HAECHI AUDIT

yAxis v3

Smart Contract Security Analysis

Published on : Oct 13, 2021

Version v1.0

HAECHI AUDIT

Smart Contract Audit Certificate

yAxis v3

Security Report Published by HAECHI AUDIT
v1.0 Oct 13, 2021

Auditor : Jasper Lee

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	-	-	-	-	-
Major	-	-	-	-	-
Minor	1	-	-	-	-
Tips	1	-	-	-	-

TABLE OF CONTENTS

2 Issues (0 Critical, 0 Major, 1 Minor, 1 Tips) Found

ABOUT US

HAECHI AUDIT believes in the power of cryptocurrency and the next paradigm it will bring. We have the vision to empower the next generation of finance. By providing security and trust in the blockchain industry, we dream of a world where everyone has easy access to blockchain technology.

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 a team of experts with years of experience in the blockchain field and have been trusted by 300+ project groups. Our notable partners include Sushiswap,1inch, Klaytn, Badger, etc.

HAECHI AUDIT is 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.

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

INTRODUCTION

This report was prepared to audit the security of yAxis v3 smart contract created by yAxis project team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by yAxis project team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the smart contract.

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 can improve the code usability or efficiency when corrected.

HAECHI AUDIT recommends the yAxis project team improve all issues discovered.

The following issue explanation 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.

SUMMARY

The codes used in this Audit can be found at GitHub (https://github.com/yaxis-project/metavault/tree/main/contracts/v3). The last commit of the code used for this Audit is 0c2298cdc27a6ae5766b23777c22aa7f5f4363c1.

Issues HAECHI AUDIT found 0 critical issues, 0 major issues, and 1 minor issues. There are 1 Tips issues explained that would improve the code's usability or efficiency upon modification.

Severity	Issue	Status
MINOR	NativeStrategyCurve3Crv does not update Crv allowance setting a new router.	(Found - v1.0)
TIPS	*MetaVault#setMin()* should fail when min is larger than MAX	(Found - v1.0)

OVERVIEW

Contracts subject to audit

- IStableSwap3Pool
- IController
- IStablesOracle
- IConverter
- ICurve3Pool
- Manager
- Vault
- ILegacyVault
- IMetaVault
- ExtendedIERC20
- VaultToken
- ISwap
- AlwaysAccess
- MetaVault
- LegacyConverter
- Harvester
- MinterWrapper
- IVaultManager
- BaseStrategy
- IStrategy
- MetaVaultNonConverter
- NativeStrategyCurve3Crv
- VaultHelper
- ILegacyController
- StablesConverter
- IManager
- IController
- IHarvester
- Controller

yAxis v3 smart contract has the following privileges.

- Owner
- Governance
- Strategist

Each privilege can access the following functions.

Role	Functions
Owner	*MinterWrapper#setMinter()* *MinterWrapper#setRate()* *Ownable#renounceOwnership()* *Ownable#transferOwnership()*
Governance	*MetaVaultNonConverter#setStrategy()* *MetaVaultNonConverter#approveForSpender()* *MetaVaultNonConverter#governanceRecoverUnsupported()* *Manager#setAllowedController()* *Manager#setAllowedConverter()* *Manager#setAllowedStrategy()* *Manager#setAllowedVault()* *Manager#setGovernance()* *Manager#setHarvester()* *Manager#setInsuranceFee()* *Manager#setInsurancePool()* *Manager#setInsurancePoolFee()* *Manager#setStakingPool()* *Manager#setStakingPoolShareFee()* *Manager#setStrategist()* *Manager#setTreasury()* *Manager#setTreasuryFee()* *Manager#setWithdrawalProtectionFee()* *NativeStrategyCurve3Crv#approveForSpender()* *NativeStrategyCurve3Crv#setRouter()* *MetaVault#setMin()* *MetaVault#setGovernance()* *MetaVault#setController()* *MetaVault#setConverter()* *MetaVault#setVaultManager()* *MetaVault#setEarnLowerLimit()* *MetaVault#setTotalDepositCap()* *MetaVault#setAcceptContractDepositor()* *MetaVault#setYaxPerBlock()* *MetaVault#setEpochEndBlock()* *MetaVault#setEpochRewardMultipler()* *MetaVault#setTreasuryWallet()* *MetaVault#claimInsurance()*

	❖ *MetaVault#earnExtra()*
Strategist	❖ *Harvester#setHarvester()* ❖ *Harvester#setSlippage()* ❖ *Manager#setHalted()* ❖ *Manager#addVault()* ❖ *Manager#recoverToken()* ❖ *Manager#removeVault()* ❖ *Manager#setController()* ❖ *Vault#seetGauge()* ❖ *Vault#setMin()* ❖ *Vault#setTotalDepositCap()* ❖ *Controller#addStrategy()* ❖ *Controller#inCaseStrategyGetStuck()* ❖ *Controller#inCaseTokenGetStuck()* ❖ *Controller#removeStrategy()* ❖ *Controller#reorderStrategies()* ❖ *Controller#setCap()* ❖ *Controller#setConverter()* ❖ *Controller#setInvestEnabled()* ❖ *Controller#setMaxStrategies()* ❖ *Controller#skim()* ❖ *Controller#withdrawAll()* ❖ *LegacyController#setVault()* ❖ *LegacyController#setConverter()* ❖ *LegacyController#setInvestEnabled()* ❖ *LegacyController#recoverUnsupportedToken()* ❖ *StableConverter#approveForSpender()* ❖ *StableConverter#recoverUnsupported()*

FINDINGS

MINOR

NativeStrategyCurve3Crv does not update Crv allowance setting a new router.

(Found - v.1.0)

```
function setRouter(
    address _router
)
    external
{
    require(msg.sender == manager.governance(), "!governance");
    router = ISwap(_router);
    IERC20(weth).safeApprove(address(_router), 0);
    IERC20(weth).safeApprove(address(_router), type(uint256).max);
}
```

[https://github.com/yaxis-project/metavault/blob/0c2298cdc27a6ae5766b23777c22aa7f5f4363c1/contracts/v3/strategies/BaseStrategy.sol#L96-L105]

```
constructor (
    string memory _name,
    address _want,
    address _crv,
    address _weth,
    address _dai,
    address _usdc,
    address _usdt,
    Gauge _gauge,
    Mintr _crvMintr,
    IStableSwap3Pool _stableSwap3Pool,
    address _controller,
    address _manager,
    address _router
)
    public
    BaseStrategy(_name, _controller, _manager, _want, _weth, _router)
{
    crv = _crv;
    dai = _dai;
    usdc = _usdc;
    usdt = _usdt;
    stableSwap3Pool = _stableSwap3Pool;
    gauge = _gauge;
    crvMintr = _crvMintr;
    IERC20(_want).safeApprove(address(_gauge), type(uint256).max);
    IERC20(_crv).safeApprove(address(_router), type(uint256).max);
```

```
        IERC20(_dai).safeApprove(address(_stableSwap3Pool), type(uint256).max);
        IERC20(_usdc).safeApprove(address(_stableSwap3Pool), type(uint256).max);
        IERC20(_usdt).safeApprove(address(_stableSwap3Pool), type(uint256).max);
        IERC20(_want).safeApprove(address(_stableSwap3Pool), type(uint256).max);
    }
```

[https://github.com/yaxis-project/metavault/blob/0c2298cdc27a6ae5766b23777c22aa7f5f4363c1/contracts/v3/strategies/NativeStrategyCurve3Crv.sol#L25-L56]

Issue

NativeStrategyCurve3Crv calls *BaseStrategy#setRouter()* when setting swap router. Once called, *BaseStrategy#setRouter()* make *NativeStrategyCurve3Crv* approve WETH on router. Crv should also be approved since *NativeStrategyCurve3Crv* is using Crv > WETH > [DAI/USDC/USDT] > 3Crv route.

Recommendation

Please implement *NativeStrategyCurve3Cr#setRouter()*, which overrides the *BaseStrategy#setRouter()* function and approves Crv on router.

💡 TIPS

MetaVault#setMin() should fail when min is larger than MAX

(Found - v.1.0)

```
function setMin(uint _min) external {
    require(msg.sender == governance, "!governance");
    min = _min;
}
```

[https://github.com/yaxis-project/metavault/blob/0c2298cdc27a6ae5766b23777c22aa7f5f4363c1/contracts/legacy/MetaVault.sol#L138-L141]

Issue

MetaVault#setMin() is a function that set value for min. It's recommended that contract interaction reverts when *_min* is larger than *MAX*.

Recommendation

Please add guard code as below :

```
function setMin(uint _min) external {
    require(msg.sender == governance, "!governance");
    require(_min <= MAX, "!_min");
    min = _min;
}
```

[https://github.com/yaxis-project/metavault/blob/0c2298cdc27a6ae5766b23777c22aa7f5f4363c1/contracts/legacy/MetaVault.sol#L138-L141]

DISCLAIMER

This report does not guarantee investment advice, the 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. 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.

```
Controller
 #constructor()
   ✓ Should initialize variables
 #addStrategy()
   ✓ Should fail if halted
   ✓ Should fail if caller is not strategist
   ✓ Should fail if strategy is not allowed
   ✓ Should fail if vault is not allowed
   ✓ Should fail if converter is not set
   ✓ Should fail if already added
   ✓ Should add strategy
 #inCaseStrategyGetStuck()
   ✓ Should fail if caller is not strategist
   ✓ Should send tokens to treasury
 #inCaseTokensGetStuck()
   ✓ Should fail if caller is not strategist
   ✓ Should send tokens to treasury
 #removeStrategy()
   ✓ Should fail if caller is not strategist
   ✓ Should fail if halted
   ✓ Should remove strategy
   ✓ Should update vault fund
   ✓ Should emit StrategyRemoved event
 #reorderStrategies()
   ✓ Should fail if caller is not strategist
   ✓ Should fail if halted
   ✓ Should fail if vault is not allowed
   ✓ Should fail if strategy1 is not for vault
   ✓ Should fail if strategy2 is not for vault
   ✓ Should change index of strategies
   ✓ Should emit StrategiesReordered event
 #setCap()
   ✓ Should fail if caller is not strategist
   ✓ Should fail if halted
```

```
  ✓ Should fail if strategy is not allowed
  ✓ Should update vault cap
  ✓ Should update want balance (convert == zero)
  ✓ Should update convert balance (convert != zero)
#setConverter()
  ✓ Should fail if caller is not strategist
  ✓ Should fail if halted
  ✓ Should fail if converter is not allowed
  ✓ Should update converter
#setInvestEnabled()
  ✓ Should fail if caller is not strategist
  ✓ Should fail if halted
  ✓ Should update globalInvestEnabled
#setMaxStrategies()
  ✓ Should fail if caller is not strategist
  ✓ Should fail if halted
  ✓ Should update maxStrategies
#skim()
  ✓ Should fail if caller is not strategist
  ✓ Should fail if strategy is not allowed
  ✓ Should update want balance
#withdrawAll()
  ✓ Should fail if caller is not strategist
  ✓ Should fail if strategy is not allowed
  ✓ Should update want balance (convert == zero)
  ✓ Should update convert balance (convert != zero)
#harvestStrategy()
  ✓ Should fail if caller is not harvester
  ✓ Should fail if halted
  ✓ Should fail if strategy is not allowed
  ✓ Should update vault balance
  ✓ Should emit harvest event
#earn()
  ✓ Should fail if caller is not vault
  ✓ Should fail if halted
  ✓ Should fail if vault is not allowed
  ✓ Should update balance
#withdraw()
  ✓ Should fail if caller is not vault
  ✓ Should fail if vault is not allowed
  ✓ Should update want balance
  ✓ Should update token balance (want != token)
```

```
LegacyController
  #constructor()
    ✓ Should initialize variables
  #setVault()
    ✓ Should fail if caller is not strategist
    ✓ Should update vault address
  #setConverter()
    ✓ Should fail if caller is not strategist
    ✓ Should update converter address
  #setInvestEnabled()
    ✓ Should fail if caller is not strategist
    ✓ Should update investEnabled
  #recoverUnsupportedToken()
    ✓ Should fail if caller is not strategist
    ✓ Should update token balance
  #balanceOf()
    ✓ Should show balance of token
  #withdrawFee()
    ✓ Should show withdrawFee
  #withdraw()
    ✓ Should fail if vault is zero address
    ✓ Should fail if caller is not metavault
    ✓ Should update balance
  #legacyDeposit()
    ✓ Should fail if converter is zero address
    ✓ Should fail if caller is not harvester

MetaVaultNonConverter
  #constructor()
    ✓ Should initialize variables
  #setStrategy()
    ✓ Should fail if caller is not governance
  #approveForSpender()
    ✓ Should fail if caller is not governance
    ✓ Should update allowance
  #token()
    ✓ Should return 3crv address
  #convert()
    ✓ Should be failed
  #convert_rate()
    ✓ Should be failed
  #convert_stables()
    ✓ Should be failed
```

```
  #calc_token_amount()
    ✓ Should be failed
  #calc_token_amount_withdraw()
    ✓ Should be failed
  #governanceRecoverUnsupported()
    ✓ Should fail if caller is not governance
    ✓ Should update token balance

StablesConverter
  #constructor()
    ✓ Should initialize variables
    ✓ Should approve tokens (DAI/USDC/USDT/3CRV)
  #approveForSpender()
    ✓ Should fail if caller is not strategist
    ✓ Should update allowance
  #recoverUnsupported()
    ✓ Should fail if caller is not strategist
    ✓ Should update token balance
  #convert()
    ✓ Should fail if caller is not allowed entity

Harvester
  #constructor()
    ✓ Should set manager
    ✓ Should set controller
    ✓ Should set legacyController
  #addStrategy()
    ✓ Should fail if caller is not controller
    ✓ Should fail if caller is not allowed
    ✓ Should add strategy
    ✓ Should emit StrategyAdded event
  #removeStrategy()
    ✓ Should fail if caller is not controller
    ✓ Should fail if caller is not allowed
    ✓ Should remove strategy
    ✓ Should emit StrategyRemoved event
  #setHarvester()
    ✓ Should fail if caller is not strategist
    ✓ Should set harvester
    ✓ Should emit HarvesterSet event
  #setSlippage()
    ✓ Should fail if caller is not strategist
    ✓ Should fail if slippage is larger than 10000
```

```
    ✓ Should set slippage
  #earn()
    ✓ Should fail if caller is not harvester
    ✓ Should update vault balance
  #harvest()
    ✓ Should fail if caller is not harvester
    ✓ Should harvest properly
    ✓ Should emit harvest event
  #legacyEarn()
    ✓ Should fail if caller is not harvester
    ✓ Should update vault balance

Manager
  #constructor()
    ✓ Should fail if yaxis address is ZERO_ADDRESS
    ✓ Should set yaxis address
    ✓ Should set governance address
    ✓ Should set strategist address
    ✓ Should set harvester address
    ✓ Should set treasury address
    ✓ Should set stakingPoolShareFee
    ✓ Should set treasuryFee
    ✓ Should set withdrawalProtectionFee
  #setHalted()
    ✓ Should fail when already halted
    ✓ Should fail if caller is not strategist
    ✓ Should update halted true
  #setAllowedController()
    ✓ Should fail when halted
    ✓ Should fail if caller is not governance
    ✓ Should fail when manager address inconsistency exists between controller and manager
    ✓ Should add allowedControllers
    ✓ Should emit AllowedController event
  #setAllowedConverter()
    ✓ Should fail when halted
    ✓ Should fail if caller is not governance (40ms)
    ✓ Should fail when manager address inconsistency exists between converter and manager
    ✓ Should add allowedConverters
    ✓ Should emit AllowedConverter event
  #setAllowedStrategy()
    ✓ Should fail when halted
    ✓ Should fail if caller is not governance
    ✓ Should fail when manager address inconsistency exists between strategy and manager
```

```
  ✓ Should add allowedStrategies
  ✓ Should emit AllowedStrategy event
#setAllowedVault()
  ✓ Should fail when halted
  ✓ Should fail if caller is not governance
  ✓ Should fail when manager address inconsistency exists between vault and manager
  ✓ Should add allowedVaults
  ✓ Should emit AllowedVault event
#setGovernance()
  ✓ Should fail when halted
  ✓ Should fail if caller is not governance
  ✓ Should update governance
  ✓ Should emit SetGovernance event
#setHarvester()
  ✓ Should fail when halted
  ✓ Should fail if caller is not governance
  ✓ Should fail when manager address inconsistency exists between harvester and manager
  ✓ Should update harvester
#setInsuranceFee()
  ✓ Should fail when halted
  ✓ Should fail if caller is not governance
  ✓ Should fail if set fee over 1%
  ✓ Should update insuranceFee
#setInsurancePool()
  ✓ Should fail when halted
  ✓ Should fail if caller is not governance
  ✓ Should update insurancePool
#setInsurancePoolFee()
  ✓ Should fail when halted
  ✓ Should fail if caller is not governance
  ✓ Should fail if set fee over 20%
  ✓ Should update insurancePoolFee
#setStakingPool()
  ✓ Should fail when halted
  ✓ Should fail if caller is not governance
  ✓ Should update stakingPool
#setStakingPoolShareFee()
  ✓ Should fail when halted
  ✓ Should fail if caller is not governance
  ✓ Should fail if set fee over 50%
  ✓ Should update stakingPoolShareFee
#setStrategist()
  ✓ Should fail when halted
```

✓ Should fail if caller is not governance
✓ Should fail if _strategist is zero address
✓ Should update pendingStrategist
✓ Should emit SetPendingStrategist event
#setTreasury()
✓ Should fail when halted
✓ Should fail if caller is not governance
✓ Should fail if _strategist is zero address
✓ Should update treasury
#setTreasuryFee()
✓ Should fail when halted
✓ Should fail if caller is not governance
✓ Should fail if set fee over 20%
✓ Should update treasuryFee
#setWithdrawalProtectionFee()
✓ Should fail when halted
✓ Should fail if caller is not governance
✓ Should fail if set fee over 1%
✓ Should update withdrawalProtectionFee
#acceptStrategist()
✓ Should fail when halted
✓ Should fail if caller is not pendingStrategist
✓ Should fail if pendingStrategist is under timelock
✓ Should update strategist
✓ Should emit SetStrategist event
#addVault()
✓ Should fail when halted
✓ Should fail if caller is not strategist
✓ Should fail if vault is not allowed
✓ Should fail if vault is already added
✓ Should emit VaultAdded event
#recoverToken()
✓ Should fail when halted
✓ Should fail if caller is not strategist
✓ Should update token balance
#removeVault()
✓ Should fail when halted
✓ Should fail if caller is not strategist
✓ Should fail if vault is not added
✓ Should emit VaultRemoved event
#setController()
✓ Should fail when halted
✓ Should fail if caller is not strategist

```
    ✓ Should fail if vault is not allowed
    ✓ Should fail if controller is not allowed
    ✓ Should update controllers
    ✓ Should emit SetController event
  #getToken()
    ✓ Should return vault token
  #getHarvestFeeInfo()
    ✓ Should return fee info

MetaVault
  #constructor()
    ✓ Should set inputTokens
    ✓ Should set token3CRV
    ✓ Should set tokenYAX
    ✓ Should set yaxPerBlock
    ✓ Should set lastRewardBlock
    ✓ Should set epochEndBlocks
    ✓ Should set governance
  #setMin()
    ✓ Should fail if caller is not governance
    ✓ Should set min
    1) [INFO] Should fail if min is larger than MAX
  #setGovernance()
    ✓ Should fail if caller is not governance
    ✓ Should set governance
  #setController()
    ✓ Should fail if caller is not governance
    ✓ Should set controller
  #setConverter()
    ✓ Should fail if caller is not governance
    ✓ Should fail if converter token is not 3CRV
    ✓ Should set converter
  #setVaultManager()
    ✓ Should fail if caller is not governance
    ✓ Should set vaultManager
  #setEarnLowerlimit()
    ✓ Should fail if caller is not governance
    ✓ Should set earnLowerlimit
  #setTotalDepositCap()
    ✓ Should fail if caller is not governance
    ✓ Should set totalDepositCap
  #setAcceptContractDepositor()
    ✓ Should fail if caller is not governance
```

```
  ✓ Should set acceptContractDepositor
#setYaxPerBlock()
  ✓ Should fail if caller is not governance
  ✓ Should set yaxPerBlock
#setEpochEndBlock()
  ✓ Should fail if caller is not governance
  ✓ Should fail if index is larger than 4
  ✓ Should fail if _epochEndBlock is before block.number
  ✓ Should fail if epochEndBlocks[_index] is before block.number
  ✓ Should set epochEndBlocks
#setEpochRewardMultipler()
  ✓ Should fail if caller is not governance
  ✓ Should fail if index is larger than 5
  ✓ Should fail if index is 0
  ✓ Should fail if epochEndBlocks[_index - 1] is before block.number
  ✓ Should set setEpochRewardMultipler
#setTreasuryWallet()
  ✓ Should fail if caller is not governance
  ✓ Should set treasuryWallet
#claimInsurance()
  ✓ Should fail if caller is not governance
  ✓ Should transfer 3crv token
  ✓ Should set insurance to zero when called by controller
#earn()
  ✓ Should do nothing if controller is zero address
  ✓ Should do nothing if invest is not enabled
  ✓ Should update vault balance
#deposit()
  ✓ Should fail if amount is zero
  ✓ Should fail if caller is contract when its not allowed
  ✓ Should deposit properly
#depositAll()
  ✓ Should deposit 3 stables properly
#stakeShares()
  ✓ Should update share balance
  ✓ Should emit Deposit event
#withdrawAll()
  ✓ Should withdraw properly
#harvest()
  ✓ Should fail if caller is not controller
  ✓ Should fail if reserve is 3CRV
  ✓ Should update token balance
#unstake()
```

```
    ✓ Should fail if requested amount is larger than balance
    ✓ Should update balance
    ✓ Should emit Withdraw event
  #withdraw()
    ✓ Should fail if requested amount is larger than balance + staked
    ✓ Should update balance
  #earnExtra()
    ✓ Should fail if caller is not governance
    ✓ Should fail if _token is 3crv
    ✓ Should fail if _token is mlvt
    ✓ Should update balance

MinterWrapper
  #constructor()
    ✓ Should set token
    ✓ Should set rate
    ✓ Should set owner
  #setMinter()
    ✓ Should fail if caller is not owner
    ✓ Should fail if minter already set
    ✓ Should fail if _minter is zero address
    ✓ Should set minter
  #setRate()
    ✓ Should fail if caller is not owner
    ✓ Should set rate
  #mint()
    ✓ Should fail if caller is not minter
    ✓ Should update balance
  #future_epoch_time_write()
    ✓ Should emit Write event
  #available_supply()
    ✓ Should return token balance

NativeStrategyCurve3Crv (BaseStrategy)
  #constructor()
    ✓ Should initialize variables
    ✓ Should approve each tokens
  #approveForSpender()
    ✓ Should fail if caller is not governance
    ✓ Should update allowance
  #setRouter()
    ✓ Should fail if caller is not governance
    ✓ Should update weth allowance
```

2) [MINOR] NativeStrategyCurve3Crv does not update crv allowance

```
#deposit()
  ✓ Should fail if caller is not controller
  ✓ Should deposit want to gauge
#harvest()
  ✓ Should fail if caller is not controller
  ✓ Should do nothing when estimated weth is 0
#skim()
  ✓ Should fail if caller is not controller
  ✓ Should update want balance
#withdraw()
  ✓ Should fail if caller is not controller
  ✓ Should fail if withdrawal asset is want
  ✓ Should update asset balance
#withdraw2()
  ✓ Should fail if caller is not controller
  ✓ Should update asset balance
#withdrawAll()
  ✓ Should fail if caller is not controller
  ✓ Should update asset balance

AlwaysAccess
#constructor()
  ✓ Should set authorized
#setAuthorized()
  ✓ Should set authorized
#allowedVaults()
  ✓ Should return authorized
#allowedControllers()
  ✓ Should return authorized
#allowedStrategies()
  ✓ Should return authorized

Vault
#constructor()
  ✓ Should set manager
  ✓ Should set token
  ✓ Should set vaultToken
  ✓ Should set min
  ✓ Should set totalDepositCap
#setGauge()
  ✓ Should fail if caller is not strategist
  ✓ Should fail if halted
```

```
    ✓ Should set gauge
  #setMin()
    ✓ Should fail if caller is not strategist
    ✓ Should fail if halted
    ✓ Should fail if min is larger than MAX
    ✓ Should set min
  #setTotalDepositCap()
    ✓ Should fail if caller is not strategist
    ✓ Should fail if halted
    ✓ Should set totalDepositCap
  #earn()
    ✓ Should fail if caller is not harvester
    ✓ Should fail if halted
    ✓ Should fail if strategy is not allowed
    ✓ Should do nothing if invest is not enabled
    ✓ Should send token to strategy
    ✓ Should emit Earn event
  #deposit()
    ✓ Should fail if halted
    ✓ Should fail if amount is 0
    ✓ Should fail if calculated totalSupply is larger than totalDepositCap
    ✓ Should mint vault token
    ✓ Should emit Deposit event
  #withdraw()
    ✓ Should burn vault token
    ✓ Should update token balance
    ✓ Should emit Withdraw event
  #withdrawAll()
    ✓ Should withdraw all

VaultHelper
  #depositVault()
    ✓ Should fail if amount is 0
    ✓ Should update token balance (gauge == 0)
    ✓ Should update token balance
  #withdrawVault()
    ✓ Should update token balance (gauge == 0)
    ✓ Should update token balance

VaultToken
  #constructor()
    ✓ Should set manager
    ✓ Should set name
```

```
  ✓ Should set symbol
  ✓ Should set totalSupply as 0
#mint()
  ✓ Should fail if caller is not vault
  ✓ Should increase totalSupply
  ✓ Should increase recipient balance
#burn()
  ✓ Should fail if caller is not vault
  ✓ Should decrease totalSupply
  ✓ Should decrease recipient balance
#transfer()
  ✓ Should fail if recipient is own vaultToken address
  ✓ Should update sender/recipient balance
#approve()
  ✓ Should fail if recipient is own vaultToken address
  ✓ Should update recipient allowance for owner
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/legacy					
IController.sol	100	100	100	100	
IConverter.sol	100	100	100	100	
IMetaVault.sol	100	100	100	100	
IVaultManager.sol	100	100	100	100	
MetaVault.sol	85.23	88.81	93.90	85.23	... 639,640,641
MetaVaultNonConverter.sol	100	100	100	100	
contract/v3					
Harvester.sol	100	100	100	100	
Manager.sol	100	100	100	100	
MinterWrapper.sol	100	100	100	100	
Vault.sol	81.25	85.71	72.73	81.25	... 268,269,270

VaultHelper.sol	100	100	100	100	
VaultToken.sol	100	100	100	100	
contract/v3/controllers					
Controller.sol	100	100	100	100	
LegacyController.sol	100	100	100	100	
contract/v3/converters					
StableConverter.sol	100	100	100	100	
contract/v3/interfaces					
ExtendedIERC20.sol	100	100	100	100	
IController.sol	100	100	100	100	
IConverter.sol	100	100	100	100	
ICurve3Pool.sol	100	100	100	100	
IHarvester.sol	100	100	100	100	
ILegacyController.sol	100	100	100	100	
ILegacyVault.sol	100	100	100	100	
IStablesOracle.sol	100	100	100	100	
IStrategy.sol	100	100	100	100	
ISwap.sol	100	100	100	100	
contract/v3/strategies					
BaseStrategy.sol	100	100	100	100	
NativeStrategyCurve3Crv.sol	100	100	100	100	
contract/v3/test					
AlwaysAccess.sol	100	100	100	100	

[Table 1] Test Case Coverage

End of Document