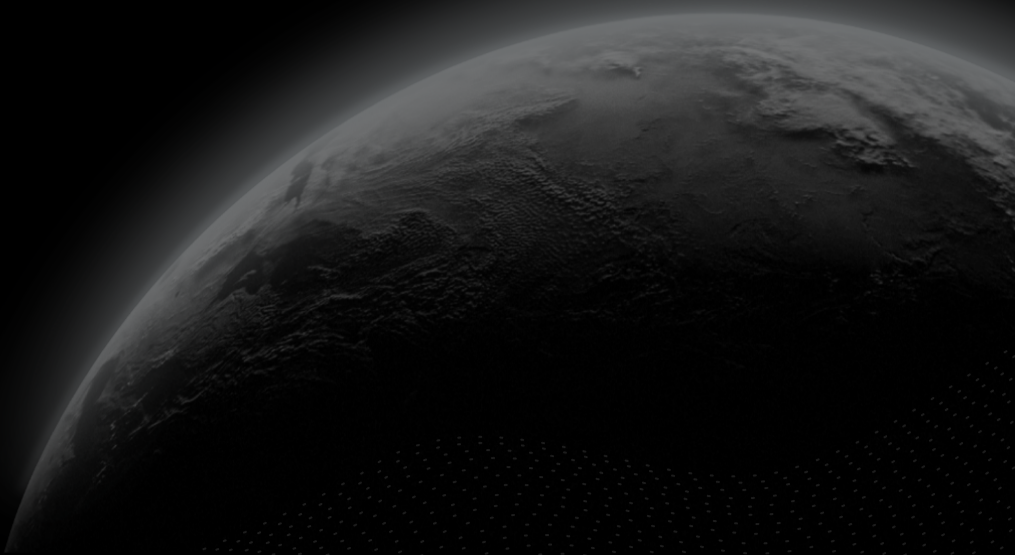




Security Assessment

ApolloX - Audit 2

CertiK Verified on May 10th, 2023





Certik Verified on May 10th, 2023

ApolloX - Audit 2

The security assessment was prepared by Certik, the leader in Web3.0 security.

Executive Summary

TYPES

Exchange

ECOSYSTEM

Arbitrum | Binance Smart
Chain (BSC) | Ethereum
(ETH)

METHODS

Formal Verification, Manual Review, Static Analysis

LANGUAGE

Solidity

TIMELINE

Delivered on 05/10/2023

KEY COMPONENTS

N/A

CODEBASE

update [1d4142c08a10b459c3625ceba84606135de3d2fd](#)base [32490e5cb13bf90af5cda621ae3464e77c250000](#)[...View All](#)

Vulnerability Summary



30

Total Findings

26

Resolved

0

Mitigated

0

Partially Resolved

4

Acknowledged

0

Declined

0

Unresolved



0

Critical

Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.



2

Major

2 Acknowledged



Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.



5

Medium

3 Resolved, 2 Acknowledged



Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.



16

Minor

16 Resolved



Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.



7

Informational

7 Resolved



Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | APOLLOX - AUDIT 2

I **Summary**

[Executive Summary](#)

[Vulnerability Summary](#)

[Codebase](#)

[Audit Scope](#)

[Approach & Methods](#)

I **Findings**

[ALP-01 : Centralization Risks in ALP.sol](#)

[AXB-02 : Centralization Risks in ApolloX.sol](#)

[FAC-01 : Potential Reentrancy Attack](#)

[LBM-01 : `brokerUpdate*\(\)` functions don't update the storage](#)

[LPF-01 : `LibPriceFacade.requestPriceCallback\(\)` can be too gas consuming](#)

[LPM-01 : `LibPairsManager.batchUpdatePairStatus\(\)` always reverts](#)

[PFF-01 : `PRICE_FEEDER_ROLE` and `KEEPER_ROLE` can manipulate the prices](#)

[AXI-01 : `supportsInterface\(\)` is inconsistent](#)

[LAM-02 : Lack of sanity check in `LibAlpManager._calculateAlpAmount\(\)`](#)

[LAR-01 : Unchecked ERC-20 `transfer\(\)`/`transferFrom\(\)` Call](#)

[LBM-02 : `LibBrokerManager.removeBroker\(\)` allows removing of `defaultBroker`](#)

[LCP-01 : Missing Validation on `latestRoundData\(\)`](#)

[LFM-01 : `LibFeeManager.chargeOpenFee\(\)` doesn't update `feeDetails.total` if `daoShareP == 0`](#)

[LPF-02 : The price from oracle explicitly converted to `uint64`](#)

[LPF-03 : `maxDelay` can be ignored by `PRICE_FEEDER_ROLE`](#)

[LTC-01 : Lack of sanity check in `TradingConfigFacet.initTradingConfigFacet\(\)`](#)

[LVB-01 : Strict comparison in `LibVault.decreaseByCloseTrade\(\)`](#)

[PMF-01 : Inconsistent checks in `_leverageMarginsCheck\(\)`](#)

[TCF-01 : Zero `entryPrice` returned by `TradingCheckerFacet.marketTradeCallbackCheck\(\)`](#)

[TOF-01 : Wrong `OrderInfo.amountIn` saved to history when new `openTrade` is created by `TradingOpenFacet.marketTradeCallback\(\)`](#)

[TPF-02 : `TradingPortalFacet.addMargin\(\)` allows to increase margin if `PairStatus.CLOSE`](#)

[TRA-01 : `TradingCloseFacet._decreaseByCloseTrade\(\)` can't extract all `openTradeAmountIns`](#)

[VFB-01 : No Upper Limits for Fees](#)

[CON-01 : Typos](#)

CON-02 : Redundant code

DIA-03 : Incompatibility with Deflationary Tokens

LAM-01 : Time Units Can Be Used

LAM-03 : `coolingDuration` can be avoided by whitelisted ALP owners

LIB-01 : Basis point values are referred as percent

LVB-02 : Redundant usage of `LibVault` namespace

I Optimizations

DIA-01 : Tautology

DIA-02 : Arguments Should Be `calldata`

FAC-03 : `_check()` argument can be declared `storage`

LAC-01 : Redundant data in `LibAccessControlEnumerable`

LIB-02 : Unnecessary Use of SafeMath

LIB-03 : `memory` variable can be used instead of `storage`

OAT-01 : `OrderAndTradeHistoryFacet.getOrderAndTradeHistory()` is gas consuming

TRA-02 : `TradingCloseFacet.transferToUserForClose()` can be optimized

I Formal Verification

Considered Functions And Scope

Verification Results

I Appendix

I Disclaimer

CODEBASE | APOLLOX - AUDIT 2















Repository


















update [1d4142c08a10b459c3625ceba84606135de3d2fd](#)


















base [32490e5cb13bf90af5cda621ae3464e77c250000](#)


















AUDIT SCOPE | APOLLOX - AUDIT 2











97 files audited ● 6 files with Acknowledged findings ● 23 files with Resolved findings ● 68 files without findings
















ID	File	SHA256 Checksum
● ALP	 contracts/ALP.sol	15f920de5d77abc3c0b16a9f24ad24c13ec7f08ccbe0c2c63b7c2a4bc119c50a
● ARF	 contracts/diamond/facets/ApxRewardFacet.sol	263d5dcaa2899fc2d198c0060e23aae3b0ccde6249d61880f9ee6b59cf5ae755
● BMF	 contracts/diamond/facets/BrokerManagerFacet.sol	97af7eb63449e13b393c05f541a25682d9aba19fdeedbe97dd033db91078b74
● PFF	 contracts/diamond/facets/PriceFacadeFacet.sol	f630cf0ee840600275c1119537d90b2236faf6e2b997166e9cd2810c0da73ef4
● LAC	 contracts/diamond/libraries/LibAccessControlEnumerable.sol	8e5f9b15cdbbc30a0a8d2e9b49e102804454de7735cc36f1f0159accfbfb153d
● AXB	 contracts/diamond/ApolloX.sol	d2dbd545e203a55bf491dc662b752f52a8f824a4c720e7d974efffeeb46145d6
● CON	 contracts/utils/Constants.sol	bc6118727ceb8d305222a3cf9830ec92843aed285465764674ce6cef7d3a2f9
● AMF	 contracts/diamond/facets/AlpManagerFacet.sol	f072f010dd6bfc845a6503d63caa0c65df07468736a627a769d00da2bcad2b1e
● LOF	 contracts/diamond/facets/LimitOrderFacet.sol	eb39a2f13598717f133f587de0b4730f1771b94217b30140917685efc7d33dbd
● OAT	 contracts/diamond/facets/OrderAndTradeHistoryFacet.sol	bdcbb123f06d400ec80ae2d4613d93e1a8c441a41310a85e5a2ba947d513458a6
● PMF	 contracts/diamond/facets/PairsManagerFacet.sol	467a2f18be5a437fc89733eb3092c37b32534a4736f2f14dd3ed44af3328dc86
● SRF	 contracts/diamond/facets/StakeRewardFacet.sol	df19200edf11c8c4b8c5d129e88a6678c6553d7ff380ec219cf868080fe383b3
● TCF	 contracts/diamond/facets/TradingCheckerFacet.sol	c5e5e7f1112e981024a8e4116462ee611c7ad26c583295e1d225ccccfb6d9735
● TRA	 contracts/diamond/facets/TradingCloseFacet.sol	3622807fedcafd0d37098446056ab46def5926d69fae74ae34271f592d440128

ID	File	SHA256 Checksum
● TOF	 contracts/diamond/facets/TradingOpenFacet.sol	2f81bd091457445c2bee8e2414c14fac9b050 6df53f75503265b2e8cd53a2710
● TPF	 contracts/diamond/facets/TradingPortalFacet.sol	3b91f6d76cd6cf7a9bd5fe23513da6fcbd4dffa6 131c4c2b6d7d173b8d5ecca0
● VFB	 contracts/diamond/facets/VaultFacet.sol	1cde88c8ba31f4b64133e96ac9195571a9d9a c019ecf5530b89eb4f7dc32829b
● LAM	 contracts/diamond/libraries/LibAlpManager.sol	20977aa2edeac0c64f928c31efe4306805c135 0c7788f065781d63ab90dd7a6b5
● LAR	 contracts/diamond/libraries/LibApxReward.sol	7850ed7c240928d6e39d121f3870c67644b1c 56003dbe03a3122b53ebc828a5c
● LBM	 contracts/diamond/libraries/LibBrokerManager.sol	777a8424367b2c01922f240952e7c2f5b1aa2 a8d4b0206135a013f468e45a7c1
● LCP	 contracts/diamond/libraries/LibChainlinkPrice.sol	e3c1326b5d547a1817445d735ac802a337f7a 470ba1eab8cb4c6860341fd5fb1
● LFM	 contracts/diamond/libraries/LibFeeManager.sol	0042478ec26a78d1e64551d9fb5764faafd910 8fcb5469c0875bc5539c73b05d
● LLO	 contracts/diamond/libraries/LibLimitOrder.sol	06d12fc4a64d7315e956eb6871600ea76d4ae 7db50966f6559e027674c2cfad2
● LPM	 contracts/diamond/libraries/LibPairsManager.sol	59153b65e28f17d34ada58c0bc5a5c09c26d0 62337b06dc9e64991dbe117da95
● LPF	 contracts/diamond/libraries/LibPriceFacade.sol	ab0eb1cc16ae86a6fd030d5528349d5006508 fa6f4da5ffe8c12f23487abad43
● LSR	 contracts/diamond/libraries/LibStakeReward.sol	21cb87df5000806324e2ff33fbf5856eddfbb04 2de3af0144f471a74aad5aa1a
● LTC	 contracts/diamond/libraries/LibTradingConfig.sol	70d688e39555fc3fd91f1f6c4cf3e0f049bb982f e28e85088c096dc6f53140a8
● LVB	 contracts/diamond/libraries/LibVault.sol	77f015c5ffb1bb3f0072a0497713b77610bf911 d6ac4d27d63cc5520e649497c
● AXI	 contracts/diamond/upgradeInitializers/ApolloXInit.sol	f435acceab751f8a6c780665b5d1425925dbd ded4b9243d5030f791b9ab416bd
● IWB	 contracts/dependencies/IWBNB.sol	977fd2f8dfa43437aa14d624768cbf85e0dc72 7b304f89c7d03d4f268190ae51
● BIT	 contracts/utils/Bits.sol	98b01bac7d4fb1e34651578762778241e7ca8 d2dc845876e2171e8a832391074

ID	File	SHA256 Checksum
● ACE	 contracts/diamond/facets/AccessControlEnumerableFacet.sol	8ede30f95bae75c5524c757d8b80bb74dccf08707165750d56f42e8e8e416614
● CPF	 contracts/diamond/facets/ChainlinkPriceFacet.sol	d489e34bb961646b9cc9844f66fce4decefb59ddd5ca2041f66d913f98d8965a
● DCF	 contracts/diamond/facets/DiamondCutFacet.sol	d340ea66cdfb4762fecb1cd63787141057f8a3463879994d1eac1702a2d43a09
● DLF	 contracts/diamond/facets/DiamondLoupeFacet.sol	0e928d5d13fede05d6378208b919d900104c47229590b30892f9130f61ccc605
● FMF	 contracts/diamond/facets/FeeManagerFacet.sol	14d1f231a13ae2c0c8db4bb0bd9747c71d282d1e36219939e90d8e3f602a8ce9
● PFB	 contracts/diamond/facets/PausableFacet.sol	65a98f9e86068aebff5f61c03ea926964a5c9a635f84dc438a9272ee59939141
● TRD	 contracts/diamond/facets/TradingConfigFacet.sol	05c6f52a8f499c6dce8c3aa89e2cae8c6061b046ffd131aeecc698b35c39d3886
● TRI	 contracts/diamond/facets/TradingCoreFacet.sol	2524aac3cc0978bd5f68a6e6d653b0b9504161415bbebc508eb24d451349b818
● TRF	 contracts/diamond/facets/TradingReaderFacet.sol	84045a01e2cd5329183049bd75676329559a587c007d22be3ac92679d6953656
● TFB	 contracts/diamond/facets/TransitionFacet.sol	9a898a5430fc8d8e67226dc3451c4a141019095267c6e439cdba51cd4a8bde5f
● LDB	 contracts/diamond/libraries/LibDiamond.sol	12395822b35ab9c0e53a1a1c0a7ace5b530df407ee41c2d00fee5c615fd2824f
● LOA	 contracts/diamond/libraries/LibOrderAndTradeHistory.sol	f07cbb8e837553706cb31fc7d04d6ccfa598e77dd676e5f81ab3fdb203c41f5e
● LTB	 contracts/diamond/libraries/LibTrading.sol	27b9caedcb0190c8a10fc473edc27cb03ea8bd9f6eeac0787632bfe270a48e0
● LIT	 contracts/diamond/libraries/LibTradingCore.sol	d85b105d9fa0227f2b4ffafac29ccb65f3a3bfd02d245267d34448bfc20cc5d3
● OSB	 contracts/diamond/security/OnlySelf.sol	2f62700e47f0f84c6e02f68faf508cfaf8515874d03d1caa02c09e929c92050f
● PAU	 contracts/diamond/security/Pausable.sol	f8d3effea268c040731ef4ba08ca472a49b995c8bdb679e07cc134ded52b6e5e
● RGB	 contracts/diamond/security/ReentrancyGuard.sol	5867ff3568a305eecef3c05085757047f8ca466d6f26b6a7b7c1d2c95f2e3da5

ID	File	SHA256 Checksum
● BIS	 contracts/utills/Bits.sol	98b01bac7d4fb1e34651578762778241e7ca8d2dc845876e2171e8a832391074
● COS	 contracts/utills/Constants.sol	3edbabd8143af5e40782952d823ae1381c3f1c5f3f090d812f1acc9fbdc4436b
● ALC	 contracts/ALP.sol	15f920de5d77abc3c0b16a9f24ad24c13ec7f08ccbe0c2c63b7c2a4bc119c50a
● ACF	 contracts/diamond/facets/AccessControlEnumerableFacet.sol	70d769fb6dae8bf4c19882752950fa39ad4d7f0b298f79453373cd481f237dd
● ALM	 contracts/diamond/facets/AlpManagerFacet.sol	6fb4fbb7365b463706aba138adafec143af66e25522700a89f0ffe3a78bc3cba
● APX	 contracts/diamond/facets/ApxRewardFacet.sol	79d3c03c960f842798d676a11988c782be41b6e4b2ab087c088aa756f719981c
● BRO	 contracts/diamond/facets/BrokerManagerFacet.sol	97af7eb63449e13b393c05f541a25682d9aba19fdeedbe97dd033db91078b74
● CHA	 contracts/diamond/facets/ChainlinkPriceFacet.sol	d489e34bb961646b9cc9844f66fce4decefb59ddd5ca2041f66d913f98d8965a
● DIA	 contracts/diamond/facets/DiamondCutFacet.sol	6754977d5831c0bad40ae4237816914f371eb070e3388d93364872bcb8d05c38
● DIM	 contracts/diamond/facets/DiamondLoupeFacet.sol	0e928d5d13fede05d6378208b919d900104c47229590b30892f9130f61ccc605
● FEE	 contracts/diamond/facets/FeeManagerFacet.sol	14d1f231a13ae2c0c8db4bb0bd9747c71d282d1e36219939e90d8e3f602a8ce9
● LIM	 contracts/diamond/facets/LimitOrderFacet.sol	6045b843562083cfa8ad657a7fd64c8c79877194b60d306876d6821d84e6f43d
● OAH	 contracts/diamond/facets/OrderAndTradeHistoryFacet.sol	53e11977530f2755bbb0f1164a156dfa3efde318bc1e065b5503f2e5de684e19
● PAI	 contracts/diamond/facets/PairsManagerFacet.sol	f435f48d9aafa2af803907131209d9d9ddb4c4c1c2a6b99916a8c8fa40f71b2a
● PFU	 contracts/diamond/facets/PausableFacet.sol	65a98f9e86068aebff5f61c03ea926964a5c9a635f84dc438a9272ee59939141
● PRI	 contracts/diamond/facets/PriceFacadeFacet.sol	f630cf0ee840600275c1119537d90b2236faf6e2b997166e9cd2810c0da73ef4
● STA	 contracts/diamond/facets/StakeRewardFacet.sol	df19200edf11c84b8c5d129e88a6678c6553d7ff380ec219cf868080fe383b3

ID	File	SHA256 Checksum
● TIM	 contracts/diamond/facets/TimeLockFacet.sol	725c0a99e1d7aa26e9a94c8aa8079626b3738157c2287e04e8320129a1d75410
● TRN	 contracts/diamond/facets/TradingCheckerFacet.sol	1524450821c8e43648399b46962f3325847ea01a3ba3aa919065b15c24042a3e
● TRG	 contracts/diamond/facets/TradingCloseFacet.sol	cb532608d06f0c103b0671792632cc871df3afbe79c7888e34ee741853c154c7
● TRC	 contracts/diamond/facets/TradingConfigFacet.sol	fa263e5c2f5eb55b890d6ce0f8f487a74d9082eeb56be0e67d1571c8b7d9ce84
● TRO	 contracts/diamond/facets/TradingCoreFacet.sol	2841a467762d402de7eb24f718e52981dc5645216d11c83a38b2ad86971b2b2d
● TRP	 contracts/diamond/facets/TradingOpenFacet.sol	f6ea980a058a7d877a792578567b39c50ec023d07e9df91cece2a42273f0529a
● TRR	 contracts/diamond/facets/TradingPortalFacet.sol	674051ff73929267db4e1c91a28fd1538b1727b856838125b09483f056b5ae04
● TRE	 contracts/diamond/facets/TradingReaderFacet.sol	84045a01e2cd5329183049bd75676329559a587c007d22be3ac92679d6953656
● TFU	 contracts/diamond/facets/TransitionFacet.sol	9a898a5430fc8d8e67226dc3451c4a141019095267c6e439cdba51cd4a8bde5f
● VFU	 contracts/diamond/facets/VaultFacet.sol	edf7c08d74885825e8e41434a825882b68e07af325fbe258dddfd9777179d8e6
● LAE	 contracts/diamond/libraries/LibAccessControlEnumerable.sol	dc16d922badf41b69e5475c0626c1310c5505ff7baa9432eaf42cfeb49331771
● LIL	 contracts/diamond/libraries/LibAlpManager.sol	e04970d31cd6887301f309f1094b9120c92b246346207dbf194b550284371b5a
● LIP	 contracts/diamond/libraries/LibApxReward.sol	1d3bfa71791d7e7db969b18271fa987c47e5472014128ae61f570aac1f0c3a5c
● LIO	 contracts/diamond/libraries/LibBrokerManager.sol	065d7d47b7e03a1c5a2fc04ac85052d84c4948c1393cc213778c8e132ee36933
● LIC	 contracts/diamond/libraries/LibChainlinkPrice.sol	ad0a834fe8444d44341ad4514f3027bd31046fc7b2b3fcf28e595567563d49e6
● LDU	 contracts/diamond/libraries/LibDiamond.sol	12395822b35ab9c0e53a1a1c0a7ace5b530df407ee41c2d00fee5c615fd2824f
● LIF	 contracts/diamond/libraries/LibFeeManager.sol	34c2b5e1cd5989ef12a24eb9eeb8ea2c0f11319f280be8fddcc57d7252d842b3

ID	File	SHA256 Checksum
● LII	 contracts/diamond/libraries/LibLimitOrder.sol	8f264939847f8bce8bf8c9990562ae1185bb6a9f9112b9c817f86faa89504063
● LOT	 contracts/diamond/libraries/LibOrderAndTradeHistory.sol	f07cbb8e837553706cb31fc7d04d6ccfa598e77dd676e5f81ab3fdb203c41f5e
● LIS	 contracts/diamond/libraries/LibPairsManager.sol	aaaae64907d2673a52babd8b972df2933a3c160f4034b5780cbae2fbae9b71544
● LIE	 contracts/diamond/libraries/LibPriceFacade.sol	0e4600f2bddcc2e0ded3353074d8ab7d399907818f19ee37f09411e472ed7425
● LID	 contracts/diamond/libraries/LibStakeReward.sol	b2b2eb46dc0c0e2dbc74151af8bff38306b15e0bb3f1e1bf18ac89dcb154f4
● LIN	 contracts/diamond/libraries/LibTimeLock.sol	2078c6cc2fe84cc9948d217ed45663347b0d688c907df24a7c6983965c4ebefd
● LTU	 contracts/diamond/libraries/LibTrading.sol	27b9caedcb0190c8a10fc473edc27cb03ea8bd9f6eeacf0787632bfe270a48e0
● LI8	 contracts/diamond/libraries/LibTradingConfig.sol	70d688e39555fc3fd91f1f6c4cf3e0f049bb982fe28e85088c096dc6f53140a8
● LIU	 contracts/diamond/libraries/LibTradingCore.sol	d85b105d9fa0227f2b4ffafac29ccb65f3a3bfd02d245267d34448bfc20cc5d3
● LVU	 contracts/diamond/libraries/LibVault.sol	a2256f92e3a33b5f7c10b3bcc3334339e5d77bc48a57b162d061c0ca3059d68f
● OSU	 contracts/diamond/security/OnlySelf.sol	2f62700e47f0f84c6e02f68faf508cfaf8515874d03d1caa02c09e929c92050f
● PAS	 contracts/diamond/security/Pausable.sol	f8d3effea268c040731ef4ba08ca472a49b995c8bdb679e07cc134ded52b6e5e
● RGU	 contracts/diamond/security/ReentrancyGuard.sol	5867ff3568a305eecef3c05085757047f8ca466d6f26b6a7b7c1d2c95f2e3da5
● All	 contracts/diamond/upgradeInitializers/ApolloXInit.sol	678551bef09f3e0ac65b85c4646830c01e3629aaf41ce5c806d95220fe815dfa
● AXU	 contracts/diamond/ApolloX.sol	7e9b7f3e12181e63e12ed87bb2c0af7eb8a64f8f68761cddb10ad5612841b2ef

APPROACH & METHODS | APOLLOX - AUDIT 2

This report has been prepared for ApolloX to discover issues and vulnerabilities in the source code of the ApolloX - Audit 2 project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

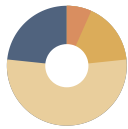
The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | APOLLOX - AUDIT 2



30

Total Findings

0

Critical

2

Major

5

Medium

16

Minor

7

Informational

This report has been prepared to discover issues and vulnerabilities for ApolloX - Audit 2. Through this audit, we have uncovered 30 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
ALP-01	Centralization Risks In ALP.Sol	Centralization / Privilege	Major	● Acknowledged
AXB-02	Centralization Risks In ApolloX.Sol	Centralization / Privilege	Major	● Acknowledged
FAC-01	Potential Reentrancy Attack	Logical Issue	Medium	● Acknowledged
LBM-01	<code>brokerUpdate*()</code> Functions Don't Update The Storage	Language Specific	Medium	● Resolved
LPF-01	<code>LibPriceFacade.requestPriceCallback()</code> Can Be Too Gas Consuming	Volatile Code	Medium	● Resolved
LPM-01	<code>LibPairsManager.batchUpdatePairStatus()</code> Always Reverts	Volatile Code	Medium	● Resolved
PFF-01	<code>PRICE_FEEDER_ROLE</code> And <code>KEEPER_ROLE</code> Can Manipulate The Prices	Centralization / Privilege	Medium	● Acknowledged
AXI-01	<code>supportsInterface()</code> Is Inconsistent	Inconsistency	Minor	● Resolved
LAM-02	Lack Of Sanity Check In <code>LibAlpManager._calculateAlpAmount()</code>	Volatile Code	Minor	● Resolved
LAR-01	Unchecked ERC-20 <code>transfer()</code> / <code>transferFrom()</code> Call	Volatile Code	Minor	● Resolved
LBM-02	<code>LibBrokerManager.removeBroker()</code> Allows Removing Of <code>defaultBroker</code>	Volatile Code	Minor	● Resolved

ID	Title	Category	Severity	Status
LCP-01	Missing Validation On <code>latestRoundData()</code>	Volatile Code	Minor	● Resolved
LFM-01	<code>LibFeeManager.chargeOpenFee()</code> Doesn't Update <code>feeDetails.total</code> If <code>daoShareP == 0</code>	Volatile Code	Minor	● Resolved
LPF-02	The Price From Oracle Explicitly Converted To <code>uint64</code>	Volatile Code	Minor	● Resolved
LPF-03	<code>maxDelay</code> Can Be Ignored By <code>PRICE_FEEDER_ROLE</code>	Volatile Code	Minor	● Resolved
LTC-01	Lack Of Sanity Check In <code>TradingConfigFacet.initTradingConfigFacet()</code>	Volatile Code	Minor	● Resolved
LVB-01	Strict Comparison In <code>LibVault.decreaseByCloseTrade()</code>	Volatile Code	Minor	● Resolved
PMF-01	Inconsistent Checks In <code>_leverageMarginsCheck()</code>	Inconsistency	Minor	● Resolved
TCF-01	Zero <code>entryPrice</code> Returned By <code>TradingCheckerFacet.marketTradeCallbackCheck()</code>	Logical Issue	Minor	● Resolved
TOF-01	Wrong <code>OrderInfo.amountIn</code> Saved To History When New <code>openTrade</code> Is Created By <code>TradingOpenFacet.marketTradeCallback()</code>	Inconsistency	Minor	● Resolved
TPF-02	<code>TradingPortalFacet.addMargin()</code> Allows To Increase Margin If <code>PairStatus.CLOSE</code>	Volatile Code	Minor	● Resolved
TRA-01	<code>TradingCloseFacet._decreaseByCloseTrade()</code> Can't Extract All <code>openTradeAmountIns</code>	Volatile Code	Minor	● Resolved
VFB-01	No Upper Limits For Fees	Logical Issue	Minor	● Resolved
CON-01	Typos	Coding Style	Informational	● Resolved

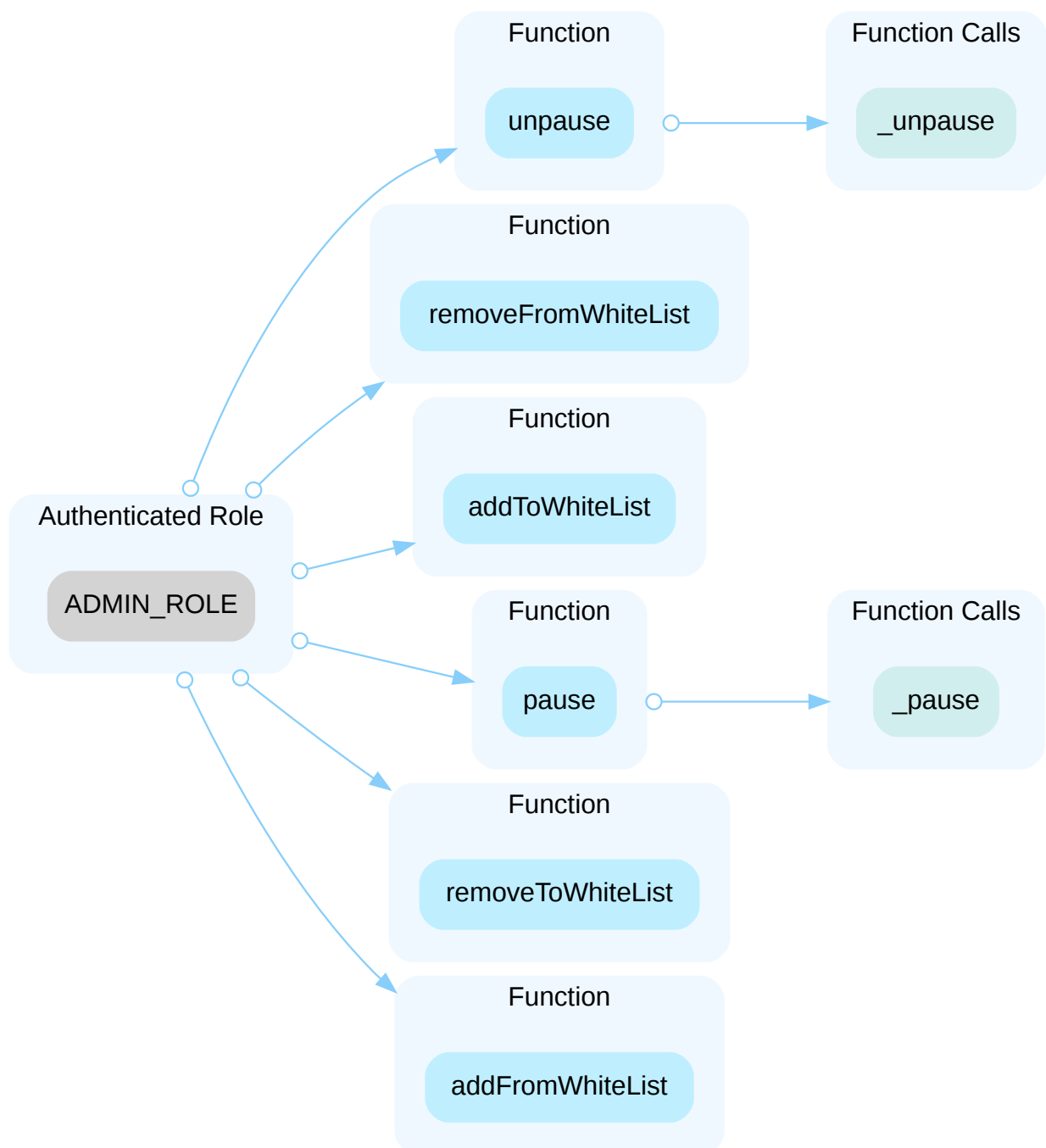
ID	Title	Category	Severity	Status
CON-02	Redundant Code	Coding Style	Informational	● Resolved
DIA-03	Incompatibility With Deflationary Tokens	Logical Issue	Informational	● Resolved
LAM-01	Time Units Can Be Used	Magic Numbers	Informational	● Resolved
LAM-03	<code>coolingDuration</code> Can Be Avoided By Whitelisted ALP Owners	Volatile Code	Informational	● Resolved
LIB-01	Basis Point Values Are Referred As Percent	Inconsistency	Informational	● Resolved
LVB-02	Redundant Usage Of <code>LibVault</code> Namespace	Coding Style	Informational	● Resolved

ALP-01 | CENTRALIZATION RISKS IN ALP.SOL

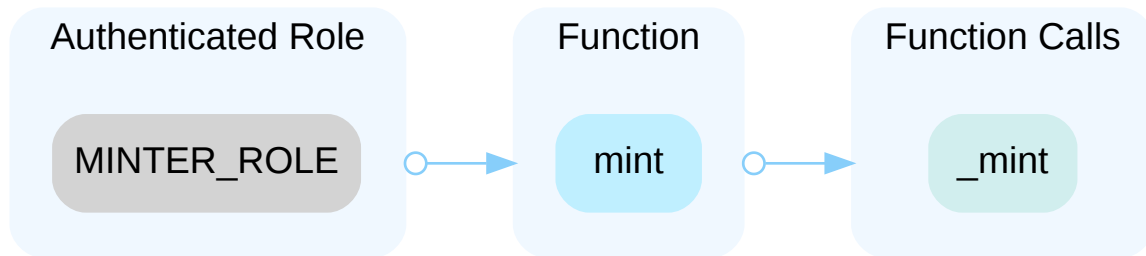
Category	Severity	Location	Status
Centralization / Privilege	● Major	contracts/ALP.sol (base): 35 , 39 , 43 , 48 , 53 , 58 , 63 , 74	● Acknowledged

Description

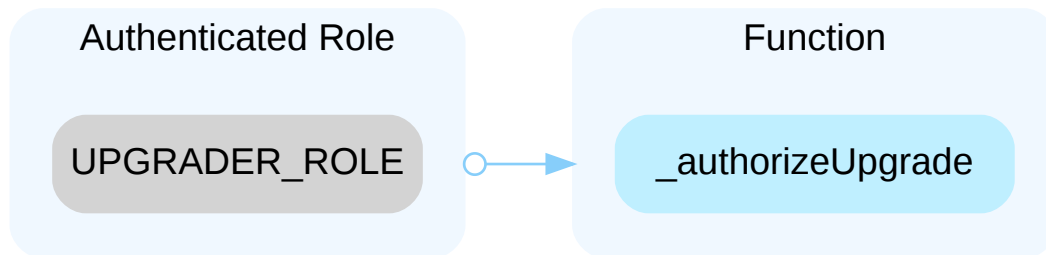
In the contract `ALP` the role `ADMIN_ROLE` has authority over the functions shown in the diagram below.



In the contract `ALP` the role `MINTER_ROLE` has authority over the functions shown in the diagram below.



In the contract `ALP` the role `UPGRADER_ROLE` has authority over the functions shown in the diagram below.



Any compromise to the privileged roles may allow the hacker to take advantage of this and

- `mint()` any amount of `ALP`
- `upgradeTo()` any other implementation contract
- `pause()` / `unpause()` , update whitelists

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign ($\frac{2}{3}$, $\frac{3}{5}$) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
OR
- Remove the risky functionality.

I Alleviation

[Project Team]: We will not implement the time lock for parameter update because we need to make rapid reactions to adjust parameters based on market situation. Moreover, we have actually added the time lock for upgrade which is managed by a multi-signature address. We plan to distribute more rights (including the management of multi-signature etc) to our DAO governance to achieve even higher decentralization.

AXB-02 | CENTRALIZATION RISKS IN APOLLOX.SOL

Category	Severity	Location	Status
Centralization / Privilege	● Major	contracts/diamond/ApolloX.sol (base): <u>35</u>	● Acknowledged

Description

In the contract `ApolloX`

- the role `DEPLOYER_ROLE` has the authority to upgrade all facets and initialize them.
- the role `DEFAULT_ADMIN_ROLE` has the authority to edit other roles.
- other roles can perform sensitive operations.

Any compromise to the privileged roles may allow the hacker to take advantage of this and

- upgrade any facet with new functionality
- add/remove pairs/brokers/commissions, etc.
- update staking reward via `updateApXPerBlock()`
- provide any prices and execute the orders

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign ($\frac{2}{3}$, $\frac{3}{5}$) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
OR
- Remove the risky functionality.

I Alleviation

[Project Team]: We will not implement the time lock for parameter update because we need to make rapid reactions to adjust parameters based on market situation. Moreover, we have actually added the time lock for upgrade which is managed by a multi-signature address. We plan to distribute more rights (including the management of multi-signature etc) to our DAO governance to achieve even higher decentralization.

FAC-01 | POTENTIAL REENTRANCY ATTACK

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/diamond/facets/ApxRewardFacet.sol (base): <u>26</u> ; contracts/diamond/facets/BrokerManagerFacet.sol (base): <u>95</u>	● Acknowledged

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects.

- `ApxRewardFacet.addReserves()` performs an external call to `rewardToken.transferFrom()` and only after that updates the contract state. Can't be exploited.
- `BrokerManagerFacet.withdrawCommission()` performs an external call to `token.safeTransfer()` and only after that updates `c.pending = 0`. This function can be exploited by anyone. As a result, the `broker` will get `allPendingCommissions` of all tokens of all other brokers.
- `LimitOrderFacet.openLimitOrder()` performs an external call to `token.safeTransferFrom()` and updates the contract state. Can't be exploited.
- `LimitOrderFacet.executeLimitOrder()` performs an external call to `token.safeTransfer()` and only after that updates the contract state via `_removeOrder()`. Can be exploited. As a result, the same order can be canceled with a refund or executed twice.

Also affected:

- `LimitOrderFacet.cancelLimitOrder()`
- `TradingPortalFacet.openMarketTrade()`
- `TradingPortalFacet.addMargin()`
- `VaultFacet.addToken()`
- `TradingCloseFacet.executeTpSlOrLiq()`
- `TradingPortalFacet.addMargin()`

and others.

Recommendation

We recommend protecting all the external functions not supposed to be re-entered by applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier to prevent reentrancy attack.

Alleviation

[Project Team]: We should be cautious when calling unknown contracts as they may be malicious and potentially cause a reentrancy attack. Therefore, it is important to ensure that all called contracts are known and trustworthy, or to use libraries such as OpenZeppelin's ReentrancyGuard for reentrancy protection. As the external contracts are added by accounts with permissions, only known contracts like USDT/USDC/WBNB are added, which are not malicious contracts. Adding `ReentrancyGuard`, however, will result in additional gas consumption.

LBM-01 | `brokerUpdate*()` FUNCTIONS DON'T UPDATE THE STORAGE

Category	Severity	Location	Status
Language Specific	● Medium	contracts/diamond/libraries/LibBrokerManager.sol (base): <u>81~85</u>	● Resolved

Description

```
81     function _checkBrokerExist(BrokerManagerStorage storage bms, uint24 id)
private view returns (Broker memory) {
82         Broker memory b = bms.brokers[id];
83         require(b.receiver != address(0), "LibBrokerManager: broker does not
exist");
84         return b;
85     }
```

`LibBrokerManager._checkBrokerExist()` returns `Broker memory`.

```
114         Broker memory b = _checkBrokerExist(bms, id);
115         address oldReceiver = b.receiver;
116         b.receiver = receiver;
```

`memory` structure is updated in `updateBrokerReceiver()` and other functions. As a result, the storage is left intact.

Recommendation

We recommend returning `Broker storage` from `_checkBrokerExist()`.

LPF-01 | `LibPriceFacade.requestPriceCallback()` CAN BE TOO GAS CONSUMING

Category	Severity	Location	Status
Volatile Code	● Medium	contracts/diamond/libraries/LibPriceFacade.sol (base): <u>120</u>	● Resolved

Description

Users can create very big number of orders and price requests via `TradingPortalFacet.openMarketTrade()` in the same block. Then `PRICE_FEEDER_ROLE` will be unable to execute `PriceFacadeFacet.requestPriceCallback()` due to gas limitation.

In `LibPriceFacade.requestPriceCallback()`

- all the requests are copied into memory from `pfs.pendingPrices[requestId]`
- all the requests are processed and then deleted from storage

As a result, the created orders will not be processed and the price feeder will be stuck.

Recommendation

We recommend limiting the number of open orders per block or introducing partial price request processing.

LPM-01 | `LibPairsManager.batchUpdatePairStatus()` ALWAYS REVERTS

Category	Severity	Location	Status
Volatile Code	● Medium	contracts/diamond/libraries/LibPairsManager.sol (base): 286	● Resolved

Description

```
286         for (UC i = ZERO; i <= uc(pairBases.length); i = i + ONE) {  
287             Pair storage pair = pms.pairs[pairBases[i.into()]];
```

Indexing `pairBases[pairBases.length]` is not allowed and will always revert.

Recommendation

We recommend using `i < uc(pairBases.length)` instead.

PFF-01 | PRICE_FEEDER_ROLE AND KEEPER_ROLE CAN MANIPULATE THE PRICES

Category	Severity	Location	Status
Centralization / Privilege	● Medium	contracts/diamond/facets/PriceFacadeFacet.sol (base): <u>51</u>	● Acknowledged

Description

The trading works this way:

1. The user calls `TradingPortalFacet.openMarketTrade()`, creates `pendingTrades` and updates `pendingPrice`
2. `PRICE_FEEDER_ROLE` provides the price via `PriceFacadeFacet.requestPriceCallback()`
3. If the price `gapPercentage <= pfs.highPriceGapP` the price is considered valid and cached to `callbackPrices`
4. `pendingPrices[requestId]` processed and `marketTradeCallback()` / `closeTradeCallback()` are called

As a result, `PRICE_FEEDER_ROLE` can change the cached price via many calls by steps not bigger than `highPriceGapP`.

Limit orders work similar way:

1. The user calls `LimitOrderFacet.openLimitOrder()`, creates `limitOrders`
2. `KEEPER_ROLE` provides the price via `LimitOrderFacet.executeLimitOrder()`
3. If the price `gapPercentage <= pfs.highPriceGapP` the price is considered valid and cached to `callbackPrices` via `PriceFacadeFacet.confirmTriggerPrice()`
4. `LibLimitOrder.executeLimitOrder()` is called

As a result, `KEEPER_ROLE` can change the cached price via many calls by steps not bigger than `highPriceGapP`.

Changing the price allows the privileged roles to manipulate the market and execute the orders not supposed to be executed.

Scenario

Consider the scenario:

1. `KEEPER_ROLE` takes any `isLong` limit order with low `order.limitPrice`, takes `beforePrice = pfs.callbackPrices[token]`.
2. `KEEPER_ROLE` calculates `newPrice` so, that $(beforePrice - newPrice) * 1e4 / beforePrice = pfs.highPriceGapP$. That means that `newPrice` is lower than `beforePrice` by $beforePrice * pfs.highPriceGapP / 1e4$.

3. `KEEPER_ROLE` calls `LimitOrderFacet.executeLimitOrder()` with `executeOrders = KeeperExecution(orderHash, newPrice)`.
4. `pfs.callbackPrices[token]` gets updated by `(newPrice, block.timestamp)`.
5. The `order` is not executed since `TradingCheckerFacade.executeLimitOrderCheck()` returns `(false, 0, 0, Refund.USER_PRICE)`.
6. `KEEPER_ROLE` repeats steps 2-4 until the order is executed.

`PRICE_FEEDER_ROLE` can perform similar price manipulations.

Recommendation

We recommend caching and using the prices only received from `LibChainlinkPrice.getPriceFromChainlink(token)`.

Alleviation

[Project Team]: `PRICE_FEEDER_ROLE` will be assigned to the Binance Oracle address to avoid relying on a single price source. We are using both Binance Oracle and Chainlink prices to ensure a diverse set of prices. When selecting a reference price, we will use the most recent price available.

AXI-01 | `supportsInterface()` IS INCONSISTENT

Category	Severity	Location	Status
Inconsistency	Minor	contracts/diamond/upgradeInitializers/ApolloXInit.sol (base): <u>16~25</u>	Resolved

Description

Diamond initialization works this way:

1. `ApolloXInit` contract is deployed with `init()` function
2. `ApolloX` contract is deployed with `ApolloXInit` address specified as `_init` argument
3. `ApolloX` constructor calls `ApolloXInit.init()` function via `delegatecall()`
4. `init()` adds 3 interfaces to `DiamondStorage.supportedInterfaces` and 3 more to `LibAccessControlEnumerable.supportedInterfaces`

Both `DiamondLoupeFacet` and `AccessControlEnumerableFacet` have `supportsInterface()` functions, each using its own storage.

It is unclear which one will be used by the Diamond and unclear why the Diamond needs both of them.

Recommendation

We recommend leaving only one `supportsInterface()` function and storing all `supportedInterfaces` at one facet.

LAM-02 | LACK OF SANITY CHECK IN

`LibAlpManager._calculateAlpAmount()`

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/diamond/libraries/LibAlpManager.sol (base): <u>85~86</u> , <u>128~129</u>	● Resolved

Description

```
85         // ∴ alpPrice_ > 0
86         // ∴ (LibVault.getTotalValueUsd() + lpUnPnlUsd) > 0
```

The code has ensured that `alpPrice > 0` and assumes that `(LibVault.getTotalValueUsd() + lpUnPnlUsd) > 0`. That value is explicitly converted to `uint256`.

However, if `totalValueUsd < 0` and `alp.totalSupply == 0`, the `_alpPrice(totalValueUsd)` returns positive value `1e8`. So, in some circumstances `LibVault.getTotalValueUsd() + lpUnPnlUsd` can be negative.

Recommendation

We recommend explicitly checking the value is non-negative before `uint256` conversion. We recommend adding `int256` `totalValueUsd = LibVault.getTotalValueUsd() + lpUnPnlUsd` to avoid redundant calculations.

LAR-01 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/diamond/libraries/LibApxReward.sol (base): <u>148</u>	● Resolved

Description

```
148      ars.rewardToken.transferFrom(msg.sender, address(this), amount);
```

The return value of the `transfer()` / `transferFrom()` call is not checked.

Recommendation

Since some ERC-20 tokens return no values and others return a `bool` value, they should be handled with care. We advise using the [OpenZeppelin's SafeERC20.sol](#) implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if `false` is returned, making it compatible with all ERC-20 token implementations.

LBM-02 | `LibBrokerManager.removeBroker()` ALLOWS REMOVING OF `defaultBroker`

Category	Severity	Location	Status
Volatile Code	Minor	contracts/diamond/libraries/LibBrokerManager.sol (base): <u>100</u>	Resolved

Description

`LibBrokerManager.removeBroker()` doesn't check that the removed broker is `defaultBroker`. `defaultBroker` is used by `updateBrokerCommission()` in case the requested broker is absent. In case it was removed, the commissions will be accumulated for the same `id` and can be withdrawn if a new broker with the same `id` will be added in the future.

Recommendation

We recommend preventing of `defaultBroker` removal.

LCP-01 | MISSING VALIDATION ON `latestRoundData()`

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/diamond/libraries/LibChainlinkPrice.sol (base): <u>65~66</u>	● Resolved

Description

```
65      (, int256 price_, uint256 startedAt_,) = oracle.latestRoundData();
66      price = uint256(price_);
```

The `price` provided by `oracle.latestRoundData()` can theoretically be negative. In this case, it is silently converted to `uint256`.

Recommendation

We recommend checking the return values of third-party services and reverting in case of unexpected.

LFM-01 | `LibFeeManager.chargeOpenFee()` DOESN'T UPDATE `feeDetails.total` IF `daoShareP == 0`

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/diamond/libraries/LibFeeManager.sol (base): <u>120-124</u> , <u>141</u>	● Resolved

Description

```
120         if (daoShare > 0) {
121             IERC20(token).safeTransfer(fms.daoRepurchase, daoShare);
122             detail.total += feeAmount;
123             detail.daoAmount += daoShare;
124         }
```

`LibFeeManager` allows `daoShareP` to be zero. However, in this case, the `LibFeeManager.chargeOpenFee()` doesn't update `feeDetails[token].total`. `FeeManagerFacet.getFeeDetails()` will return incorrect results.

`chargeCloseFee()` is also affected.

Recommendation

We recommend updating the `detail.total` in any case.

LPF-02 | THE PRICE FROM ORACLE EXPLICITLY CONVERTED TO `uint64`

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/diamond/libraries/LibPriceFacade.sol (base): <u>170</u>	● Resolved

Description

`LibPriceFacade.getPriceFromCacheOrOracle()` gets the `uint256` price by `LibChainlinkPrice.getPriceFromChainlink()` and then explicitly converts it to `uint64`. This can lead to a accidental hidden overflow that will get unnoticed.

Recommendation

We recommend explicitly checking that the provided by the third-party values fit into `uint64`.

LPF-03 | `maxDelay` CAN BE IGNORED BY `PRICE_FEEDER_ROLE`

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/diamond/libraries/LibPriceFacade.sol (base): <u>135~138</u>	● Resolved

Description

```
135      // The time interval is too long.
136      // receive the current price but not use it
137      // and wait for the next price to be feed.
138      if (block.timestamp > updatedAt + pfs.maxDelay) {
```

In `LibPriceFacade.requestPriceCallback()` the `PRICE_FEEDER_ROLE` provides the `price`. If the `beforePrice` extracted by `getPriceFromCacheOrOracle()` was stored there more than `pfs.maxDelay` time ago, then the provided `price` is "rejected".

However, since that `price` is saved to `pfs.callbackPrices[pendingPrice.token]` with the current `block.timestamp`, the next call to `requestPriceCallback()` by `PRICE_FEEDER_ROLE` with the same arguments will be successful: the price will be used, callbacks called, `pendingPrices` deleted.

Recommendation

We recommend clarifying the intended logic of `pfs.maxDelay`.

Alleviation

[Project Team]: We use both Binance Oracle and Chainlink price feeds. If the Chainlink price has not been updated for a period exceeding `maxDelay`, we consider it unreliable and only use the Binance Oracle price. To ensure the accuracy of the price, we reject the first price and accept the second price, which is equivalent to a two-step confirmation process.

LTC-01 | LACK OF SANITY CHECK IN

`TradingConfigFacet.initTradingConfigFacet()`

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/diamond/libraries/LibTradingConfig.sol (base): <u>35</u>	● Resolved

Description

```
35         require(tcs.executionFeeUsd == 0 && tcs.minNotionalUsd == 0 &&
tcs.maxTakeProfitP == 0, "LibTradingConfig: Already initialized");
```

`TradingConfigFacet.initTradingConfigFacet()` is supposed to be called once by `DEPLOYER_ROLE`. The check above is supposed to ensure that. However, all three argument values can and probably will be 0, `initTradingConfigFacet()` doesn't enforce the arguments to be non-zero.

Recommendation

We recommend adding `require(minNotionalUsd > 0 && maxTakeProfitP > 0)` to make the function consistent with other library setters.

LVB-01 | STRICT COMPARISON IN

`LibVault.decreaseByCloseTrade()`

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/diamond/libraries/LibVault.sol (base): <u>214</u>	● Resolved

Description

```
214         require(index.into() > 0 && otherTokenAmountUsd < totalBalanceUsd,  
"LibVault: Insufficient funds in the treasury");
```

The code requires `otherTokenAmountUsd` to be strictly less than `totalBalanceUsd`, however, equal balances are also enough to finish the operation.

Recommendation

We recommend using `otherTokenAmountUsd <= totalBalanceUsd` instead.

PMF-01 | INCONSISTENT CHECKS IN `_leverageMarginsCheck()`

Category	Severity	Location	Status
Inconsistency	Minor	contracts/diamond/facets/PairsManagerFacet.sol (base): <u>141</u>	Resolved

Description

`PairsManagerFacet._leverageMarginsCheck()` performs checks of `leverageMargins`.

The check `lm.tier >= leverageMargins[(i + ONE).into()].tier` is redundant since `lm.tier != (i + ONE).into()` check is performed.

It is not ensured that `lm.initialLostP > nextLm.initialLostP`.

Recommendation

We recommend rewriting the conditions in `require()` form (ensuring the conditions are satisfied instead of looking for unsatisfied). We recommend adding the missing condition and removing redundant one.

TCF-01 | ZERO `entryPrice` RETURNED BY `TradingCheckerFacet.marketTradeCallbackCheck()`

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/diamond/facets/TradingCheckerFacet.sol (base): 389	● Resolved

Description

```
389         return (false, 0, 0, entryPrice, Refund.TP);
```

`entryPrice` is zero here. `tuple.entryPrice` is supposed to be returned. The value is unused by the caller.

Recommendation

We recommend using `tuple.entryPrice` here.

TOF-01 | WRONG `OrderInfo.amountIn` SAVED TO HISTORY WHEN NEW `openTrade` IS CREATED BY `TradingOpenFacet.marketTradeCallback()`

Category	Severity	Location	Status
Inconsistency	● Minor	contracts/diamond/facets/TradingOpenFacet.sol (base): 47	● Resolved

Description

`TradingOpenFacet._marketTrade()` calls `OrderAndTradeHistoryFacet.marketTrade()` with `OrderInfo.amountIn` argument `ot.margin + ot.openFee`. The real `amountIn` value was bigger by `ot.executionFee`.

For example, when `LibLimitOrder.openLimitOrder()` calls `OrderAndTradeHistoryFacet.createLimitOrder()`, it uses full `amountIn` transferred from the user.

`TradingPortalFacet.addMargin()` also saves the full `margin` received via `OrderAndTradeHistory.updateMargin()`.

Recommendation

We recommend saving `ot.margin + ot.openFee + ot.executionFee` in a call to `OrderAndTradeHistoryFacet.marketTrade()`.

TPF-02 | `TradingPortalFacet.addMargin()` ALLOWS TO INCREASE MARGIN IF `PairStatus.CLOSE`

Category	Severity	Location	Status
Volatile Code	Minor	contracts/diamond/facets/TradingPortalFacet.sol (base): <u>138</u>	Resolved

Description

`TradingPortalFacet.addMargin()` can be executed even if `TradingConfig.marketTrading` is unset or `getPairForTrading(ot.pairBase).status` is `PairStatus.CLOSE`.

Recommendation

We recommend clarifying the intended behavior.

Alleviation

[Project Team]: The act of adding collateral does not change any behavior of LP or ALP, and is intentionally designed to be independent of the trading pair and market conditions.

TRA-01 | `TradingCloseFacet._decreaseByCloseTrade()` CAN'T EXTRACT ALL `openTradeAmountIns`

Category	Severity	Location	Status
Volatile Code	Minor	contracts/diamond/facets/TradingCloseFacet.sol (base): <u>181</u>	Resolved

Description

`TradingCloseFacet._decreaseByCloseTrade()` calculates the total `openTradeAmountIns` in USD as `totalBalanceUsd`.

`otherTokenAmountUsd` is the amount in USD required to fulfill the request.

```
181         require(otherTokenAmountUsd < totalBalanceUsd, "TradingCloseFacet:
Insufficient funds in the openTrade");
```

The `require()` statement checks if `otherTokenAmountUsd` is strictly below `totalBalanceUsd`. However, equal amounts also should be acceptable.

Recommendation

We recommend using `<=` instead of `<`.

VFB-01 | NO UPPER LIMITS FOR FEES

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/diamond/facets/VaultFacet.sol (base): 32 , 44	● Resolved

Description

There is no upper limit restricting parameter of `VaultFacet.addToken()`, potentially enabling even more than 100% of `feeBasisPoints`, `taxBasisPoints`.

Recommendation

We recommend setting an upper limit for fees.

CON-01 | TYPOS

Category	Severity	Location	Status
Coding Style	● Informational	contracts/diamond/facets/AlpManagerFacet.sol (base): 49 , 58 ; contracts/diamond/facets/LimitOrderFacet.sol (base): 47 ; contracts/diamond/facets/TradingCheckerFacet.sol (base): 35 , 202 ; contracts/diamond/facets/VaultFacet.sol (base): 47 ; contracts/diamond/libraries/LibLimitOrder.sol (base): 42 ; contracts/diamond/libraries/LibPriceFacade.sol (base): 137 ; contracts/diamond/libraries/LibVault.sol (base): 69 , 134 ; contracts/utills/Constants.sol (base): 20	● Resolved

Description

```
20      bytes32 constant STAKE_OPERATOR_ROLE = keccak256("STAKE_OPERATOR");
```

In Constants.sol all the other roles contain the "ROLE" word, like "TOKEN_OPERATOR_ROLE". For consistency, we recommend updating the `STAKE_OPERATOR_ROLE` string literal and hash.

```
69      require(vs.wbnb == address(0), "LibAlpManager: Already initialized");
```

In `LibVault.initialize()` `LibAlpManager` is mentioned.

```
134     function updateAsMagin(address tokenAddress, bool asMagin) internal {
```

In `LibVault.updateAsMagin()` the word `Margin` is written as `Magin`.

```
137         // and wait for the next price to be feed.
```

"feed" is supposed to be "fed".

```
58      require(alpAmount >= minAlp, "LibLiquidity: insufficient ALP output");
```

"LibLiquidity" is supposed to be "AlpManagerFacet".

"LimitBookFacet" is supposed to be "LimitOrderFacet".

```
35         // stopLoss price below the liquidation price is meaningless
```

When the order is not `isLong`, the `stopLoss` price **above** the liquidation price is meaningless.

```
202          // Comparison of the values of price and limitPrice + slippege
```

"slippege" is supposed to be "slippage".

Recommendation

We recommend fixing the typos.

CON-02 | REDUNDANT CODE

Category	Severity	Location	Status
Coding Style	● Informational	contracts/diamond/ApolloX.sol (base): <u>41~46</u> ; contracts/utills/Constants.sol (base): <u>4~6</u>	● Resolved

Description

```
41     LibDiamond.DiamondStorage storage ds;  
42     bytes32 position = LibDiamond.DIAMOND_STORAGE_POSITION;  
43     // get diamond storage  
44     assembly {  
45         ds.slot := position  
46     }
```

The code in `ApolloX.fallback()` reimplements the `LibDiamond.diamondStorage()`. Can be rewritten as `LibDiamond.DiamondStorage storage ds = LibDiamond.diamondStorage();`.

```
4 type Price8 is uint64;  
5 type Qty10 is uint80;  
6 type Usd18 is uint96;
```

The types and constants `PRICE_DECIMALS` - `FUNDING_FEE_RATE_DIVISOR` from `Constants` library are never used.

Recommendation

We recommend following the recommendations.

DIA-03 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/diamond/facets/ApxRewardFacet.sol (base): 28 ; contracts/diamond/facets/StakeRewardFacet.sol (base): 32 , 37 ; contracts/diamond/libraries/LibApxReward.sol (base): 105 , 124 , 148 , 149 ; contracts/diamond/libraries/LibStakeReward.sol (base): 63 , 66 , 78 , 79	● Resolved

Description

When transferring deflationary ERC20 tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrived to the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Recommendation

We recommend carefully managing the token list supported by the project and avoiding adding deflationary tokens.

Alleviation

[Project Team]: As the external contracts are added by accounts with permissions, only known contracts like USDT/USDC/WBNB are added.

LAM-01 | TIME UNITS CAN BE USED

Category	Severity	Location	Status
Magic Numbers	● Informational	contracts/diamond/libraries/LibAlpManager.sol (base): <u>35</u>	● Resolved

Description

```
34      // default 30 minutes
35      ams.coolingDuration = 1800;
```

Time unit `minutes` can be used.

Recommendation

We recommend using `30 minutes` instead of `1800` and removing the comment.

LAM-03 `coolingDuration` CAN BE AVOIDED BY WHITELISTED ALP OWNERS

Category	Severity	Location	Status
Volatile Code	● Informational	contracts/diamond/libraries/LibAlpManager.sol (base): <u>15~16</u>	● Resolved

Description

`LibAlpManager` manages `lastMintedAt` parameter of each user and doesn't allow to `burnAlp()` / `burnAlpBNB()` if `coolingDuration` has not yet expired since last mint.

However, members of `ALP.fromWhiteList` and `ALP.toWhiteList` can avoid that limitation and burn immediately by transferring of minted ALP to another address.

Recommendation

We recommend adding to whitelists only the accounts that are not supposed to burn ALP.

Alleviation

[Project Team]: Currently only one contract address, ApolloX, has been added to the whitelist of ALP contracts.

LIB-01 | BASIS POINT VALUES ARE REFERRED AS PERCENT

Category	Severity	Location	Status
Inconsistency	● Informational	contracts/diamond/libraries/LibBrokerManager.sol (base): <u>20</u> ; contracts/diamond/libraries/LibFeeManager.sol (base): <u>20~21</u> , <u>33</u> ; contracts/diamond/libraries/LibPriceFacade.sol (base): <u>34~35</u> , <u>126</u> ; contracts/diamond/libraries/LibVault.sol (base): <u>42</u>	● Resolved

Description

```
42      uint16 securityMarginP;    // %
```

Many values hold basis points (1.0 is represented as 10000), however, they commented as `%` and have the `P` suffix in their names.

```
126     uint gapPercentage = priceGap * 1e4 / beforePrice;
```

Using the word "percentage" for the value in basis points is incorrect. The "percentage" refers to a value out of 100, while basis points refer to a value out of 10000

Recommendation

We recommend updating the comments to "// basis points" to avoid ambiguity and replacing `P` suffix with `BPS`. We recommend renaming `LibVault.AvailableToken.weight` to `weightBPS`, etc.

Alleviation

Comments were updated.

LVB-02 | REDUNDANT USAGE OF LibVault NAMESPACE

Category	Severity	Location	Status
Coding Style	● Informational	contracts/diamond/libraries/LibVault.sol (base): 249	● Resolved

Description

```
249      LibVault.VaultStorage storage vs = LibVault.vaultStorage();
```

In `LibVault` library it is not required to mention `LibVault` namespace to access own structures and methods.

Recommendation

We recommend omitting of `LibVault` namespace wherever possible. Like this:

```
249      VaultStorage storage vs = vaultStorage();
```

OPTIMIZATIONS | APOLLOX - AUDIT 2

ID	Title	Category	Severity	Status
DIA-01	Tautology	Gas Optimization	Optimization	● Resolved
DIA-02	Arguments Should Be <code>calldata</code>	Gas Optimization	Optimization	● Resolved
FAC-03	<code>_check()</code> Argument Can Be Declared <code>storage</code>	Gas Optimization	Optimization	● Resolved
LAC-01	Redundant Data In <code>LibAccessControlEnumerable</code>	Gas Optimization	Optimization	● Acknowledged
LIB-02	Unnecessary Use Of SafeMath	Gas Optimization	Optimization	● Resolved
LIB-03	<code>memory</code> Variable Can Be Used Instead Of <code>storage</code>	Gas Optimization	Optimization	● Resolved
OAT-01	<code>OrderAndTradeHistoryFacet.getOrderAndTradeHistory()</code> Is Gas Consuming	Gas Optimization	Optimization	● Resolved
TRA-02	<code>TradingCloseFacet._transferToUserForClose()</code> Can Be Optimized	Coding Style	Optimization	● Resolved

DIA-01 | TAUTOLOGY

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/diamond/facets/ApxRewardFacet.sol (base): 14 , 15 ; c ontracts/diamond/libraries/LibApxReward.sol (base): 155	● Resolved

Description

Comparisons that are always `true` are unnecessary.

```
14     require(_apxPerBlock >= 0, "Invalid _apxPerBlock");
15     require(_startBlock >= 0, "Invalid _startBlock");
```

```
155     require(_apxPerBlock >= 0, "apxPerBlock greater than 0");
```

Recommendation

We recommend clarifying the intended behavior (if zero values are expected or not) and either removing `require()` or using strict comparisons (`>`). We recommend updating the error messages to reflect the expected conditions.

DIA-02 | ARGUMENTS SHOULD BE `calldata`

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/diamond/facets/OrderAndTradeHistoryFacet.sol (base): 19 ; contracts/diamond/facets/PairsManagerFacet.sol (base): 61~63 , 119 ; contracts/diamond/facets/TradingCheckerFacet.sol (base): 226 , 424 ; contracts/diamond/facets/VaultFacet.sol (base): 28 , 35 , 53 ; contracts/diamond/libraries/LibVault.sol (base): 79	● Resolved



Description

Non changed arguments of external functions are declared as `memory`.

Recommendation

We recommend declaring the non changed arguments of external functions as `calldata` to save gas.

FAC-03 | `_check()` ARGUMENT CAN BE DECLARED `storage`

Category	Severity	Location	Status
Gas Optimization	 Optimization	contracts/diamond/facets/TradingCloseFacet.sol (base): <u>384</u> ; c ontracts/diamond/facets/TradingPortalFacet.sol (base): <u>20</u>	 Resolved

Description

`TradingPortalFacet._check()` accepts `memory ot` argument. All the function callers provide `storage` data structure.

`TradingCloseFacet._removeOpenTrade()` is also affected.

Recommendation

We recommend declaring `ot` argument as `storage` to avoid redundant copying.

LAC-01 | REDUNDANT DATA IN LibAccessControlEnumerable

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/diamond/libraries/LibAccessControlEnumerabl e.sol (base): <u>60-64</u>	● Acknowledged

Description

```
60     if (!hasRole(role, account)) {  
61         acs.roles[role].members[account] = true;  
62         emit RoleGranted(role, account, msg.sender);  
63     }  
64     acs.roleMembers[role].add(account);
```

`acs.roleMembers` can be updated only if `!hasRole(role, account)` (account doesn't have the role already).

`RoleData.members` and `RoleData` structure in general are redundant. `roleMembers` uses `EnumerableSet.AddressSet` to store members of role in an enumerable way. As a result, holding members as part of `roles` structure is not required.

Recommendation

We recommend replacing `mapping(bytes32 => RoleData) roles` structure with `mapping(bytes32 => bytes32) roleAdmins`. We recommend using `acs.roleMembers[role].contains(account)` in `hasRole()`.

Alleviation

[Project Team]: This contract is already running online, and modifying the data storage layout may cause unforeseen problems. These are the contracts we have already deployed:

<https://louper.dev/diamond/0x1b6F2d3844C6ae7D56ceb3C3643b9060ba28FEb0?network=binance>

LIB-02 | UNNECESSARY USE OF SAFEMATH

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/diamond/libraries/LibApxReward.sol (base): 173 , 174 , 175 ; contracts/diamond/libraries/LibStakeReward.sol (base): 6 , 4 , 65 , 76 , 77	● Resolved

Description

With Solidity compiler versions 0.8.0 or newer, arithmetic operations will automatically revert in case of integer overflow or underflow. `SafeMath` library is used for `uint256` type in `LibApxReward` and `LibStakeReward` contracts.

Recommendation

We recommend removing the usage of `SafeMath` library and using the built-in arithmetic operations provided by the Solidity programming language.

LIB-03 | `memory` VARIABLE CAN BE USED INSTEAD OF `storage`

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/diamond/libraries/LibChainlinkPrice.sol (base): <u>45~46</u> ; contracts/diamond/libraries/LibVault.sol (base): <u>263~264</u>	● Resolved

Description

```
261         address tokenAddress = vs.tokenAddresses[i.into()];
262         LibVault.AvailableToken storage at = vs.tokens[tokenAddress];
263         uint256 price = LibPriceFacade.getPrice(at.tokenAddress);
264         uint256 balance = vs.treasury[at.tokenAddress];
```

In `getTotalValueUsd()` `tokenAddress` variable can be used instead of `at.tokenAddress` storage field to save gas.

```
45         address priceFeed = pf.feedAddress;
46         require(pf.feedAddress != address(0), "LibChainlinkPrice: Price feed
does not exist");
```

In `removeChainlinkPriceFeed()` `priceFeed` variable can be used instead of `pf.feedAddress` storage field to save gas.

Recommendation

We recommend using `memory` variables instead of `storage` fields.

OAT-01 | OrderAndTradeHistoryFacet.getOrderAndTradeHistory() IS GAS CONSUMING

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/diamond/facets/OrderAndTradeHistoryFacet.sol (base): 64	● Resolved

Description

`OrderAndTradeHistoryFacet.getOrderAndTradeHistory()` is an `external` `view` function. `view` functions can be limited by the amount of computational resources available on a particular node. If a `view` function is particularly resource-intensive, it may cause nodes to become overwhelmed and unable to execute it.

```
64      ActionInfo[] memory infos = hs.actionInfos[user];
```

The function copies all the `hs.actionInfos[user]` array from `storage` into `memory`. The array can be extremely big and copying can be expensive in terms of gas.

Recommendation

We recommend omitting the copying of the whole array and accessing the `hs.actionInfos[user]` elements directly:

```
71      UC oldest = uc(hs.actionInfos[user].length - start - 1);
72      ...
73      ActionInfo memory ai = hs.actionInfos[user][(oldest -
ai).into()];
```

TRA-02 | TradingCloseFacet._transferToUserForClose() CAN BE OPTIMIZED

Category	Severity	Location	Status
Coding Style	● Optimization	contracts/diamond/facets/TradingCloseFacet.sol (base): <u>221~224</u>	● Resolved

Description

```
221         if (userReceive > 0) {
222             _closeTradeReceived(tradeHash, to, settleTokens[0].token,
userReceive);
223         }
224         settleTokens[0].amount -= userReceive;
```

`settleTokens[0].amount` can be updated only if `userReceive > 0`.

The function contains code repetitions and can be refactored.

It is recommended to check at line 267 that

```
267     require(userReceiveUsd == 0, "TradingCloseFacet: Insufficient funds in the
openTrade");
```

Recommendation

We recommend performing function refactoring.

FORMAL VERIFICATION | APOLLOX - AUDIT 2

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied automated formal verification (symbolic model checking) to prove that well-known functions in the smart contracts adhere to their expected behavior.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-transfer-revert-zero	<code>transfer</code> Prevents Transfers to the Zero Address
erc20-transfer-correct-amount	<code>transfer</code> Transfers the Correct Amount in Non-self Transfers
erc20-transfer-correct-amount-self	<code>transfer</code> Transfers the Correct Amount in Self Transfers
erc20-transfer-change-state	<code>transfer</code> Has No Unexpected State Changes
erc20-transfer-false	If <code>transfer</code> Returns <code>false</code> , the Contract State Is Not Changed
erc20-transfer-exceed-balance	<code>transfer</code> Fails if Requested Amount Exceeds Available Balance
erc20-transfer-never-return-false	<code>transfer</code> Never Returns <code>false</code>
erc20-transferfrom-revert-from-zero	<code>transferFrom</code> Fails for Transfers From the Zero Address
erc20-transferfrom-revert-to-zero	<code>transferFrom</code> Fails for Transfers To the Zero Address
erc20-transferfrom-correct-amount	<code>transferFrom</code> Transfers the Correct Amount in Non-self Transfers

Property Name	Title
erc20-transferfrom-correct-amount-self	<code>transferFrom</code> Performs Self Transfers Correctly
erc20-transferfrom-correct-allowance	<code>transferFrom</code> Updated the Allowance Correctly
erc20-transferfrom-fail-exceed-allowance	<code>transferFrom</code> Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-fail-exceed-balance	<code>transferFrom</code> Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-change-state	<code>transferFrom</code> Has No Unexpected State Changes
erc20-totalsupply-succeed-always	<code>totalSupply</code> Always Succeeds
erc20-transferfrom-false	If <code>transferFrom</code> Returns <code>false</code> , the Contract's State Is Unchanged
erc20-transferfrom-never-return-false	<code>transferFrom</code> Never Returns <code>false</code>
erc20-totalsupply-correct-value	<code>totalSupply</code> Returns the Value of the Corresponding State Variable
erc20-totalsupply-change-state	<code>totalSupply</code> Does Not Change the Contract's State
erc20-balanceof-succeed-always	<code>balanceOf</code> Always Succeeds
erc20-balanceof-correct-value	<code>balanceOf</code> Returns the Correct Value
erc20-balanceof-change-state	<code>balanceOf</code> Does Not Change the Contract's State
erc20-allowance-succeed-always	<code>allowance</code> Always Succeeds
erc20-allowance-correct-value	<code>allowance</code> Returns Correct Value
erc20-allowance-change-state	<code>allowance</code> Does Not Change the Contract's State
erc20-approve-revert-zero	<code>approve</code> Prevents Approvals For the Zero Address
erc20-approve-succeed-normal	<code>approve</code> Succeeds for Admissible Inputs
erc20-approve-correct-amount	<code>approve</code> Updates the Approval Mapping Correctly
erc20-approve-change-state	<code>approve</code> Has No Unexpected State Changes
erc20-approve-false	If <code>approve</code> Returns <code>false</code> , the Contract's State Is Unchanged
erc20-approve-never-return-false	<code>approve</code> Never Returns <code>false</code>

Property Name	Title	
erc20-transfer-succeed-normal	transfer	Succeeds on Admissible Non-self Transfers
erc20-transfer-succeed-self	transfer	Succeeds on Admissible Self Transfers
erc20-transfer-recipient-overflow	transfer	Prevents Overflows in the Recipient's Balance
erc20-transferfrom-succeed-normal	transferFrom	Succeeds on Admissible Non-self Transfers
erc20-transferfrom-succeed-self	transferFrom	Succeeds on Admissible Self Transfers
erc20-transferfrom-fail-recipient-overflow	transferFrom	Prevents Overflows in the Recipient's Balance

Verification Results

For the following contracts, model checking established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract ALP (contracts/ALP.sol) In Commit 1d4142c08a10b459c3625ceba84606135de3d2fd

Verification of ERC-20 Compliance

Detailed results for function transfer

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● True	
erc20-transfer-correct-amount	● True	
erc20-transfer-correct-amount-self	● True	
erc20-transfer-change-state	● True	
erc20-transfer-false	● True	
erc20-transfer-exceed-balance	● True	
erc20-transfer-never-return-false	● True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● True	
erc20-transferfrom-revert-to-zero	● True	
erc20-transferfrom-correct-amount	● True	
erc20-transferfrom-correct-amount-self	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-change-state	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-never-return-false	● True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

In the remainder of this section, we list all contracts where model checking of at least one property was not successful. There are several reasons why this could happen:

- Model checking reports a counterexample that violates the property. Depending on the counterexample, this occurs if
 - The specification of the property is too generic and does not accurately capture the intended behavior of the smart contract. In that case, the counterexample does not indicate a problem in the underlying smart contract. We report such instances as being "inapplicable".
 - The property is applicable to the smart contract. In that case, the counterexample showcases a problem in the smart contract and a corresponding finding is reported separately in the Findings section of this report. In the following tables, we report such instances as "invalid". The distinction between spurious and actual counterexamples is done manually by the auditors.
- The model checking result is inconclusive. Such a result does not indicate a problem in the underlying smart contract. An inconclusive result may occur if
 - The model checking engine fails to construct a proof. This can happen if the logical deductions necessary are beyond the capabilities of the automated reasoning tool. It is a technical limitation of all proof engines and cannot be avoided in general.

- The model checking engine runs out of time or memory and did not produce a result. This can happen if automatic abstraction techniques are ineffective or of the state space is too big.

Detailed Results For Contract ALP (contracts/ALP.sol) In Commit 32490e5cb13bf90af5cda621ae3464e77c250000

Verification of ERC-20 Compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● True	
erc20-transfer-correct-amount	● True	
erc20-transfer-succeed-normal	● Inapplicable	Not in scope
erc20-transfer-succeed-self	● Inapplicable	Not in scope
erc20-transfer-correct-amount-self	● True	
erc20-transfer-change-state	● True	
erc20-transfer-exceed-balance	● True	
erc20-transfer-false	● True	
erc20-transfer-never-return-false	● True	
erc20-transfer-recipient-overflow	● Inapplicable	Not in scope

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● True	
erc20-transferfrom-revert-to-zero	● True	
erc20-transferfrom-correct-amount	● True	
erc20-transferfrom-correct-amount-self	● True	
erc20-transferfrom-succeed-normal	● Inapplicable	Not in scope
erc20-transferfrom-succeed-self	● Inapplicable	Not in scope
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-change-state	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-never-return-false	● True	
erc20-transferfrom-fail-recipient-overflow	● Inapplicable	Not in scope

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-succeed-normal	● True	
erc20-approve-revert-zero	● True	
erc20-approve-correct-amount	● True	
erc20-approve-false	● True	
erc20-approve-change-state	● True	
erc20-approve-never-return-false	● True	

APPENDIX | APOLLOX - AUDIT 2

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Language Specific	Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.
Inconsistency	Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.
Magic Numbers	Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified using symbolic model checking. Each such contract was compiled into a mathematical model which reflects all its possible behaviors with respect to the property. The

model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

Technical Description

The model also formalizes a simplified execution environment of the Ethereum blockchain and a verification harness that performs the initialization of the contract and all possible interactions with the contract. Initially, the contract state is initialized non-deterministically (i.e. by arbitrary values) and over-approximates the reachable state space of the contract throughout any actual deployment on chain. All valid results thus carry over to the contract's behavior in arbitrary states after it has been deployed.

Assumptions and Simplifications

The following assumptions and simplifications apply to our model:

- Gas consumption is not taken into account, i.e. we assume that executions do not terminate prematurely because they run out of gas.
- The contract's state variables are non-deterministically initialized before invocation of any function. That ignores contract invariants and may lead to false positives. It is, however, a safe over-approximation.
- The verification engine reasons about unbounded integers. Machine arithmetic is modeled using modular arithmetic based on the bit-width of the underlying numeric Solidity type. This ensures that over- and underflow characteristics are faithfully represented.
- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for Property Specification

All properties are expressed in linear temporal logic (LTL). For that matter, we treat each invocation of and each return from a public or an external function as a discrete time step. Our analysis reasons about the contract's state upon entering and upon leaving public or external functions.

Apart from the Boolean connectives and the modal operators "always" (written \Box) and "eventually" (written \Diamond), we use the following predicates as atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `started(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond`.
- `willSucceed(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond` and considers only those executions that do not revert.
- `finished(f, [cond])` Indicates that execution returns from contract function `f` in a state satisfying formula `cond`. Here, formula `cond` may refer to the contract's state variables and to the value they had upon entering the function (using the `old` function).

- `reverted(f, [cond])` Indicates that execution of contract function `f` was interrupted by an exception in a contract state satisfying formula `cond`.

The verification performed in this audit operates on a harness that non-deterministically invokes a function of the contract's public or external interface. All formulas are analyzed w.r.t. the trace that corresponds to this function invocation.

Description of the Analyzed ERC-20 Properties

The specifications are designed such that they capture the desired and admissible behaviors of the ERC-20 functions `transfer`, `transferFrom`, `approve`, `allowance`, `balanceOf`, and `totalSupply`. In the following, we list those property specifications.

Properties related to function `transfer`

erc20-transfer-revert-zero

`transfer` Prevents Transfers to the Zero Address. Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address. Specification:

```
[(started(contract.transfer(to, value), to == address(0)) ==>
  <>(reverted(contract.transfer) || finished(contract.transfer(to, value), return
    == false)))
```

erc20-transfer-succeed-normal

`transfer` Succeeds on Admissible Non-self Transfers. All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender`,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call. Specification:

```
[(started(contract.transfer(to, value), to != address(0) && to != msg.sender &&
  value >= 0 && value <= _balances[msg.sender] && _balances[to] + value <
  0x10000000000000000000000000000000000000000000000000000000000000000 &&
  _balances[to] >= 0 && _balances[msg.sender] <
  0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transfer(to, value), return == true)))
```

erc20-transfer-succeed-self

`transfer` Succeeds on Admissible Self Transfers. All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and

- the supplied gas suffices to complete the call. Specification:

```

[](started(contract.transfer(to, value), to != address(0) && to == msg.sender &&
    value >= 0 && value <= _balances[msg.sender] && _balances[msg.sender] >= 0 &&
    _balances[msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
<>(finished(contract.transfer(to, value), return == true)))

```

erc20-transfer-correct-amount

`transfer` Transfers the Correct Amount in Non-self Transfers. All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address. Specification:

```

[](willSucceed(contract.transfer(to, value), to != msg.sender && _balances[to] >= 0
    && value >= 0 && _balances[to] + value <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[msg.sender] >= 0 && _balances[msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
<>(finished(contract.transfer(to, value), return == true ==>
    _balances[msg.sender] == old(_balances[msg.sender]) - value && _balances[to]
    == old(_balances[to]) + value)))

```

erc20-transfer-correct-amount-self

`transfer` Transfers the Correct Amount in Self Transfers. All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender`. Specification:

```

[](willSucceed(contract.transfer(to, value), to == msg.sender && _balances[to] >= 0
    && _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
<>(finished(contract.transfer(to, value), return == true ==> _balances[to] ==
    old(_balances[to]))))

```

erc20-transfer-change-state

`transfer` Has No Unexpected State Changes. All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses. Specification:

```

[](willSucceed(contract.transfer(to, value), p1 != msg.sender && p1 != to) ==>
    <>(finished(contract.transfer(to, value), return == true ==> (_totalSupply ==
        old(_totalSupply) && _allowances == old(_allowances) && _balances[p1] ==
        old(_balances[p1]) && other_state_variables ==
        old(other_state_variables)))))

```


erc20-transfer-exceed-balance

`transfer` Fails if Requested Amount Exceeds Available Balance. Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail. Specification:

```
[(started(contract.transfer(to, value), value > _balances[msg.sender] &&
  _balances[msg.sender] >= 0 && value <
  0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(reverted(contract.transfer) || finished(contract.transfer(to, value), return
    == false)))
```

erc20-transfer-recipient-overflow

`transfer` Prevents Overflows in the Recipient's Balance. Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow. Specification:

```
[(started(contract.transfer(to, value), to != msg.sender && _balances[to] + value
  >= 0x1000000000000000000000000000000000000000000000000000000000000000 &&
  _balances[to] >= 0 && _balances[to] <
  0x1000000000000000000000000000000000000000000000000000000000000000 &&
  _balances[msg.sender] <
  0x1000000000000000000000000000000000000000000000000000000000000000 && value >
  0 && value <= _balances[msg.sender]) ==> <>(reverted(contract.transfer) ||
  finished(contract.transfer(to, value), return == false) ||
  finished(contract.transfer(to, value), _balances[to] > old(_balances[to]) +
    value -
    0x1000000000000000000000000000000000000000000000000000000000000000))
```

erc20-transfer-false

If `transfer` Returns `false`, the Contract State Is Not Changed. If the `transfer` function in contract `contract` fails by returning `false`, it must undo all state changes it incurred before returning to the caller. Specification:

```
[(willSucceed(contract.transfer(to, value)) ==> <>(finished(contract.transfer(to,
  value), return == false ==> (_balances == old(_balances) && _totalSupply ==
  old(_totalSupply) && _allowances == old(_allowances) &&
  other_state_variables == old(other_state_variables))))
```

erc20-transfer-never-return-false

`transfer` Never Returns `false`. The transfer function must never return `false` to signal a failure. Specification:

```
[(!(finished(contract.transfer, return == false)))
```

Properties related to function `transferFrom`

erc20-transferfrom-revert-from-zero

`transferFrom` Fails for Transfers From the Zero Address. All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail. Specification:

```
[(started(contract.transferFrom(from, to, value), from == address(0)) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom, return ==
    false)))
```

erc20-transferfrom-revert-to-zero

`transferFrom` Fails for Transfers To the Zero Address. All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail. Specification:

```
[(started(contract.transferFrom(from, to, value), to == address(0)) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom, return ==
    false)))
```

erc20-transferfrom-succeed-normal

`transferFrom` Succeeds on Admissible Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call. Specification:

```
[(started(contract.transferFrom(from, to, value), from != address(0) && to !=
  address(0) && from != to && value <= _balances[from] && value <=
  _allowances[from][msg.sender] && _balances[to] + value <
  0x10000000000000000000000000000000000000000000000000000000000000000 && value >=
  0 && _balances[to] >= 0 && _balances[from] >= 0 && _balances[from] <
  0x10000000000000000000000000000000000000000000000000000000000000000 &&
  _allowances[from][msg.sender] >= 0 && _allowances[from][msg.sender] <
  0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true)))
```

erc20-transferfrom-succeed-self

`transferFrom` Succeeds on Admissible Self Transfers. All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`, and

- the supplied gas suffices to complete the call. Specification:

```

[](started(contract.transferFrom(from, to, value), from != address(0) && from == to
  && value <= _balances[from] && value <= _allowances[from][msg.sender] && value
  >= 0 && _balances[from] <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _allowances[from][msg.sender] <
      0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true)))

```

erc20-transferfrom-correct-amount

`transferFrom` Transfers the Correct Amount in Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`. Specification:

```

[](willSucceed(contract.transferFrom(from, to, value), from != to && value >= 0 &&
  _balances[from] >= 0 && _balances[from] <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] + value <
      0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true ==>
    _balances[from] == old(_balances[from]) - value && _balances[to] ==
      old(_balances[to] + value))))

```

erc20-transferfrom-correct-amount-self

`transferFrom` Performs Self Transfers Correctly. All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`). Specification:

```

[](willSucceed(contract.transferFrom(from, to, value), from == to && value >= 0 &&
  value < 0x10000000000000000000000000000000000000000000000000000000000000000 &&
  _balances[from] >= 0 && _balances[from] <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true ==>
    _balances[from] == old(_balances[from]))))

```

erc20-transferfrom-correct-allowance

`transferFrom` Updated the Allowance Correctly. All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`. Specification:

erc20-transferfrom-change-state

- The balance entry for the address in `dest` ,
- The balance entry for the address in `from` ,
- The allowance for the address in `msg.sender` for the address in `from` . Specification:

erc20-transferfrom-fail-exceed-balance

erc20-transferfrom-fail-exceed-allowance

`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance. Any call of the form `transferFrom(from,`

```

[](started(contract.transferFrom(from, to, value), msg.sender != from && value >
    _allowances[from][msg.sender] && _allowances[from][msg.sender] >= 0 && value <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
    <>(reverted(contract.transferFrom) || finished(contract.transferFrom(from, to,
        value), return == false)))

```

`transferFrom` Prevents Overflows in the Recipient's Balance. Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail. Specification:

[illegible]

If `transferFrom` Returns `false`, the Contract's State Is Unchanged. If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller. Specification:

```
[](willSucceed(contract.transferFrom(from, to, value)) ==>
  <>(finished(contract.transferFrom(from, to, value), return == false ==>
    (_balances == old(_balances) && _totalSupply == old(_totalSupply) &&
      _allowances == old(_allowances) && other_state_variables ==
        old(other_state_variables))))))
```

`transferFrom` Never Returns `false`. The `transferFrom` function must never return `false`. Specification:

```
[!(!finished(contract.transferFrom, return == false)))
```

`totalSupply` Always Succeeds. The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas. Specification:

```
[(started(contract.totalSupply) ==> <>(finished(contract.totalSupply)))
```

erc20-totalsupply-correct-value

`totalSupply` Returns the Value of the Corresponding State Variable. The `totalSupply` function must return the value that is held in the corresponding state variable of contract `contract`. Specification:

```
[(willSucceed(contract.totalSupply) ==> <>(finished(contract.totalSupply, return
  == _totalSupply)))
```

erc20-totalsupply-change-state

`totalSupply` Does Not Change the Contract's State. The `totalSupply` function in contract `contract` must not change any state variables. Specification:

```
[(willSucceed(contract.totalSupply) ==> <>(finished(contract.totalSupply,
  _totalSupply == old(_totalSupply) && _balances == old(_balances) &&
  _allowances == old(_allowances) && other_state_variables ==
  old(other_state_variables)))
```

Properties related to function `balanceOf`

erc20-balanceof-succeed-always

`balanceOf` Always Succeeds. Function `balanceOf` must always succeed if it does not run out of gas. Specification:

```
[(started(contract.balanceOf) ==> <>(finished(contract.balanceOf)))
```

erc20-balanceof-correct-value

`balanceOf` Returns the Correct Value. Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`. Specification:

```
[(willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf(owner),
  return == _balances[owner]))
```

erc20-balanceof-change-state

`balanceOf` Does Not Change the Contract's State. Function `balanceOf` must not change any of the contract's state variables. Specification:

```
[(willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf(owner),
  _totalSupply == old(_totalSupply) && _balances == old(_balances) &&
  _allowances == old(_allowances) && other_state_variables ==
  old(other_state_variables)))
```

Properties related to function `allowance`

erc20-allowance-succeed-always

`allowance` Always Succeeds. Function `allowance` must always succeed, assuming that its execution does not run out of gas. Specification:

```
[(started(contract.allowance) ==> <>(finished(contract.allowance)))
```

erc20-allowance-correct-value

`allowance` Returns Correct Value. Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`. Specification:

```
[(willSucceed(contract.allowance(owner, spender)) ==>  
  <>(finished(contract.allowance(owner, spender), return ==  
    _allowances[owner][spender]))]
```

erc20-allowance-change-state

`allowance` Does Not Change the Contract's State. Function `allowance` must not change any of the contract's state variables. Specification:

```
[(willSucceed(contract.allowance(owner, spender)) ==>  
  <>(finished(contract.allowance(owner, spender), _totalSupply == old(_totalSupply)  
    && _balances == old(_balances) && _allowances == old(_allowances) &&  
    other_state_variables == old(other_state_variables)))]
```

Properties related to function `approve`

erc20-approve-revert-zero

`approve` Prevents Approvals For the Zero Address. All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address. Specification:

```
[(started(contract.approve(spender, value), spender == address(0)) ==>  
  <>(reverted(contract.approve) || finished(contract.approve(spender, value),  
    return == false))]
```

erc20-approve-succeed-normal

`approve` Succeeds for Admissible Inputs. All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas. Specification:

```

[](started(contract.approve(spender, value), spender != address(0)) ==>
  <>(finished(contract.approve(spender, value), return == true)))

```

erc20-approve-correct-amount

`approve` Updates the Approval Mapping Correctly. All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`. Specification:

```

[](willSucceed(contract.approve(spender, value), spender != address(0) && value >=
  0 && value <
  0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.approve(spender, value), return == true ==>
    _allowances[msg.sender][spender] == value)))

```

erc20-approve-change-state

`approve` Has No Unexpected State Changes. All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes. Specification:

```

[](willSucceed(contract.approve(spender, value), spender != address(0) && (p1 !=
  msg.sender || p2 != spender)) ==> <>(finished(contract.approve(spender,
  value), return == true ==> _totalSupply == old(_totalSupply) && _balances
  == old(_balances) && _allowances[p1][p2] == old(_allowances[p1][p2]) &&
  other_state_variables == old(other_state_variables))))

```

erc20-approve-false

If `approve` Returns `false`, the Contract's State Is Unchanged. If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller. Specification:

```

[](willSucceed(contract.approve(spender, value)) ==>
  <>(finished(contract.approve(spender, value), return == false ==> (_balances ==
    old(_balances) && _totalSupply == old(_totalSupply) && _allowances ==
    old(_allowances) && other_state_variables == old(other_state_variables)))))

```

erc20-approve-never-return-false

`approve` Never Returns `false`. The function `approve` must never returns `false`. Specification:

```

[](!(finished(contract.approve, return == false)))

```


DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.



