6th MAY 2021

SMART CONTRACT AUDIT REPORT

-

version v2.0

Smart Contract Security Audit and General Analysis

HAECHI AUDIT

Table of Contents

4 Issues (0 Critical, 2 Major, 2 Minor) Found

Table of Contents

About HAECHI AUDIT

01. Introduction

02. Summary

- Issues

03. Overview

- Contracts Subject to Audit
- Roles

04. Issues Found

- Major : The authority of Gov is set relatively large. (Found - v.1.0) (Intended - v.2.0)
- Major : balanceSnapshot can decrease by more than the actual withdrawal amount. (Found - v.1.0) (Intended - v.2.0)
- Minor : Tokens can be withdrawn even when the contract is paused. (Found - v.1.0) (Intended - v.2.0)
- Minor : There is a possibility that the inCaseTokensGetStuck() function would always be reverted. (Found - v.1.0) (Acknowledged - v.2.0)

05. Disclaimer

Appendix A. Test Results

About HAECHI AUDIT

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. HAECHI AUDIT provides specialized and professional smart contract security auditing and development services.

We are experts with years of experience in the research and development of blockchain technology. We are the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Cryptocurrency exchanges worldwide trust HAECHI AUDIT's security audit reports. Indeed, numerous clients have successfully listed on Huobi, OKEX, Upbit, and Bithumb, etc. after passing HAECHI AUDIT smart contract security audit.

Representative clients and partners include the Global Blockchain Project and Fortune Global 500 corporations, in addition to Ground X (Kakao Corp. subsidiary), Carry Protocol, Metadium, LG, Hanwha, and Shinhan Bank. Over 60 clients have benefited from our most reliable smart contract security audit and development services.

Inquiries: audit@haechi.io
Website: audit.haechi.io

01. Introduction

This report aims to audit the security of BeltFi earnV2 smart contract created by OZYS team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by OZYS team is soundly implemented and designed as specified in the published materials as well as whether the smart contract is secure in terms of security.

The issues discovered are categorized into CRITICAL, MAJOR, MINOR, TIPS according to their importance.

CRITICAL	Critical issues must be resolved as critical flaws that can harm a wide range of users.
MAJOR	Major issues require correction because they either have security problems or are implemented not as intended.
MINOR	Minor issues can potentially cause problems and therefore require correction.
TIPS	Tips issues are those that can improve the usability or efficiency of the code when corrected.

HAECHI AUDIT recommends OZYS team improve all issues discovered.

The following issue description uses the format of {file name}#{line number}, {contract name}#{function/variable name} to specify the code. For instance, *Sample.sol:20* points to the 20th line of Sample.sol file, and *Sample#fallback()* means the fallback() function of the Sample contract.

Please refer to the Appendix to check all results of the tests conducted for this report.

02. Summary

The codes used in this Audit can be found at Github (https://github.com/BeltFi/belt-contract/tree/main/contracts/earnV2). The last commit of the code used in this Audit is "af319aa4da0861b3b25f61f69a1fd935573b755b".

Issues

HAECHI AUDIT found 0 critical issues, 2 major issues, and 2 minor issues. There was 0 issue categorized as Tips as those that can improve the code's usability and efficiency.

Severity	Issue	Status
MAJOR	The authority of Gov is set relatively large	(Found - v.1.0) (Intended - v.2.0)
MAJOR	balanceSnapshot can decrease by more than the actual withdrawal amount	(Found - v.1.0) (Intended - v.2.0)
MINOR	Tokens can be withdrawn even when the contract is paused	(Found - v.1.0) (Intended - v.2.0)
MINOR	There is a possibility that the inCaseTokensGetStuck() function would always be reverted	(Found - v.1.0) (Acknowledged - v.2.0)

Update

[v2.0] - OZYS team confirmed that three issues are intended implementations and one issue has been acknowledged.

03. Overview

Contracts Subject to Audit

- defi
    - acryptos.sol
    - alpaca.sol
    - alphaHomora.sol
    - autoFarm.sol
    - ellipsis.sol
    - fortube.sol
    - pancake.sol
    - venus.sol
- strategies
    - Strategy.sol
- strategies/acrypto
    - StrategyACrypto.sol
    - StrategyACryptoImpl.sol
    - StrategyACryptoStorage.sol
- strategies/alpaca
    - StrategyAlpaca.sol
    - StrategyAlpacaImpl.sol
    - StrategyAlpacaStorage.sol
- strategies/alpha
    - StrategyAlpha.sol

    - StrategyAlphaImpl.sol
    - StrategyAlphaStorage.sol
- strategies/Auto
    - StrategyAuto.sol
    - StrategyAutoImpl.sol
    - StrategyAutoStorage.sol
- strategies/ellipsis
    - StrategyElliipsis.sol
    - StrategyElliipsisImpl.sol
    - StrategyElliipsisStorage.sol
- strategies/fortube
    - StrategyFortube.sol
    - StrategyFortubeImpl.sol
    - StrategyFortubeStorage.sol
- strategies/venus
    - StrategyVenusV2.sol
    - StrategyVenusV2Impl.sol
    - StrategyVenusV2Storage.sol
    - StrategyVenusV2WithRepaymentImpl.sol
- tokens
    - MultiStrategyToken.sol
    - MultiStrategyTokenImpl.sol
    - MultiStrategyTokenStorage.sol

- SingleStrategyToken.sol
- SingleStrategyToken2.sol
- SingleStrategyTokenImpl.sol
- SingleStrategyTokenImpl2.sol
- SingleStrategyTokenStorage.sol
- StrategyToken.sol

Roles

BeltFi earnV2 Smart contract has the following authorities.

- **Owner**
- **Gov**

Each authority can access the following functions.

Role	Functions
Owner	*StrategyACryptoImpl#deposit()* *StrategyACryptoImpl#withdraw()* *StrategyAlpacaImpl#deposit()* *StrategyAlpacaImpl#withdraw()* *StrategyAlphaImpl#deposit()* *StrategyAlphaImpl#withdraw()* *StrategyAutoImpl#deposit()* *StrategyAutoImpl#withdraw()* *StrategyEllipsisImpl#deposit()* *StrategyEllipsisImpl#withdraw()* *StrategyFortubeImpl#deposit()* *StrategyFortubeImpl#withdraw()* *StrategyVenusV2Impl#deposit()* *StrategyVenusV2Impl#withdraw()*
Gov	*Strategy#setWithdrawFee()* *StrategyACryptoImpl#pause()* *StrategyACryptoImpl#unpause()* *StrategyACryptoImpl#setbuyBackRate()* *StrategyACryptoImpl#setGov()*

	StrategyACryptoImpl#setHarvestFee() *StrategyACryptoImpl#inCaseTokensGetStuck()* *StrategyAlpacaImpl#pause()* *StrategyAlpacaImpl#unpause()* *StrategyAlpacaImpl#setbuyBackRate()* *StrategyAlpacaImpl#setGov()* *StrategyAlpacaImpl#inCaseTokensGetStuck()* *StrategyAlphaImpl#pause()* *StrategyAlphaImpl#unpause()* *StrategyAlphaImpl#setbuyBackRate()* *StrategyAlphaImpl#setGov()* *StrategyAlphaImpl#inCaseTokensGetStuck()* *StrategyAutoImpl#pause()* *StrategyAutoImpl#unpause()* *StrategyAutoImpl#setbuyBackRate()* *StrategyAutoImpl#setGov()* *StrategyAutoImpl#inCaseTokensGetStuck()* *StrategyEllipsisImpl#pause()* *StrategyEllipsisImpl#unpause()* *StrategyEllipsisImpl#setbuyBackRate()* *StrategyEllipsisImpl#setSafetyCoeff()* *StrategyEllipsisImpl#setGov()* *StrategyEllipsisImpl#inCaseTokensGetStuck()* *StrategyFortubeImpl#rebalance()* *StrategyFortubeImpl#pause()* *StrategyFortubeImpl#unpause()* *StrategyFortubeImpl#setbuyBackRate()* *StrategyFortubeImpl#setSafetyCoeff()* *StrategyFortubeImpl#setGov()* *StrategyFortubeImpl#wrapBNB()* *StrategyVenusV2Impl#rebalance()* *StrategyVenusV2Impl#pause()* *StrategyVenusV2Impl#unpause()* *StrategyVenusV2Impl#setbuyBackRate()* *StrategyVenusV2Impl#setSafetyCoeff()* *StrategyVenusV2Impl#setGov()* *StrategyVenusV2Impl#wrapBNB()* *MultiStrategyTokenImpl#rebalance()* *MultiStrategyTokenImpl#changeRatio()* *MultiStrategyTokenImpl#setStrategyActive()* *MultiStrategyTokenImpl#setRebalanceThreshold()* *MultiStrategyTokenImpl#inCaseTokensGetStuck()* *SingleStrategyTokenImpl#setEntranceFee()* *SingleStrategyTokenImpl2#inCaseTokensGetStuck()*

04. Issues Found

Major : The authority of Gov is set relatively large. (Found - v.1.0) (Intended - v.2.0)

MAJOR

```
448.    function inCaseTokensGetStuck(
449.    address _token,
450.    uint256 _amount,
451.    address _to
452.) public {
453.    require(msg.sender == govAddress, "!gov");
454.    require(_token != address(this), "!safe");
455.    if (_token == address(0)) {
456.        require(address(this).balance >= _amount, "amount greater than holding");
457.        _wrapBNB(_amount);
458.    } else if (_token == token) {
459.        require(balance() >= _amount, "amount greater than holding");
460.    }
461.    IERC20(_token).safeTransfer(_to, _amount);
462.}
```

Problem Statement

inCaseTokensGetStuck() is a function able to call only Gov address and sends a specific _token to _to for the amount of _amount.

However, because Gov address can transfer the token existing in the contract to a specific address, it may cause loss to users of the system and therefore is considered to be a factor that reduces the overall reliability of the contract.

The above issue may occur in StrategyACryptoImpl, StrategyAlpacaImpl, StrategyAlphaImpl, StrategyAutoImpl, StrategyEllipsisImpl, StrategyFortubeImpl, StrategyVenusV2Impl, StrategyVenusV2WithRepaymentImpl, MultiStrategyTokenImpl, SingleStrategyTokenImpl2 contract.

Recommendation

We advise you to restrict the types of tokens that can be sent.

Update

[v2.0] - OZYS team has implemented the function so that, as an exception, Gov can process a type of token not handled in strategy enters the contract. OZYS team also confirmed that the above issue is an intended implementation.

Major : balanceSnapshot can decrease by more than the actual withdrawal amount. (Found - v.1.0) (Intended - v.2.0)

MAJOR

```
188.   function withdraw(uint256 _wantAmt)
189.       override
190.       public
191.       onlyOwner
192.       nonReentrant
193.       returns (uint256)
194.   {
195.       _wantAmt =
      _wantAmt.mul(withdrawFeeDenom.sub(withdrawFeeNumer)).div(withdrawFeeDenom);
196.       balanceSnapshot = balanceSnapshot.sub(_wantAmt);
197.
198.       if(_stakedWantTokens() < _wantAmt) {
199.           _wantAmt = _stakedWantTokens();
200.       }
201.
202.       uint256 wantBal = IERC20(wantAddress).balanceOf(address(this));
203.       _withdraw(_wantAmt);
204.       wantBal = IERC20(wantAddress).balanceOf(address(this)).sub(wantBal);
205.       IERC20(wantAddress).safeTransfer(owner(), wantBal);
206.
207.       return wantBal;
208.   }
209.
```

Problem Statement

StrategyACryptoImpl#withdraw() function is implemented to change _wantAmt, the withdrawal amount, inside the function and updates the withdrawal result by decreasing the balanceSnapshot as much as _wantAmt. At this time, there is a possibility that balanceSnapshot decreases by more than the actual withdrawal amount.

Example

In the case of _stakedWantTokens() < _wantAmt(), the amount withdrawn in fact is _stakedWantTokens(). However, balanceSnapShot would decrease by _wantAmt(), which is larger than the amount withdrawn.

Recommendation

If the above situation is unintended, we recommend decreasing balanceSnapshot as much as the amount withdrawn by implementing the balanceSnapshot reduction statement after all changes to _wantAmt are concluded.

Update

[v2.0] - OZYS team implemented the code so that the actual withdrawal amount would differ from the amount of change in balanceSnapshot when the strategy has withdrawal fees and confirmed that the above issue is an intended implementation.

Minor : Tokens can be withdrawn even when the contract is paused.

(Found - v.1.0) (Intended - v.2.0)

MINOR

```
210.   function withdraw(uint256 _wantAmt)
211.       override
212.       public
213.       onlyOwner
214.       nonReentrant
215.       returns (uint256)
216.   {
217.       _wantAmt =
       _wantAmt.mul(withdrawFeeDenom.sub(withdrawFeeNumer)).div(withdrawFeeDenom);
218.       balanceSnapshot = balanceSnapshot.sub(_wantAmt);
219.
220.       if(_stakedWantTokens() < _wantAmt) {
221.           _wantAmt = _stakedWantTokens();
222.       }
223.
224.       uint256 wantBal = IERC20(wantAddress).balanceOf(address(this));
225.       _withdraw(_wantAmt);
226.       wantBal = IERC20(wantAddress).balanceOf(address(this)).sub(wantBal);
227.       IERC20(wantAddress).safeTransfer(owner(), wantBal);
228.
229.       return wantBal;
230.   }
231.
```

Problem Statement

Because there is no whenNotPaused modifier in the *withdraw()* function, tokens may be moved by using the *withdraw()* function even when the contract is paused by using the *pause()* function.

This issue may occur in StrategyACryptoImpl, StrategyAlpacaImpl, StrategyAlphaImpl, StrategyAutoImpl, StrategyEllipsisImpl, StrategyFortubeImpl, StrategyVenusV2Impl, StrategyVenusV2WithRepaymentImpl contract.

Recommendation

We advise controlling token movements when the contract is paused by adding whenNotPaused modifier to the *withdraw()* function as the *deposit()*, *earn()* functions where tokens move.

Update

[v2.0] - OZYS team distributed the owner of the strategy as singleMinter, implemented the code so that deposit pause / withdraw pause will be separately designed in singleMinter, and confirmed that the above issue is an intended implementation.

Minor : There is a possibility that the inCaseTokensGetStuck() function would always be reverted. (Found - v.1.0) (Acknowledged - v.2.0)

MINOR

```
463.    function inCaseTokensGetStuck(
464.    address _token,
465.    uint256 _amount,
466.    address _to
467.) public {
468.    require(msg.sender == govAddress, "!gov");
469.    require(_token != address(this), "!safe");
470.    if (_token == address(0)) {
471.        require(address(this).balance >= _amount, "amount greater than holding");
472.        _wrapBNB(_amount);
473.    } else if (_token == token) {
474.        require(balance() >= _amount, "amount greater than holding");
475.    }
476.    IERC20(_token).safeTransfer(_to, _amount);
477.}
```

Problem Statement

The *inCaseTokensGetStuck()* function basically sends _token to _to as much as _amount. It is determined that, at this time, the function is implemented to send WBNB when _token is address(0).

However, in the case of _token == address(0), the IERC20(_token).safeTransfer() statement in the 461st line always reverts.

This issue can occur in MultiStrategyTokenImpl, SingleStrategyTokenImpl2 contract.

Recommendation

We recommend adding a statement that transmits WBNB by categorizing cases when _token == address(0).

Update

[v2.0] - OZYS team acknowledges the above issue and confirmed that it will correct the issue in the contract version to be updated.

05. Disclaimer

This report does not guarantee investment advice, suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects in Ethereum and Solidity. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results are unit test results that cover the key logic of the smart contract subject to the security audit. Parts marked in red are test cases that failed to pass the test due to having issues.

```
StratACryptoSingle
  constructor()
    ✓ should set harvestFee
    ✓ should set govAddress
    ✓ should set vaultAddress
    ✓ should set BELTAddress
    ✓ should set vaultAddress
    ✓ should set wantAddress
    ✓ should set wantToBELTPath
    ✓ should set pancakeRouterAddress
  deposit()
    ✓ should transfer want owner -> address(this)
    ✓ should call vault.deposit
    ✓ should call acsFarm.deposit
    ✓ should update balanceSnapshot
  withdraw()
    1) should fail when paused
    ✓ should call vault.withdraw
    ✓ should call acsFarm.withdraw
    ✓ should transfer want address(this) -> owner
    2) should decrease wantAmt when wantAmt > stakedWantTokens()
  earn()
    ✓ should harvest ascFarm
    ✓ should swap ACS to Belt
    ✓ should transfer belt SACS -> buybackAddress
    ✓ should swap ACS to Want
    ✓ should call vault.deposit
    ✓ should call acsFarm.deposit
    ✓ should update balanceSnapshot
    ✓ should update lastEarnBlock
  pause()
    ✓ should fail when msg.sender is not owner
    ✓ should set approve 0 (56ms)
  unpause()
    ✓ should fail when msg.sender is not owner
```

```
    ✓ should set approve 0 (91ms)
  wantLockedTota()l
    ✓ should return total locked want
  setbuyBackRate()
    ✓ should fail when msg.sender is not owner
    ✓ should update buyBackRate
  setGov()
    ✓ should fail when msg.sender is not owner
    ✓ should update gov
  setHarvestFee()
    ✓ should fail when msg.sender is not owner
    ✓ should update harvestFee
  inCaseTokensGetStuck()
    ✓ should fail when msg.sender is not owner
    ✓ should fail when token is acs
    ✓ should fail when token is want
    ✓ should fail when token is vault
    ✓ should transfer token SACS -> to (40ms)
  buyBack()
    ✓ should return earnedAmt when buyBackRate is 0
  buyBackWant()
    ✓ should withdraw when curWantBal < buyBackWant (84ms)

StratAlpacaSingle
  constructor()
    ✓ should set poolID
    ✓ should set govAddress
    ✓ should set vaultAddress
    ✓ should set BELTAddress
    ✓ should set vaultAddress
    ✓ should set wantAddress
    ✓ should set wantToBELTPath
    ✓ should set uniRouterAddress
  deposit()
    ✓ should transfer want owner -> address(this) (72ms)
    ✓ should call vault.deposit (70ms)
    ✓ should call launch.deposit (165ms)
    ✓ should update balanceSnapshot (74ms)
  withdraw()
    ✓ should call vault.withdraw (58ms)
    ✓ should call launch.withdraw (58ms)
    ✓ should transfer want address(this) -> owner (63ms)
```

```
  earn()
    ✓ should harvest launch
    ✓ should swap alpa to Belt
    ✓ should transfer belt SAS -> buybackAddress
    ✓ should swap alpa to Want
    ✓ should call vault.deposit
    ✓ should call launch.deposit
    ✓ should update balanceSnapshot
    ✓ should update lastEarnBlock
  pause()
    ✓ should fail when msg.sender is not owner
    ✓ should set approve 0 (55ms)
  unpause()
    ✓ should fail when msg.sender is not owner
    ✓ should set approve 0 (86ms)
  wantLockedTota()l
    ✓ should return total locked want
  setbuyBackRate()
    ✓ should fail when msg.sender is not owner
    ✓ should update buyBackRate
  setGov()
    ✓ should fail when msg.sender is not owner
    ✓ should update gov
  inCaseTokensGetStuck()
    ✓ should fail when msg.sender is not owner
    ✓ should fail when token is acs
    ✓ should fail when token is want
    ✓ should fail when token is vault
    ✓ should transfer token SAS -> to (40ms)
  buyBack()
    ✓ should return earnedAmt when buyBackRate is 0
  buyBackWant()
    ✓ should withdraw when curWantBal < buyBackWant (63ms)

StratAlphaSingle
  constructor()
    ✓ should set govAddress
    ✓ should set BELTAddress
    ✓ should set wantAddress
    ✓ should set wantToBELTPath
    ✓ should set pancakeRouterAddress
  deposit()
```

```
    ✓ should transfer want owner -> address(this) (55ms)
    ✓ should call bank.deposit
    ✓ should update balanceSnapshot (54ms)
  withdraw()
    ✓ should call bank.withdraw (40ms)
    ✓ should transfer want address(this) -> owner (55ms)
  earn()
    ✓ should swap alpa to Belt
    ✓ should transfer belt SAS -> buybackAddress
    ✓ should swap alpa to Want
    ✓ should call bank.deposit
    ✓ should update balanceSnapshot
    ✓ should update lastEarnBlock
  pause()
    ✓ should fail when msg.sender is not owner
    ✓ should set approve 0 (51ms)
  unpause()
    ✓ should fail when msg.sender is not owner
    ✓ should set approve 0 (76ms)
  wantLockedTotal()
    ✓ should return total locked want
  setbuyBackRate()
    ✓ should fail when msg.sender is not owner
    ✓ should update buyBackRate
  setGov()
    ✓ should fail when msg.sender is not owner
    ✓ should update gov
  inCaseTokensGetStuck()
    ✓ should fail when msg.sender is not owner
    ✓ should fail when token is acs
    ✓ should fail when token is want
    ✓ should fail when token is vault
    ✓ should transfer token SAS -> to (40ms)
  buyBack()
    ✓ should return earnedAmt when buyBackRate is 0
  buyBackWant()
    ✓ should withdraw when curWantBal < buyBackWant (82ms)

StratAutoSingle
  constructor()
    ✓ should set govAddress
    ✓ should set BELTAddress
```

```
  ✓ should set wantAddress
  ✓ should set wantToBELTPath
  ✓ should set pancakeRouterAddress
deposit()
  ✓ should transfer want owner -> address(this) (47ms)
  ✓ should call autoFarm.deposit (45ms)
  ✓ should update balanceSnapshot
withdraw()
  ✓ should call autoFarm.withdraw
  ✓ should transfer want address(this) -> owner (42ms)
earn()
  ✓ should swap alpa to Belt
  ✓ should transfer belt SAS -> buybackAddress
  ✓ should swap alpa to Want
  ✓ should call autoFarm.deposit
  ✓ should update balanceSnapshot
  ✓ should update lastEarnBlock
pause()
  ✓ should fail when msg.sender is not owner
  ✓ should set approve 0 (51ms)
unpause()
  ✓ should fail when msg.sender is not owner
  ✓ should set approve 0 (99ms)
wantLockedTotal()
  ✓ should return total locked want
setbuyBackRate()
  ✓ should fail when msg.sender is not owner
  ✓ should update buyBackRate
setGov()
  ✓ should fail when msg.sender is not owner
  ✓ should update gov
inCaseTokensGetStuck()
  ✓ should fail when msg.sender is not owner
  ✓ should fail when token is acs
  ✓ should fail when token is want
  ✓ should transfer token SAS -> to (45ms)
buyBack()
  ✓ should return earnedAmt when buyBackRate is 0
buyBackWant()
  ✓ should withdraw when curWantBal < buyBackWant (58ms)

StrategyEllipsis
```

```
constructor()
  ✓ should set govAddress
  ✓ should set BELTAddress
  ✓ should set wantAddress
  ✓ should set pancakeRouterAddress
deposit()
  ✓ should transfer want owner -> address(this) (101ms)
  ✓ should call swap.addLiquidity (111ms)
  ✓ should call stake.deposit (91ms)
withdraw()
  ✓ should transfer want address(this) -> owner (173ms)
  ✓ should call stake.withdraw (126ms)
  ✓ should call swap.remove_liquidity_one_coin (107ms)
earn()
  ✓ should call stake.withdraw
  ✓ should call dist.exit
  ✓ should swap eps to want
  ✓ should call stake.deposit
  ✓ should transfer belt SAS -> buybackAddress
  ✓ should update lastEarnBlock
pause()
  ✓ should fail when msg.sender is not owner
  ✓ should set approve 0 (75ms)
unpause()
  ✓ should fail when msg.sender is not owner
  ✓ should set approve 0 (157ms)
wantLockedTotal()
  ✓ should return total locked want (118ms)
setbuyBackRate()
  ✓ should fail when msg.sender is not owner
  ✓ should update buyBackRate
setGov()
  ✓ should fail when msg.sender is not owner
  ✓ should update gov
inCaseTokensGetStuck()
  ✓ should fail when msg.sender is not owner
  ✓ should fail when token is eps
  ✓ should fail when token is eps3
  ✓ should fail when token is want
  ✓ should transfer token SAS -> to (40ms)
buyBack()
  ✓ should return earnedAmt when buyBackRate is 0
```

```
  setSafetyCoeff()
    ✓ should fail when msg.sender is not owner
    ✓ should fail when _safetyDenom is 0
    ✓ should fail when _safetyDenom > _safetyNumer
    ✓ should update numer, denom
  getTokenIndexInt()
    ✓ should return 0 when tokenAddr = busdAddress
    ✓ should return 1 when tokenAddr = usdcAddress

StrategyFortube
  constructor()
    ✓ should set govAddress
    ✓ should set BELTAddress
    ✓ should set wantAddress
    ✓ should set bankAddress
    ✓ should set wantToBELTPath
    ✓ should set uniRouterAddress
  deposit()
    ✓ should transfer want owner -> address(this) (84ms)
    ✓ should update balanceSnapshot (80ms)
  withdraw()
    ✓ should withdraw wbn (403ms)
    ✓ should rerpay borrow (412ms)
    ✓ should transfer want address(this) -> owner (227ms)
  earn()
    ✓ should swap for to Belt
    ✓ should transfer belt SF -> buybackAddress
    ✓ should swap alpa to Want
    ✓ should call bank.deposit
    ✓ should update balanceSnapshot
    ✓ should update lastEarnBlock
  pause()
    ✓ should fail when msg.sender is not owner
    ✓ should set approve 0 (68ms)
    ✓ should set approve 0 (64ms)
  unpause()
    ✓ should fail when msg.sender is not owner
    ✓ should set approve max (86ms)
    ✓ should set approve max (107ms)
  wantLockedTotal()
    ✓ should return total locked want
  setbuyBackRate()
```

```
  ✓ should fail when msg.sender is not owner
  ✓ should update buyBackRate
setGov()
  ✓ should fail when msg.sender is not owner
  ✓ should update gov
inCaseTokensGetStuck()
  ✓ should fail when msg.sender is not owner
  ✓ should fail when token is earned
  ✓ should fail when token is want
  ✓ should fail when token is vault
  ✓ should transfer token SAS -> to (58ms)
buyBack()
  ✓ should return earnedAmt when buyBackRate is 0
buyBackWant()
  ✓ should withdraw when curWantBal < buyBackWant (74ms)
  ✓ should withdraw when curWantBal < buyBackWant (115ms)
  ✓ should return 0 when input is 0
wrapBNB()
  ✓ should fail when msg.sender is owner
  ✓ should fail when want != wbn
  ✓ should wrapBNB (50ms)
unwrapBNB()
  ✓ should unwrapBNB
wantLockedInHere()
  ✓ should return locked want (45ms)
rebalance()
  ✓ should fail when msg.sender is owner
  ✓ should fail when _borrowRate > BORROW_RATE_MAX
  ✓ should fail when _borrowDepth > BORROW_DEPTH_MAX
  ✓ should update borrowRate (71ms)
  ✓ should update borrowDepth (80ms)
deleverageOnce()
  ✓ should withdrawUnderlying (55ms)
  ✓ should _repayBorrow (49ms)
  ✓ should withdrawUnderlying (95ms)
leverage()
  ✓ should deposit bank (122ms)
  ✓ should borrow bank (99ms)
farm()
  ✓ should deposit bank (114ms)
  ✓ should deposit bank when want is wbnb (138ms)
deleverage()
```

```
    ✓ just return when _delevPartial & wantBal>=_minAmt

StrategyVenusV2
  constructor()
    ✓ should set govAddress
    ✓ should set BELTAddress
    ✓ should set wantAddress
    ✓ should set wantToBELTPath
    ✓ should set uniRouterAddress
  deposit()
    ✓ should transfer want owner -> address(this) (93ms)
    ✓ should update balanceSnapshot (77ms)
  withdraw()
    ✓ should withdraw wbn (404ms)
    ✓ should rerpay borrow (316ms)
    ✓ should transfer want address(this) -> owner (221ms)
  earn()
    ✓ should claim venus
    ✓ should swap for to Belt
    ✓ should transfer belt SF -> buybackAddress
    ✓ should swap alpa to Want
    ✓ should call bank.deposit
    ✓ should update balanceSnapshot
    ✓ should update lastEarnBlock
  pause()
    ✓ should fail when msg.sender is not owner
    ✓ should set approve 0
    ✓ should set approve 0 (54ms)
  unpause()
    ✓ should fail when msg.sender is not owner
    ✓ should set approve max (88ms)
    ✓ should set approve max (89ms)
  wantLockedTotal()
    ✓ should return total locked want
  setbuyBackRate()
    ✓ should fail when msg.sender is not owner
    ✓ should fail when new rate is too big
    ✓ should update buyBackRate
  setGov()
    ✓ should fail when msg.sender is not owner
    ✓ should update gov
  inCaseTokensGetStuck()
```

```
  ✓ should fail when msg.sender is not owner
  ✓ should fail when token is earned
  ✓ should fail when token is want
  ✓ should fail when token is vault
  ✓ should transfer token SAS -> to (47ms)
buyBack()
  ✓ should return earnedAmt when buyBackRate is 0
buyBackWant()
  ✓ should withdraw when curWantBal < buyBackWant (90ms)
  ✓ should withdraw when curWantBal < buyBackWant (136ms)
  ✓ should return 0 when input is 0
wrapBNB()
  ✓ should fail when msg.sender is owner
  ✓ should fail when want != wbn
  ✓ should wrapBNB (50ms)
unwrapBNB()
  ✓ should unwrapBNB
wantLockedInHere()
  ✓ should return locked want (47ms)
rebalance()
  ✓ should fail when msg.sender is owner
  ✓ should fail when _borrowRate > BORROW_RATE_MAX
  ✓ should fail when _borrowDepth > BORROW_DEPTH_MAX
  ✓ should update borrowRate (76ms)
  ✓ should update borrowDepth (78ms)
deleverageOnce()
  ✓ should withdrawUnderlying (51ms)
  ✓ should _repayBorrow (52ms)
  ✓ should withdrawUnderlying (98ms)
leverage()
  ✓ should deposit bank
  ✓ should borrow bank (99ms)
farm()
  ✓ should deposit bank
 ✓ should deposit bank when want is wbnb
deleverage()
  ✓ just return when _delevPartial & wantBal>=_minAmt

MultiStrategyToken
 constructor()
  ✓ should set token
  ✓ should set strategies
```

```
  ✓ should set govAddress
  ✓ should set entranceFeeNumer
  ✓ should set entranceFeeDenom
  ✓ should set isWbnb
  ✓ should set ratioTotal
  ✓ should set depositActiveCount
  ✓ should set withdrawActiveCount
  ✓ should set rebalanceThresholdNumer
  ✓ should set rebalanceThresholdDenom
deposit()
  ✓ should fail when amount is zero
  ✓ should fail share < minShare (179ms)
 valid case
   ✓ should deposit strategy token
   ✓ should mint strategy token to msg.sender
   ✓ should transfer toekn msg.sender -> contract
   ✓ should transfer toekn msg.sender -> contract (190ms)
depositBNB()
  ✓ should fail when wnbn != token
  ✓ should fail when msg.value is zero
 valid case
   ✓ should mint strategy token to msg.sender
   ✓ should wrap wbnb
withdraw()
  ✓ should fail when share is zero
  ✓ should fail share > isBalance
 valid case
   ✓ should burn strategy token of msg.sender (360ms)
   ✓ should strategy#withdraw (365ms)
   ✓ should transfer token to msg.sender (370ms)
   ✓ should burn strategy token of msg.sender (430ms)
withdrawBNB()
  ✓ should fail when wnbn != token
 valid case
   ✓ should burn strategy token of msg.sender
   ✓ should strategy#withdraw
balance()
  ✓ should return token balanceOf contract
getPricePerFullShare()
  ✓ should return Price Per FullShare (258ms)
amountToShares()
  ✓ should return amount when initialized
```

```
  ✓ should return appropriate share when not initialized yet (252ms)
setEntranceFee()
  ✓ should be reverted
unwrapBNB()
  ✓ should withdraw wbnb (38ms)
inCaseTokensGetStuck()
  ✓ should fail when msg.sender is not owner
  ✓ should fail when token is this.address
  ✓ should fail when insufficient balance
  ✓ should fail when insufficient balance()
  ✓ should transfer token MST -> to (49ms)
  4) should wrapBNB
  ✓ should wrapBNB (52ms)
strategyCount()
  ✓ should return strategy length
setRebalanceThreshold()
  ✓ should fail when msg.sender is not gov
  ✓ should fail when denom is 0
  ✓ should fail when denom < numer
  ✓ should update rebalanceThreshold param
setStrategyActive()
  ✓ should fail when msg.sender is not gov
  ✓ should fail when index is over str count
  ✓ should fail when isActive = b
 isDeposit
    ✓ should decrease depositActiveCount()
    ✓ should increase depositActiveCount() (45ms)
 !isDeposit
    ✓ should decrease withdrawActiveCount()
    ✓ should increase depositActiveCount() (43ms)
sharesToAmount()
  ✓ should return appropriate amount (264ms)
amountToShares()
  ✓ should return appropriate amount (262ms)
getMaxWithdrawableShares()
  ✓ should fail when totalSupply = 0
  ✓ should return appropriate amountToShares (290ms)
changeRatio()
  ✓ should fail when msg.sender is not gov
  ✓ should fail when index > str.length
  ✓ should update ratios
  ✓ should update ratioTotal
```

```
  rebalance()
    ✓ should fail when msg.sender is not gov
    ✓ should withdraw stoken (251ms)
    ✓ should deposit stoken (258ms)
  findMostLockedStrategy()
    ✓ should return appropriate lockedMostAddr when cuttent > lockedBalance (112ms)

SingleStrategyToken2
  constructor()
    ✓ should set token
    ✓ should set strategy
    ✓ should set govAddress
    ✓ should set entranceFeeNumer
    ✓ should set entranceFeeDenom
    ✓ should set isWbnb
  deposit()
    ✓ should fail when amount is zero
    ✓ should fail share < minShare (75ms)
    valid case
      ✓ should mint strategy token to msg.sender
      ✓ should transfer toekn msg.sender -> contract
      ✓ should transfer toekn msg.sender -> contract (78ms)
  depositBnb()
    ✓ should fail when wnbn != token
    ✓ should fail when msg.value is zero
    valid case
      ✓ should mint strategy token to msg.sender
      ✓ should wrap wbnb
  withdraw()
    ✓ should fail when share is zero
    ✓ should fail share > isBalance
    valid case
      ✓ should burn strategy token of msg.sender
      ✓ should strategy#withdraw
      ✓ should transfer token to msg.sender
  withdrawBNB()
    ✓ should fail when wnbn != token
    valid case
      ✓ should burn strategy token of msg.sender
      ✓ should strategy#withdraw
  withdrawBNB()
    ✓ should return token balanceOf contract
```

```
  getPricePerFullShare()
    ✓ should return Price Per FullShare (85ms)
  amountToShares()
    ✓ should return amount when initialized
    ✓ should return appropriate share when not initialized yet (81ms)
  setEntranceFee()
    ✓ should be reverted
  unwrapBNB()
    ✓ should withdraw wbnb
  inCaseTokensGetStuck()
    ✓ should fail when msg.sender is not owner
    ✓ should fail when token is this.address
    ✓ should fail when insufficient balance
    ✓ should fail when insufficient balance()
    ✓ should transfer token SST2 -> to (56ms)
    5) should wrapBNB
    ✓ should wrapBNB (54ms)

SingleStrategyToken
  constructor()
    ✓ should set token
    ✓ should set strategy
    ✓ should set govAddress
    ✓ should set entranceFeeNumer
    ✓ should set entranceFeeDenom
    ✓ should set isWbnb
  deposit()
    ✓ should fail when amount is zero
    ✓ should fail share < minShare (57ms)
    valid case
      ✓ should mint strategy token to msg.sender
      ✓ should transfer toekn msg.sender -> contract
      ✓ should transfer toekn msg.sender -> contract (79ms)
  depositBNB()
    ✓ should fail when wnbn != token
    ✓ should fail when msg.value is zero
    valid case
      ✓ should mint strategy token to msg.sender
      ✓ should wrap wbnb
  withdraw()
    ✓ should fail when share is zero
    ✓ should fail share > isBalance
```

```
  valid case
    ✓ should burn strategy token of msg.sender
    ✓ should strategy#withdraw
    ✓ should transfer token to msg.sender
withdrawBNB()
  ✓ should fail when wnbn != token
  valid case
    ✓ should burn strategy token of msg.sender
    ✓ should strategy#withdraw
withdrawBNB()
  ✓ should return token balanceOf contract
getPricePerFullShare()
  ✓ should return Price Per FullShare (96ms)
amountToShares()
  ✓ should return amount when initialized
  ✓ should return appropriate share when not initialized yet (88ms)
setEntranceFee()
  ✓ should be reverted
unwrapBNB()
  ✓ should withdraw wbnb
supplyStrategy()
  ✓ should fail when unsufficient supply (40ms)
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/earnV2/defi					
acryptos.sol	**100**	**100**	**100**	**100**	
alpaca.sol	**100**	**100**	**100**	**100**	
alphaHomora.sol	**100**	**100**	**100**	**100**	
autoFarm.sol	**100**	**100**	**100**	**100**	
ellipsis.sol	**100**	**100**	**100**	**100**	
fortube.sol	**100**	**100**	**100**	**100**	
pancake.sol	**100**	**100**	**100**	**100**	
venus.sol	**100**	**100**	**100**	**100**	
contracts/earnV2/strategies/					
Strategy.sol	**100**	**100**	**100**	**100**	

contracts/earnV2/strategies/acrypto					
StrategyACrypto.sol	**100**	**100**	**100**	**100**	
StrategyACryptoImpl.sol	**100**	**100**	**100**	**100**	
StrategyACryptoStorage.sol	**100**	**100**	**100**	**100**	
contracts/earnV2/strategies/alpaca					
StrategyAlpaca.sol	**100**	**100**	**100**	**100**	
StrategyAlpacaImpl.sol	**100**	**100**	**100**	**100**	
StrategyAlpacaStorage.sol	**100**	**100**	**100**	**100**	
contracts/earnV2/strategies/alpha					
StrategyAlpha.sol	**100**	**100**	**100**	**100**	
StrategyAlphaImpl.sol	**100**	**100**	**100**	**100**	
StrategyAlphaStorage.sol	**100**	**100**	**100**	**100**	
contracts/earnV2/strategies/auto					
StrategyAuto.sol	**100**	**100**	**100**	**100**	
StrategyAutoImpl.sol	**100**	**100**	**100**	**100**	
StrategyAutoStorage.sol	**100**	**100**	**100**	**100**	
contracts/earnV2/strategies/ellipsis					
StrategyEllipsis.sol	**100**	**100**	**100**	**100**	
StrategyEllipsisImpl.sol	**100**	**100**	**100**	**100**	
StrategyEllipsisStorage.sol	**100**	**100**	**100**	**100**	
contracts/earnV2/strategies/fortube					
StrategyFortube.sol	**100**	**100**	**100**	**100**	
StrategyFortubeImpl.sol	**100**	**100**	**100**	**100**	
StrategyFortubeStorage.sol	**100**	**100**	**100**	**100**	
contracts/earnV2/strategies/venus					
StrategyVenusV2.sol	**100**	**100**	**100**	**100**	
StrategyVenusV2Impl.sol	**100**	**100**	**100**	**100**	

StrategyVenusV2Storage.sol	**100**	**100**	**100**	**100**	
StrategyVenusV2WithRepaymentImpl.sol	**100**	**100**	**100**	**100**	
contracts/earnV2/tokens					
MultiStrategyToken.sol	**100**	**100**	**100**	**100**	
MultiStrategyTokenImpl.sol	**100**	**100**	**100**	**100**	
MultiStrategyTokenStorage.sol	**100**	**100**	**100**	**100**	
SingleStrategyToken.sol	**100**	**100**	**100**	**100**	
SingleStrategyToken2.sol	**100**	**100**	**100**	**100**	
SingleStrategyTokenImpl.sol	**100**	**100**	**100**	**100**	
SingleStrategyTokenImpl2.sol	**100**	**100**	**100**	**100**	
SingleStrategyTokenStorage.sol	**100**	**100**	**100**	**100**	
StrategyToken.sol	**100**	**100**	**100**	**100**	

[표 1] Test Case Coverage