossPoolOracle.sol
29 contracts/interface/IGovernable.sol
30 contracts/interface/IController.sol
31 contracts/interface/ITreasury.sol
32 contracts/interface/external/IMulticall.sol
33 contracts/interface/external/IWETH.sol
34 contracts/interface/external/IAToken.sol
35 contracts/interface/external/ICToken.sol
36 contracts/interface/ISyntheticAsset.sol
37 contracts/interface/IDebtToken.sol
```

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	5	3

LIKELIHOOD

IMPACT

		(HAL-01)		
	(HAL-02) (HAL-03) (HAL-04) (HAL-05)			
(HAL-07)		(HAL-06)		
(HAL-08) (HAL-09)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
RACE CONDITION ON APPROVE FUNCTION MAY LEADS TO ALLOWANCE MANIPULATION	Medium	RISK ACCEPTED
MISSING RE-ENTRANCY PROTECTION	Low	SOLVED - 03/04/2022
INCOMPATIBILITY WITH INFLATIONARY TOKENS	Low	RISK ACCEPTED
EXTERNAL FUNCTION CALLS WITHIN LOOP	Low	RISK ACCEPTED
MISSING ZERO-ADDRESS CHECK	Low	SOLVED - 03/04/2022
IGNORE RETURN VALUES	Low	SOLVED - 03/04/2022
USAGE OF BLOCK-TIMESTAMP	Informational	SOLVED - 03/04/2022
CACHE ARRAY LENGTH IN FOR LOOPS CAN SAVE GAS	Informational	SOLVED - 03/04/2022
POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED - 03/04/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) RACE CONDITION ON APPROVE FUNCTION MAY LEADS TO ALLOWANCE MANIPULATION - MEDIUM

Description:

In the SyntheticAsset.sol contract, approve allows setting a new allowance. However, changing the allowance using this method carries the risk that someone could use both the old and the new allowance by unfortunate transaction ordering. For example, Alice first sets the allowance to 0, checks, then sets the allowance to N, the front running still exists, and Alice may not know that Bob has already transferred his M tokens before resetting the allowance to 0.

Code Location:

Listing 2: SyntheticAsset.sol (Line 58)

```
58     function approve(address spender, uint256 amount) public
↳ virtual override returns (bool) {
59         _approve(_msgSender(), spender, amount);
60         return true;
61     }
```

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

A possible solution to mitigate this race condition is to reduce the spender allowance to 0 and then set the desired value. Otherwise, build the following function. If the current allowance for _spender is equal to _currentValue, override it with _value and return true, otherwise return

false.

Listing 3

```
1 function approve(address _spender, uint256 _currentValue, uint256
↳ _value) returns (bool success)
```

Reference

Issue in Approval

Remediation Plan:

RISK ACCEPTED: The Vesper team accepted the risk of this issue. However, the team claims to adhere to the ERC20 standard, as this is a well-known issue with ERC20 approvals.

3.2 (HAL-02) MISSING RE-ENTRANCY PROTECTION - LOW

Description:

Some contracts within the scope of Vesper Synth develop branch are missing the nonReentrant guard. In this function, external calls that follow an external call are identified, making it vulnerable to a Reentrancy attack.

- DepositToken.sol contract function transferFrom missing nonReentrant guard.

To protect against cross-function reentrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the function with a recursive call. OpenZeppelin has its own mutex implementation called ReentrancyGuard which provides a modifier to any function called “nonReentrant” that protects the function with a mutex against the Reentrancy attacks.

Code Location:

Listing 4: DepositToken.sol (Lines 236,241)

```
231     function transferFrom(
232         address _sender,
233         address _recipient,
234         uint256 _amount
235     ) public override returns (bool) {
236         _transferWithChecks(_sender, _recipient, _amount);
237 
238         uint256 currentAllowance = allowance[_sender][_msgSender()
 ↳ ];
239         require(currentAllowance >= _amount, "amount-exceeds-
 ↳ allowance");
240         unchecked {
241             _approve(_sender, _msgSender(), currentAllowance -
 ↳ _amount);
242         }
```

```
243
244         return true;
245     }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Change the code to follow the checks-effects-interactions pattern and use ReentrancyGuard via the nonReentrant modifier.

Remediation Plan:

SOLVED: The Vesper team solved the above issue in commit e6104311cda89a4b651a4e33b90f150962646e42. On commit, the team added a nonReentrant modifier to the transferFrom function above.

3.3 (HAL-03) INCOMPATIBILITY WITH INFLATIONARY TOKENS - LOW

Description:

In multiple functions, Vesper Synth contracts use OpenZeppelin safeTransferFrom and safeTransfer to handle token transfers. These functions call transferFrom and transfer internally in the token contract to actually execute the transfer. However, the actual amount transferred is not verified, i.e., the delta of previous (before the transfer) and current (after the transfer) balance. As a result, a malicious user can enumerate a custom ERC20 token with the transferFrom or transfer function modified in such a way that (for example, it does not transfer any tokens at all and the attacker will continue to have their liquidity pool tokens minted anyway). In this case, the creator of the contract sets both tokens in the constructor, so they are trusted, but it would be good practice to do this check.

Code Location:

Listing 5: WETHGateway.sol

```
#51: _depositToken.safeTransferFrom(msg.sender, address(this),
↳ _amount);
#68: _token.safeTransfer(_to, _amount);
```

- Missing check on after transfer

Listing 6: Treasury.sol

```
#37: _token.safeTransfer(_to, _amount);
#53: if (_balance > 0) _depositToken.safeTransfer(_newTreasury,
↳ _balance);
#54: if (_underlyingBalance > 0) _depositToken.underlying().
↳ safeTransfer(_newTreasury, _underlyingBalance);
#63: _vsAsset.safeTransfer(_newTreasury, _balance);
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Whenever tokens are transferred, the delta of the previous (before the transfer) and current (after the transfer) token balance should be verified to match the amount of tokens declared.

Remediation Plan:

RISK ACCEPTED: The Vesper team accepted the risk of this issue. However, the team claims that the transfer delta value is not checked for the following reason, i.e., both DepositToken and SyntheticToken are not deflationary and gas-saving tokens.

3.4 (HAL-04) EXTERNAL FUNCTION CALLS WITHIN LOOP - LOW

Description:

External calls inside a loop increase gas usage or can lead to a denial of service attack. In Controller.sol, Treasury.sol contract functions discovered that there are for loop on i variable that iterates through debtTokensOfAccount.length(_account), _depositTokens.length, and _syntheticAssets.length these loops have external calls inside a loop. If this integer evaluates to large numbers, this can cause a DoS.

Code Location:

Listing 7: Controller.sol (Lines 250,251)

```
247     function debtOf(address _account) public view override returns
 ↳  (uint256 _debtInUsd) {
248         for (uint256 i = 0; i < debtTokensOfAccount.length(
 ↳ _account); ++i) {
249             IDebtToken _debtToken = IDebtToken(debtTokensOfAccount
 ↳ .at(_account, i));
250             ISyntheticAsset _syntheticAsset = _debtToken.
 ↳ syntheticAsset();
251             uint256 _amountInUsd = oracle.convertToUsd(
 ↳ _syntheticAsset, _debtToken.balanceOf(_account));
252             _debtInUsd += _amountInUsd;
253         }
254     }
```

Listing 8: Controller.sol (Lines 271,273)

```
263     function depositOf(address _account)
264         public
265         view
266         override
267         returns (uint256 _depositInUsd, uint256
 ↳ _mintableLimitInUsd)
268     {
```

```
269         for (uint256 i = 0; i < depositTokensOfAccount.length(
 ↳ _account); ++i) {
270             IDepositToken _depositToken = IDepositToken(
 ↳ depositTokensOfAccount.at(_account, i));
271             uint256 _amountInUsd = oracle.convertToUsd(
 ↳ _depositToken, _depositToken.balanceOf(_account));
272             _depositInUsd += _amountInUsd;
273             _mintableLimitInUsd += _amountInUsd.wadMul(
 ↳ _depositToken.collateralizationRatio());
274         }
275     }
```

Listing 9: Treasury.sol (Lines 50,51,54,61)

```
44     function migrateTo(address _newTreasury) external
 ↳ onlyController {
45         address[] memory _depositTokens = controller.
 ↳ getDepositTokens();
46
47         for (uint256 i = 0; i < _depositTokens.length; ++i) {
48             IDepositToken _depositToken = IDepositToken(
 ↳ _depositTokens[i]);
49
50             uint256 _balance = _depositToken.balanceOf(address(
 ↳ this));
51             uint256 _underlyingBalance = _depositToken.underlying
 ↳ ().balanceOf(address(this));
52
53             if (_balance > 0) _depositToken.safeTransfer(
 ↳ _newTreasury, _balance);
54             if (_underlyingBalance > 0) _depositToken.underlying()
 ↳ .safeTransfer(_newTreasury, _underlyingBalance);
55         }
56
57         address[] memory _syntheticAssets = controller.
 ↳ getSyntheticAssets();
58
59         for (uint256 i = 0; i < _syntheticAssets.length; ++i) {
60             IERC20 _vsAsset = IERC20(_syntheticAssets[i]);
61             uint256 _balance = _vsAsset.balanceOf(address(this));
62             if (_balance > 0) {
63                 _vsAsset.safeTransfer(_newTreasury, _balance);
64             }
65         }
```

```
66     }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended to set the maximum length a for loop can iterate to and avoid using multiple loops in the same function. If possible, use a pull over push strategy for external calls.

Reference
External Calls Recommendation

Remediation Plan:

RISK ACCEPTED: The Vesper team accepted the risk of this issue. However, The team claims that it is unlikely for the current system as even a user has up to 100 deposit tokens and 100 synthetic tokens even, then the cost would be ~1/3 of the block gas limit (30M).

3.5 (HAL-05) MISSING ZERO-ADDRESS CHECK - LOW

Description:

Lack of zero address validation has been found when assigning user-supplied address values to state variables directly.

- In contract UpgraderBase.sol:
  - updateMulticall lacks a zero address check on _multicall.

Code Location:

Zero Address Validation is missing before address assignment to a state variable.

Listing 10: UpgraderBase.sol

```
1  multicall = _multicall (#13)
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Although administrative restrictions are imposed to this function due to the role-based access controls (RBAC), it is recommended to add proper address validation when assigning user-supplied input to a variable. This could be as simple as using the following statement:

Listing 11

```
require(address_input != 0, "Address is zero")
```

Remediation Plan:

SOLVED: The Vesper team solved the above issue in commit 45a89f1da452ecef982230a8aab2b0fe079d1e80. On commit, the team has added a proper zero address validation.

Listing 12: Updated UpgraderBase.sol

```
    function updateMulticall(address _multicall) public onlyOwner
↳ {
        require(_multicall != address(0), "address-is-null");
        require(_multicall != multicall, "new-same-as-current");
        multicall = _multicall;
    }
```

3.6 (HAL-06) IGNORE RETURN VALUES - LOW

Description:

The return value of an external call is not stored in a local or state variable. In the Controller.sol contract, there are instances where an external method is called, and the return value is ignored.

Code Location:

Listing 13: Controller.sol (Line 660)

```
658     function addToDebtTokensOfAccount(address _account) external {
659         _requireMsgSenderIsDebtToken();
660         debtTokensOfAccount.add(_account, _msgSender());
661     }
```

Listing 14: Controller.sol (Line 640)

```
638     function addToDepositTokensOfAccount(address _account)
 ↳ external {
639         require(depositTokens.contains(_msgSender()), "caller-is-
 ↳ not-deposit-token");
640         depositTokensOfAccount.add(_account, _msgSender());
641     }
```

Listing 15: Controller.sol (Line 670)

```
668     function removeFromDebtTokensOfAccount(address _account)
 ↳ external {
669         _requireMsgSenderIsDebtToken();
670         debtTokensOfAccount.remove(_account, _msgSender());
671     }
```

Listing 16: Controller.sol (Line 650)

```
648     function removeFromDepositTokensOfAccount(address _account)
 ↳ external {
649         require(depositTokens.contains(_msgSender()), "caller-is-
 ↳ not-deposit-token");
650         depositTokensOfAccount.remove(_account, _msgSender());
651     }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Add return value check to prevent an unexpected contract lock. Return value check will help to handle exceptions in a better way.

Remediation Plan:

SOLVED: The Vesper team solved the above issue in the commit 4bb94cfa40aa3b003beed92eae0319137eecd3a5. On commit, the team added require checks for return value validation for the above code.

3.7 (HAL-07) USAGE OF BLOCK-TIMESTAMP - INFORMATIONAL

Description:

Some Vesper Synth contracts use block.timestamp. Miners can influence the value of block.timestamp to a certain degree, so testers should be warned that this may come at some risk if miners collude in time manipulation to influence price oracles. It is important to follow the 15-second rule, i.e., if the contract is not based on an interval of less than 15 seconds, it is fine to use block.timestamp.

Code Location:

Listing 17: DefaultOracle.sol

```
#91: return block.timestamp - _timeOfLastUpdate > assets[_asset].
↳ stalePeriod;
```

Listing 18: DepositToken.sol

```
#35: require(block.timestamp >= lastDepositOf[_account] +
↳ minDepositTime, "min-deposit-time-have-not-passed");
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to follow the 15-second rule, i.e., if the time-dependent event scale can vary by 15 seconds and maintain integrity, it is safe to use a block.timestamp.

Reference:

Ethereum Yellow Paper

Remediation Plan:

SOLVED: The Vesper team has below checks in place, ensuring check conditions are much higher than 15 seconds; thus, a block manipulation would be unlikely.

Listing 19: DepositToken.sol

```
1 modifier onlyIfMinDepositTimePassed(address _account) {
2    require(block.timestamp >= lastDepositOf[_account] +
↳ minDepositTime, "min-deposit-time-have-not-passed");
3    _;
4 }
```

Listing 20: DefaultOracle.sol

```
1 function _priceIsStale(IERC20 _asset, uint256 _timeOfLastUpdate)
↳ private view returns (bool) {
2    return block.timestamp - _timeOfLastUpdate > assets[_asset].
↳ stalePeriod;
3 }
```

3.8 (HAL-08) CACHE ARRAY LENGTH IN FOR LOOPS CAN SAVE GAS - INFORMATIONAL

Description:

Reading the length of the array at each iteration of the loop requires 6 gas (3 for `mload` and 3 to place `memory_offset`) onto the stack. Caching the length of the array on the stack saves about 3 gas per iteration.

Code Location:

Listing 21: Controller.sol (Line 248)

```
247     function debtOf(address _account) public view override returns
 ↳  (uint256 _debtInUsd) {
248         for (uint256 i = 0; i < debtTokensOfAccount.length(
 ↳ _account); ++i) {
249             IDebtToken _debtToken = IDebtToken(debtTokensOfAccount
 ↳ .at(_account, i));
250             ISyntheticAsset _syntheticAsset = _debtToken.
 ↳ syntheticAsset();
251             uint256 _amountInUsd = oracle.convertToUsd(
 ↳ _syntheticAsset, _debtToken.balanceOf(_account));
252             _debtInUsd += _amountInUsd;
253         }
254     }
```

Listing 22: Controller.sol (Line 269)

```
263     function depositOf(address _account)
264         public
265         view
266         override
267         returns (uint256 _depositInUsd, uint256
 ↳ _mintableLimitInUsd)
268     {
269         for (uint256 i = 0; i < depositTokensOfAccount.length(
 ↳ _account); ++i) {
```

```
270             IDepositToken _depositToken = IDepositToken(
 ↳ depositTokensOfAccount.at(_account, i));
271             uint256 _amountInUsd = oracle.convertToUsd(
 ↳ _depositToken, _depositToken.balanceOf(_account));
272             _depositInUsd += _amountInUsd;
273             _mintableLimitInUsd += _amountInUsd.wadMul(
 ↳ _depositToken.collateralizationRatio());
274         }
275     }
```

Listing 23: Treasury.sol (Lines 47,59)

```
44     function migrateTo(address _newTreasury) external
 ↳ onlyController {
45         address[] memory _depositTokens = controller.
 ↳ getDepositTokens();
46 
47         for (uint256 i = 0; i < _depositTokens.length; ++i) {
48             IDepositToken _depositToken = IDepositToken(
 ↳ _depositTokens[i]);
49 
50             uint256 _balance = _depositToken.balanceOf(address(
 ↳ this));
51             uint256 _underlyingBalance = _depositToken.underlying
 ↳ ().balanceOf(address(this));
52 
53             if (_balance > 0) _depositToken.safeTransfer(
 ↳ _newTreasury, _balance);
54             if (_underlyingBalance > 0) _depositToken.underlying()
 ↳ .safeTransfer(_newTreasury, _underlyingBalance);
55         }
56 
57         address[] memory _syntheticAssets = controller.
 ↳ getSyntheticAssets();
58 
59         for (uint256 i = 0; i < _syntheticAssets.length; ++i) {
60             IERC20 _vsAsset = IERC20(_syntheticAssets[i]);
61             uint256 _balance = _vsAsset.balanceOf(address(this));
62             if (_balance > 0) {
63                 _vsAsset.safeTransfer(_newTreasury, _balance);
64             }
65         }
66     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider the length of the cache array. The example code can be seen below.

Listing 24: Controller.sol (Lines 248,249)

```
247     function debtOf(address _account) public view override returns
 ↳  (uint256 _debtInUsd) {
248         uint256 debtTokensOfAccount_length = debtTokensOfAccount.
 ↳ length(_account);
249         for (uint256 i = 0; i < debtTokensOfAccount_length; ++i) {
250             IDebtToken _debtToken = IDebtToken(debtTokensOfAccount
 ↳ .at(_account, i));
251             ISyntheticAsset _syntheticAsset = _debtToken.
 ↳ syntheticAsset();
252             uint256 _amountInUsd = oracle.convertToUsd(
 ↳ _syntheticAsset, _debtToken.balanceOf(_account));
253             _debtInUsd += _amountInUsd;
254         }
255     }
```

Remediation Plan:

SOLVED: The Vesper team solved the above issue in commit 40afc1fef76bdaf31dfb7d68b393b6b32cd6002f. On commit, the team updated the code, and the above code now caches the length of the array.

3.9 (HAL-09) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied into memory, while external functions can read directly from calldata. Reading calldata is cheaper than allocating memory. Public functions need to write arguments to memory because public functions can be called internally. Internal calls are passed internally via pointers to memory. Therefore, the function expects its arguments to be located in memory when the compiler generates the code for an internal function. Moreover, methods do not necessarily have to be public if they are only called within the contract; in such case, they should be marked as internal.

Code Location:

Below are smart contracts and their corresponding functions affected:
ChainlinkPriceProvider.sol:
convert()

Controller.sol:
addDepositToken addSyntheticAsset() initialize() setController()

DebtToken.sol:
allowance() approve() burn() decreaseAllowance() increaseAllowance() initialize() mint() transfer() transferFrom()

DefaultOracle.sol:
update()

DepositToken.sol:
addDepositToken() approve() burn() burnForWithdraw() decreaseAllowance() increaseAllowance() initialize() lockedBalanceOf() mint() seize() toggleIsActive() transfer() transferFrom() updateCollateralizationRatio() updateMaxTotalSupplyInUsd() updateMinDepositTime()

InterestBearingOracle.sol:
convertFromUsd() convertToUsd()

Manageable.sol:
setController()

SyntheticAsset.sol:
addSyntheticAsset(address) approve() burn() decreaseAllowance() increaseAllowance() initialize() interestRatePerBlock() mint() seize() toggleIsActive() transfer() transferFrom() updateInterestRate() updateMaxTotalSupplyInUsd()

Treasury.sol:
initialize() initialize()

UpgraderBase.sol:
updateMulticall()

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider, as much as possible, declaring external variables instead of public variables. As for best practices, you should use external if you expect that the function to be called only externally, and use public if you need to call the function internally. In short, public functions can be accessed by everyone, external functions can only be accessed externally, and internal functions can only be called inside the contract.

Remediation Plan:

SOLVED: The Vesper team solved the above issue in commit fe6d5d76d53f9c5ca548f55cbd6bfd41b8ba0f21.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

```
Reentrancy in DebtToken._mint(address,uint256) (contracts/DebtToken.sol#94-105):
        External calls:
        - _beforeTokenTransfer(address(0),_account,_amount) (contracts/DebtToken.sol#97)
                - controller.addToDebtTokensOfAccount(_to) (contracts/DebtToken.sol#135)
        State variables written after the call(s):
        - interestRateOf[_account] = debtIndex (contracts/DebtToken.sol#101)
        - principalOf[_account] += _amount (contracts/DebtToken.sol#100)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

Controller.liquidate(ISyntheticAsset,address,uint256,IDepositToken) (contracts/Controller.sol#475-523) uses a dangerous strict equality:
        - require(bool,string)(_newDebtInUsd == 0 || _newDebtInUsd >= debtFloorInUsd,debt-lt-floor) (contracts/Controller.sol#499)
Controller.repay(ISyntheticAsset,address,uint256) (contracts/Controller.sol#435-466) uses a dangerous strict equality:
        - require(bool,string)(_newDebtInUsd == 0 || _newDebtInUsd >= debtFloorInUsd,debt-lt-floor) (contracts/Controller.sol#459)
Controller.swap(ISyntheticAsset,ISyntheticAsset,uint256) (contracts/Controller.sol#531-580) uses a dangerous strict equality:
        - require(bool,string)(_inNewDebtInUsd == 0 || _inNewDebtInUsd >= debtFloorInUsd,asset-in-debt-lt-floor) (contracts/Controller.sol#554)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in Controller.updateTreasury(ITreasury,bool) (contracts/Controller.sol#768-776):
        External calls:
        - treasury.migrateTo(address(_newTreasury)) (contracts/Controller.sol#772)
        State variables written after the call(s):
        - treasury = _newTreasury (contracts/Controller.sol#775)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

Controller.mint(ISyntheticAsset,uint256,address)._feeAmount (contracts/Controller.sol#415) is a local variable never initialized
Controller.repay(ISyntheticAsset,address,uint256)._feeAmount (contracts/Controller.sol#447) is a local variable never initialized
Controller.withdraw(IDepositToken,uint256,address)._feeAmount (contracts/Controller.sol#371) is a local variable never initialized
Controller.deposit(IDepositToken,uint256,address)._feeAmount (contracts/Controller.sol#341) is a local variable never initialized
Controller.swap(ISyntheticAsset,ISyntheticAsset,uint256)._feeAmount (contracts/Controller.sol#569) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

Controller.addSyntheticAsset(address) (contracts/Controller.sol#585-592) ignores return value by syntheticAssets.add(_syntheticAsset) (contracts/Controller.sol#589)
Controller.removeSyntheticAsset(ISyntheticAsset) (contracts/Controller.sol#597-605) ignores return value by syntheticAssets.remove(address(_syntheticAsset)) (contracts/Controller.sol#602)
Controller.addDepositToken(address) (contracts/Controller.sol#610-618) ignores return value by depositTokens.add(_depositToken) (contracts/Controller.sol#614)
Controller.removeDepositToken(IDepositToken) (contracts/Controller.sol#623-631) ignores return value by depositTokens.remove(address(_depositToken)) (contracts/Controller.sol#628)
Controller.addToDepositTokensOfAccount(address) (contracts/Controller.sol#638-641) ignores return value by depositTokensOfAccount.add(_account,_msgSender()) (contracts/Controller.sol#640)
Controller.removeFromDepositTokensOfAccount(address) (contracts/Controller.sol#648-651) ignores return value by depositTokensOfAccount.remove(_account,_msgSender()) (contracts/Cont
```

```
Reentrancy in DepositToken._burn(address,uint256) (contracts/DepositToken.sol#126-141):
        External calls:
        - _beforeTokenTransfer(_account,address(0),_amount) (contracts/DepositToken.sol#129)
                - controller.addToDepositTokensOfAccount(_to) (contracts/DepositToken.sol#161)
        State variables written after the call(s):
        - balanceOf[_account] = accountBalance - _amount (contracts/DepositToken.sol#134)
Reentrancy in DepositToken._mint(address,uint256) (contracts/DepositToken.sol#114-124):
        External calls:
        - _beforeTokenTransfer(address(0),_account,_amount) (contracts/DepositToken.sol#117)
                - controller.addToDepositTokensOfAccount(_to) (contracts/DepositToken.sol#161)
        State variables written after the call(s):
        - balanceOf[_account] += _amount (contracts/DepositToken.sol#120)
Reentrancy in DepositToken._transfer(address,address,uint256) (contracts/DepositToken.sol#92-112):
        External calls:
        - _beforeTokenTransfer(sender,recipient,_amount) (contracts/DepositToken.sol#100)
                - controller.addToDepositTokensOfAccount(_to) (contracts/DepositToken.sol#161)
        State variables written after the call(s):
        - balanceOf[sender] = senderBalance - _amount (contracts/DepositToken.sol#105)
        - balanceOf[recipient] += _amount (contracts/DepositToken.sol#107)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

```
Controller.debtOf(address) (contracts/Controller.sol#247-254) has external calls inside a loop: _syntheticAsset = _debtToken.syntheticAsset() (contracts/Controller.sol#250)
Controller.debtOf(address) (contracts/Controller.sol#247-254) has external calls inside a loop: _amountInUsd = oracle.convertToUsd(_syntheticAsset,_debtToken.balanceOf(_account)) (
contracts/Controller.sol#251)
Controller.depositOf(address) (contracts/Controller.sol#263-275) has external calls inside a loop: _amountInUsd = oracle.convertToUsd(_depositToken,_depositToken.balanceOf(_account
)) (contracts/Controller.sol#271)
Controller.depositOf(address) (contracts/Controller.sol#263-275) has external calls inside a loop: _mintableLimitInUsd += _amountInUsd.wadMul(_depositToken.collateralizationRatio()
) (contracts/Controller.sol#273)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Reentrancy in Controller.deposit(IDepositToken,uint256,address) (contracts/Controller.sol#323-351):
        External calls:
        - _depositToken.underlying().safeTransferFrom(_sender,address(treasury),_amount) (contracts/Controller.sol#336)
        - _depositToken.mint(address(treasury),_feeAmount) (contracts/Controller.sol#344)
        - _depositToken.mint(_onBehalfOf,_amountToDeposit) (contracts/Controller.sol#348)
        Event emitted after the call(s):
        - CollateralDeposited(_depositToken,_sender,_onBehalfOf,_amount,_feeAmount) (contracts/Controller.sol#350)
Reentrancy in Controller.liquidate(ISyntheticAsset,address,uint256,IDepositToken) (contracts/Controller.sol#475-523):
        External calls:
        - accrueInterest(_syntheticAsset) (contracts/Controller.sol#487)
                - _interestAccumulated = _syntheticAsset.debtToken().accrueInterest() (contracts/Controller.sol#309)
                - _syntheticAsset.mint(address(treasury),_interestAccumulated) (contracts/Controller.sol#313)
        - _syntheticAsset.burn(_liquidator,_amountToRepay) (contracts/Controller.sol#514)
        - _syntheticAsset.debtToken().burn(_account,_amountToRepay) (contracts/Controller.sol#515)
        - _depositToken.seize(_account,_liquidator,_toLiquidator) (contracts/Controller.sol#516)
        - _depositToken.seize(_account,address(treasury),_toProtocol) (contracts/Controller.sol#519)
        Event emitted after the call(s):
        - PositionLiquidated(_liquidator,_account,_syntheticAsset,_amountToRepay,_depositToSeize,_toProtocol) (contracts/Controller.sol#522)
```

```
DefaultOracle._priceIsStale(IERC20,uint256) (contracts/oracle/DefaultOracle.sol#90-92) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp - _timeOfLastUpdate > assets[_asset].stalePeriod (contracts/oracle/DefaultOracle.sol#91)
DefaultOracle.convertToUsd(IERC20,uint256) (contracts/oracle/DefaultOracle.sol#180-191) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_amountInUsd > 0 && ! _priceIsStale(_asset,_lastUpdatedAt),price-is-invalid) (contracts/oracle/DefaultOracle.sol#190)
DefaultOracle.convertFromUsd(IERC20,uint256) (contracts/oracle/DefaultOracle.sol#199-210) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_amount > 0 && ! _priceIsStale(_asset,_lastUpdatedAt),price-is-invalid) (contracts/oracle/DefaultOracle.sol#209)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

UpgraderBase.updateMulticall(address)._multicall (contracts/upgraders/UpgraderBase.sol#12) lacks a zero-check on :
                - multicall = _multicall (contracts/upgraders/UpgraderBase.sol#13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

UpgraderBase.updateMulticall(address)._multicall (contracts/upgraders/UpgraderBase.sol#12) lacks a zero-check on :
                - multicall = _multicall (contracts/upgraders/UpgraderBase.sol#13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

UpgraderBase.updateMulticall(address)._multicall (contracts/upgraders/UpgraderBase.sol#12) lacks a zero-check on :
                - multicall = _multicall (contracts/upgraders/UpgraderBase.sol#13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

UpgraderBase.updateMulticall(address)._multicall (contracts/upgraders/UpgraderBase.sol#12) lacks a zero-check on :
                - multicall = _multicall (contracts/upgraders/UpgraderBase.sol#13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

UpgraderBase.updateMulticall(address)._multicall (contracts/upgraders/UpgraderBase.sol#12) lacks a zero-check on :
                - multicall = _multicall (contracts/upgraders/UpgraderBase.sol#13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

UpgraderBase.updateMulticall(address)._multicall (contracts/upgraders/UpgraderBase.sol#12) lacks a zero-check on :
                - multicall = _multicall (contracts/upgraders/UpgraderBase.sol#13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

UpgraderBase.updateMulticall(address)._multicall (contracts/upgraders/UpgraderBase.sol#12) lacks a zero-check on :
                - multicall = _multicall (contracts/upgraders/UpgraderBase.sol#13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

```
Treasury.migrateTo(address) (contracts/Treasury.sol#44-66) has external calls inside a loop: _balance = _depositToken.balanceOf(address(this)) (contracts/Treasury.sol#50)
Treasury.migrateTo(address) (contracts/Treasury.sol#44-66) has external calls inside a loop: _underlyingBalance = _depositToken.underlying().balanceOf(address(this)) (contracts/Tre
asury.sol#51)
Treasury.migrateTo(address) (contracts/Treasury.sol#44-66) has external calls inside a loop: _depositToken.underlying().safeTransfer(_newTreasury,_underlyingBalance) (contracts/Tre
asury.sol#54)
Treasury.migrateTo(address) (contracts/Treasury.sol#44-66) has external calls inside a loop: _balance_scope_1 = _vsAsset.balanceOf(address(this)) (contracts/Treasury.sol#61)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

initialize(string,string,uint8,IController,ISyntheticAsset) should be declared external:
        - DebtToken.initialize(string,string,uint8,IController,ISyntheticAsset) (contracts/DebtToken.sol#17-35)
transfer(address,uint256) should be declared external:
        - DebtToken.transfer(address,uint256) (contracts/DebtToken.sol#48-53)
allowance(address,address) should be declared external:
        - DebtToken.allowance(address,address) (contracts/DebtToken.sol#55-60)
approve(address,uint256) should be declared external:
        - DebtToken.approve(address,uint256) (contracts/DebtToken.sol#62-67)
transferFrom(address,address,uint256) should be declared external:
        - DebtToken.transferFrom(address,address,uint256) (contracts/DebtToken.sol#69-75)
increaseAllowance(address,uint256) should be declared external:
        - DebtToken.increaseAllowance(address,uint256) (contracts/DebtToken.sol#77-82)
decreaseAllowance(address,uint256) should be declared external:
        - DebtToken.decreaseAllowance(address,uint256) (contracts/DebtToken.sol#84-89)
mint(address,uint256) should be declared external:
        - DebtToken.mint(address,uint256) (contracts/DebtToken.sol#154-156)
burn(address,uint256) should be declared external:
        - DebtToken.burn(address,uint256) (contracts/DebtToken.sol#163-165)
setController(IController) should be declared external:
        - Manageable.setController(IController) (contracts/access/Manageable.sol#46-49)
```

According to the test results, some findings found by these tools were considered as false positives, while some of these findings were real security concerns. All relevant findings were reviewed by the auditor and relevant findings were addressed in the report as security concerns.

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. Only security-related findings are shown below.

Results:

Report for DebtToken.sol

Line	SWC Title	Severity	Short Description
33	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
173	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

Report for DepositToken.sol

Line	SWC Title	Severity	Short Description
35	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.

Report for proxy/ERC1967/ERC1967Proxy.sol

Line	SWC Title	Severity	Short Description
19	(SWC-112) Delegatecall to Untrusted Callee	High	The contract delegates execution to another contract with a user-supplied address.

Report for proxy/ERC1967/ERC1967Upgrade.sol

Line	SWC Title	Severity	Short Description
9	(SWC-112) Delegatecall to Untrusted Callee	High	The contract delegates execution to another contract with a user-supplied address.
25	(SWC-112) Delegatecall to Untrusted Callee	High	The contract delegates execution to another contract with a user-supplied address.

All relevant valid findings were founded during the manual code review.

THANK YOU FOR CHOOSING

// HALBORN