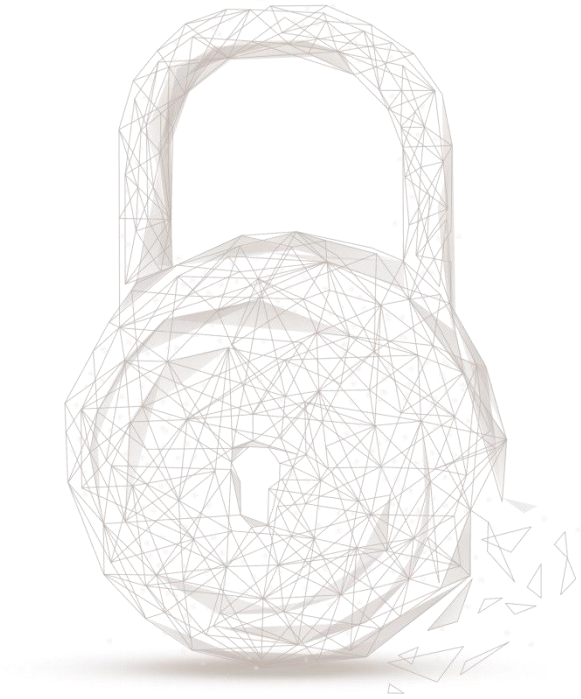
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202105311856

Report query name：xDVG

Smart Contracts Link 1：

https://github.com/daoventures/vipDVG

Smart Contract commit 1:

Start: 9dd7e6d3aeff89c06adcc2ae48736632d970de78

Final: 2662a9ed4b2e8fd2184b7cef96dfc3fd43174435

Audit File Name	Audit File Link
DVGUniBot.sol	https://github.com/daoventures/vipDVG/blob/main/contracts/DVGUniBot.sol
xDVG.sol	https://github.com/daoventures/vipDVG/blob/main/contracts/xDVG.sol

Smart Contracts Link 2：

https://github.com/daoventures/dao-protocol/

Smart Contract commit 2:

Start: 3b54f0f2052dac746019cf2aa4605b0264393e2c

Final: 2cb53c1a64e2acc14f3f35ecbb7a11ca54385d80

Audit File Name	Audit File Link
DAOVaultFactory.sol	https://github.com/daoventures/dao-protocol/blob/develop/HarvestFarmer-admin/contracts/factories/DAOVaultFactory.sol
DAOVault.sol	https://github.com/daoventures/dao-protocol/blob/develop/HarvestFarmer-admin/contracts/vaults/DAOVault.sol
HarvestFarmerFactory.sol	https://github.com/daoventures/dao-protocol/blob/develop/HarvestFarmer-admin/contracts/factories/HarvestFarmerFactory.sol
HarvestFarmer.sol	https://github.com/daoventures/dao-protocol/blob/develop/HarvestFarmer-admin/contracts/strategies/HarvestFarmer.sol

Start Date：2021.04.19

Completion Date：2021.05.31

Overall Result：Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass
		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of project xDVG, including Coding Standards, Security, and Business Logic. **The xDVG project passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.

- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing ETH.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.

- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use of the security risk of 'tx.origin' in the contract.
- Result: Pass

2.10 Replay Attack

- Description: Check whether the implementation possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

3.1 Business analysis of Contract xDVG

- Description: This contract implements the function of evenly sharing income, and users can obtain xDVG tokens as share certificates by deposit the designated "dvg" tokens. When the user burns xDVG and exchanges it into "dvg", he will get a moderate percentage of the " dvg" in the contract.

```
26      function deposit(uint256 _amount) public {
27          // Gets the amount of DVG locked in the contract
28          uint256 totalDVG = dvg.balanceOf(address(this));
29          // Gets the amount of xDVG in existence
30          uint256 totalShares = totalSupply();
31          uint256 what;
32          // If no xDVG exists, mint it 1:1 to the amount put in
33          if (totalShares == 0) {
34              what = _amount;
35              _mint(msg.sender, _amount);
36          }
37          // Calculate and mint the amount of xDVG the DVG is worth. The ratio will change overtime
38          else {
39              what = _amount.mul(totalShares).div(totalDVG);
40              _mint(msg.sender, what);
41          }
42          // Lock the DVG in the contract
43          dvg.safeTransferFrom(msg.sender, address(this), _amount);
44
45          emit Deposit(msg.sender, _amount, what);
46      }
```

Figure 1 source code of xDVG

```
49      function withdraw(uint256 _share) public {
50          // Gets the amount of xDVG in existence
51          uint256 totalShares = totalSupply();
52          // Calculates the amount of DVG the xDVG is worth
53          uint256 what = _share.mul(dvg.balanceOf(address(this))).div(totalShares);
54          _burn(msg.sender, _share);
55          dvg.safeTransfer(msg.sender, what);
56
57          emit Withdraw(msg.sender, what, _share);
58      }
```

Figure 2 source code of xDVG

- Related Functions: *deposit*, *withdraw*
- Safety Suggestion: None
- Result: Pass

3.2 Business analysis of Contract DAOVaultFactory and HarvestFarmerFactory

(1) DAOVaultFactory

- Description: This contract implements the function of cloning the DAOVault contract. Calling the *createValut* function of the contract can create and initialize a DAOVault contract.
- Related Functions*: cerateVault*
- Safety Suggestion: None
- Result: Pass

(2) HarvestFarmerFactory

- Description: This contract implements the function of cloning the HarvestFarmer contract. Calling the *createStrategy* function of the contract can create and initialize a HarvestFarmer contract.
- Related Functions: *createStrategy*
- Safety Suggestion: None
- Result: Pass

3.3 Business analysis of Contract DVGUniBot

- Description: This contract implements a swap function. Anyone can call the *buyDVG* function to exchange the specified tokens in the specified wallet for DVG and send it to the xDVG contract.

```
58      function buyDVG(IERC20 _token) public payable onlyEOA nonReentrant returns(uint256 dvgAmount) {
59          require(token[_token].allowed, "Token not allowed");
60          require(_token.balanceOf(wallet) >= minAmount.mul(token[_token].decimals), "Token balance of wallet not enough");
61
62          uint256 amount_ = amount.mul(token[_token].decimals);
63          address weth = router.WETH();
64
65          _token.safeTransferFrom(wallet, address(this), amount_);
66
67          address[] memory path = new address[](2);
68          path[0] = address(_token);
69          path[1] = weth;
70
71          uint[] memory amounts = router.swapExactTokensForETH(amount_, 0, path, address(this), block.timestamp);
72
73          path[0] = weth;
74          path[1] = address(dvg);
75          amounts = router.swapExactETHForTokens{value:amounts[amounts.length - 1]}(0, path, address(xdvg), block.timestamp);
76
77          dvgAmount = amounts[1];
78
79          emit BuyDVG(msg.sender, _token, dvgAmount);
80      }
```

Figure 3 source code of function buyDVG

- Related Functions: *buyDVG*, *setWallet*, *setAmount*, *setToken*
- Safety Suggestion: None
- Result: Pass

3.4 Business analysis of Contract DAOVault

(1) deposit

- Description: Users can call this function to deposit and obtain share tokens. The number of share tokens obtained is proportional to the total number of tokens that have been stored and profitable.

```
132     function deposit(uint256 _amount) external onlyEOA {
133         require(_amount > 0, "Amount must > 0");
134 
135         uint256 _pool = strategy.getPseudoPool().add(token.balanceOf(address(this))).sub(_fees);
136         token.safeTransferFrom(msg.sender, address(this), _amount);
137 
138         uint256 _networkFeePercentage;
139         /**
140          * Network fees
141          * networkFeeTier2 is used to set each tier minimun and maximun
142          * For example networkFeeTier2 is [50000, 100000],
143          * Tier 1 = _depositAmount < 50001
144          * Tier 2 = 50001 <= _depositAmount <= 100000
145          * Tier 3 = _depositAmount > 100000
146          *
147          * networkFeePercentage is used to set each tier network fee percentage
148          * For example networkFeePercentage is [100, 75, 50]
149          * which mean network fee for Tier 1 = 1%, Tier 2 = 0.75%, Tier 3 = 0.5%
150          *
151          * customNetworkFeeTier is treat as tier 4
152          * customNetworkFeePercentage will be used in customNetworkFeeTier
153          */
154         if (_amount < networkFeeTier2[0]) {
155             // Tier 1
156             _networkFeePercentage = networkFeePercentage[0];
157         } else if (_amount <= networkFeeTier2[1]) {
158             // Tier 2
159             _networkFeePercentage = networkFeePercentage[1];
160         } else if (_amount < customNetworkFeeTier) {
161             // Tier 3
162             _networkFeePercentage = networkFeePercentage[2];
163         } else {
164             // Custom Tier
165             _networkFeePercentage = customNetworkFeePercentage;
166         }
167         uint256 _fee = _amount.mul(_networkFeePercentage).div(10000 /*DENOMINATOR*/);
168         _amount = _amount.sub(_fee);
169         _fees = _fees.add(_fee);
170 
171         uint256 _shares = totalSupply() == 0
172             ? _amount
173             : _amount.mul(totalSupply()).div(_pool);
174         _mint(msg.sender, _shares);
175     }
```

Figure 4 source code of function deposit

- Related Functions: *safeTransferFrom, getPseudoPool*
- Safety Suggestion: None
- Result: Pass

(2) Withdraw

- Description: Users can call this function to burn shares tokens and withdraw their deposited tokens and rewards.

```
183     function withdraw(uint256 _shares) external onlyEOA {
184         uint256 _balanceOfVault = (token.balanceOf(address(this))).sub(_fees);
185         uint256 _withdrawAmt = (_balanceOfVault.add(strategy.pool()).mul(_shares).div(totalSupply()));
186 
187         // USDT.transfer doesn't check if amount is 0. Therefor we will check it here.
188         require(0 < _withdrawAmt, "Amount must > 0");
189 
190         if (_withdrawAmt > _balanceOfVault) {
191             uint256 _diff = strategy.withdraw(_withdrawAmt.sub(_balanceOfVault));
192             token.safeTransfer(msg.sender, _balanceOfVault.add(_diff));
193         } else {
194             token.safeTransfer(msg.sender, _withdrawAmt);
195         }
196 
197         _burn(msg.sender, _shares);
198     }
```

Figure 5 source code of function withdraw

- Related Functions: *withdraw*, *safeTransfer*
- Safety Suggestion: *Strategy.pool* is used here as a benchmark, which is different from the one used in deposit, and there will be a little error.
- Fix result: The project team believes that the difference can be ignored.
- Result: Pass

3.5 Business analysis of Contract HarvestFarmer

(1) deposit

- Description: The Vault contract calls this function to deposit funds to the hfVault and hfStake contracts for profit.

```
193     function deposit(uint256 _amount) external onlyVault notVesting {
194         token.safeTransferFrom(msg.sender, address(this), _amount);
195         hfVault.deposit(_amount);
196         pool = pool.add(_amount);
197         hfStake.stake(hfVault.balanceOf(address(this)));
198     }
```

Figure 6 source code of function deposit

- Related Functions: *safeTransferFrom*, *deposit*, *stake*
- Safety Suggestion: None
- Result: Pass

(2) withdraw

- Description: The Vault contract withdraws the specified amount of funds to the Vault contract by calling this function.

```
208     function withdraw(uint256 _amount) external onlyVault notVesting returns (uint256) {
209         uint256 _fTokenBalance = (hfStake.balanceOf(address(this))).mul(_amount).div(pool);
210         hfStake.withdraw(_fTokenBalance);
211         hfVault.withdraw(hfVault.balanceOf(address(this)));
212 
213         uint256 _withdrawAmt = token.balanceOf(address(this));
214         token.safeTransfer(msg.sender, _withdrawAmt);
215         pool = pool.sub(_amount);
216         return _withdrawAmt;
217     }
```

Figure 7 source code of function withdraw

- Related Functions: *withdraw*, *safeTransfer*
- Safety Suggestion: None
- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the project xDVG. The project xDVG passed all audit items, The overall audit result is **Pass.**

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com