Autofarm.network V2

Security Code Review

https://twitter.com/VidarTheAuditor - 19 January 2021

Overview

Project Summary

Project Name	Autofarm
Description	
Platform	**Binance Smart Chain**, Solidity
Contracts	https://github.com/autofarm-network/autofarmV2
	commit ad26111fb74b3c63e5af463d766cf2aafa8471c4

Executive Summary

Binance Smart Chain contracts were provided.

We have run extensive static analysis of the codebase as well as standard security assessment utilising industry approved tools.

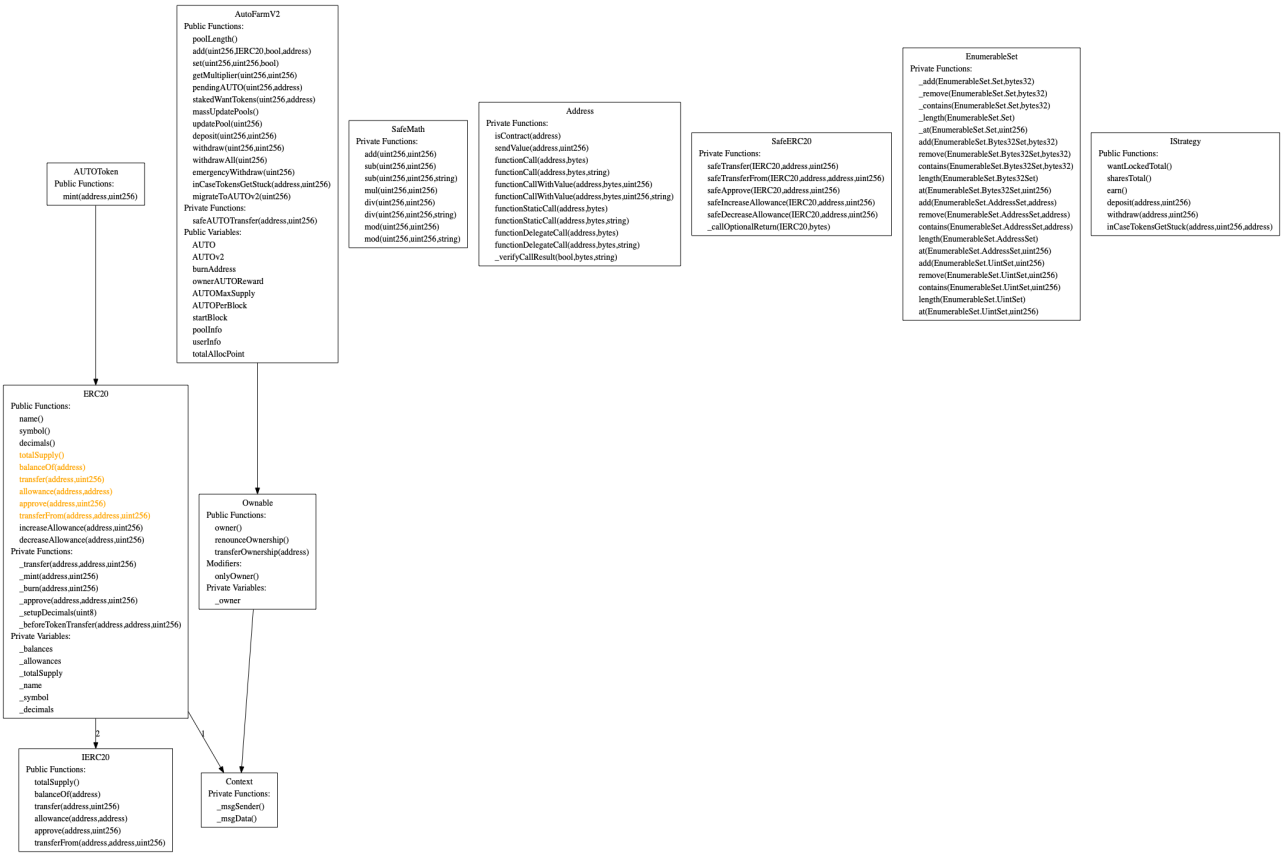
There were some recommendations being issued regarding some parts of the system. They were fixed by the team and verified.

Disclaimer: The analysis did not include any tokenomics analysis (e.g. APY rates etc).

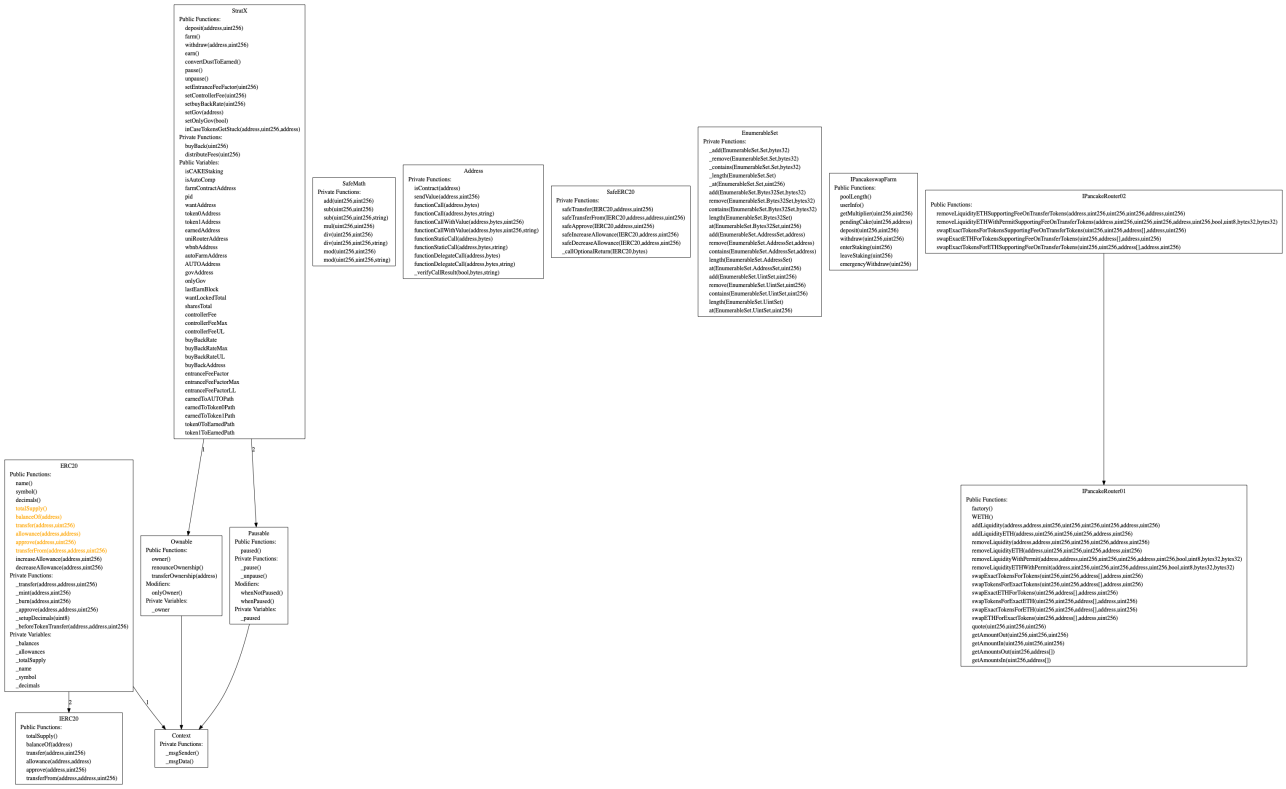
Architecture & Standards

Please find below the calling architecture of the reviewed contracts.

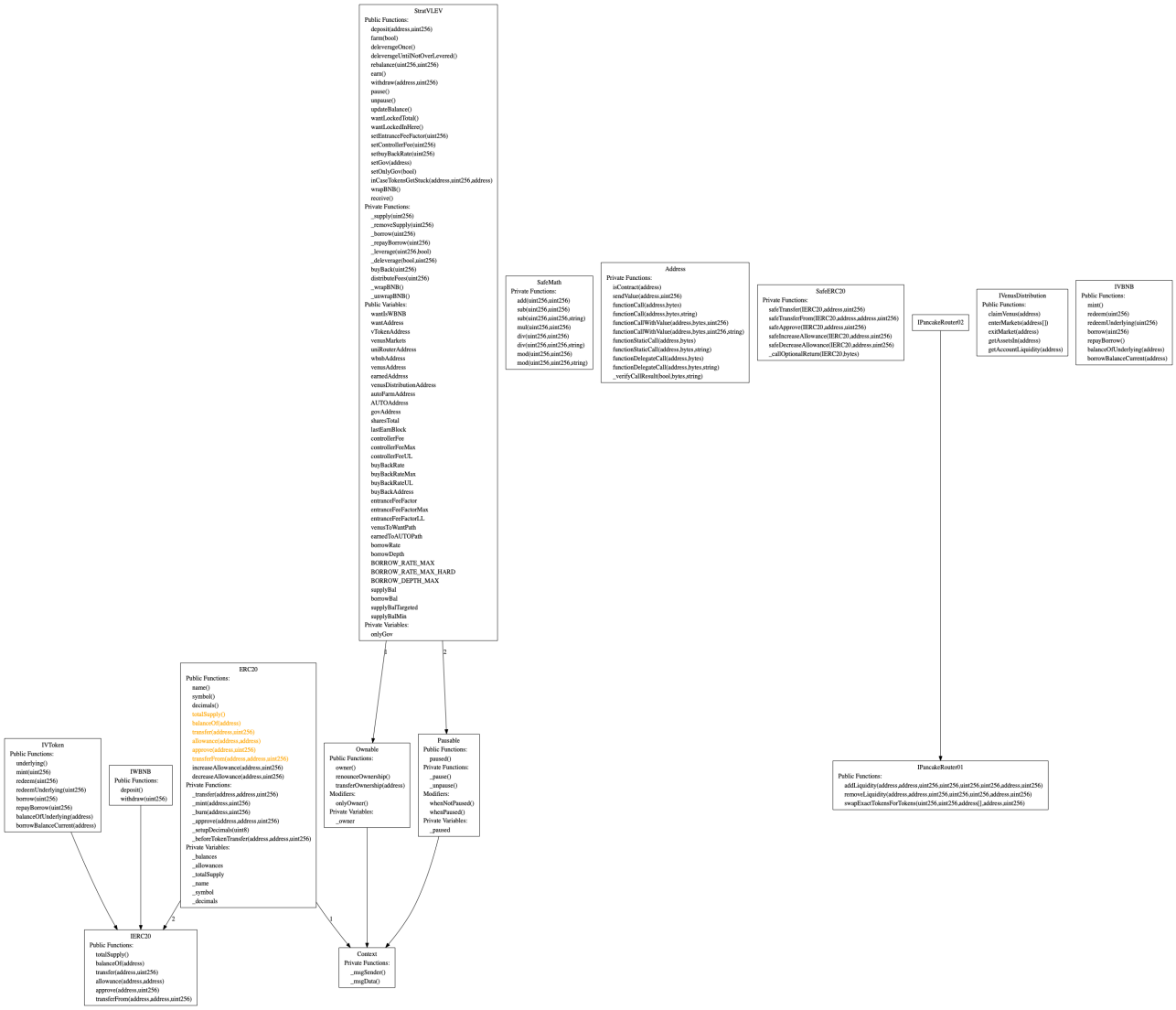
AutoFarmV2:



StratX (first strategy)



StratVLEV


StratVLEV
Public Functions:
deposit(address,uint256)
farm(bool)
deleverageOnce()
deleverageUntilNotOverLevered()
rebalance(uint256,uint256)
earn()
withdraw(address,uint256)
pause()
unpause()
updateBalance()
wantLockedTotal()
wantLockedInHere()
setEntrancerFeeFactor(uint256)
setControllerFee(uint256)
setbuyBackRate(uint256)
setGov(address)
setOnlyGov(bool)
inCaseTokensGetStuck(address,uint256,address)
wrapBNB()
receive()
Private Functions:
_supply(uint256)
_removeSupply(uint256)
_borrow(uint256)
_repayBorrow(uint256)
_leverage(uint256,bool)
_deleverage(bool,uint256)
buyBack(uint256)
distributeFees(uint256)
_wrapBNB()
_unwrapBNB()
Public Variables:
wantLWBNB
wantAddress
vTokenAddress
venusMarkets
uniRouterAddress
wbnbAddress
venusAddress
earnedAddress
venusDistributionAddress
autoFarmAddress
AUTOAddress
govAddress
sharesTotal
lastEarnBlock
controllerFee
controllerFeeMax
controllerFeeUL
buyBackRate
buyBackRateMax
buyBackRateUL
buyBackAddress
entranceFeeFactor
entranceFeeFactorMax
entranceFeeFactorLL
venusToWantPath
earnedToAUTOPath
borrowRate
borrowDepth
BORROW_RATE_MAX
BORROW_RATE_MAX_HARD
BORROW_DEPTH_MAX
supplyBal
borrowBal
supplyBalTargeted
supplyBalMin
Private Variables:
onlyGov
SafeMath
Private Functions:
add(uint256,uint256)
sub(uint256,uint256)
sub(uint256,uint256,string)
mul(uint256,uint256)
div(uint256,uint256)
div(uint256,uint256,string)
mod(uint256,uint256)
mod(uint256,uint256,string)
Address
Private Functions:
isContract(address)
sendValue(address,uint256)
functionCall(address,bytes)
functionCall(address,bytes,string)
functionCallWithValue(address,bytes,uint256)
functionCallWithValue(address,bytes,uint256,string)
functionStaticCall(address,bytes)
functionStaticCall(address,bytes,string)
functionDelegateCall(address,bytes)
functionDelegateCall(address,bytes,string)
_verifyCallResult(bool,bytes,string)
SafeERC20
Private Functions:
safeTransfer(IERC20,address,uint256)
safeTransferFrom(IERC20,address,address,uint256)
safeApprove(IERC20,address,uint256)
safeIncreaseAllowance(IERC20,address,uint256)
safeDecreaseAllowance(IERC20,address,uint256)
_callOptionalReturn(IERC20,bytes)
IPancakeRouter02
IVenusDistribution
Public Functions:
claimVenus(address)
enterMarkets(address[])
exitMarket(address)
getAssetsIn(address)
getAccountLiquidity(address)
IVBNB
Public Functions:
mint()
redeem(uint256)
redeemUnderlying(uint256)
borrow(uint256)
repayBorrow()
balanceOfUnderlying(address)
borrowBalanceCurrent(address)
ERC20
Public Functions:
name()
symbol()
decimals()
totalSupply()
balanceOf(address)
transfer(address,uint256)
allowance(address,address)
approve(address,uint256)
transferFrom(address,address,uint256)
increaseAllowance(address,uint256)
decreaseAllowance(address,uint256)
Private Functions:
_transfer(address,address,uint256)
_mint(address,uint256)
_burn(address,uint256)
_approve(address,address,uint256)
_setupDecimals(uint8)
_beforeTokenTransfer(address,address,uint256)
Private Variables:
_balances
_allowances
_totalSupply
_name
_symbol
_decimals
Ownable
Public Functions:
owner()
renounceOwnership()
transferOwnership(address)
Modifiers:
onlyOwner()
Private Variables:
_owner
Pausable
Public Functions:
paused()
Private Functions:
_pause()
_unpause()
Modifiers:
whenNotPaused()
whenPaused()
Private Variables:
_paused
IVToken
Public Functions:
underlying()
mint(uint256)
redeem(uint256)
redeemUnderlying(uint256)
borrow(uint256)
repayBorrow(uint256)
balanceOfUnderlying(address)
borrowBalanceCurrent(address)
IWBNB
Public Functions:
deposit()
withdraw(uint256)
IPancakeRouter01
Public Functions:
addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256)
removeLiquidity(address,address,uint256,uint256,uint256,address,uint256)
swapExactTokensForTokens(uint256,uint256,address[],address,uint256)
IERC20
Public Functions:
totalSupply()
balanceOf(address)
transfer(address,uint256)
allowance(address,address)
approve(address,uint256)
transferFrom(address,address,uint256)
Context
Private Functions:
_msgSender()
_msgData()
1
2
1
2
1
2

AUTOToken contract is fully BIP20 compatible.

```
# Check AUTOToken

## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> () (correct return value)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> () (correct return value)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [✓] transfer(address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> () (correct return value)
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> () (correct return value)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> () (correct return value)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> () (correct return value)
        [✓] symbol() is view
[✓] decimals() is present
        [✓] decimals() -> () (correct return value)
        [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed


        [✓] AUTOToken has increaseAllowance(address,uint256)
```

Findings

Number of contracts: 13+15+11 (including inherited ones)

Use: SafeMath

Name	# functions	ERCS	ERC20 info	Complex code	Features
SafeMath	8			No	
Address	11			No	Send ETH Delegatecall Assembly
SafeERC20	6			No	Send ETH Tokens interaction
EnumerableSet	20			No	
AUTOToken	27	ERC20	∞ Minting Approve Race Cond.	No	
IStrategy	6			No	
AutoFarmV2	22			No	Send ETH Tokens interaction

AUTOToken has minting capabilities, however the owner is moved to AutoFarm contract so it **does not** possesses any risks. See Deployment section for more details.

Name	# functions	ERCS	ERC20 info	Complex code	Features
SafeMath	8			No	
ERC20	26	ERC20	No Minting Approve Race Cond.	No	
Address	11			No	Send ETH Delegatecall Assembly
SafeERC20	6			No	Send ETH Tokens interaction
IPancakeRouter02	3			No	
IVenusDistribution	5			No	
IWBNB	8	ERC20	No Minting Approve Race Cond.	No	Receive ETH
IVBNB	7			No	Receive ETH
IVToken	14	ERC20	∞ Minting Approve Race Cond.	No	
StratVLEV	43			No	Receive ETH Send ETH Tokens interaction

Name	# functions	ERCS	ERC20 info	Complex code	Features
SafeMath	8			No	
ERC20	26	ERC20	No Minting Approve Race Cond.	No	
Address	11			No	Send ETH Delegatecall Assembly
SafeERC20	6			No	Send ETH Tokens interaction
EnumerableSet	20			No	
IPancakeswapFarm	9			No	
IPancakeRouter02	24			No	Receive ETH
StratX	28			Yes	Send ETH Tokens interaction

Static Analysis Findings

High issues:

Possible re-entrancy: **[FIXED]**

```
Reentrancy in StratVLEV.farm(bool) (StratVLEV.sol#1546-1557):
        External calls:
        - _unwrapBNB() (StratVLEV.sol#1548)
                - IWBNB(wbnbAddress).withdraw(wbnbBal) (StratVLEV.sol#1882)
        - _leverage(address(this).balance,_withLev) (StratVLEV.sol#1549)
                - IVToken(vTokenAddress).borrow(_amount) (StratVLEV.sol#1505)
                - IVBNB(vTokenAddress).mint{value: _amount}() (StratVLEV.sol#1494)
                - IVToken(vTokenAddress).mint(_amount) (StratVLEV.sol#1496)
        - _leverage(wantLockedInHere(),_withLev) (StratVLEV.sol#1551)
                - IVToken(vTokenAddress).borrow(_amount) (StratVLEV.sol#1505)
                - IVBNB(vTokenAddress).mint{value: _amount}() (StratVLEV.sol#1494)
                - IVToken(vTokenAddress).mint(_amount) (StratVLEV.sol#1496)
        - updateBalance() (StratVLEV.sol#1554)
                - supplyBal = IVToken(vTokenAddress).balanceOfUnderlying(address(this)) (StratVLEV.sol#1809)
                - borrowBal = IVToken(vTokenAddress).borrowBalanceCurrent(address(this)) (StratVLEV.sol#1810)
        - deleverageUntilNotOverLevered() (StratVLEV.sol#1556)
                - supplyBal = IVToken(vTokenAddress).balanceOfUnderlying(address(this)) (StratVLEV.sol#1809)
                - IVToken(vTokenAddress).redeemUnderlying(_amount) (StratVLEV.sol#1501)
                - borrowBal = IVToken(vTokenAddress).borrowBalanceCurrent(address(this)) (StratVLEV.sol#1810)
                - IVBNB(vTokenAddress).repayBorrow{value: _amount}() (StratVLEV.sol#1510)
                - IVToken(vTokenAddress).repayBorrow(_amount) (StratVLEV.sol#1512)
                - IWBNB(wbnbAddress).withdraw(wbnbBal) (StratVLEV.sol#1882)
        External calls sending eth:
        - _leverage(address(this).balance,_withLev) (StratVLEV.sol#1549)
                - IVBNB(vTokenAddress).mint{value: _amount}() (StratVLEV.sol#1494)
        - _leverage(wantLockedInHere(),_withLev) (StratVLEV.sol#1551)
                - IVBNB(vTokenAddress).mint{value: _amount}() (StratVLEV.sol#1494)
        - deleverageUntilNotOverLevered() (StratVLEV.sol#1556)
                - IVBNB(vTokenAddress).repayBorrow{value: _amount}() (StratVLEV.sol#1510)
        State variables written after the call(s):
        - deleverageUntilNotOverLevered() (StratVLEV.sol#1556)
                - borrowBal = IVToken(vTokenAddress).borrowBalanceCurrent(address(this)) (StratVLEV.sol#1810)
Reentrancy in StratVLEV.withdraw(address,uint256) (StratVLEV.sol#1743-1773):
        External calls:
        - _deleverage(true,_wantAmt.sub(wantBal)) (StratVLEV.sol#1757)
                - supplyBal = IVToken(vTokenAddress).balanceOfUnderlying(address(this)) (StratVLEV.sol#1809)
                - IVToken(vTokenAddress).redeemUnderlying(_amount) (StratVLEV.sol#1501)
                - IVBNB(vTokenAddress).repayBorrow{value: _amount}() (StratVLEV.sol#1510)
                - borrowBal = IVToken(vTokenAddress).borrowBalanceCurrent(address(this)) (StratVLEV.sol#1810)
                - IVToken(vTokenAddress).repayBorrow(_amount) (StratVLEV.sol#1512)
                - IWBNB(wbnbAddress).withdraw(wbnbBal) (StratVLEV.sol#1882)
                - IWBNB(wbnbAddress).deposit{value: bnbBal}() (StratVLEV.sol#1874)
                - IVToken(vTokenAddress).redeem(vTokenBal) (StratVLEV.sol#1661)
        - _wrapBNB() (StratVLEV.sol#1759)
                - IWBNB(wbnbAddress).deposit{value: bnbBal}() (StratVLEV.sol#1874)
        - IERC20(wantAddress).safeTransfer(autoFarmAddress,_wantAmt) (StratVLEV.sol#1768)
        - farm(true) (StratVLEV.sol#1770)
                - IVToken(vTokenAddress).borrow(_amount) (StratVLEV.sol#1505)
                - supplyBal = IVToken(vTokenAddress).balanceOfUnderlying(address(this)) (StratVLEV.sol#1809)
                - IVToken(vTokenAddress).redeemUnderlying(_amount) (StratVLEV.sol#1501)
                - borrowBal = IVToken(vTokenAddress).borrowBalanceCurrent(address(this)) (StratVLEV.sol#1810)
                - IVBNB(vTokenAddress).mint{value: _amount}() (StratVLEV.sol#1494)
                - IVBNB(vTokenAddress).repayBorrow{value: _amount}() (StratVLEV.sol#1510)
                - IVToken(vTokenAddress).repayBorrow(_amount) (StratVLEV.sol#1512)
                - IWBNB(wbnbAddress).withdraw(wbnbBal) (StratVLEV.sol#1882)
                - IVToken(vTokenAddress).mint(_amount) (StratVLEV.sol#1496)
        External calls sending eth:
        - _deleverage(true,_wantAmt.sub(wantBal)) (StratVLEV.sol#1757)
                - IVBNB(vTokenAddress).repayBorrow{value: _amount}() (StratVLEV.sol#1510)
                - IWBNB(wbnbAddress).deposit{value: bnbBal}() (StratVLEV.sol#1874)
        - _wrapBNB() (StratVLEV.sol#1759)
                - IWBNB(wbnbAddress).deposit{value: bnbBal}() (StratVLEV.sol#1874)
        - farm(true) (StratVLEV.sol#1770)
                - IVBNB(vTokenAddress).mint{value: _amount}() (StratVLEV.sol#1494)
                - IVBNB(vTokenAddress).repayBorrow{value: _amount}() (StratVLEV.sol#1510)
        State variables written after the call(s):
        - farm(true) (StratVLEV.sol#1770)
                - borrowBal = IVToken(vTokenAddress).borrowBalanceCurrent(address(this)) (StratVLEV.sol#1810)
        - farm(true) (StratVLEV.sol#1770)
                - supplyBal = IVToken(vTokenAddress).balanceOfUnderlying(address(this)) (StratVLEV.sol#1809)
        - farm(true) (StratVLEV.sol#1770)
                - supplyBalMin = borrowBal.mul(1000).div(BORROW_RATE_MAX_HARD) (StratVLEV.sol#1812)
        - farm(true) (StratVLEV.sol#1770)
                - supplyBalTargeted = borrowBal.mul(1000).div(borrowRate) (StratVLEV.sol#1811)
```

The recommendation is to use the *check-effects-interactions* pattern or use nonReentrant modifier (https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard-nonReentrant--)

Medium issues:

Divide before multiply:

```
AutoFarmV2.pendingAUTO(uint256,address) (AutoFarmV2.sol#1486-1507) performs a multiplication on the result of a division:
        -AUTOReward = multiplier.mul(AUTOPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (AutoFarmV2.sol#1498-1501)
        -accAUTOPerShare = accAUTOPerShare.add(AUTOReward.mul(1e12).div(sharesTotal)) (AutoFarmV2.sol#1502-1504)
AutoFarmV2.updatePool(uint256) (AutoFarmV2.sol#1536-1562) performs a multiplication on the result of a division:
        -AUTOReward = multiplier.mul(AUTOPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (AutoFarmV2.sol#1547-1550)
        -AUTOToken(AUTOv2).mint(owner(),AUTOReward.mul(ownerAUTOReward).div(1000)) (AutoFarmV2.sol#1552-1555)
AutoFarmV2.updatePool(uint256) (AutoFarmV2.sol#1536-1562) performs a multiplication on the result of a division:
        -AUTOReward = multiplier.mul(AUTOPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (AutoFarmV2.sol#1547-1550)
        -pool.accAUTOPerShare = pool.accAUTOPerShare.add(AUTOReward.mul(1e12).div(sharesTotal)) (AutoFarmV2.sol#1558-1560)
```

Solidity integer division might truncate. It may happen that performing multiplication before division might reduce precision.

[Recommendation] All those operation have to be carefully analysed for any precision reduction based on the code business logic. **Not a significant risk.**

Possible re-entrancy:

```
Reentrancy in AutoFarmV2.deposit(uint256,uint256) (AutoFarmV2.sol#1565-1593):
        External calls:
        - updatePool(_pid) (AutoFarmV2.sol#1566)
                - AUTOToken(AUTOv2).mint(owner(),AUTOReward.mul(ownerAUTOReward).div(1000)) (AutoFarmV2.sol#1552-1555)
                - AUTOToken(AUTOv2).mint(address(this),AUTOReward) (AutoFarmV2.sol#1556)
        - safeAUTOTransfer(msg.sender,pending) (AutoFarmV2.sol#1576)
                - IERC20(AUTOv2).transfer(_to,AUTOBal) (AutoFarmV2.sol#1669)
                - IERC20(AUTOv2).transfer(_to,_AUTOAmt) (AutoFarmV2.sol#1671)
        - pool.want.safeTransferFrom(address(msg.sender),address(this),_wantAmt) (AutoFarmV2.sol#1580-1584)
        - pool.want.safeIncreaseAllowance(pool.strat,_wantAmt) (AutoFarmV2.sol#1586)
        - sharesAdded = IStrategy(poolInfo[_pid].strat).deposit(msg.sender,_wantAmt) (AutoFarmV2.sol#1587-1588)
        State variables written after the call(s):
        - user.shares = user.shares.add(sharesAdded) (AutoFarmV2.sol#1589)
        - user.rewardDebt = user.shares.mul(pool.accAUTOPerShare).div(1e12) (AutoFarmV2.sol#1591)
Reentrancy in AutoFarmV2.emergencyWithdraw(uint256) (AutoFarmV2.sol#1648-1663):
        External calls:
        - IStrategy(poolInfo[_pid].strat).withdraw(msg.sender,amount) (AutoFarmV2.sol#1657)
        - pool.want.safeTransfer(address(msg.sender),amount) (AutoFarmV2.sol#1659)
        State variables written after the call(s):
        - user.shares = 0 (AutoFarmV2.sol#1661)
        - user.rewardDebt = 0 (AutoFarmV2.sol#1662)
```

```
Reentrancy in AutoFarmV2.withdraw(uint256,uint256) (AutoFarmV2.sol#1596-1641):
        External calls:
        - updatePool(_pid) (AutoFarmV2.sol#1597)
                - AUTOToken(AUTOv2).mint(owner(),AUTOReward.mul(ownerAUTOReward).div(1000)) (AutoFarmV2.sol#1552-1555)
                - AUTOToken(AUTOv2).mint(address(this),AUTOReward) (AutoFarmV2.sol#1556)
        - safeAUTOTransfer(msg.sender,pending) (AutoFarmV2.sol#1615)
                - IERC20(AUTOv2).transfer(_to,AUTOBal) (AutoFarmV2.sol#1669)
                - IERC20(AUTOv2).transfer(_to,_AUTOAmt) (AutoFarmV2.sol#1671)
        - sharesRemoved = IStrategy(poolInfo[_pid].strat).withdraw(msg.sender,_wantAmt) (AutoFarmV2.sol#1624-1625)
        State variables written after the call(s):
        - user.shares = 0 (AutoFarmV2.sol#1628)
        - user.shares = user.shares.sub(sharesRemoved) (AutoFarmV2.sol#1630)
```

The recommendation is to use the *check-effects-interactions* pattern or use nonReentrant modifier (https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard-nonReentrant--)

Low/Informational issues:

Different versions of solidity used:

```
Different versions of Solidity is used in :
        - Version used: ['0.6.12', '^0.6.0', '^0.6.12']
        - 0.6.12 (StratVLEV.sol#7)
        - ^0.6.0 (StratVLEV.sol#1309)
        - ^0.6.0 (StratVLEV.sol#1333)
        - ^0.6.12 (StratVLEV.sol#1385)
```

[Recommendation] Stick to one version of Solidity.

Dynamic Tests

We have run fuzzing/property-based testing of Solidity smarts contracts. It was using sophisticated grammar-based fuzzing campaigns based on a contract ABI to falsify user-defined predicates or Solidity assertions.

There were also dynamic tests run on EVM byte code to detect common vulnerabilities including integer underflows, owner-overwrite-to-Ether-withdrawal, and others.

```
==== Multiple Calls in a Single Transaction ====
SWC ID: 113
Severity: Low
Contract: AutoFarmV2
Function name: emergencyWithdraw(uint256)
PC address: 5055
Estimated Gas Usage: 27812 - 211453
Multiple calls are executed in the same transaction.
This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be c
aused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be tru
sted (i.e. they're part of your own codebase).
--------------------
In file: /tmp/AutoFarmV2.sol:1654

IStrategy(poolInfo[_pid].strat).sharesTotal()
```

```
==== External Call To User-Supplied Address ====
SWC ID: 107
Severity: Low
Contract: AutoFarmV2
Function name: emergencyWithdraw(uint256)
PC address: 5336
Estimated Gas Usage: 27812 - 211453
A call to a user-supplied address is executed.
An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within
this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or r
eentrancy guards are in place.
--------------------
In file: /tmp/AutoFarmV2.sol:1657

IStrategy(poolInfo[_pid].strat).withdraw(msg.sender, amount)
```

[Manual Check] IStrategy contract is fully controlled by the codebase, providing proper tests are done it is **NOT an issue**.

```
==== State access after external call ====
SWC ID: 107
Severity: Medium
Contract: AutoFarmV2
Function name: emergencyWithdraw(uint256)
PC address: 5560
Estimated Gas Usage: 27812 - 211453
Write to persistent state following external call
The contract account state is accessed after an external call to a user defined address. To prevent reentrancy issues, consider accessing the state only before the call, esp
ecially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.
--------------------
In file: /tmp/AutoFarmV2.sol:1661

user.shares = 0
```

```
==== State access after external call ====
SWC ID: 107
Severity: Medium
Contract: AutoFarmV2
Function name: emergencyWithdraw(uint256)
PC address: 5570
Estimated Gas Usage: 27812 - 211453
Write to persistent state following external call
The contract account state is accessed after an external call to a user defined address. To prevent reentrancy issues, consider accessing the state only before the call, esp
ecially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.
--------------------
In file: /tmp/AutoFarmV2.sol:1662

user.rewardDebt = 0
```

[Recommendation] It may cause possible re-entrancy attack, use nonReentrant modifier (https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard-nonReentrant--) **[FIXED]**

Manual Checks

StratX params:

4. buyBackRate
150 *uint256*

5. buyBackRateMax
10000 *uint256*

6. buyBackRateUL
800 *uint256*

7. controllerFee
20 *uint256*

8. controllerFeeMax
10000 *uint256*

9. controllerFeeUL
300 *uint256*

StratVLEV params:

10. buyBackRate
150 *uint256*

11. buyBackRateMax
10000 *uint256*

12. buyBackRateUL
800 *uint256*

13. controllerFee
60 *uint256*

14. controllerFeeMax
10000 *uint256*

15. controllerFeeUL
300 *uint256*

Automatic Tests

The project lacks any automatic testing and tests scripts. We did not run any functional tests provided by the team, due to lack of such scripts provided. Hence the full business logic functionality was not tested.

[**Recommendation**]: Create comprehensive test cases and implement them as scripts or mocha tests using the hardhat infrastructure.

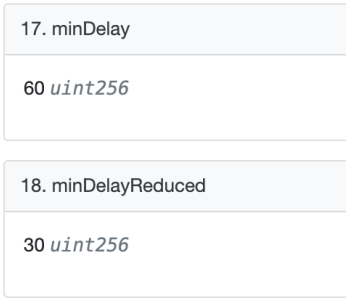
[**Disclaimer**] There were no tests conducted testing full system functionality due to lack of proper test cases and/or test scripts.

Deployment & Contract Ownership

The contracts are currently deployed on BSC Mainnet:

- **AUTOv2 - https://bscscan.com/address/0x12300ea3d3444e9106c65913de24c58462abfe24**
- **AutofarmV2 - https://bscscan.com/address/0xf6c361f82c744881fb58563e6c0563601007cde2**
- **TimelockController - https://bscscan.com/address/0x28579eca0a326e53340edd93e18698d379876a04**
- **StratX - https://bscscan.com/address/0x5f93026bcf9120ccf719882fcde147a52da8ca5c**
- **StratVLEV - https://bscscan.com/address/0x89e1509810378f73b4bc0c1decb87fcca6fbe9a1**

TimelockController controls multiple functions with significant importance to the overall system. Currently the delay in the time lock is set to 60/30 seconds.



Recommendations:

- Deploy a governance on top of the ownership of all contracts.
    - It could be a proper multi-sig controlled setup or full Compound like governance system - https://medium.com/compound-finance/compound-governance-5531f524cf68
    - Increase the TimelockController delay to significant value accepted by the project community.

Disclaimer

The information appearing in this report is for general purposes only and is not intended to provide any legal security guarantees to any individual or entity. As one review is not enough to provide 100% security against any attacks or bugs, it is advisable to conduct more reviews or/and audits.

The report does not provide personalised investment advice or recommendations, especially does not provide advice to conclude any transactions and it does not provide investment, financial, legal or tax advice.

We are not responsible or liable for any loss which results from the report.

The report should not be considered as an investment advice.