HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Rumi Finance - Defi Blue
Date: March 21, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Blue Swan Labs
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token; Hedge Fund; Delta Neutral Yield Farming
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.rumi.finance/
Changelog	20.01.2023 - Initial Review 02.03.2023 - Second Review 21.03.2023 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Rumi Finance - Defi Blue (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository	https://bitbucket.org/blueswanio/bs-non-custodial-optimism-vault-hardhat
Commit	e8583d84acd6c722c6427701f7b7574df0722099
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/AHStrategy.sol SHA3: 033b8982d05d487069a4866ba51d7c2ef0e6d0296076997d8757140229d90442 File: ./contracts/deps/Controller.sol SHA3: 770ab4803a35517549bf70d78d07290b19bc4c9b6da3b09b8c4b4cf03ecf19f0 File: ./contracts/deps/SettV3.sol SHA3: df0c5f4cdd6b639f2291dfa0993952d73ccae2a025655e6e72e96372e0a94804 File: ./contracts/libraries/BitMath.sol SHA3: ebbb6cbff6857d61fd7f507e014276d227a648fac37324e63217da0571d8f830 File: ./contracts/libraries/FixedPoint128.sol SHA3: 113cc07aef8fec2ac943540438956848a00d79af066b396be5d5b029e7f16249 File: ./contracts/libraries/FixedPoint96.sol SHA3: 1a7355695c5cf2b2e5450621a9e47d3cf6549a33067b1fd650f9e3909302b781 File: ./contracts/libraries/FullMath.sol SHA3: db0a08150647a30f2b3fdfb240a4aac73553e4ef0ee485ee62fc213d8c64aa91 File: ./contracts/libraries/HomoraMath.sol SHA3: 1e4bee5c4d4e4f2d8269024acf64f863325998d61351335ebbb4be62dcbd077b File: ./contracts/libraries/LiquidityAmounts.sol SHA3: 49b635275599c11bacb330a5f09167c4fee6b90a28cc7873276b3c700374b41d File: ./contracts/libraries/LowGasSafeMath.sol

	SHA3: 3d5f137e0f0322d2e700b2749dac94c22e01fa497ae70b663f61973580fb9584 File: ./contracts/libraries/SqrtPriceMath.sol SHA3: 506860400069f7402e383881ff323475988d25b72618a4840d2ec3a255ce9fd9 File: ./contracts/libraries/SwapMath.sol SHA3: e47e3399c0f9c683e6157d06a5172ed349247022184fdcd5fa5c09ef9a715b3c File: ./contracts/libraries/TickBitMap.sol SHA3: d67e26b8d700142907580ad15e4c3d4012a48166a435fd25ecf84d5ef2e5361f File: ./contracts/libraries/TickMath.sol SHA3: ee1a765f1e54c0cbb87f5b763c658794598109d0652c149157b508a8797ba8e7 File: ./contracts/UniswapV3SpellIntegrationOp.sol SHA3: a66f6422521d3fcad7d60aabddfd8506aa561a74f6aedbc9e4ed4988c86d6895 File: ./deps/BaseStrategy.sol SHA3: 1def2f2c8014c00f5a04f5d69e692985631df3ec2e4f50153a610804407cfbb5 File: ./deps/SettAccessControl.sol SHA3: bb6c255f6ce06a423e42616e965d56f8ed8d44ad32d9a276b32c83ed7b7042d6 File: ./deps/SettAccessControlDefended.sol SHA3: 0bfac94805828c11a3ac5399a9964abed8def8241cff48216b3c2d5669ad7d9f

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Second review scope

Repository	https://bitbucket.org/blueswanio/bs-non-custodial-optimism-vault-hardhat
Commit	f824453867b2c5bf30f6c334436cbca0f2d6974c
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/deps/Controller.sol SHA3: 23b6adec35219c6736fa830520a37e8f3fb37837aeee18bc87c603938de34d32 File: ./contracts/deps/SettV3.sol SHA3: eea8360622561a60ac04c87243aff57ba3a3ed86ebe0a7bfece4166c31ed22c2 File: ./contracts/libraries/LiquidityMath.sol SHA3: 191af298bc74859d3a989633b0b68f78a826f8294e1115156e6ab8cfbaf48f41 File: ./contracts/libraries/MaskBitPos.sol SHA3: f3cbc52ace9e3f728efc7be89360cc6b9e266996cad2e3dbcfb87199855720c0 File: ./contracts/StrategyAlphaHomora.sol SHA3: 16976c99c230a9abcc50269738dbbb4b3d89b4a5ffcd49c04c71a061a378e7f1 File: ./contracts/UniswapV3SpellIntegrationOp.sol SHA3: ee20594c50d4a29e50dbaf81f3289c2b2fcfef1f87db8ecd4956c59cb54e9fed File: ./interfaces/badger/IController.sol SHA3: ce80946d21a00035d74b832e48b886543c91e573dfc5ad9e022a7ac5643135a5 File: ./interfaces/badger/IStrategy.sol SHA3: 75d96601d0835d6e2895735005bc87feda519e61ed0f089bbfebffeff83588a5 File: ./interfaces/erc20/IWETH.sol SHA3: 4dbb1912577aeadcc80ca20588b4f8229b8791b541dbed54f64f5563a7042d43 File: ./interfaces/homora/IBank.sol SHA3: 0358570645a0abfb6f12ccd5649b385a41488486bc3014915b962ca49dabecbe File: ./interfaces/homora/IBankOP.sol SHA3: 27f40399047733215822c7b0838742425b6911be1e6aed8411343d835ca5f5b2 File: ./interfaces/homora/IBaseOracle.sol SHA3: 6c4b6168b430d8a8dd618589b35d7a9b5899d5df5f150b8be1c9ef5970a8ca82 File: ./interfaces/homora/IGovernable.sol SHA3: c8562b1e0906e8ec9f3fcf98aa2a89ea07a20542053e82e18f88a223666ecad8

	File: ./interfaces/homora/IOracle.sol SHA3: a25ac953e489462dafa362ce42cd1a9575540cd63daaf2ab6f8828f9213d27c5 File: ./interfaces/homora/IUniswapv3OptimalSwap.sol SHA3: bf1510337fd32c0a9322d5daeb2623d8185e67dfcaa64aa36783c13f0e95910c File: ./interfaces/homora/IUniswapV3Spell.sol SHA3: 8fd4db94a1afb46ef724e927f2fb4cd62f357429051a330eb7693515b22fec40 File: ./interfaces/homora/IWUniswapV3Position.sol SHA3: 924ba38ae4318984227090075ec877b4d5a0006cad6c23e8b4937e8090a44c5e File: ./interfaces/optimism/gasEstimator.sol SHA3: 2a1917131496861af8fa6c99b5a1aca083caa899b359476f887772a365aa6cc3 File: ./interfaces/uniswap/IUniswapRouterV2.sol SHA3: a03c5b9b99d76ff83c46c4179ad3bb617e99e5d8de825d439ffde366f139978b File: ./interfaces/UniswapV3/INonfungiblePositionManager.sol SHA3: 01fa235b2b8b556548229f30c519703774c4a14510416ff6826788f6121d02b6 File: ./interfaces/UniswapV3/Interfaces.sol SHA3: 689514444632c868337b15aeeb0cd28734ced6341f8c022c4a870af71b6d6191 File: ./interfaces/UniswapV3/ISwapRouter.sol SHA3: 973e7523ff1967cc124868d131d64d261ad2d18498c5147c6b5945950622878a

Third review scope

Repository	https://bitbucket.org/blueswanio/bs-non-custodial-optimism-vault-hardhat
Commit	0d0354db7c3ba3c85791e77412d39648542d8356
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/BaseIntegration.sol SHA3: c97fc52a255b9b1c6609fcbcc84a0a64c60b5674dcb171a9239b3fa63bd6066f File: ./contracts/deps/BaseStrategy.sol SHA3: ccd975c62ee3025ae15336a1f536e34b189d1e54ff577c85003ad55f4a3a0761 File: ./contracts/deps/Controller.sol SHA3: 2e32cf6366eb746e4ea18dd579863d9d74504d7c0c83120def30e9318fbc7b1d

	File: ./contracts/deps/SettAccessControl.sol SHA3: 7c9ecde86ae43e77d34a0a943bf940362357a11165f4ecd9524e040ddf28e15f File: ./contracts/deps/SettAccessControlDefended.sol SHA3: 42febcabfa5d98e52f9de896eb737c87b5f61b3094a542b1b1dd479568ffe230 File: ./contracts/deps/SettV3.sol SHA3: 24df38117a4f25bd1288ff7041e5d82f402f02013a16b1a1645fba82f7598cf4 File: ./contracts/libraries/LiquidityMath.sol SHA3: 191af298bc74859d3a989633b0b68f78a826f8294e1115156e6ab8cfbaf48f41 File: ./contracts/libraries/MaskBitPos.sol SHA3: f3cbc52ace9e3f728efc7be89360cc6b9e266996cad2e3dbcfb87199855720c0 File: ./contracts/StrategyAlphaHomora.sol SHA3: 0972159ddbf0c1c56cb9a11f6fa0d684d59d0fe5f20d818a433dd0e759466c0e File: ./contracts/UniswapV3SpellIntegrationOp.sol SHA3: 92928be6f39a0c299b7d77e186c2ab51babb70910b52e7a65db51ec3f3290400 File: ./interfaces/badger/IController.sol SHA3: ce80946d21a00035d74b832e48b886543c91e573dfc5ad9e022a7ac5643135a5 File: ./interfaces/badger/IStrategy.sol SHA3: 75d96601d0835d6e2895735005bc87feda519e61ed0f089bbfebffeff83588a5 File: ./interfaces/erc20/IWETH.sol SHA3: 4dbb1912577aeadcc80ca20588b4f8229b8791b541dbed54f64f5563a7042d43 File: ./interfaces/homora/IBank.sol SHA3: 0358570645a0abfb6f12ccd5649b385a41488486bc3014915b962ca49dabecbe File: ./interfaces/homora/IBankOP.sol SHA3: 27f40399047733215822c7b0838742425b6911be1e6aed8411343d835ca5f5b2 File: ./interfaces/homora/IBaseOracle.sol SHA3: 6c4b6168b430d8a8dd618589b35d7a9b5899d5df5f150b8be1c9ef5970a8ca82 File: ./interfaces/homora/IGovernable.sol SHA3: c8562b1e0906e8ec9f3fcf98aa2a89ea07a20542053e82e18f88a223666ecad8 File: ./interfaces/homora/IOracle.sol SHA3: a25ac953e489462dafa362ce42cd1a9575540cd63daaf2ab6f8828f9213d27c5 File: ./interfaces/homora/IUniswapv3OptimalSwap.sol SHA3: bf1510337fd32c0a9322d5daeb2623d8185e67dfcaa64aa36783c13f0e95910c File: ./interfaces/homora/IUniswapV3Spell.sol

	SHA3: 8fd4db94a1afb46ef724e927f2fb4cd62f357429051a330eb7693515b22fec40 File: ./interfaces/homora/IWUniswapV3Position.sol SHA3: 924ba38ae4318984227090075ec877b4d5a0006cad6c23e8b4937e8090a44c5e File: ./interfaces/uniswap/IUniswapRouterV2.sol SHA3: a03c5b9b99d76ff83c46c4179ad3bb617e99e5d8de825d439ffde366f139978b File: ./interfaces/UniswapV3/INonfungiblePositionManager.sol SHA3: 01fa235b2b8b556548229f30c519703774c4a14510416ff6826788f6121d02b6 File: ./interfaces/UniswapV3/Interfaces.sol SHA3: 689514444632c868337b15aeeb0cd28734ced6341f8c022c4a870af71b6d6191 File: ./interfaces/UniswapV3/ISwapRouter.sol SHA3: 973e7523ff1967cc124868d131d64d261ad2d18498c5147c6b5945950622878a

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to assets loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect the code quality

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **8** out of **10**.

- Functional requirements are provided.
- Technical description is partially provided.

Code quality

The total Code Quality score is **10** out of **10**.

- The code follows style guide and best practices.
- The development environment is configured.

Test coverage

Code coverage of the project is **94.81%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative test cases coverage is missing.

Security score

As a result of the audit, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.6**.

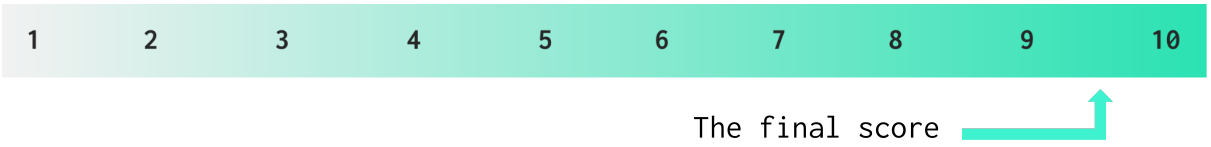

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
20 January 2023	16	17	8	2
02 March 2023	1	3	2	1
21 March 2023	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Passed
Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed

Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

Blue Swan is a crypto hedge fund with the following contracts:

- *AHstrategy* – The contract defines the specific strategy that interacts with Alpha Homora and UniswapV3 Pools to execute a Leveraged Delta Neutral Yield Farming Strategy.
- *BaseStrategy* – The contract holds the base class for all strategies that are held under the portfolio. It holds the functional structure for deposits, withdrawals and other permissioned calls to the Strategy contracts.
- *Controller* – The contract couples the Vault and Strategies together, allowing these to be interchangeable. A Controller can couple several Vaults and Strategies together, allowing for a N to the N combination. Normally one controller would be used per chain where we deploy the Vaults and Strategies. This will keep an inventory of them.
- *SettV3* - The contract that will interact with external users and other smart contracts that want to initiate an investment with us. It keeps accounting of investment through emitting shares (ERC20-LPToken) while tracking a value (NAV) of its managed portfolio of strategies.
- *UniswapV3SpellIntegrationOp* - The contract that interacts with UniswapV3 to open/add/remove/close users' positions.
- *SettAccessControl* - The contract that sets permissioned roles for the “Sett” ecosystem.
- *SettAccessControlDefended* - The contract to prevent unwanted contract access to “Sett” permissions.

Privileged roles

- The Governance address is responsible for setting strategist, controller, guardian, treasury, keeper, and governance, reward, vault addresses, as well as approve/revoke contract addresses and pause/unpause the contracts. The Governance can set thresholds, leverage, ticks, min harvest amount. Additionally, the Governance role has the ability to recover tokens that may become stuck in the contract.
- The Strategist address can withdraw the entire token balance of the strategy contract and can set strategy, vault addresses. Additionally, the Strategist role has the ability to recover tokens that may become stuck in the Controller.sol contract.
- The owner of the UniswapV3SpellIntegrationOp contract can open, close, increase, reduce, reinvest, harvest and clear positions.
- The Guardian address can pause the SettV3.sol contract.
- The keeper is an address that can call earn() to deposit tokens from a SettV3.sol into the associated active Strategy. Designed for use by a trusted bot in lieu of having this function publicly callable.

Risks

- The contract interacts with Alpha Homora V2 and UniswapV3 contracts, which are **out-of-scope** for this audit.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

■■■■ Critical

C01. Denial Of Service Vulnerability

The removeStrategy() function does not effectively eliminate the targeted address from the "wantsArray" list as intended. Instead, it alters the targeted address to the default value of 0x00. Subsequently, any external function calls made to remove elements of the wantsArray list will revert.

This can lead to denial of service vulnerability.

Path: ./contracts/deps/SettV3.sol : removeStrategy()

Recommendation: Copy the last index value to will be removed index, then pop the last element from the list.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

C02. Invalid Calculations

The mulDivRoundingUp() function calculates the a*b/denominator by rounding up the value. This calculation is being used in various places like calculateAmountOut() function in UniswapV3SpellIntegrationOp.sol contract. There is an error in this calculation. In normal rounding up, the number should only be increased by 1 if the decimal part is greater than or equal to 5. This function, however, rounds up every decimal.

For more information, this PoC can be examined.

This can lead users to have more funds than they should.

Path: ./contracts/libraries/Fullmath.sol: mulDivRoundingUp()

Recommendation: Re-implement this function, so that it applies the correct rounding up rule.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

C03. Denial of Service Vulnerability

Within the for loop, the function makes an external call to the *controller.withdraw()* function. However, it does not check the *_toWithdraw* amount is greater than zero. In the scenario that this amount is equal to zero, the contract will revert without finishing the *for* loop.

This can lead to a block in users' fund withdrawal activities.

Path: ./contracts/deps/SettV3.sol: _withdraw()

Recommendation: Implement a balance check before performing the *controller.withdraw()* function.

Status: Fixed (Revised commit: 0d0354db7c3ba3c85791e77412d39648542d8356)

■■■ High

H01. Invalid Calculations

The calculateFee() and totalSupply() functions return value based on 1e18 decimal. Inside the epochHarvest() and getApproximateWithdrawableAmount() functions sharesToMint calculation divided is 1e14 (100*PRECISION) decimal.

This may actually return larger results than one intended to calculate.

Paths: ./contracts/deps/SettV3.sol : epochHarvest(), getApproximateWithdrawableAmount()

./contracts/AHStrategy.sol: _swapStableUSDC(), _ammCheck(), _checkPositionHealth()

Recommendation: Divide by 1e18 instead of 1e14.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

H02. Invalid Calculations

The square root function sqrt() of the HomoraMath.sol function should return a rounded-down value due to the nature of the calculation. In other words, the result of the computation must always be less than or equal to the input. However, the current implementation does not support this behavior in some cases.

For more information, this PoC can be examined.

This may lead to wrong results being used in the fund computations.

Path: ./contracts/libraries/HomoraMath.sol: sqrt()

Recommendation: Re-implement square root functionality.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

H03. Invalid Calculations

The getNextSqrtPriceFromAmount0RoundingUp() function returns the next sqrt price for a given delta of token0. It benefits from two formulas; one for normal cases and the other for when overflows occur. The overflow formula is the following: “liquidity / (liquidity / sqrtPX96 +- amount)”. When the issued function’s output is checked against the formula’s output range, there are some exceptions.

For more information, this PoC can be examined.

This may lead to wrong results being used in the fund computations.

Path: ./contracts/libraries/SqrtPriceMath.sol : getNextSqrtPriceFromAmount0RoundingUp()

Recommendation: Re-implement square root functionality.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

H04. Front Running Attack

The contract performs swaps on Uniswap through Alpha Homora. However, in those swaps, they do not consider the case where minOut is 0. The minOut values are calculated through given slippage.

This may lead to sandwich attacks.

Path: ./contracts/UniswapV3SpellIntegrationOp.sol: _convertToStable()

Recommendation: Implement checks for the case where minOut is zero.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

H05. Requirements Violation

The note comment states that the function must exclude any tokens used in the yield, but the code does not implement it.

Path: ./deps/BaseStrategy.sol: withdrawOther()

Recommendation: Either implement the missing logic or remove the related statement from the documentation

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

H06. Invalid Calculations

The _calculateAdminFee() function first calculates the adminFeeEpoch in a weekly manner (adminFee*timeSinceEpoch/(7*24*60*60)). However, when calculating the annual admin fee, it divides the value by 52 instead of (7*24*60*60*52).

This can lead to incorrectly calculated admin fees.

Path: ./contracts/deps/SettV3.sol: _calculateAdminFee()

Recommendation: Re-implement the calculation logic.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

H07. Insufficient Balance

While setting strategy allocation, the implementation withdraws all the funds from the related strategy to rebalance the environment; it then makes a call to earn() function, which transfers underlying

tokens to the controller. However, there may not be enough funds to send to the controller after rebalancing.

This can lead to imbalances.

Path: ./contracts/deps/SettV3.sol: setStrategiesAllocation()

Recommendation: Make sure there are enough funds.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

H08. Invalid Calculations

The `balanceOfPool()` function has this calculation: `(stableBalance*uint(10)**6)/stableDecimals` some of the stable tokens have 1e18 decimals. In such a scenario, this calculation may lead to an inaccurate result when applied to these tokens.

The `_setPnl()` function has this calculation: `(pnlInt*int(10)**6)/int(10)**ERC20Upgradeable(stableToken).decimals()` some of the stable tokens have 1e18 decimals. In such a scenario, this calculation may lead to an inaccurate result when applied to these tokens.

Path: ./contracts/AHStrategy.sol : balanceOfPool(), _setPnl()

Recommendation: Change the `(10)**6` to `stableDecimals.`

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

H09. Non-Finalized Code

The file has an exploiter contract which is used for testing purposes. The production code should not contain any functions or variables that are being used solely in the test environment. This will allow malicious parties to manipulate the code or users to trigger them accidentally.

Path: ./contracts/deps/SettV3.sol

Recommendation: Remove any variables, contracts, or other entities that are associated with the testing process.

Status: Fixed (Revised commit: 0d0354db7c3ba3c85791e77412d39648542d8356)

H10. Invalid Calculations

While calculating the pool balance in the strategy contract the function, calculates the pool balance with the following formula:

`balanceOfPoolInt()+getHarvestable()+IERC20Upgradeable(stableToken).balanceOf(address(this))`

Then function checks if the stable token is not USDC, if it is, it adds the active USDC balance to the calculated balance. This means that the USDC balance is used in the calculation twice.

This may lead to calculating more balance than it actually has.

Path: ./contracts/StrategyAlphaHomora.sol: balanceOfPool()

Recommendation: Re-implement the formula so that it will not add the USDC balance twice.

Status: Fixed (Revised commit: 0d0354db7c3ba3c85791e77412d39648542d8356)

Medium

M01. Inefficient Gas Model

The numbers of iterations of the loop in the functions are uncontrolled as it depends on stored data.

The numbers of iterations of the loop in the functions are uncontrolled as it depends on stored data and it makes external calls.

Path: ./contracts/deps/SettV3.sol : _harvest(), _deposit(), _withdraw(), setStrategiesAllocation(), _earn()

Recommendation: Implement loop length limitations.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

M02. Invalid Calculations

In the getApproximateWithdrawableAmount() function, it is possible for the calculation *_shares -= sharesAsFee* to yield unexpected results. Specifically, if the *fee* variable holds a sufficiently large value, the value of *sharesAsFee* can surpass the value of *_shares*.

Path: ./contracts/deps/SettV3.sol : getApproximateWithdrawableAmount()

Recommendation: Implement the necessary *_shares* is greater than *sharesAsFee* checks.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

M03. Inconsistent Data

The *getPositionBalance()* function makes an external call to *bank.getPositionInfo()*, which assigns the return value to the variable *collSize*, which is defined as a uint. Subsequently, the value of *collSize* is passed as an argument to another function as *uint128(collSize)*. However, if the value of *collSize* exceeds the maximum value of *uint128*, an overflow condition may occur.

The delta variable, which is defined as a *uint256*, is improperly cast to a *uint128*, which may result in an overflow.

The setTicks() function casts absTick, uint256, to int24.

Paths: ./contracts/UniswapV3SpellIntegrationOp.sol : _getPositionTokens(), _calculateAmountOut()

./contracts/AHStrategy.sol : _setTicks()

Recommendation: Refactor the logic in case of the overflow factor.

Status: Mitigated. Uniswap's implementation is being used.

M04. Inconsistent Data

The constructor takes in two input parameters: a dynamic array of config elements and a dynamic array of ticks. Within the constructor, six elements from the config array and two elements from the ticks array are utilized. It is important to ensure that the length of both lists is properly verified before proceeding.

Path: ./contracts/AHStrategy.sol.sol : constructor()

Recommendation: Re-implement the code so that it will use structs instead of two arrays.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

M05. Tautology

The *setThresholds()* function has a requirement that contains a contradiction. Specifically, the requirement that *_slippage >= 0* is in conflict with the definition of the *_slippage* variable as a *uint*. By definition, variables of type uint are always equal to or greater than ZERO.

Path: ./contracts/AHStrategy.sol : setThresholds()

Recommendation: Remove related *require* statement.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

M06. Contradiction

If the value of *priceChange* is less than all elements in the *gradientBreakPoints* list, at the end of the loop, the contract is reverting due to the *(uint i = gradientBreakPoints.length-1; i>=0; i--)* statement.

Path: ./contracts/UniswapV3SpellIntegrationOp.sol : getGradient()

Recommendation: Remove *=* operator from the statement and assign *uint i* as *i = gradientBreakPoints.length*. Replace the

gradientBreakPoints[i] and *gradients[i]* with *gradientBreakPoints[i-1]* and *gradients[i-1]* respectively.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

M07. Best Practice Violation

The Checks-Effects-Interactions pattern is violated. During the function, some state variables are updated after the external calls.

Paths: ./contracts/deps/Controller.sol : setStrategy()

./contracts/UniswapV3SpellIntegrationOp.sol: ensureApprove(), _convertToStable()

Recommendation: Implement the function according to the Checks-Effects-Interactions pattern.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

M08. Invalid Calculations

The *getAmount0Delta()* reverts in some cases due to the *customMulDiv()* function calculation overflow.

For more information, this PoC can be examined.

Path: ./contracts/libraries/SqrtPriceMath.sol : getAmount0Delta(), customMulDiv()

Recommendation: Refactor the calculation for the overflow factor.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

M09. Best Practice Violation

The LiquidityAmounts.sol (LiquidityAmounts and LiquidtyMath), SafeCast.sol, UnsafeCast.sol, and UniswapV3IntegrationOp.sol contracts contain multiple contracts in the same file.

Path: ./contracts/UniswapV3IntegrationOp.sol

Recommendation: Separate defined contracts into individual files.

Status: Fixed (Revised commit: 0d0354db7c3ba3c85791e77412d39648542d8356)

M10. Best Practice Violation

The StrategyAlphaHomora.sol and SettV3.sol, contracts contain multiple contracts in the same file.

Paths: ./contracts/StrategyAlphaHomora.sol

./contracts/SettV3.sol

Recommendation: Separate defined contracts into individual files.

Status: Fixed (Revised commit: 0d0354db7c3ba3c85791e77412d39648542d8356)

M11. Contradiction

The implementation contains commented code which looks like it should be uncommented to finalize the code.

Paths: ./contracts/deps/Controller.sol: earn()

./contracts/AHStrategy.sol: constructor()

./deps/BaseStrategy.sol: withdrawOther()

Recommendation: Remove the commented code or finalize its implementation.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

M12. Contradiction

_calculateFee's NatSpec block was placed in the block of another function.

withdrawAll() and withdrawAllForRebalance() functions' NatSpec state that they only allow partial withdrawals. However, the functions implemented to withdraw all of the funds.

Paths: ./contracts/deps/SettV3.sol: _calculateAdminFee()

./deps/BaseStrategy.sol: withdrawAll(), withdrawAllForRebalance(),

Recommendation: Change the NatSpec into the correct one.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

M13. Contradiction

The withdrawAllFromRebalance() function performs an authentication check it allows strategists, governance, and vault tokens. However, the error message does not include vault tokens.

Path: ./contracts/deps/Controller.sol: withdrawAllFromRebalance()

Recommendation: Update the error message.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

M14. Best Practice Violation

The UniswapV3SpellIntegrationOp.sol contract contains a fallback() function. However, there is no need for such a functionality.

Path: ./contracts/UniswapV3SpellIntegrationOp.sol: fallback()

Recommendation: Update the error message.

Status: Mitigated. Needed for Homora's spell contract.

M15. Inconsistent Data

Consider limiting the variables in order to prevent high fees, unexpected calculations etc.

Paths: ./contracts/AHStrategy.sol : ammCheckThreshold, debtRatioThreshold, volatilityThreshold, gradients, gradientBreakPoints

./contracts/deps/SettV3.sol: performanceFee, adminFee, withdrawalFee, min

Recommendation: Provide conscious limits for stored configuration values.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

M16. Invalid Calculations

In the _swapStableUSDC() function, it is possible for the calculation `expectedOut = balance*uint(10)**ERC20Upgradeable(tokenOut).decimals()/uint(10)**ERC20Upgradeable(tokenIn).decimals()` to yield unexpected results. Specifically, if the `tokenOut` decimal is different from the `tokenIn` decimal.

Path: ./contracts/AHStrategy.sol : _swapStableUSDC()

Recommendation: Implement the necessary checks.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

M17. Requirement Violation

The commented line states the performance fee is equal to `20%` but it is actually `2%`.

This can lead to misunderstanding about the contract.

Path: ./contracts/deps/SettV3.sol : initialize()

Recommendation: Either change the comment line or the logic.

Status: Fixed (Revised commit: 0d0354db7c3ba3c85791e77412d39648542d8356)

Low

L01. Misleading Contract Name

The contract name is mismatched with the file name.

Path: ./contracts/AHStrategy.sol

Recommendation: Either change the contract or file name.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L02. Floating Pragma

The project uses floating pragmas ^0.8.16, ^0.8.17.

Paths: ./contracts/UniswapV3SpellIntegrationOp.sol,

./contracts/AHStrategy.sol,

./deps/BaseStrategy.sol,

./deps/SettAccessControl.sol,

./deps/SettAccessControlDefended.sol,

./contracts/proxy/AdminUpgradeabilityProxy.sol,

./contracts/deps/Controller.sol,

./contracts/deps/SettV3.sol,

./contracts/libraries/BitMath.sol,

./contracts/libraries/FixedPoint96.sol,

./contracts/libraries/FixedPoint128.sol,

./contracts/libraries/FullMath.sol,

./contracts/libraries/HomoraMath.sol,

./contracts/libraries/LiquidityAmounts.sol,

./contracts/libraries/LowGasSafeMath.sol,

./contracts/libraries/SqrtPriceMath.sol,

./contracts/libraries/SwapMath.sol,

./contracts/libraries/TickBitMap.sol,

./contracts/libraries/TickMath.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L03. State Variables Can Be Declared Immutable

Variable`s PRECISION, MAX_INT, stableToken value is set in the constructor. These variables can be declared immutable.

This will lower the Gas taxes.

Paths: ./contracts/AHStrategy.sol:

./contracts/UniswapV3SpellIntegrationOp.sol:

Recommendation: Declare mentioned variables as immutable.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L04. Commented Code Parts

In the contract AHStrategy.sol lines 334-348 are commented parts of code.

This reduces code quality.

Path: ./contracts/AHStrategy.sol

Recommendation: Remove commented parts of code.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L05. Missing Events

Events for critical state changes should be emitted for tracking things off-chain.

Paths: ./deps/BaseStrategy.sol : setController(), setGuardian(), setWithdrawalMaxDeviationThreshold(),

./contracts/deps/Controller.sol: setStrategy(), approveStrategy(), revokeStrategy(), setRewards(), setVault(),

./deps/SettAccessControlDefended.sol: approveContractAccess(), revokeContractAccess(),

./deps/SettAccessControl.sol: setStrategist(), setKeeper(), setGovernance()

./contracts/deps/SettV3.sol: setFees(), setMin(), setController(), setGuardian(), setTreasury()

./contracts/AHStrategy.sol: setLeverage(), setThresholds(), setMinHarvestRequired()

Recommendation: Create and emit related events.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L06. State Variable Default Visibility

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variables (pnl, stableDecimals, volatileDecimals, PRECISION, MAX_INT, bank, factory, npm, router, stableToken, MAX_DEPOSIT.

Paths: ./contracts/AHStrategy.sol

./contracts/UniswapV3SpellIntegrationOp.sol

./contracts/deps/SettV3.sol

Recommendation: Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L07. Functions that Can Be Declared External

In order to save Gas, public functions that are never called in the contract should be declared as external.

Paths: ./deps/SettAccessControl.sol: setGovernance()

./contracts/deps/Controller.sol: approveStrategy(), revokeStrategy(), setRewards()

./contracts/AHStrategy.sol: getAmounts()

Recommendation: Use the external attribute for functions never called from the contract.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L08. Missing Zero Address Validation

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

Paths: ./deps/SettAccessControl.sol: setGovernance(), setKeeper(), setStrategist()

./contracts/deps/Controller.sol: setRewards(), withdrawAll()

./deps/BaseStrategy.sol: setController(), setGuardian()

./deps/SettAccessControlDefended.sol: approveContractAccess()

./contracts/deps/SettV3.sol: initialize()

./contracts/AHStrategy.sol : constructor()

./contracts/UniswapV3SpellIntegrationOp.sol: constructor

Recommendation: Implement zero address checks.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L09. Use of Hard-Coded Values

Hard-coded values are used in computations.

Paths: ./contracts/deps/SettV3.sol: _calculateAdminFee()

./contracts/StrategyAlphaHomora.sol : constructor()

Recommendation: Convert these variables into constants.

Status: Fixed (Revised commit: 0d0354db7c3ba3c85791e77412d39648542d8356)

L10. Unused Function

The functions created but not used in the project should be deleted. This will make a more Gas efficient contract.

Paths: ./contracts/AHStrategy.sol : withdrawSome()

./deps/BaseStrategy.sol: isTendable()

Recommendation: Remove unused function.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L11.Redundant Mathematical Operation

The mathematical operation `withdrawalFee = 1 * PRECISION` is redundant.

Path: ./contracts/deps/SettV3.sol: initialize()

Recommendation: Remove redundant mathematical operations.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L12. Redundant Import

The usage of IController is unnecessary for the contract.

Path: ./deps/BaseStrategy.sol

Recommendation: Remove the redundant import.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L13. Similar Modifiers

Modifiers with similar functionalities should be merged into one. onlyGovernance(), onlyAuthorizedActors() have similar functionalities.

Path: ./deps/SettAccessControl.sol: onlyGovernance(), onlyAuthorizedActors()

Recommendation: Merge modifiers.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L14. Boolean Equality

Boolean constants can be used directly and do not need to be compared to true or false.

Path: ./contracts/deps/Controller.sol: setStrategy()

Recommendation: Remove boolean equality.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L15. Zero Valued Transactions

The function withdraw can execute a zero-valued transaction if _amount input parameter is ZERO.

This can lead to a transaction with zero value to be sent.

Paths: ./deps/BaseStrategy.sol: withdraw()

./contracts/deps/Controller.sol inCaseTokensGetStuck(), inCaseStrategyTokenGetStuck()

Recommendation: Implement conditional checks for the zero-valued transaction.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

L16. Unimplemented Function

The function has no implementation.

Path: ./deps/BaseStrategy.sol: _postDeposit()

Recommendation: Implement the function or remove it.

Status: Fixed (Revised commit: f824453867b2c5bf30f6c334436cbca0f2d6974c)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, Consultant cannot guarantee the explicit security of the audited smart contracts.