Security Review - Report
NM-0056: ZKX Protocol (Layer 1 and Layer 2)

NETHERMIND

(Sep 28, 2022)

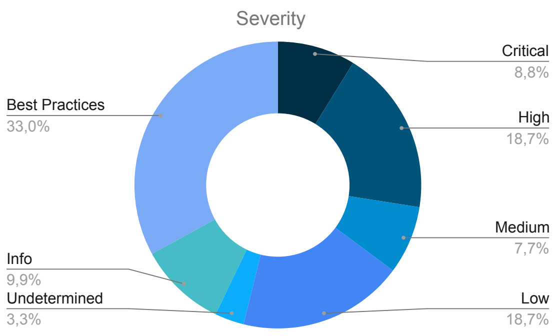
Contents

1 Executive Summary

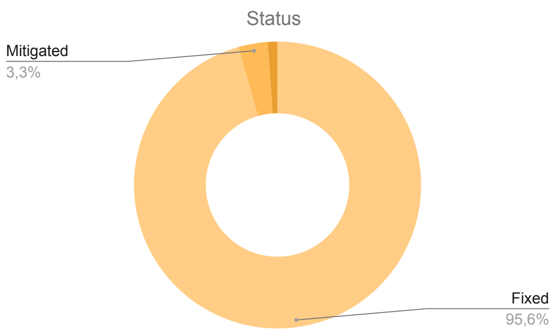
This document presents the security review performed by Nethermind on the Layer 1 and Layer 2 contracts of the ZKX Protocol written in the Solidity and Cairo languages. ZKX is a permissionless protocol for derivatives built on StarkNet, StarkWare's ZK rollup. The provided audit code is currently under development, with the ZKX team primarily focusing on the L2 implementation. The L1 implementation mostly serves as a proof-of-concept for the L2 functionalities. Since the protocol is under development, this audit should be treated as an assessment of the current state of the ZKX protocol implementation to guide further development, and not as a deployment readiness audit. **During the initial audit**, we raised issues related to best practices, and most of this audit effort was taken on performing low level validations of the code base. However, the audit team recommends a second pass on the code base for checking high level issues and more complex exploits.

During the initial audit, we have raised 94 points of attention. **During the reaudit**, the ZKX team has fixed 87 issues and identified three false-positive issues. One issue has been acknowledge, while 3 issues have been mitigated. **The codebase is composed of** 261 lines of Solidity code and 6,500 lines of Cairo code. There are no integration tests between L1 (Solidity) and L2 (Cairo). The ZKX protocol also depends on the ZKX-Nodes to be fully functional. The source code of the ZKX-Nodes are out of the scope of this audit, and they are assumed to function as expected.

The distribution of issues is summarized in the figure below. **This document is organized as follows:** Section 2 presents file in the scope of this audit. Section 3 summarizes the findings in a table. Section 4 discusses the risk rating methodology adopted for this audit. Section 5 details each finding. Section 6 discusses the documentation provided for this audit. Section 7 presents the output of the automated test suite. Section 8 concludes the audit report.



(a)



(b)

Distribution of Issues: Critical (8), **High** (17), **Medium** (7), **Low** (17), **Undetermined** (3), **Informational** (9), **Best Practices** (30).
Distribution of Status: Fixed (87), **Acknowledged** (1), **Mitigated** (3), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Aug. 22, 2022
Response from Client	Sep. 13, 2022
Final Report	Sep. 28, 2022
Methods	Manual Review, Automated Analysis
Repository	ZKX Protocol
Commit Hash (Initial Audit)	7a5c7533d22be9e4eb943caa7df4ffe23bb0d3bc
Commit Hash (Final Audit)	https://github.com/zkxteam/zkxprotocol/tree/audit-fixes-L2
Documentation	Confidential document composed of 101 pages
Documentation Assessment	High
Unit Tests Assessment	High
L1-L2 Integration Tests	Nonexistent

2 Contracts

	L1 Contracts (Solidity)	Lines of Code	Lines of Comments	Comments Ratio	Blank Lines	Total Lines
1	L1/contracts/L1ZKXContract.sol	242	108	44.6%	54	404
2	L1/contracts/Constants.sol	8	4	50.0%	5	17
3	L1/contracts/IStarknetCore.sol	11	11	100.0%	2	24
	Total	**261**	**123**	**47.1%**	**61**	**445**

	L2 Contracts (Cairo)	Lines of Code	Lines of Comments	Comments Ratio	Blank Lines	Total Lines
1	L2/interfaces/ITradingFeesUser.cairo	21	0	0.0%	4	25
2	L2/interfaces/ITradingFeesAdmin.cairo	19	0	0.0%	2	21
3	L2/contracts/Trading.cairo	578	99	17.1%	100	777
4	L2/contracts/AccountDeployer.cairo	117	5	4.3%	45	167
5	L2/contracts/CallFeeBalance.cairo	31	7	22.6%	6	44
6	L2/contracts/Markets.cairo	208	43	20.7%	39	290
7	L2/contracts/Math_64x61.cairo	242	38	15.7%	49	329
8	L2/contracts/ArrayTesting.cairo	91	0	0.0%	21	112
9	L2/contracts/AdminAuth.cairo	45	25	55.6%	6	76
10	L2/contracts/InsuranceFund.cairo	167	31	18.6%	36	234
11	L2/contracts/WithdrawalFeeBalance.cairo	100	45	45.0%	27	172
12	L2/contracts/ABRFund.cairo	119	25	21.0%	30	174
13	L2/contracts/AuthorizedRegistry.cairo	55	28	50.9%	10	93
14	L2/contracts/Liquidate.cairo	668	145	21.7%	87	900
15	L2/contracts/Holding.cairo	145	27	18.6%	34	206
16	L2/contracts/ABR.cairo	535	162	30.3%	100	797
17	L2/contracts/Asset.cairo	281	80	28.5%	54	415
18	L2/contracts/LiquidityFund.cairo	167	34	20.4%	38	239
19	L2/contracts/ABRPayment.cairo	220	36	16.4%	30	286
20	L2/contracts/DataTypes.cairo	183	28	15.3%	19	230
21	L2/contracts/Account.cairo	1000	223	22.3%	200	1423
22	L2/contracts/FeeDiscount.cairo	33	12	36.4%	5	50
23	L2/contracts/MarketPrices.cairo	116	38	32.8%	29	183
24	L2/contracts/WithdrawalRequest.cairo	108	43	39.8%	24	175
25	L2/contracts/EmergencyFund.cairo	229	42	18.3%	63	334
26	L2/contracts/TradingFees.cairo	231	54	23.4%	33	318
27	L2/contracts/Constants.cairo	43	1	2.3%	4	48
28	L2/contracts/FeeBalance.cairo	63	16	25.4%	15	94
29	L2/contracts/AccountRegistry.cairo	124	29	23.4%	22	175
30	L2/contracts/interfaces/IMarkets.cairo	19	2	10.5%	8	29
31	L2/contracts/interfaces/IAuthorizedRegistry.cairo	6	0	0.0%	1	7
32	L2/contracts/interfaces/IQuoteL1Fee.cairo	16	2	12.5%	3	21
33	L2/contracts/interfaces/IEmergencyFund.cairo	22	2	9.1%	15	39
34	L2/contracts/interfaces/IAdminAuth.cairo	6	0	0.0%	1	7
35	L2/contracts/interfaces/ILiquidityFund.cairo	16	2	12.5%	8	26
36	L2/contracts/interfaces/IWithdrawalRequest.cairo	11	2	18.2%	5	18
37	L2/contracts/interfaces/IFeeBalance.cairo	10	2	20.0%	6	18
38	L2/contracts/interfaces/ILiquidate.cairo	10	0	0.0%	5	15
39	L2/contracts/interfaces/IInsuranceFund.cairo	16	2	12.5%	8	26
40	L2/contracts/interfaces/IWithdrawalFeeBalance.cairo	12	2	16.7%	7	21
41	L2/contracts/interfaces/IABRPayment.cairo	6	1	16.7%	3	10
42	L2/contracts/interfaces/IAsset.cairo	44	2	4.5%	14	60
43	L2/contracts/interfaces/IHolding.cairo	14	2	14.3%	7	23
44	L2/contracts/interfaces/IMarketPrices.cairo	13	2	15.4%	5	20
45	L2/contracts/interfaces/IABR.cairo	14	0	0.0%	2	16
46	L2/contracts/interfaces/IAccount.cairo	40	0	0.0%	14	54
47	L2/contracts/interfaces/ITrading.cairo	15	2	13.3%	6	23
48	L2/contracts/interfaces/ITradingFees.cairo	23	2	8.7%	14	39
49	L2/contracts/interfaces/IFeeDiscount.cairo	10	2	20.0%	4	16
50	L2/contracts/interfaces/IABRFund.cairo	8	0	0.0%	2	10
51	L2/contracts/interfaces/IAccountRegistry.cairo	12	2	16.7%	4	18
52	L2/contracts/libraries/Utils.cairo	21	3	14.3%	7	31
53	L2/contracts/libraries/RelayLibrary.cairo	197	27	13.7%	53	277
	Total	**6500**	**1377**	**21.2%**	**1334**	**9211**

3 Summary of Issues

	Finding	Severity	Update
1	A zero address account can be added to the account registry	Critical	Fixed
2	Any L1 user can steal another users' funds deposited on L2	Critical	Fixed
3	Anyone can call the functions `add_user_tokens(...)` and `remove_user_tokens(...)` and change the users' balance	Critical	Fixed
4	Datatype size to store token balance is not sufficient	Critical	Fixed
5	Missing input validation in function `add_withdrawal_request(...)`	Critical	Fixed
6	No access control in function `pay_abr(...)`	Critical	Mitigated
7	Testing between L1 and L2 is required	Critical	Fixed
8	User can lose funds on deposit to incorrect L2 address	Critical	Fixed
9	Assets can be overwritten in `addAsset(...)`	High	Fixed
10	Assets can have matching tickers in `addAsset(...)`	High	Fixed
11	Data integrity of fee mappings can be compromised	High	Fixed
12	Events not used in protocol	High	Fixed
13	Existing asset can be added multiple times	High	Fixed
14	Fee and discount tiers may have empty entries	High	Fixed
15	Function `modify_core_settings(...)` can lead to fund losses	High	Fixed
16	Function `update_market_price(...)` not checking for market existence	High	Fixed
17	Market prices can be set to arbitrary values (including zero and negative values)	High	Fixed
18	Missing validation on first entry in fee arrays	High	Fixed
19	Non-existing asset can be removed	High	Fixed
20	Not checking the array bounds in function `populate_account_registry(...)`	High	Fixed
21	Proposed administrator can call admin-only functions	High	Fixed
22	Standard collateral can be set negative or zero	High	Fixed
23	Weak validation of L2 addresses	High	Fixed
24	`L1 to L2` message cancellation is not supported	High	Fixed
25	`node_operator_L2_address` can changed by malicious users	High	Mitigated
26	Admins cannot be added when admin count is one	Medium	Fixed
27	Fee and discount tiers aren't guaranteed to be ordered	Medium	Fixed
28	Missing input validation in `update_admin_mapping(...)`	Medium	Fixed
29	Missing use of `SafeERC20` wrapper	Medium	Fixed
30	Missing validations in core functions of fund related contracts	Medium	Fixed
31	Removing an asset does not unset its token contract address	Medium	Fixed
32	Risk of negative array length in function `remove_from_account_registry(...)`	Medium	Fixed
33	Administrator can cease all `L1 ERC-20` deposits and withdrawals	Low	Fixed
34	Administrator can transfer L1 funds at any time	Low	Fixed
35	Function `get_account_registry(...)` returns an unbounded array	Low	Fixed
36	Functions not validating the input parameters in `Math_64x61.cairo`	Low	Fixed
37	Looping over unbounded arrays	Low	Fixed
38	Missing `felt` input validation	Low	Fixed
39	Missing input validation in `addAsset(...)`	Low	Fixed
40	Missing overflow and negative input checks for `Math64x61` interpreted felts	Low	Fixed
41	Missing validations in function `add_user_tokens(...)`	Low	Fixed
42	No lower bound in the withdrawal fee in `set_standard_withdraw_fee(...)`	Low	Fixed
43	Off-by-one error in `Math64x61_assert64x61(...)`	Low	Fixed
44	Redundant storage of `L1_zkx_address`	Low	Fixed
45	Risk of fund losses in transferring to `address(0x0)`	Low	Fixed
46	Risk of markets having the same `id` in function `addMarket(...)`	Low	Fixed
47	Risk of removing markets with open positions	Low	Fixed
48	Two-stage public key change is needed	Low	Fixed
49	`Math64x61_ceil(...)` returning wrong output on whole numbers	Low	Fixed
50	Admin can be removed without approval	Undetermined	Fixed
51	Market functions can return removed markets	Undetermined	Fixed
52	Market prices can be set by anybody once TTL expires	Undetermined	Mitigated
53	Address comparison does not guarantee equality	Info	Fixed
54	Assets cannot be removed from `assets_array`	Info	Fixed
55	Function `find_std(...)` is actually computing the variance	Info	Fixed
56	Missing event emission	Info	Fixed
57	Possibility of modifying the leverage of removed and non-existent markets	Info	Fixed
58	Unnecessary `assert_not_zero` in `update_contract_registry(...)`	Info	Fixed
59	Unnecessary `namespace` declaration	Info	Fixed
60	Unused contract	Info	Fixed

	Finding	Severity	Update
61	Unused storage variable	Info	Fixed
62	Application is not using the StarkNet boolean library	Best Practices	Fixed
63	Consider implementing pause functionality	Best Practices	Acknowledged
64	Constant defined as zero	Best Practices	Fixed
65	Different versions of Solidity are used	Best Practices	Fixed
66	Functions with high lines of code	Best Practices	Fixed
67	Inconsistent code style	Best Practices	Fixed
68	Inconsistent contract structure	Best Practices	Fixed
69	Incorrect error message in `liquidate_position(...)`	Best Practices	Fixed
70	Incorrect error message in `withdraw(...)`	Best Practices	Fixed
71	Missing address input validation	Best Practices	Fixed
72	Missing error messages on assert fail	Best Practices	Fixed
73	Missing parameter in `NatSpec` comment	Best Practices	Fixed
74	Missing use of action constants	Best Practices	Fixed
75	Order status and types should be defined as constants	Best Practices	Fixed
76	Repeated caller authentication code	Best Practices	Fixed
77	Repeated code in fund contracts	Best Practices	Fixed
78	Replicated code in function `pay_abr_users_positions(...)`	Best Practices	Fixed
79	Test contracts should be in a separate directory	Best Practices	Fixed
80	Test function present in non-test contract	Best Practices	Fixed
81	Unindexed events	Best Practices	Fixed
82	Unnecessary assert	Best Practices	Fixed
83	Unnecessary balance check before Ether transfers	Best Practices	Fixed
84	Unnecessary check before `ERC-20` token transfer	Best Practices	Fixed
85	Unnecessary returns	Best Practices	Fixed
86	Unnecessary use of `alloc_locals`	Best Practices	Fixed
87	Unused `SafeMath` library	Best Practices	Fixed
88	Unused imports	Best Practices	Fixed
89	Unused variables in `is_valid_signature_order(...)`	Best Practices	Fixed
90	Using `AccessControl` instead of `Ownable`	Best Practices	Fixed
91	Withdrawal status values could be constants	Best Practices	Fixed

4 Risk Rating Methodology

The risk rating methodology used by Nethermind follows the principles established by the OWASP Foundation. The severity of each finding is determined two factors: **Likelihood** and **Impact**.

Likelihood is a measure of how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding other factors are also considered. These can include but are not limited to: Motive, opportunity, exploit accessibility, ease of discovery and ease of exploit.

Impact is a measure of the damage that may be caused if the finding were to be exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue can cause significant damage such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding other factors are also considered. These can include but are not limited: Data/state integrity, loss of availability, financial loss, reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

5 Issues

5.1 L1 General Findings

5.1.1 [Best Practices] Different versions of Solidity are used

File(s): `L1/contracts/*`

Description: The application uses the versions ['^0.8.0', '^0.8.7'], which may lead to compatibility issues.

Recommendation(s): Lock the Solidity version of all files to one version.

Status: Fixed

Update from the client: Solidity version was updated to 0.8.14 and made strict.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/81

5.2 `L1/contracts/Constants.sol`

5.2.1 [Best Practices] Constant defined as zero

File(s): `L1/contracts/Constants.sol`

Description: The file `Constant.sol` defines a set of constants used in the L1 contracts. These constants are shown below:

```
uint256 constant WITHDRAWAL_INDEX = 0;
uint256 constant ADD_ASSET_INDEX = 1;
uint256 constant REMOVE_ASSET_INDEX = 2;
```

The constant `WITHDRAWAL_INDEX` is defined as `zero`. Since Ethereum storage and EVM memory is initially `zero`, comparisons or operations involving the constant `WITHDRAWAL_INDEX` and uninitialized storage or memory may introduce unexpected behavior.

Recommendation(s): Consider defining the constant `WITHDRAWAL_INDEX` to a non-zero value.

Status: Fixed

Update from the client: Withdrawal index was changed to 3.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/82

5.3 `L1/contracts/IStarknetCore.sol`

5.3.1 [High] `L1 to L2` message cancellation is not supported

File(s): `L1/contracts/IStarknetCore.sol`

Description: The `IStarknetCore` interface doesn't allow for L1 -> L2 message cancellation. This may be needed when L2 message consumption is not functioning properly (EG: Bug in the ZKX L2 contracts or censorship from the StarkNet sequencer). If the message consumption mechanism is broken, this could result in the user losing custody over their asset forever as explained here.

Recommendation(s): Consider adding message cancellation support to the L1 component of the ZKX protocol.

Status: Fixed

Update from the client: We added support for cancellation of messages related to deposits.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/93

5.4 `L1/contracts/L1ZKXContract.sol`

5.4.1 [Critical] Any L1 user can steal another users' funds deposited on L2

File(s): `L1/contracts/L1ZKXContract.sol`

Description: When a user deposits funds with the functions `depositToL1(...)` or `depositEthToL1(...)`, any L2 account address can be passed as the `userL2Address_` parameter including the address of a ZKX account contract that is owned by another user. An attacker can call an L1 deposit function with the argument `userL2Address_` set to a victim L2 ZKX account contract address. When the ZKX account contract's `@l1_handler deposit(...)` function is called in Starknet, on L1300 the storage variable `L1_address` is overwritten with the attacker's L1 address. When the victim decides to withdraw, the message sent to L1 will contain the attacker's L1 address which will only allow the attacker to withdraw funds. Relevant code snippets are shown below:

```
@l1_handler
func deposit{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    from_address : felt, user : felt, amount : felt, assetID_ : felt
):
    ...
    # Update the L1 address
    L1_address.write(user)
    ...
end
```

Recommendation(s): Consider mitigating this attack by implementing a strict one to one relationship between L1 addresses and L2 ZKX contract accounts, removing the need for users to input a recipient L2 address.

Status: Fixed

Update from the client:

– L1-to-L2 relation was made strict ;
– user's L1 address is written to L2 account storage during deployment ;
– user's L1 address is immutable after deployment ;
– user's L2 account is single source of truth for user's L1 address ;

PR on L1 Link: https://github.com/zkxteam/zkxprotocol/pull/85 **PR on L2 Link**: https://github.com/zkxteam/zkxprotocol/pull/87

5.4.2 [Critical] User can lose funds on deposit to incorrect L2 address

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The functions `depositToL1(...)` and `depositToL2(...)` allow users to deposit `ERC-20` tokens and Ether for use on their L2 ZKX account contract. When depositing for the first time their recipient L2 address is stored and used for withdrawal logic. On a second call to a deposit function the user is not required to set the recipient L2 address to the same as the entry in the `l2ContractAddress` mapping. This means that the L1 deposit functions support one L1 address depositing to multiple L2 addresses. When a user wishes to withdraw, the L2 address to withdraw from is loaded from `l2ContractAddress` which contains the L2 address of the first deposit recipient, meaning that withdraw functions only support one withdrawing L1 address to one L2 address. Any deposits made by a user after their first deposit where the L2 recipient address does not match the entry in the mapping `l2ContractAddress` cannot be withdrawn and are lost. Relevant code snippets are shown below:

The deposit logic. Notice how `userL2Address` does not have to be equal to the entry in `l2ContractAddress` after the first call.

```
function depositEthToL1(uint256 userL2Address_)
    payable
    external
isValidL2Address(userL2Address_)
{
    // If not yet set, store L2 address linked to sender's L1 address
    uint256 senderAsUint256 = uint256(uint160(address(msg.sender)));
    if (l2ContractAddress[senderAsUint256] == 0) {
        l2ContractAddress[senderAsUint256] = userL2Address_;
    }

    ...
}
```

The withdrawal logic. Notice how `userL2Address` will always be the L2 recipient address from `msg.sender`'s deposit.'

```
function withdraw(
    uint256 userL1Address_,
    uint256 ticker_,
    uint256 amount_,
    uint256 requestId_
) external {
    require(msg.sender == address(uint160(userL1Address_)), "Sender is not withdrawal recipient");
    uint256 userL2Address = l2ContractAddress[userL1Address_];
    ...
}
```

The relationship constraint of one L1 address to one L2 address is not enforced, so unaware users may deposit to multiple L2 addresses and lose funds.

Recommendation(s): All protocol constraints should be enforced, rather than relying on user knowledge. Consider implementing a strict one to one relationship between L1 addresses and L2 ZKX contract accounts, removing the need for users to input a recipient L2 address.

Status: Fixed

Update from the client:

- L1 and L2 contracts were updated to fight this issue ;
- L2 Account deployment logic was changed: user's L1 address is required as constructor argument ;
- user's L1 address is written to storage during initialization and is never mutated after ;
- L1-to-L2 mapping completely removed from L1ZKXContract ;
- user's L2 address should be provided as an input parameter for `deposit` function ;
- there's a check in L2 Account contract message-handling function that prevents deposits from anyone except owner ;
- deposit message can't be consumed if sender's L1 address is not equal to owner's ;
- in case of wrong L2 address provided during deposit, message won't be consumed and user can cancel deposit later ;

PR Link: https://github.com/zkxteam/zkxprotocol/pull/85

5.4.3 [High] Existing asset can be added multiple times

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The function `updateAssetListInL1(...)` is used to update the `assetList` to reflect that asset that has been added on the L2 component of the protocol. There is no check to ensure that the to-be-added asset's ticker is already in use. If an already-existing ticker is added again its `assetID` entry will be overwritten with the new `assetId_` and the `assetList` storage array will contain two entries of the same ticker. This may cause unexpected behavior and would affect `removeAssetFromList(...)` as it only deletes the first ticker it finds, leaving the other duplicate ticker remaining with a blank `assetID` mapping entry.

Recommendation(s): When designing `L1 <-> L2` communication it is dangerous to assume that an incoming message from another layer is always correct. In the case of a smart contract bug on the other layer, unexpected messages may be sent. The function `updateAssetListInL1(...)` should not assume that the incoming `ADD_ASSET_INDEX` L2 message will always be correct. Consider adding a check to see if the to-be-added ticker already exists.

Status: Fixed

Update from the client: Fixed, before adding an asset there's a check that the asset doesn't exist yet.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/91

5.4.4 [High] Non-existing asset can be removed

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The function `removeAssetFromList(...)` is used to update the `assetList` to reflect that an asset has been removed on the L2 component of the protocol. There is no check to ensure that the to-be-removed asset exists before removal. If a non-existing asset is removed the `for` loop will cycle to the end of `assetList` and `index` will not have been set so it remains at zero. The asset at index zero is then replaced with the last entry in the `assetList` and then the final item in the array is removed with `assetList.pop()`. This means that removing an asset that does not exist on L1 will lead to the removal of the first asset in `assetList`. The relevant code is shown below:

```
// Remove the asset from the asset list
uint256 index;
for (uint256 i = 0; i < assetList.length; i++) {
    if (assetList[i] == ticker_) {
        index = i;
        break;
    }
}
assetList[index] = assetList[assetList.length - 1];
assetList.pop();
```

Recommendation(s): When designing L1 <-> L2 communication it is dangerous to assume that an incoming message from another layer is always correct. In the case of a smart contract bug on the other layer, unexpected messages may be sent. The function `removeAssetFromList(...)` should not assume that the incoming `REMOVE_ASSET_INDEX` L2 message will always be correct. Consider adding a check to see if the to-be-removed asset exists.

Status: Fixed

Update from the client: Asset-management logic was updated. Before any operation with an asset there's a check to ensure the asset exists.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/91

5.4.5 [High] Weak validation of L2 addresses

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The modifier `isValidL2Address(...)` is used to ensure a given address is within the range of a StarkNet address. This is a weak validation as any value that is within the range of a Cairo `felt` will pass this check. This opens the opportunity for attackers to pass malicious addresses (as shown in other findings) or for users to accidentally input an incorrect L2 address during deposits which can lead to lost funds. The modifier is shown below:

```
modifier isValidL2Address(uint256 l2Address_) {
    require(l2Address_ != 0 && l2Address_ < FIELD_PRIME, "L2_ADDRESS_OUT_OF_RANGE");
    _;
}
```

Recommendation(s): Consider implementing a mechanism that monitors and stores ZKX account contracts, and a modifier that reverts when a non-ZKX account contract address is passed is an argument.

Status: Fixed

Update from the client:

- deposit logic was improved ;
- if user deposits to wrong L2 address, message will never be consumed on L2 ;
- we've added support for deposit cancellation ;

PR Link: https://github.com/zkxteam/zkxprotocol/pull/93

5.4.6 [Medium] Missing use of `SafeERC20` wrapper

File(s): `L1/contracts/L1ZKXContract.sol`

Description: All `ERC-20` token transfers in the contract `L1ZKXContract` do not consider `ERC-20` tokens that do not comply with the `ERC-20` token standard, such as returning false on an unsuccessful transfer rather than reverting. This may lead to failing transfers being considered a success by the ZKX protocol.

Recommendation(s): Consider using the OpenZeppelin `SafeERC20` wrapper to handle non-compliant `ERC-20` tokens. Once `SafeERC20` is implemented, the manual balance checks before and after the transfers of tokens can be removed.

Status: Fixed

Update from the client: Started to use `SafeERC20` for token transfers.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/83

5.4.7 [Medium] Removing an asset does not unset its token contract address

File(s): `L1/contracts/L1ZKXContract.sol`

Description: When an asset is removed with the function `L1ZKXContract.removeAssetFromList(...)` the token contract address in the mapping `tokenContractAddress` is not unset for the removed `ticker_`. This may lead to unexpected behavior if a new asset is added with the same ticker as a previously deleted asset, where the new asset will have its token contract shared with the deleted asset. With the current implementation once an asset is removed with `removeAssetFromList(...)` the protocol administrator must then manually call `L1ZKXContract.setTokenContractAddress(...)` to clear the token contract address entry, but this is not guaranteed to happen.

Recommendation(s): When removing an asset from the protocol ensure that there is no leftover data related to the asset in contract storage.

Status: Fixed

Update from the client: Asset-management logic was updated. Data related to assets is now stored in `Asset` struct, which is completely removed on `removeAssetFromList()` call. Also tests for this logic were added.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/91

5.4.8 [Low] Administrator can cease all `L1` `ERC-20` deposits and withdrawals

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The function `L1ZKXContract.setTokenContractAddress(...)` has no safety or time delay features. The administrator can set the token contract address for any given ticker to `address(0x0)` at any time, which will revert ERC-20 token transfer functions used in `L1ZKXContract.depositToL1(...)` and `L1ZKXContract.withdraw(...)`. The admin function `L1ZKXContract.transferFunds(...)` will still work however.

Recommendation(s): This isn't necessarily a security issue, but it does pose potential risk to the protocol users. Consider ways to increase trust such as using a MultiSig wallet for the administrator address or adding features that allows users to protect themselves from a malicious or compromised admin address.

Status: Fixed

Update from the client: Contract management on L1 will be carried out through `MultisigAdmin` contract developed specifically for this purpose. All transactions must be confirmed by a required number of admins. Also it introduces timelock for transaction execution. It should improve protocol transparency and increase users trust.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/142

5.4.9 [Low] Administrator can transfer L1 funds at any time

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The functions `L1ZKXContract.transferFunds(...)` and `L1ZKXContract.transferEth(...)` have no safety or time delay features. The administrator can transfer funds at any time.

Recommendation(s): This poses potential risk to the protocol users. Consider ways to reduce risk such as using a MultiSig wallet for the administrator address or adding features that allows users to protect themselves from a malicious or compromised admin address. The transfer functions are shown below:

```
function transferFunds(address recipient_, uint256 amount_, address tokenAddress_)
    external
    onlyRole(DEFAULT_ADMIN_ROLE)
{
    uint256 balance = IERC20(tokenAddress_).balanceOf(address(this));
    require(amount_ <= balance, "Not enough ERC-20 tokens to withdraw");
    IERC20(tokenAddress_).transfer(recipient_, amount_);
}
```

```
function transferEth(address payable recipient_, uint256 amount_)
    external
    onlyRole(DEFAULT_ADMIN_ROLE)
{
    require(amount_ <= address(this).balance, "ETH to be transferred is more than the balance");
    recipient_.transfer(amount_);
}
```

Status: Fixed

Update from the client: This issue becomes less dangerous with introduction of `MultisigAdmin` contract usage for `L1ZKXContract` management. Transactions require multiple confirmations by admins to be executed and there's a delay between transaction confirmation and execution.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/142

5.4.10 [Low] Looping over unbounded arrays

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The storage array `assetList` contains a list of all assets supported by the protocol represented as a ticker. When adding assets to `assetList`, the code does not present any upper limit for the size of the array. The function `removeAssetFromList(...)` is used to remove assets and it uses a `for` loop to cycle through each entry in the array to find a matching ticker. If there are too many assets in the array, it may cost a significant amount of gas or in a worst case scenario it may not be possible to complete the function. The relevant code is shown below:

```
// Remove the asset from the asset list
uint256 index;
for (uint256 i = 0; i < assetList.length; i++) {
    if (assetList[i] == ticker_) {
        index = i;
        break;
    }
}
```

Recommendation(s): Consider implementing an upper limit for the length of the `assetList` array or the use or an EnumerableMap data structure. Alternatively, consider breaking the loops into individual functions to allow users to continue the loop process in a separate transaction.

Status: Fixed

Update from the client: Asset-management logic was improved. We got rid of the for-loop for finding the asset's index. Asset index is now stored in `Asset` struct alongside with its token address, asset ID and `bool` value indicating asset presence.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/91

5.4.11 [Low] Missing `felt` input validation

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The following functions do not validate that relevant arguments are within the range of a Cairo `felt` datatype.

```
constructor(...)                // Relevant argument(s): `assetContractAddress_`, `withdrawalRequestContractAddress`
depositToL2(...)                // Relevant argument(s): `amount_`
setTokenContractAddress(...)    // Relevant argument(s): `ticker_`
setAssetContractAddress(...)    // Relevant argument(s): `assetContractAddress_`
setWithdrawalRequestAddress(...) // Relevant argument(s): `withdrawalRequestContractAddress_`
```

Recommendation(s): Consider verifying that the listed functions and their relevant arguments are within the range of a Cairo `felt` datatype. For arguments that are supposed to represent StarkNet addresses, consider using the modifier `isValidL2Address(...)`.

Status: Fixed

Update from the client:

- necessary felt validation added ;
- if value is used in payload of L2-to-L1 message, it's not checked; with invalid value message consumption will fail anyway ;
- `ticker` is not checked because if it's not present — tx will fail; if `ticker` is present, it was already validated when adding asset ;

PR Link: https://github.com/zkxteam/zkxprotocol/pull/88

5.4.12 [Low] Risk of fund losses in transferring to `address(0x0)`

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The function `L1ZKXContract.transferEth(...)` lacks a zero-check on the address `recipient_`. Thus, `ETH` can be transferred to `address(0x0)`, which causes the loss of funds. Moreover, the function also does not check if `amount_` is greater than zero. The function is reproduced below.

```
function transferEth(address payable recipient_, uint256 amount_)
    external
    onlyRole(DEFAULT_ADMIN_ROLE)
{
    require(amount_ <= address(this).balance, "ETH to be transferred is more than the balance");
    recipient_.transfer(amount_);
}
```

Recommendation(s): Always perform a zero-check on the address receiving funds. In order to save gas, revert the operation in case the amount to be sent is zero.

Status: Fixed

Update from the client: Zero-check was added.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/83

5.4.13 [Info] Address comparison does not guarantee equality

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The functions `L1ZKXContract.withdraw(...)` and `L1ZKXContract.withdrawEth(...)` have equality comparisons between `msg.sender` and the `uint256` argument `userL1Address_`. It is possible to pass this check while `userL1Address_` is not equal to `msg.sender` by passing a valid address followed by data above the first 160 bits. During the cast these extra bits will be truncated and the equality check will pass, while `userL1Address_` is not equal to `msg.sender`. It should be noted that passing this check will still lead to a revert due to `userL2Address` reading zero from the mapping, which will then fail when zero as the address when consuming a message from `starknetCore`. The relevant code is shown below:

```
require(msg.sender == address(uint160(userL1Address_)), "Sender is not withdrawal recipient");
uint256 userL2Address = l2ContractAddress[userL1Address_];

// Construct withdrawal message payload.
uint256[] memory withdrawal_payload = new uint256[](5);
withdrawal_payload[0] = WITHDRAWAL_INDEX;
withdrawal_payload[1] = userL1Address_;
withdrawal_payload[2] = ETH_TICKER;
withdrawal_payload[3] = amount_;
withdrawal_payload[4] = requestId_;

// Consume the message from the StarkNet core contract.
// This will revert the (Ethereum) transaction if the message does not exist.
starknetCore.consumeMessageFromL2(userL2Address, withdrawal_payload);
```

Recommendation(s): Consider verifying that the full value of `userL1Address_` is equal to `msg.sender` rather than the truncated result of the casts.

Status: Fixed

Update from the client:

– `msg.sender` is not used anymore for withdrawal ;
– withdraw recipient is determined on L2 when withdrawal is initiated ;

PR Link: https://github.com/zkxteam/zkxprotocol/pull/83

5.4.14 [Info] Missing event emission

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The following functions do not emit events on important state changes:

```
constructor(...)                 // Should emit an event stating that the contract has been deployed and its deployment
↪ parameters
setAssetContractAddress(...)     // Should emit an event containing the previous address and the new address
setWithdrawalRequestAddress(...) // Should emit an event containing the previous address and the new address
transferFunds(...)               // Should emit an event stating that the administrator has moved funds
transferEth(...)                 // Should emit an event stating that the administrator has moved funds
```

Recommendation(s): Consider emitting events for the functions listed above to improve protocol logging.

Status: Fixed

Update from the client: Events for all these functions were added.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/86

5.4.15 [Best Practices] Missing address input validation

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The constructor does not validate the input parameter `starknetCore_` for `address(0x0)`. Most of the time this does not represent any risk since constructors are called during deployment. However, validating input parameters is considered a good programming practice and can reduce the chances of deploying contracts with wrong parameters. The constructor is reproduced below:

```
constructor(
    IStarknetCore starknetCore_,
    uint256 assetContractAddress_,
    uint256 withdrawalRequestContractAddress_
) {
    starknetCore = starknetCore_;
    assetContractAddress = assetContractAddress_;
    withdrawalRequestContractAddress = withdrawalRequestContractAddress_;
    _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
}
```

Recommendation(s): Consider validating the input parameter `starknetCore_` for `address(0x0)`.

Status: Fixed

Update from the client: Checks for all three constructor arguments were added.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/81

5.4.16 [Best Practices] Missing parameter in NatSpec comment

File(s): L1/contracts/L1ZKXContract.sol

Description: The NatSpec documentation for the function depositToL2(...) is missing an @param description for the argument userL2Address. The function and it's NatSpec documentation is shown below:

```
/**
 * @dev function to deposit funds to L2 Account contract
 * @param userL1Address_ - Users L1 wallet address
 * @param collateralId_ - ID of the collateral
 * @param amount_ - The amount of tokens to be deposited
 **/
function depositToL2(
    uint256 userL1Address_,
    uint256 userL2Address_,
    uint256 collateralId_,
    uint256 amount_
) private {...}
```

Recommendation(s): Improve the NatSpec documentation by adding a description for the argument userL2Address.

Status: Fixed

Update from the client: Added.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/81

5.4.17 [Best Practices] Unindexed events

File(s): L1/contracts/L1ZKXContract.sol

Description: To improve off-chain analytics, events can have topics which allow events to be easily searched. Topics can be added to event arguments with the indexed keyword. No events in the L1ZKXContract make use of topics.

Recommendation(s): Consider adding topics to important event arguments to improve off-chain analytics.

Status: Fixed

Update from the client: indexed modifier was added to relevant event arguments.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/89

5.4.18 [Best Practices] Unnecessary balance check before Ether transfers

File(s): L1/contracts/L1ZKXContract.sol

Description: Before calling the Solidity transfer(...) function to transfer Ether, a check is done to prevent the contract from transferring more Ether than it owns. This is done in L1ZKXContract.withdrawEth(...) and L1ZKXContract.transferEth(...) and is shown below:

```
require(amount_ <= address(this).balance, "ETH to be transferred is more than the balance");
payable(msg.sender).transfer(amount_);
```

This check is unnecessary as the Ethereum network does not allow transfers to exceed the balance of an account on a protocol level.

Recommendation(s): Consider removing the unnecessary check before Ether transfers.

Status: Fixed

Update from the client: Removed.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/83

5.4.19 [Best Practices] Unnecessary check before `ERC-20` token transfer

File(s): `L1/contracts/L1ZKXContract.sol`

Description: In the admin-only function `L1ZKXContract.transferFunds(...)` a check is done before transferring tokens to ensure that the contract owns enough tokens for the transfer. This check is unnecessary balance checks are done within the `ERC-20` contract token itself.

Recommendation(s): Consider removing the unnecessary balance check in `transferFunds(...)`.

Status: Fixed

Update from the client: Removed.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/83

5.4.20 [Best Practices] Unused `SafeMath` library

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The OpenZeppelin `SafeMath` library is imported and declared to be used for datatype `uint256` however no `SafeMath` functionality is ever used in the contract.

```
using SafeMath for uint256;
```

Recommendation(s): Consider removing the unused `SafeMath` import to improve the overall readability and maintainability of the codebase.

Status: Fixed

Update from the client: `SafeMath` import removed.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/81

5.4.21 [Best Practices] Using `AccessControl` instead of `Ownable`

File(s): `L1/contracts/L1ZKXContract.sol`

Description: The OpenZeppelin `AccessControl` library is used to handle access control in the `L1ZKXContract`, however only one role is used throughout the contract which is set during contract deployment. The OpenZeppelin `Ownable` library appears to be much more appropriate for the current L1 implementation as only one role is currently supported.

Recommendation(s): Consider replacing the more complex `AccessControl` library with `Ownable` for handling access control.

Status: Fixed

Update from the client: Switched to `Ownable` instead of `AccessControl`.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/81

5.5 L2 General Findings

5.5.1 [Critical] Datatype size to store token balance is not sufficient

File(s): `L2/contracts/*`

Description: Balances of ERC-20 tokens are stored as `uint256` on Ethereum, which is 256 bits. When a user deposits funds to the protocol, the L2 contract represents balances as `Math64x61` which is 125 bits. This can cause many unexpected behaviors and can lead to loss of funds in cases where the L2 deposit calls revert, leaving funds locked in L1 as the L2 balance never updates but the L1 transaction finalizes. This reduced datatype to store balance causes withdrawals to fail with any withdrawal amount over approximately 18.4 tokens when the token decimals is 18.

```
256 bits (uint256) highest value:
115792089237316195423570985008687907853269984665640564039457584007913129639935

125 bits (Math64x61) highest value:
42535295865117307932921825928971026432
```

Recommendation(s): Consider using a larger data type for balances and amounts throughout the L2 to prevent potential loss of information.

Status: Fixed

Update from the client: Below the client shows the changes made in the code.

Previously:

```
func fromFelt(x):
  let res = x * 2^64
  assert_in_64x61_range(res)
  return res
end
```

Let's imagine conversion with 20 ETH which actually is 20 * 10ˆ18 wei units

```
1. decimals = 18
2. decimal_part = pow(10, decimals)
3. decimal_part_64x61 = fromFelt(decimal_part) // 10^18 * 2^64 □ OK, fits in Math64x61 range
4. amount = 20 ETH // 20 * 10^18
5. amount_64x61 = fromFelt(amount) // 20 * 10^18 * 2^64 □ Math64x61 OVERFLOW
6. amount_decimal_representation = div(amount_64x61, decimal_part_64x61)
```

The problem is that we multiply BOTH values by 2ˆ64 just to divide them one line later. And it caused overflow even for 20 ETH

So we won't reach line 6, because fromFelt conversion on line 5 will fail, because value will be out of valid Math64x61 range.

Now:

```
1. decimals = 18
2. decimal_part = pow(10, decimals)
3. amount = 20 ETH // 20 * 10^18
4. amount_decimal_representation = div(amount, decimal_part)
```

PR Link: https://github.com/zkxteam/zkxprotocol/pull/148

5.5.2 [Critical] Testing between L1 and L2 is required

File(s): `L2/contracts/*`

Description: A test suite that connects the L1 contracts to L2 contracts for comprehensive testing is necessary to ensure that both L1 and L2 will correctly communicate on deployment. Without proper L1 to L2 testing there may be unexpected behaviors that will only be seen upon deployment.

Recommendation(s): Implement 100% coverage testing including communication between L1 and L2 before deployment to ensure cross chain communication behaves as intended.

Status: Fixed

Update from the client: Added integrations tests for L1 and l2 communication.

PR Links:

– https://github.com/zkxteam/zkxprotocol/pull/131 ;
– https://github.com/zkxteam/zkxprotocol/pull/135 ;

5.5.3 [High] Events not used in protocol

File(s): `L2/contracts/*`

Description: The L2 implementation does not emit any events. Events can be very useful for off-chain analytics and monitoring the overall health of the protocol.

Recommendation(s): Add events to the L2 implementation to record every important state transition. Emit events during contract deployment in constructors to track deployment parameters.

Status: Fixed

Update from the client: Events have been added.

PR Links:

– https://github.com/zkxteam/zkxprotocol/pull/106 ;
– https://github.com/zkxteam/zkxprotocol/pull/108 ;
– https://github.com/zkxteam/zkxprotocol/pull/109 ;
– https://github.com/zkxteam/zkxprotocol/pull/112 ;
– https://github.com/zkxteam/zkxprotocol/pull/114 ;

5.5.4 [Low] Missing overflow and negative input checks for `Math64x61` interpreted felts

File(s): `L2/contracts/L2/contracts/*`

Description: In general, the code does not check for overflows. In functions that are supposed to add amounts, the code does not validate if the amount to be added is positive. For example the function `Holding.fund(...)` receives the `amount` to increase the `balance_mapping` by, but there are no checks for overflows once `amount` is added and `amount` is not tested to ensure it's a positive number.

```
@external
func fund{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    asset_id_ : felt, amount : felt
):
    ...
    ############################################
    # @audit not checking for overflow
    #        not checking for negative amounts
    ############################################
    let current_amount : felt = balance_mapping.read(asset_id=asset_id_)
    balance_mapping.write(asset_id=asset_id_, value=current_amount + amount)

    return ()
end
```

This also applies to functions where amounts are subtracted instead of added. In functions that are supposed to reduce some value, the code does not validate if that reduction is positive. Subtraction of negative values is the equivalent to increasing the value. For example the function `Holding.defund(...)` has the input parameter `amount` checked to ensure it is less than or equal to the `current_amount`. If we pass a negative value for `amount` this test will pass and the funds will be increased instead of reduced. The code snippet is shown below.

```
@external
func defund{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    asset_id_ : felt, amount : felt
):
    ...
    ############################################
    # @audit not checking for negative amounts
    ############################################
    let current_amount : felt = balance_mapping.read(asset_id=asset_id_)
    with_attr error_message("Amount to be deducted is more than asset's balance"):
        assert_le(amount, current_amount)
    end
    balance_mapping.write(asset_id=asset_id_, value=current_amount - amount)

    return ()
end
```

Recommendation(s): Check for overflows (on the positive side when adding values and also on the negative side when subtracting values). Make sure that the input parameters are in the proper range in accordance with the semantics of the function. When depositing or withdrawing ensure that the amount is positive.

- When performing `res=a+b`, the result `res` must be greater than `a` and `b`;
- When performing `res=a-b`, the result `res` must be smaller than `a`;
- When performing `res=a*b`, the operation `res/b=a` must hold;
- When performing `res=a/b`, `b` must be different from `zero` and the condition `res*b=a` must hold;

Status: Fixed

Update from the client: In most places calculations were updated to use Math64x61 library functions: mul/add/sub/div. Inside this functions input parameters and the result are validated to be in a Math64x61 range. If all these checks pass, the calculation result is also valid.

PR Links:

- https://github.com/zkxteam/zkxprotocol/pull/100 ;
- https://github.com/zkxteam/zkxprotocol/pull/148 ;

5.5.5 [Low] Redundant storage of `L1_zkx_address`

File(s): `L2/contracts/*`

Description: The address of the L1 ZKX contract is stored across multiple contracts. Some of these contracts have the ability to change the L1 address and some do not. In the case that the L1 address needs to be changed, only contracts that have the `set_l1_address(...)` function are able to change addresses, meaning that contracts without the setter will not be able to update to the newer L1 address.

Recommendation(s): Consider having the L1 ZKX contract address stored in one place (such as `AddressRegistry`), where each time a contract needs the L1 ZKX address it can fetch the address from the same source. In case the L1 ZKX address needs to be updated this process can be done in one transaction compared to calling the setter function per-contract.

Status: Fixed

Update from the client: Improved the code by storing `L1_zkx_address` in the authorized address registry.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/103

5.5.6 [Best Practices] Application is not using the StarkNet boolean library

File(s): `L2/contracts/*`

Description: Many contracts use the felt values `1` and `0` to represent binary values (such as permission bits in `AuthAdmin.cairo`). Readability can be improved by using the StarkNet library `starkware.cairo.common.bool` to replace values `1` and `0` with `TRUE` and `FALSE`.

Recommendation(s): Consider using the StarkNet boolean library in all places where `felt` values are treated as binary to improve readability.

Status: Fixed

Update from the client: All contracts now makes use of starknet boolean library.

PR Links:

– https://github.com/zkxteam/zkxprotocol/pull/108 ;
– https://github.com/zkxteam/zkxprotocol/pull/109 ;
– https://github.com/zkxteam/zkxprotocol/pull/112 ;

5.5.7 [Best Practices] Consider implementing pause functionality

File(s): `L2/contracts/*`

Description: The application is not pausable. Although there is some debate in the community about pausable contracts, pausing the application can be a quick and effective way to stop fund losses in case the application is attacked.

Recommendation(s): Consider adding pausable functionality to the application as a safeguard in the case of critical conditions in the protocol.

Status: Acknowledged

Update from the client: We do acknowledge the finding, however as you know, ZKX is designed to be a DAO-driven protocol. For us, this means that every decision about the functioning of the protocol has to be made by the community governing the protocol and not a single (or a small group of) admin(s) making decisions on behalf of the protocol. The Pause function is an excellent example of such functionality that might be used as a security backstop to enforce resistance to the financial vulnerability of the protocol, but due to the time required for the DAO voting to take place vs emergency application of a handbrake functionality might not be the prime candidate for a functionality driven by DAO voting. We are still working on the balanced approach to the Pause function, and its implementation will be added as part of the changes where we connect control endpoints to the DAO voting smart contracts.

Update from Nethermind: We agree on your considerations.

5.5.8 [Best Practices] Functions with high lines of code

File(s): `L2/contracts/*`

Description: There are functions in the protocol that have very high LOC which affects readability, which makes it challenging to understand function logic. An example of such a function is `Trading.check_and_execute(...)`, which is a 600 line function that handles the full process of executing a trade. A function of this size should be broken down into smaller components.

Recommendation(s): Consider refactoring large functions to consist of smaller, more well defined functions to improve the overall readability and maintainability of the codebase.

Status: Fixed

Update from the client: Broke down the large functions into smaller components in Trading.cairo

PR Link: https://github.com/zkxteam/zkxprotocol/pull/140

5.5.9 [Best Practices] Inconsistent code style

File(s): `L2/contracts/*`

Description: Using a consistent code style across a codebase improves the readability and maintainability of the code. There are functions and arguments with differing styles throughout different files. Some examples are shown below:

```
# Camel case function name
func removeMarket(...)

# Snake case function name
func populate_markets(...)
```

```
# Trailing underscore on argument names
@constructor
func constructor{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    registry_address_ : felt, version_ : felt
):
    ...
end

# No trailing underscore on argument names
@view
func getMarket_from_assets{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    asset_id : felt, collateral_id : felt
) -> (market_id : felt):
    ...
end
```

Recommendation(s): Consider using a consistent code style across the protocol to improve the overall readability and maintainability of the code.

Status: Fixed

Update from the client: Made the code style consistent across the protocol.

PR Links:

– https://github.com/zkxteam/zkxprotocol/pull/137 ;
– https://github.com/zkxteam/zkxprotocol/pull/141 ;

5.5.10 [Best Practices] Inconsistent contract structure

File(s): `L2/contracts/*`

Description: The Cairo contract files do not follow a consistent structure and vary between different files. Contract structure is the order in which components of a contract are declared (functions, events, constructors, imports etc). A lack of contract structure affects readability and makes it difficult to find where certain parts of a contract should be.

Recommendation(s): To improve contract readability, consider discussing with your team and setting some standards to be applied consistently across all contracts. An example structure is provided below:

```
1.  Starknet contract declaration
2.  Builtins
3.  Imports
4.  Namespace declarations
5.  Constants
6.  Structs
7.  Events
8.  Storage variables
9.  Constructor
10. Getter functions
11. Setter functions
12. Remaining functions
```

Status: Fixed

Update from the client: Contract structure is now consistent across all contracts.

PR Links:

– https://github.com/zkxteam/zkxprotocol/pull/106 ;
– https://github.com/zkxteam/zkxprotocol/pull/108 ;
– https://github.com/zkxteam/zkxprotocol/pull/109 ;

5.5.11 [Best Practices] Missing error messages on assert fail

File(s): `L2/contracts/*`

Description: There are assert statements that are not enclosed with a `with_attr` error message. In the case that these assert statements fail the user will not receive any information about the error.

Recommendation(s): Consider doing a cleanup pass over the entire codebase to ensure that all assert statement fails are enclosed with a `with_attr` error message.

Status: Fixed

Update from the client: Added error messages for all assert statements.

PR Links:

– https://github.com/zkxteam/zkxprotocol/pull/106 ;
– https://github.com/zkxteam/zkxprotocol/pull/108 ;
– https://github.com/zkxteam/zkxprotocol/pull/109 ;
– https://github.com/zkxteam/zkxprotocol/pull/112 ;
– https://github.com/zkxteam/zkxprotocol/pull/114 ;

5.5.12 [Best Practices] Order status and types should be defined as constants

File(s): `L2/contracts/*`

Description: Order status and types are not defined as constants. They are referred to within the code as integers, which affects readability as anybody reading the code must find out what the numbers actually represent through comments in other files or separate documentation. A code snippet from `Account.cairo` demonstrates this below:

```
if orderDetails.portionExecuted - size == 0:
    if request.orderType == 3:
        assert status_ = 7
    else:
        assert status_ = 4
    end
else:
    if request.orderType == 4:
        assert status_ = 5
    else:
        if request.orderType == 3:
            assert status_ = 6
        else:
            assert status_ = 3
        end
    end
end
```

Recommendation(s): Consider defining constants to represent these values instead, similar to how address indexes and actions are defined in `Constants.sol`. This will improve readability and maintainability.

Status: Fixed

Update from the client: Changed the code to make use of constants.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/104

5.5.13 [Best Practices] Repeated caller authentication code

File(s): `L2/contracts/*`

Description: Functions that need to verify the caller is the correct address through `AuthorizedRegistry` or verify that the caller has the correct permissions though `AdminAuth` do not have wrapper functions. This authentication code is repeated in many functions. This has caused approximately 700 lines of repeated code throughout the L2 implementation. The shown example function below is from `Account.cairo`.

```
@external
func transfer_from{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    assetID_ : felt, amount : felt
) -> ():
    alloc_locals

    # Check if the caller is trading contract
    let (caller) = get_caller_address()
    let (registry) = registry_address.read()
    let (version) = contract_version.read()
    let (balance_) = balance.read(assetID=assetID_)

    let (trading_address) = IAuthorizedRegistry.get_contract_address(
        contract_address=registry, index=Trading_INDEX, version=version
    )

    with_attr error_message("Caller is not authorized to do transferFrom in account contract."):
        assert caller = trading_address
    end

    balance.write(assetID=assetID_, value=balance_ - amount)
    return ()
end
```

In this function there are 14 significant lines of code. Twelve of these lines are for authentication, only to have one line to make a storage change followed by another line to return.

Recommendation(s): Consider implementing a generic authentication function to reduce repeated code. A potential solution could be to have such a function revert with no permission, so the caller contract only needs to surround one function call with a `with_attr` error message. An example of the above function using a generic function is shown below:

```
@external
func transfer_from{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    assetID_ : felt, amount : felt
) -> ():
    with_attr error_message("Caller is not authorized to do transferFrom in account contract."):
        verify_caller(index=Trading_INDEX);
    end

    balance.write(assetID=assetID_, value=balance_ - amount)
    return ()
end
```

Status: Fixed

Update from the client: Implemented `verify_caller_authority` function to reduce repeated code.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/97

5.5.14 [Best Practices] Test contracts should be in a separate directory

File(s): `L2/contracts/*`

Description: There are contracts that are used exclusively for testing, however they are not placed in a test directory which may confuse readers as to which contracts are necessary for deployment and which are only for testing or development purposes.

Recommendation(s): Place all testing related contracts in a specific folder.

Status: Fixed

Update from the client: Isolated test contracts from contracts to be deployed.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/125

5.5.15 [Best Practices] Unnecessary returns

File(s): `L2/contracts/*`

Description: There are functions in the L2 implementation that are hardcoded to return `1` on completion to indicate that they have run successfully. This return is not necessary as the function not reverting is already an indicator that the function has run successfully. Furthermore, some calling functions don't even verify that the function returned `1`. An example from `Account.cairo` is shown below:

```
func add_to_array{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    id_ : felt
) -> (res : felt):
    let (arr_len) = position_array_len.read()
    position_array.write(index=arr_len, value=id_)
    position_array_len.write(arr_len + 1)

    #################################################################
    # @audit This function will always return 1
    #        This means that the return holds no valuable information
    #################################################################

    return (1)
end
```

Recommendation(s): Consider removing returns from functions where the return value carries no information. Any function that will always return the same value does not need to return at all.

Status: Fixed

Update from the client: Removed unnecessary returns.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/136

5.5.16 [Best Practices] Unnecessary use of `alloc_locals`

File(s): `L2/contracts/*`

Description: Many contracts have functions that start with `alloc_locals` even though they do not make use of any local variables.

Recommendation(s): To improve code readability, consider removing `alloc_locals` from functions that do not use local variables.

Status: Fixed

Update from the client: Removed unnecessary use of `alloc_locals`

PR Links:

- https://github.com/zkxteam/zkxprotocol/pull/104 ;
- https://github.com/zkxteam/zkxprotocol/pull/106 ;
- https://github.com/zkxteam/zkxprotocol/pull/108 ;
- https://github.com/zkxteam/zkxprotocol/pull/109 ;

5.5.17 [Best Practices] Unused imports

File(s): `L2/contracts/*`

Description: Many contracts have unused imports. All imports in contract should be used as unused imports are unnecessary and affect readability.

Recommendation(s): Consider doing a cleanup pass to remove all unused imports from the codebase.

Status: Fixed

Update from the client: Removed unused imports.

PR Links:

- https://github.com/zkxteam/zkxprotocol/pull/106 ;
- https://github.com/zkxteam/zkxprotocol/pull/108 ;
- https://github.com/zkxteam/zkxprotocol/pull/109 ;
- https://github.com/zkxteam/zkxprotocol/pull/112 ;
- https://github.com/zkxteam/zkxprotocol/pull/114 ;

5.6 `L2/contracts/Account.cairo`

5.6.1 [High] `node_operator_L2_address` can changed by malicious users

File(s): L2/contracts/Account.cairo

Description: The function `withdraw(...)` can be called as long as a valid signature is provided. `withdraw(...)` has an argument `node_operator_L2_address` which is placed in the withdrawal history to track what ZKX node conducted the withdrawal. If a user decides to call `withdraw(...)` directly, they can control the argument `node_operator_L2_address` and set it to any value of their choosing. This will save potentially incorrect data into withdrawal history.

Recommendation(s): Consider ways to handle `node_operator_L2_address` if it is not being called by a ZKX node.

Status: Mitigated

Update from the client: Such calls for withdrawal will come only from ValidatorRouter contract which will ensure that the call has been signed by sufficient number of whitelisted signature nodes (ZKX nodes). The called function will also have to be changed to ensure it accepts calls only from the ValidatorRouter contract (this is pending and will be done after audit changes are merged into main).

PR Link: https://github.com/zkxteam/zkxprotocol/pull/79

Update from the client: We agree on your considerations.

5.6.2 [Low] Two-stage public key change is needed

File(s): L2/contracts/Account.cairo

Description: In the case that a user decides to use a different L1 public key they can use the function `set_public_key(...)`. If a user accidentally inputs the incorrect address then access to the account contract will be lost. The user will be unable to call `set_public_key(...)` again to change the public key back because that function is only callable through `execute(...)` which verifies a signature against the public key. Since the public key would have been set incorrectly there is no way to change the public key back. A relevant code snippet is shown below:

```
func set_public_key{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    new_public_key : felt
):
    assert_only_self()
    public_key.write(new_public_key)
    return ()
end
```

Recommendation(s): Consider adding a two-stage process when changing the public key of an account contract. This will allow the user to recover from a situation where they otherwise would have permanently lost access to their StarkNet ZKX account contract.

Status: Fixed

Update from the client: This function has been removed.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/96

5.6.3 [Best Practices] Incorrect error message in `liquidate_position(...)`

File(s): `L2/contracts/Account.cairo`

Description: The function `liquidate_position(...)` has a check to ensure that the amount to be sold is not more than the portion executed. The check is done with `assert_le` (assert less than or equal) but the error message only states "less than", rather than "less than or equal". A relevant code snippet is shown below:

```
with_attr error_message("Amount to be sold should be less than portion executed"):
    assert_le(amount_to_be_sold_, order_details.portionExecuted)
end
```

Recommendation(s): Consider changing the`with_attr` error message to correctly reflect the check in the code.

Status: Fixed

Update from the client: Error message has been modified

PR Link: https://github.com/zkxteam/zkxprotocol/pull/104

5.6.4 [Best Practices] Incorrect error message in `withdraw(...)`

File(s): `L2/contracts/Account.cairo`

Description: The function `withdraw(...)` has a check to ensure that the amount of fees to be paid for a withdrawal is no more than the balance of the contract. The check is done with `assert_le` (assert less than or equal) but the error message only states "less than", rather than "less than or equal". A relevant code snippet is shown below:

```
with_attr error_message("Fee amount should be less than fee collateral balance"):
    assert_le(standard_fee, fee_collateral_balance)
end
```

Recommendation(s): Consider changing the `with_attr` error message to correctly reflect the check in the code.

Status: Fixed

Update from the client: Error message has been updated.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/134

5.6.5 [Best Practices] Unused variables in `is_valid_signature_order(...)`

File(s): `L2/contracts/Account.cairo`

Description: The function `is_valid_signature_order(...)` has three unused variables. The variables are shown below:

```
let (caller) = get_caller_address()
let (registry) = registry_address.read()
let (version) = contract_version.read()
```

Recommendation(s): Consider removing the unused variables shown above.

Status: Fixed

Update from the client: Unused variables have been removed.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/104

5.7 `L2/contracts/AccountRegistry.cairo`

5.7.1 [Critical] A zero address account can be added to the account registry

File(s): `L2/contracts/AccountRegistry.cairo`

Description: The function `add_to_account_registry(...)` allows a zero address to be added to the registry as there is no input validation for the argument `address_`. Once this address is added it cannot be removed as the function `remove_from_account_registry(...)` does an existence check to prevent non-existing values from being removed. The existence check is done by checking the value in `account_registry` at the given index. If the value is zero then the function will revert. This will affect the recursive function `populate_account_registry(...)` that does a recursive check and uses a similar existence check to determine when to stop recursion.

Recommendation(s): Consider preventing a zero address from being added to the account registry.

Status: Fixed

Update from the client: Checks have been added on the address to be added to the account registry.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/134

5.7.2 [High] Not checking the array bounds in function `populate_account_registry(...)`

File(s): `L2/contracts/AccountRegistry.cairo`

Description: The function `populate_account_registry(...)` uses recursive calls to retrieve the registry accounts. Instead of using the storage variable `account_registry_len` for controlling the recursion, the function relies on the address stored in the registry. Since the registry is an unbounded array, this strategy can lead to unexpected behavior when the next memory slot is not empty. The strategy can also truncate the list of addresses when some intermediary array element has value `zero`. In fact, adding array elements with value `zero` is allowed by the function `add_to_account_registry(...)`.

```
func populate_account_registry{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    account_registry_len_ : felt, account_registry_list_ : felt*
) -> (account_registry_len_ : felt, account_registry_list_ : felt*):
    alloc_locals
    let (address) = account_registry.read(index=account_registry_len_)

    if address == 0:
        return (account_registry_len_, account_registry_list_)
    end

    assert account_registry_list_[account_registry_len_] = address
    return populate_account_registry(account_registry_len_ + 1, account_registry_list_)
end
```

Recommendation(s): Use the array length for defining the last element of the array. Never rely on the array elements for keeping track of the array length. This recommendation applies to all functions of this application.

Status: Fixed

Update from the client: We now make use of array length to keep track of last element.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/99

5.7.3 [Medium] Risk of negative array length in function `remove_from_account_registry(...)`

File(s): `L2/contracts/AccountRegistry.cairo`

Description: The function `remove_from_account_registry(...)` does not check if the input parameter `id_` is greater than (or equal to) zero and whether `id_` is smaller than the array length (`reg_len`). The function also does not assert that the array length (`reg_len`) is greater than zero before subtracting one unit, which can lead to storing a negative value as the array length.

```
@external
func remove_from_account_registry{
    syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr
}(id_ : felt) -> (res : felt):
    ...
    let (caller) = get_caller_address()
    let (registry) = registry_address.read()
    let (version) = contract_version.read()
    let (auth_address) = IAuthorizedRegistry.get_contract_address(
        contract_address=registry, index=AdminAuth_INDEX, version=version
    )

    let (access) = IAdminAuth.get_admin_mapping(
        contract_address=auth_address, address=caller, action=MasterAdmin_ACTION
    )
    assert_not_zero(access)

    let (account_address) = account_registry.read(index=id_)
    if account_address == 0:
        with_attr error_message("account address does not exists in that index"):
            assert 1 = 0
        end
    end

    let (reg_len) = account_registry_len.read()
    let (last_account_address) = account_registry.read(index=reg_len - 1)

    account_registry.write(index=id_, value=last_account_address)
    account_registry.write(index=reg_len - 1, value=0)
    account_registry_len.write(reg_len - 1)
    account_present.write(address=account_address, value=0)
    return (1)
end
```

Recommendation(s): Assert that the input parameter `id_` is greater than or equal to zero, `id_` is less than `reg_len`. Also ensure that `reg_len` is greater than zero before subtracting one unit and making any changes in the registry.

Status: Fixed

Update from the client: Validations have been added.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/99

5.7.4 [Low] Function `get_account_registry(...)` returns an unbounded array

File(s): `L2/contracts/AccountRegistry.cairo`

Description: The function `get_account_registry(...)` returns the list of the registry accounts. Since this is an unbounded array, the execution of the function is not guaranteed. Since there is no maximum number of account contracts that can be registered, the call may revert due to extensive computational overhead.

```
@view
func get_account_registry{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}() -> (
    account_registry_len : felt, account_registry : felt*
):
    alloc_locals
    let (account_registry_list : felt*) = alloc()
    let (account_registry_len_, account_registry_list_) = populate_account_registry(
        0, account_registry_list
    )
    return (account_registry_len=account_registry_len_, account_registry=account_registry_list_)
end
```

Recommendation(s): Consider implementing a pagination scheme by grabbing a set range of elements from the array at a time. Make successive calls to this function in order to get the list of all accounts.

Status: Fixed

Update from the client: Implemented pagination.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/99

5.8 L2/contracts/AdminAuth.cairo

5.8.1 [High] Proposed administrator can call admin-only functions

File(s): L2/contracts/AdminAuth.cairo

Description: The storage map admin_mapping stores a value allowed for each address and a given action. If allowed is 1 then a user does have permission for that given action, and if 0 then they do not. However when an address has been proposed to be an admin (but has not yet been approved) their allowed value in admin_mapping stores the address of the admin that proposed them. The purpose of this is to prevent the same admin from proposing and then approving by ensuring that the approving address is not the same as the proposing address. The issue is that the majority of permission checks for MasterAdmin_ACTION check that a user has permission with assert_not_zero(allowed). When an admin is proposed their allowed value is the address of the proposer which is non-zero, allowing them to pass auth checks. Relevant code snippets are shown below.

```
##############################################################################
# Snippet 1: Writing proposer to the addresses `admin_mapping` `allowed` value.
##############################################################################

let (caller) = get_caller_address()
let status : felt = admin_mapping.read(address=address, action=MasterAdmin_ACTION)

if status == 0:
    admin_mapping.write(address=address, action=MasterAdmin_ACTION, value=caller)
    # Second approval for granting/revoking admin role
else:
    assert_not_equal(status, caller)
    admin_mapping.write(address=address, action=MasterAdmin_ACTION, value=value)
end
```

```
##############################################################################
#Snippet 2: The insecure admin authentication code.
##############################################################################

let (access) = IAdminAuth.get_admin_mapping(
    contract_address=admin_auth, address=caller, action=MasterAdmin_ACTION
)
with_attr error_message("Caller does not have permission to update base abr value"):
    assert_not_zero(access)
end
```

Recommendation(s): Consider handling admin proposals with a separate storage map to admin_mapping and change authentication code to assert access = 1 rather than assert_not_zero(access) to make the authentication checks more specific.

Status: Fixed

Update from the client: Admin management logic has been improved such that an address can perform admin only actions only after getting two approvals.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/102

5.8.2 [Medium] Admins cannot be added when admin count is one

File(s): L2/contracts/AdminAuth.cairo

Description: The process of adding a new master admin requires a two step process with a proposal and approval. If one admin has removed the other leaving only one master admin, there is no other address to approve a proposal. This leaves the protocol in a state where no new admins can be added.

Recommendation(s): Consider enforcing a minimum number of master admin addresses to prevent an admin count of one.

Status: Fixed

Update from the client: Enforced the presence of minimum number of master admins in the system always.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/102

5.8.3 [Medium] Missing input validation in `update_admin_mapping(...)`

File(s): `L2/contracts/AdminAuth.cairo`

Description: In the function `update_admin_mapping(...)` the argument `value_` represents the permission bit for a given `action_`. The `value_` parameter is expected to be `1` or `0`, but a lack of input validation allows any value to be used. It should be also noted that as mentioned in a previous finding, the use for the StarkNet boolean library is recommended for these permission bits. The problem with a lack of input validation is that different functions check the permission bit in different ways. Some only require that the value is not zero and some ensure that the value is equal to one.

Recommendation(s): Ensure that the argument `value_` is equal to `1` or `0`.

Status: Fixed

Update from the client: Input validations have been added.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/102

5.8.4 [Undetermined] Admin can be removed without approval

File(s): `L2/contracts/AdminAuth.cairo`

Description: The process for adding a new master admin requires a two step process with a proposal and approval. When removing an admin it can be done immediately without the need for approval. A malicious or compromised admin address may remove other admin addresses to gain full control of the protocol for the given version.

Recommendation(s): This finding is undetermined as the audit team does not have documentation to indicate whether this is an intended feature or not. Consider discussing whether an approval process should exist for removing an admin.

Status: Fixed

Update from the client: Admins can only be removed with two approvals now.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/102

5.8.5 [Best Practices] Missing use of action constants

File(s): `L2/contracts/AdminAuth.cairo`

Description: In the `AdminAuth` constructor there are two writes to storage for the first two master admin roles. These writes to storage use the direct value which represents a master admin, rather than using the constant `MasterAdmin_ACTION` which affects readability. A comment does describe what the values represent, however directly using the proper constant is also recommended. The constructor is shown below:

```
@constructor
func constructor{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    address1 : felt, address2 : felt
):
    admin_mapping.write(address=address1, action=0, value=1)
    admin_mapping.write(address=address2, action=0, value=1)
    return ()
end
```

Recommendation(s): Consider replacing the `action=0` value with the appropriate constant `action=MasterAdmin_ACTION` to improve readability and maintainability.

Status: Fixed

Update from the client: Logic for Admin management is updated.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/102

5.9 `L2/contracts/AuthorizedRegistry.cairo`

5.9.1 [Info] Unnecessary `assert_not_zero` in `update_contract_registry(...)`

File(s): `L2/contracts/AuthorizedRegistry.cairo`

Description: The function `update_contract_registry(...)` has two duplicate assert statements. One is surrounded by a `with_attr` containing an error message, however the other assert statement checks for the same condition just before, so if the error occurs there will be no error message. The code is shown below.

```
assert_not_zero(access)
with_attr error_message("Caller does not have permission to update contract registry"):
    assert_not_zero(access)
end
```

Recommendation(s): Remove the unnecessary `assert_not_zero` shown above.

Status: Fixed

Update from the client: Removed unnecessary assert statement.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/77

5.9.2 [Info] Unnecessary `namespace` declaration

File(s): `L2/contracts/AuthorizedRegistry.cairo`

Description: The namespace `IAdminAuth` is declared to be used as an interface with the `AdminAuth` contract to call the function `get_-admin_mapping`. There is no need to declare this interface in the contract as the file `L2/contracts/interfaces/IAdminAuth.cairo` already declares this interface so it can simply be imported. The code is shown below.

```
# @notice AdminAuth interface
@contract_interface
namespace IAdminAuth:
    func get_admin_mapping(address : felt, action : felt) -> (allowed : felt):
    end
end
```

Recommendation(s): Consider removing the declared `IAdminAuth` interface and import `IAdminAuth` from `L2/contracts/interfaces/IAdminAuth.cairo` instead.

Status: Fixed

Update from the client: Removed namespace declaration.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/104

5.10 `L2/contracts/Markets.cairo`

5.10.1 [Best Practices] Unnecessary assert

File(s): `L2/contracts/Markets.cairo`

Description: In the function `addMarket(...)` a check is done to ensure that the value of a tradable field should be less than or equal to `2`. A comparison is done and then an assert is used to ensure that the return from the comparison is `1`. This logic can be combined into one statement with `assert_le_felt(...)`. The relevant code snippet is shown below:

```
    # Value of tradable field should be less than or equal to 2
    let (is_less) = is_le(newMarket.tradable, 2)
    assert_not_zero(is_less)
```

Recommendation(s): Consider replacing the above code with `assert_le_felt(...)`.

Status: Fixed

Update from the client: We now do the validation using `assert_le`

PR Link: https://github.com/zkxteam/zkxprotocol/pull/123

5.11 `L2/contracts/Trading.cairo`

5.11.1 [Undetermined] Market prices can be set by anybody once TTL expires

File(s): `L2/contracts/Trading.cairo`

Description: The function `execute_batch(...)` gets the price for the given `marketID` and checks if the price has expired its TTL before executing orders. If the price is expired then the price is set to the argument `execution_price`. There is no authentication on the `execute_batch(...)` function allowing anybody to set the price of any expired market price to any value. The market prices are used by `Liquidate.cairo` for liquidation checks which could lead to unexpected behavior such as positions being marked for liquidation when they shouldn't be or positions that should be liquidated not being marked for liquidation. The relevant code snippet is shown below.

```
tempvar timestamp = market_prices.timestamp
tempvar time_difference = current_timestamp - timestamp
let (status) = is_le(time_difference, ttl)

# update market price
if status == 0:
    IMarketPrices.update_market_price(
        contract_address=market_prices_contract_address,
        id=marketID,
        price=execution_price
    )
    ...
```

Recommendation(s): Consider adding an authentication check to `execute_batch(...)` to ensure that it can only be called by ZKX nodes.

Status: Mitigated

Update from the client: Such calls for trade execution will come only from ValidatorRouter contract which will ensure that the call has been signed by sufficient number of whitelisted signature nodes (ZKX nodes). The called function will also have to be changed to ensure it accepts calls only from the ValidatorRouter contract (this is pending and will be done after audit changes are merged into main).

PR Link: https://github.com/zkxteam/zkxprotocol/pull/79

5.11.2 [Info] Unused storage variable

File(s): `L2/contracts/Trading.cairo`

Description: The storage variable `net_acc` and its respective getter are defined but not used within the contract.

Recommendation(s): Consider whether the storage variable `net_acc` is needed within the `Trading` contract.

Status: Fixed

Update from the client: This storage variable has been removed.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/125

5.12 `L2/contracts/TradingFees.cairo`

5.12.1 [High] Fee and discount tiers may have empty entries

File(s): `L2/contracts/TradingFees.cairo`

Description: The fees and discounts are determined by the function `get_user_fees_and_discount(...)` which calls the recursive functions `find_user_base_fee(...)` and `find_user_discount(...)`. These functions are recursive, starting at the highest tier and decrementing the tier in the recursion call each time until an appropriate fee and discount has been discovered. The max tier for both fees and discounts can be set directly with function calls (`update_max_base_fee_tier(...)` and `update_max_discount_tier(...)`) or through the functions `update_base_fees(...)` and `update_discount(...)` where the argument `tier_` is higher than the current max. This allows the max tier to be set in such a way that allows for many empty storage slots in the fee and discount storage variables. When this empty data is read by the function `find_user_base_fee(...)` this can lead to the recursive function selecting a tier from an empty entry which leads to the user paying zero fees. The function is shown below:

```
@external
func update_base_fees{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    tier_ : felt, fee_details : BaseFee
):
    alloc_locals
    let (caller) = get_caller_address()
    let (registry) = registry_address.read()
    let (version) = contract_version.read()
    let (auth_address) = IAuthorizedRegistry.get_contract_address(
        contract_address=registry, index=AdminAuth_INDEX, version=version
    )
    let (access) = IAdminAuth.get_admin_mapping(
        contract_address=auth_address, address=caller, action=ManageFeeDetails_ACTION
    )
    assert_not_zero(access)

    let (current_max_base_fee_tier) = max_base_fee_tier.read()
    let (result) = is_le(current_max_base_fee_tier, tier_)
    if result == 1:

        ###########################################################################
        # @audit-issue The parameter `tier_` can be much higher than the current max.
        #              This can lead to many empty entries that will be read by
        #              the recursive function `find_user_base_fee()`.
        #
        #              EG: You can call this with `tier_ = 30,000` but if there are
        #                  only entries in 1,2,3,4 then you will have
        #                  entries 1,2,3,4,30000 with real values and the remaining
        #                  will be empty, but still readable by other functions.
        #
        #              The same issue applies to discounts and `find_user_discount()`
        ###########################################################################

        max_base_fee_tier.write(value=tier_)
        tempvar syscall_ptr = syscall_ptr
        tempvar pedersen_ptr : HashBuiltin* = pedersen_ptr
        tempvar range_check_ptr = range_check_ptr
    else:
        tempvar syscall_ptr = syscall_ptr
        tempvar pedersen_ptr : HashBuiltin* = pedersen_ptr
        tempvar range_check_ptr = range_check_ptr
    end

    base_fee_tiers.write(tier=tier_, value=fee_details)
    return ()
end
```

Recommendation(s): Consider adjusting the implementation of `update_base_fees(...)` and `update_discount(...)` to prevent the max fee and discount tiers from being set in a way that allows empty entries to be read by other functions. Consider removing the ability to manually set the max tier for fees and discounts.

Status: Fixed

Update from the client: Modified code so that fee and discount tiers will not allow empty entries.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/100

5.12.2 [High] Missing validation on first entry in fee arrays

File(s): `L2/contracts/TradingFees.cairo`

Description: When making an entry into the storage arrays `base_fee_tiers` or `discount_fee_tiers` it is possible for the first entry in the array to have a `numberOfTokens` value to be non-zero. If the first entry in a fee array has a non-zero `numberOfTokens` , unexpected behavior regarding fee selection may occur. The next recurse may cause the array slot at `PRIME-1` to be read, leading to zero fees or zero discount on fees. The relevant code is shown below.

```
func find_user_base_fee{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    number_of_tokens_ : felt, tier_ : felt
) -> (base_fee_maker : felt, base_fee_taker : felt):
    alloc_locals

    ####################################################################################
    # @audit-issue If the first entry in the array has a numberOfTokens that is
    #              above zero the `is_nn` will return `0`. This will then cause
    #              a recursion call so the next base_fee_tiers read will be on index `0 - 1`.
    #              This may lead to unexpected fee selection behavior.
    ####################################################################################
    let (fee_details) = base_fee_tiers.read(tier=tier_)
    let sub_result = number_of_tokens_ - fee_details.numberOfTokens

    let (result) = is_nn(sub_result)
    if result == 1:
        return (base_fee_maker=fee_details.makerFee, base_fee_taker=fee_details.takerFee)
    else:
        return find_user_base_fee(number_of_tokens_, tier_ - 1)
    end
end
```

Recommendation(s): Consider adding a check in all functions that can set values in base_fee_tiers and discount_fee_tiers to ensure that the first entry in either of these arrays cannot have a numberOfTokens less than or equal to zero.

Status: Fixed

Update from the client: Now that tiers are ordered and number of tokens in those tiers are also in increasing order, this issue will not happen.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/100

5.12.3 [Medium] Fee and discount tiers aren't guaranteed to be ordered

File(s): L2/contracts/TradingFees.cairo

Description: The storage variables base_fee_tiers and discount_tiers store data related to fees and discounts. Based on the recursive function find_user_base_fee(...) and find_user_discount(...) it appears that the intention is that the higher the tier, the better it is for the user. When adding new discounts or fees with update_base_fees(...) or update_discount(...) there aren't any checks to ensure that base_fee_tiers and discount_tiers will remain ordered. If a fee entry containing very high fees is added to the highest tier, when get_user_fee_and_discount(...) is called it will return the first fee that doesn't result in negative tokens. This can cause users to be charged for fees that are higher than they should actually pay if the fee and discount storage variable entries were ordered. Relevant code is shown below:

```
################################################################################
# @audit-issue This function is called with argument `tier_` equal to `max_base_fee_tier`
################################################################################

func find_user_base_fee{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    number_of_tokens_ : felt, tier_ : felt
) -> (base_fee_maker : felt, base_fee_taker : felt):
    alloc_locals

    ##################################################################################
    # @audit Since `base_fee_tiers` isn't guaranteed to be ordered `fee_details`
    #        may not represent the best fee for the user. This fee will likely
    #        pass the `is_nn(...)` check and the user will have paid a higher
    #        fee than they should have.
    ##################################################################################

    let (fee_details) = base_fee_tiers.read(tier=tier_)
    let sub_result = number_of_tokens_ - fee_details.numberOfTokens
    let (result) = is_nn(sub_result)
    if result == 1:
        return (base_fee_maker=fee_details.makerFee, base_fee_taker=fee_details.takerFee)
    else:
        return find_user_base_fee(number_of_tokens_, tier_ - 1)
    end
end
```

Recommendation(s): Consider adjusting the implementation of update_base_fees(...) and update_discount(...) to ensure that the storage variables base_fee_tiers and discount_tiers will remain ordered.

Status: Fixed

Update from the client: Modified code so that fee and discount tiers will always be in order.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/100

5.13 L2/contracts/Math_64x61.cairo

5.13.1 [Low] Functions not validating the input parameters in Math_64x61.cairo

File(s): L2/contracts/Math_64x61.cairo

Description: Several Math64x61 functions do not validate the input parameters for valid ranges. It is possible that invalid input parameters (i.e., out of range) can lead to a valid output passing the test Math64x61_assert64x61(res).

Recommendation(s): Assert that the input parameters are within a valid range by calling Math64x61_assert64x61(...).

Status: Fixed

Update from the client: Validations have been added.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/100

5.13.2 [Low] Off-by-one error in Math64x61_assert64x61(...)

File(s): L2/contracts/Math_64x61.cairo

Description: The function Math64x61_assert64x61(...) is designed to assert that the input parameter fits in 125 bits. Instead of using assert_le(...) for the upper bound check, the function should use assert_lt(...). When we have 125 bits, we can represent numbers from 0 up to (2^125)-1. The function is reproduced below.

```
func Math64x61_assert64x61{range_check_ptr}(x : felt):
    assert_le(x, Math64x61_BOUND)
    assert_le(-Math64x61_BOUND, x)
    return ()
end
```

Recommendation(s): Replace assert_le(...) for the upper bound check with assert_lt(...).

Status: Fixed

Update from the client: Replaced assert_le with assert_lt

PR Link: https://github.com/zkxteam/zkxprotocol/pull/100

5.13.3 [Low] Math64x61_ceil(...) returning wrong output on whole numbers

File(s): L2/contracts/Math_64x61.cairo

Description: The function Math64x61_ceil(...) is missing the condition where mod_val is zero. The function is reproduced below.

```
# Calculates the ceiling of a 64.61 value
func Math64x61_ceil{range_check_ptr}(x : felt) -> (res : felt):
    let (int_val, mod_val) = signed_div_rem(x, Math64x61_ONE, Math64x61_BOUND)
    let res = (int_val + 1) * Math64x61_ONE
    Math64x61_assert64x61(res)
    return (res)
end
```

Recommendation(s): When mod_val is equal to zero, the function must return int_val * Math64x61_ONE, instead of (int_val + 1) * Math64x61_ONE.

Status: Fixed

Update from the client: Corrected the logic.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/100

5.14 L2/contracts/QuoteL1Fee.cairo

5.14.1 [Info] Unused contract

File(s): `L2/contracts/AccountDeployer.cairo`

Description: The contract `QuoteL1Fee` is not interacted with at any point within the protocol.

Recommendation(s): Consider whether this contract needs to remain in the codebase.

Status: Fixed

Update from the client: Moved it to a test folder.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/134

5.15 L2/contracts/ABR.cairo

5.15.1 [Info] Function `find_std(...)` is actually computing the variance

File(s): `L2/contracts/ABR.cairo`

Description: The function `find_std(...)` is supposed to get the standard deviation of the sample. However, in its actual form, it is computing the variance, which is the square of the standard deviation. The function is reproduced below.

```
func find_std{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    array_len : felt, array : felt*, mean : felt, window_size : felt, sum : felt
) -> (sum : felt):
    alloc_locals

    # If reached the end of the array, return
    if window_size == 0:
        return (sum)
    end

    # Calculates the difference between the array element and the mean
    let (diff) = Math64x61_sub([array], mean)

    # Calculates the square root of the difference
    let (diff_sq) = Math64x61_mul(diff, diff)

    # Recursively call the next array element
    return find_std(array_len, array + 1, mean, window_size - 1, sum + diff_sq)
end
```

Recommendation(s): The development team must decide if they need the variance or the standard deviation. The standard deviation is the square root of `sum`.

Status: Fixed

Update from the client: Renamed the function name for better understanding.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/134

5.16 L2/contracts/Asset.cairo

5.16.1 [High] Assets can be overwritten in `addAsset(...)`

File(s): `L2/contracts/Asset.cairo`

Description: There is no check to ensure that the `id` of a newly created asset matches that of an existing asset. This will overwrite the entry in the `asset` storage variable for the given array however the L1 ZKX contract will still add the asset like normal rather than modifying an existing entry. This leads to an inconsistency between the asset lists in L1 and L2. Furthermore, when overwriting an existing asset updates to `assets_array` and `assets_array_len` are updated as normal.

Recommendation(s): Consider adding a check to ensure that an existing asset cannot be overwritten when a new asset is added.

Status: Fixed

Update from the client: Asset-management logic is updated. We check for asset existence before adding a new asset.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/110

5.16.2 [High] Assets can have matching tickers in `addAsset(...)`

File(s): `L2/contracts/Asset.cairo`

Description: In the function `addAsset(...)` there is no check to ensure that the `Asset.ticker` value of the to-be-added asset matches that of an existing asset in the L2 side of the protocol. In the current implementation this function will work normally however the message sent to the ZKX L1 contract will cause the function `L1ZKXContract.updateAssetListInL1(...)` to add the new asset in `assetList` but overwrite the previous tickers entry in `assetID`. This affects the data integrity of assets on the L1 side and can affect withdrawals in unexpected ways.

Recommendation(s): Consider adding a check to ensure that the `Asset.ticker` of the to-be-added asset does not match any existing tickers in the protocol.

Status: Fixed

Update from the client: Asset-management logic is updated. We check for asset existence before adding a new asset.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/110

5.16.3 [High] Function `modify_core_settings(...)` can lead to fund losses

File(s): `L2/contracts/Asset.cairo`

Description: The function `modify_core_settings(...)` makes changes to the asset settings, including the number of decimals and the short name. It is unclear:

– When can this function be called? ;
– What happens if we change an asset that is actively being traded? ;
– How changing the number of decimals can affect the deposited amount? ;

Moreover, the new settings are not reflected on L1, which can lead to loss of integrity. The function is reproduced below.

```
@external
func modify_core_settings{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    id : felt,
    short_name : felt,
    tradable : felt,
    collateral : felt,
    token_decimal : felt,
    metadata_id : felt,
):
    alloc_locals
    ...
    let (_asset : Asset) = asset.read(id=id)

    asset.write(
        id=id,
        value=Asset(asset_version=_asset.asset_version, ticker=_asset.ticker, short_name=short_name, tradable=tradable,
        collateral=collateral, token_decimal=token_decimal, metadata_id=metadata_id, tick_size=_asset.tick_size,
        ↪  step_size=_asset.step_size,
        minimum_order_size=_asset.minimum_order_size, minimum_leverage=_asset.minimum_leverage,
        ↪  maximum_leverage=_asset.maximum_leverage,
        currently_allowed_leverage=_asset.currently_allowed_leverage,
        ↪  maintenance_margin_fraction=_asset.maintenance_margin_fraction,
        initial_margin_fraction=_asset.initial_margin_fraction,
        ↪  incremental_initial_margin_fraction=_asset.incremental_initial_margin_fraction,
        incremental_position_size=_asset.incremental_position_size,
        ↪  baseline_position_size=_asset.baseline_position_size,
        maximum_position_size=_asset.maximum_position_size),
    )
    return ()
end
```

Recommendation(s): Double-check if this function is necessary. Calling this function can potentially lead to fund losses. The function should revert in case users have funds deposited into the target asset.

Status: Fixed

Update from the client: Removed `token_decimal` argument from `modify_core_settings` function of Asset contract to prevent pottential fund losses.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/139

5.16.4 [Low] Missing input validation in `addAsset(...)`

File(s): `L2/contracts/Asset.cairo`

Description: The function `addAsset(...)` does not validate any parts of the to-be-added asset object. It is possible to add an asset with all zero values which will affect other parts of the protocol such as `populate_assets(...)` which causes the list of assets to return before all assets have been listed.

Recommendation(s): Consider validating inputs for asset objects. It may be worth discussing with the team to establish some valid ranges for certain inputs, for example min and max values for asset `step_size` and `tick_size`.

Status: Fixed

Update from the client: Added input validations for all asset properties.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/148

5.16.5 [Info] Assets cannot be removed from `assets_array`

File(s): `L2/contracts/Asset.cairo`

Description: When an asset is removed from the protocol through the function `removeAsset(...)`, the asset is not removed from the `assets_array` storage variable. This can lead to calls to the function `returnAllAssets(...)` returning previously removed assets.

Recommendation(s): Consider updating the `assets_array` entries to reflect the removed asset within the function `remoneAsset(...)`.

Status: Fixed

Update from the client: Asset-management logic is updated. Data related to assets is now stored in Asset struct, which is completely removed on removeAsset() call.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/110

5.17 `L2/contracts/ABRPayment.cairo`

5.17.1 [Critical] No access control in function `pay_abr(...)`

File(s): `L2/contracts/ABRPayment.cairo`

Description: The function `pay_abr(...)` is `external` and it does not validate the caller. The inline comments state that this function is supposed to be called only by the node. However, the code responsible for checking the signature is empty. The code is shown below.

```
# @notice Function to be called by the node
# @param account_addresses_len - Length of thee account_addresses array being passed
# @param account_addresses - Account addresses array
@external
func pay_abr{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    account_addresses_len : felt, account_addresses : felt*
):
    # ## Signature checks go here ####
    # (...empty...)
    ...
end
```

Recommendation(s): Implement proper access control to this function.

Status: Mitigated

Update from the client: This call will be made by ValidatorRouter contract which will check that call has been signed by a certain number of whitelisted signature nodes (ZKX nodes). The called contract (ABRPayment.cairo) will also have to be changed to accept calls only from such ValidatorRouter contract (this will be done after audit related fixes are merged into main branch)

PR Link: https://github.com/zkxteam/zkxprotocol/pull/79

5.17.2 [Best Practices] Replicated code in function `pay_abr_users_positions(...)`

File(s): `L2/contracts/ABRPayment.cairo`

Description: The function `pay_abr_users_positions(...)` has three possible actions: a) pay the users; b) receive from users; c) do nothing in case the user has no open positions. The function has replicated code for paying users and receiving from users. Replicating code is not a good programming practice and should be avoided. The function also uses a hard coded value to define the direction. A better approach is to define constants and make these constants different from `zero`.

Recommendation(s): Consider improving the quality and readability of this code by eliminating the replication of code blocks.

Status: Fixed

Update from the client: Redundant code has been converted to functions and used as required.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/112

5.18 `L2/contracts/WithdrawalRequest.cairo`

5.18.1 [Critical] Missing input validation in function `add_withdrawal_request(...)`

File(s): `L2/contracts/WithdrawalRequest.cairo`

Description: This function is intended to be called through `Account.withdraw(...)`. However it can be called directly with `Account.execute(...)`. Since all the validations happen outside of the function, when called directly with `Account.execute(...)` the data is not validated, allowing for existing entries to be overwritten which affects the integrity of the storage variable `withdrawal_request_mapping`.

```
@external
func add_withdrawal_request{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    request_id_ : felt,
    user_l1_address_ : felt,
    ticker_ : felt,
    amount_ : felt,
):
    let (registry) = registry_address.read()
    let (version) = contract_version.read()
    let (caller) = get_caller_address()

    # fetch account registry contract address
    let (account_registry_address) = IAuthorizedRegistry.get_contract_address(
        contract_address=registry, index=AccountRegistry_INDEX, version=version
    )
    # check whether caller is registered user
    let (present) = IAccountRegistry.is_registered_user(
        contract_address=account_registry_address, address_=caller
    )

    with_attr error_message("Called account contract is not registered"):
        assert_not_zero(present)
    end

    # Create a struct with the withdrawal Request
    let new_request = WithdrawalRequest(
        user_l1_address=user_l1_address_,
        user_l2_address=caller,
        ticker=ticker_,
        amount=amount_,
    )

    withdrawal_request_mapping.write(request_id=request_id_, value=new_request)
    return ()
end
```

Recommendation(s): All the functions should work as a black box where all the validations happen inside the function body, otherwise it is susceptible to exploits. Consider implementing a way to prevent arbitrary calls from a ZKX account contracts `Account.execute(...)` function to `add_withdrawal_request(...)`, while still allowing the regular calls from `Account.withdraw(...)`.

Status: Fixed

Update from the client: execute functon has been removed from Account contract. So, now arbitary calls through ZKX account contract is not possible.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/124

5.19 L2/contracts/WithdrawalFeeBalance.cairo

5.19.1 [High] Data integrity of fee mappings can be compromised

File(s): L2/contracts/L2/contracts/WithdrawalFeeBalance.cairo

Description: The function update_withdrawal_fee_mapping(...) is used to track withdrawal requests. It appears that this function is only expected to be called through Account.withdraw(...) however it is possible for a user to use a ZKX account contract's execute(...) function to call update_withdrawal_fee_mapping(...) directly. There is authentication to ensure that the caller is a registered ZKX account contract, but a user can call this function through Account.execute(...) and pass any values for arguments. This can be used to overwrite existing data or write new data. This compromises the integrity of withdrawal_fee_mapping and total_withdrawal_fee_per_asset storage maps.

Recommendation(s): Implement a strategy to prevent arbitrary calls from a ZKX account contracts Account.execute(...) function to update_withdrawal_fee_mapping(...), while still allowing the regular calls from Account.withdraw(...).

Status: Fixed

Update from the client: execute functon has been removed from Account contract. So, now arbitary calls through ZKX account contract is not possible.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/124

5.19.2 [Low] No lower bound in the withdrawal fee in set_standard_withdraw_fee(...)

File(s): L2/contracts/L2/contracts/WithdrawalFeeBalance.cairo

Description: The function set_standard_withdraw_fee(...) is responsible for setting the withdrawal fee_ for a given collateral_id_. The function can only be called by users having the role AdminAuth_INDEX. The inline comment states that 0.02 USDC is the standard withdrawal fee. However, the function does not present any limitation for this fee. A relevant code snippet is shown below.

```
@external
func set_standard_withdraw_fee{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    fee_ : felt, collateral_id_ : felt
):
    alloc_locals
    # Auth Check
    let (caller) = get_caller_address()
    let (registry) = registry_address.read()
    let (version) = contract_version.read()
    let (auth_address) = IAuthorizedRegistry.get_contract_address(
        contract_address=registry, index=AdminAuth_INDEX, version=version
    )

    let (access) = IAdminAuth.get_admin_mapping(
        contract_address=auth_address, address=caller, action=MasterAdmin_ACTION
    )
    assert_not_zero(access)

    standard_withdraw_fee.write(value=fee_)
    standard_withdraw_fee_collateral_id.write(value=collateral_id_)
    return ()
end
```

Recommendation(s): Although the DAO is responsible for setting this fee, we recommend ensuring that the fee cannot be negative. It is a good practice to also validate if collateral_id_ maps to an existing collateral asset.

Status: Fixed

Update from the client: Checks have been added to ensure that fee cannot be negative and also we are validating whether the asset is collateral in the system.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/116

5.20 `L2/contracts/FeeDiscount.cairo`

5.20.1 [Critical] Anyone can call the functions `add_user_tokens(...)` and `remove_user_tokens(...)` and change the users' balance

File(s): `L2/contracts/L2/contracts/FeeDiscount.cairo`

Description: Although adding the authentication is planned, at this moment the functions `add_user_tokens(...)` and `remove_user_tokens(...)` have no authentication and have `external` visibility. This is dangerous and compromises fee functionality in the protocol. The code for both functions is reproduced below.

```
@external
func add_user_tokens{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    address : felt, value : felt
):
    # Authorization needs to be added in the future
    let number_of_tokens : felt = user_tokens.read(address=address)
    user_tokens.write(address=address, value=number_of_tokens + value)
    return ()
end
```

```
@external
func remove_user_tokens{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    address : felt, value : felt
):
    # Authorization needs to be added in the future
    let number_of_tokens : felt = user_tokens.read(address=address)
    assert_nn(number_of_tokens - value)
    user_tokens.write(address=address, value=number_of_tokens - value)
    return ()
end
```

Recommendation(s): The authentication must be added immediately. Since the code is extensive, it can be difficult to track this in the future.

Status: Fixed

Update from the client: A new role has been added for authentication.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/122

5.20.2 [Low] Missing validations in function `add_user_tokens(...)`

File(s): `L2/contracts/L2/contracts/FeeDiscount.cairo`

Description: The function does not check for overflow when adding more tokens to the user. The function also does not assert that `value` is greater than `zero`.

```
@external
func add_user_tokens{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    address : felt, value : felt
):
    ...
--> user_tokens.write(address=address, value=number_of_tokens + value)
    ...
end
```

Recommendation(s): The authentication must be added immediately. Since the codebase is extensive, it can be difficult to track this in the future. Assert that `value` is greater than `zero`.

Status: Fixed

Update from the client: Overflow checks have been added.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/122

5.21 L2/contracts/Markets.cairo

5.21.1 [Low] Risk of markets having the same id in function addMarket(...)

File(s): L2/contracts/Markets.cairo

Description: The function addMarket(...) receives as input parameters the id of the new market and the variable newMarket having its setup parameters. The function does not check if id already exists and if newMarket has valid parameters. The code is reproduced below.

```
@external
func addMarket{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    id : felt, newMarket : Market
):
    ...
    #################################################################
    # @audit-issue Risk of duplicated ids.
    # @audit-issue Hard Coded numbers.
    ##################################################################
    # Value of tradable field should be less than or equal to 2
    let (is_less) = is_le(newMarket.tradable, 2)
    assert_not_zero(is_less)

    # Getting asset details
    let (asset_address) = IAuthorizedRegistry.get_contract_address(
        contract_address=registry, index=Asset_INDEX, version=version
    )
    let (asset1 : Asset) = IAsset.getAsset(contract_address=asset_address, id=newMarket.asset)
    let (asset2 : Asset) = IAsset.getAsset(
        contract_address=asset_address, id=newMarket.assetCollateral
    )
    assert_not_zero(asset2.collateral)
    assert_not_zero(asset1.ticker)

    ##################################################################
    # @audit-issue Market parameters are not checked for validity
    #              before being written to storage.
    ##################################################################
    if newMarket.tradable == 2:
        market.write(
            id=id,
            value=Market(asset=newMarket.asset, assetCollateral=newMarket.assetCollateral,
                leverage=newMarket.leverage, tradable=asset1.tradable, ttl=newMarket.ttl),
        )
        market_mapping.write(
            asset_id=newMarket.asset, collateral_id=newMarket.assetCollateral, value=id
        )
    else:
        if newMarket.tradable == 1:
            assert_not_zero(asset1.tradable)
        end

        market.write(id=id, value=newMarket)
        market_mapping.write(
            asset_id=newMarket.asset, collateral_id=newMarket.assetCollateral, value=id
        )
    end

    let (curr_len) = markets_array_len.read()
    markets_array.write(index=curr_len, value=id)
    markets_array_len.write(value=curr_len + 1)
    return ()
end
```

Recommendation(s): Validate the input parameters. Make sure that id is unique. Assert that newMarket contains valid parameters.

Status: Fixed

Update from the client: Market-management logic was updated. Before adding a new market, we check whether market with same id exists.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/123

5.21.2 [Low] Risk of removing markets with open positions

File(s): `L2/contracts/Markets.cairo`

Description: The function `removeMarket(...)` does not present any check to ensure that a market has no open positions. Before removing the markets, all positions should be closed. The code is reproduced below.

```
@external
func removeMarket{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(id : felt):
    ##################################################################
    # @audit-issue Risk of removing markets having open positions
    #              Not checking if market was already removed
    ##################################################################

    alloc_locals
    # Auth Check
    let (caller) = get_caller_address()
    let (registry) = registry_address.read()
    let (version) = contract_version.read()
    let (auth_address) = IAuthorizedRegistry.get_contract_address(
        contract_address=registry, index=AdminAuth_INDEX, version=version
    )

    let (access) = IAdminAuth.get_admin_mapping(
        contract_address=auth_address, address=caller, action=ManageMarkets_ACTION
    )
    assert_not_zero(access)

    market.write(id=id, value=Market(asset=0, assetCollateral=0, leverage=0, tradable=0, ttl=0))
    return ()
end
```

Recommendation(s): Close all the open positions in the market prior to removal. Double check for more side effects when removing a market that was active.

Status: Fixed

Update from the client: We have added different states to markets now. When a market is in archived state, it won't show up in the list of markets, this can be used to temporarily pause trading in a specific market. Then there is non tradable state, during which users won't be able to open new positions, but existing positions can be closed. Before removing a certain market, it will be in non tradable state for a certain time period. Also markets can now be removed only if it is in non tradable state.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/698

5.21.3 [Undetermined] Market functions can return removed markets

File(s): `L2/contracts/Markets.cairo`

Description: The functions `populate_markets(...)` and `getMarket(...)` return removed markets. There is no documentation to state whether this is an intended behavior or not.

Recommendation(s): Double check if this is the intended behavior.

Status: Fixed

Update from the client: `populate_markets(...)`function now returns the markets which exists.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/698

5.21.4 [Info] Possibility of modifying the leverage of removed and non-existent markets

File(s): `L2/contracts/Markets.cairo`

Description: The function `modifyLeverage(...)` should revert in case a previously removed or non-existent market is passed as input parameter. The code is reproduced below.

```
@external
func modifyLeverage{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    id : felt, leverage : felt
):
    alloc_locals
    ...
    let (_market : Market) = market.read(id=id)

    ####################################################################
    # @audit-issue This operation cannot be performed when the market
    #              was removed (fields equal to zero)
    ####################################################################
    market.write(
        id=id,
        value=Market(asset=_market.asset, assetCollateral=_market.assetCollateral, leverage=leverage,
        ↪  tradable=_market.tradable, ttl=_market.ttl),
    )
    return ()
end
```

Recommendation(s): Check if the market was removed or does not exist before updating the leverage.

Status: Fixed

Update from the client: Market-management logic was updated. Before modifying leverage, we check whether the market exists.

PR Link: https://github.com/zkxteam/zkxprotocol/pull/123

5.22 Fund Related Contracts

5.22.1 [Medium] Missing validations in core functions of fund related contracts

File(s): `L2/contracts/InsuranceFund.cairo`, `L2/contracts/ABRFund.cairo`, `L2/contracts/LiquidityFund.cairo`, `L2/contracts/Holding.cairo`, `L2/contracts/EmergencyFund.cairo`

Description: The functions `deposit(..)`, `fund(...)`, `defund(...)`, `withdraw(...)` and similar functions such as `fund_holding(...)`, `fund_liquidity(...)`, `fund_insurance(...)`, `defund_holding(...)`, `defund_insurance(...)`, `defund_liquidity(...)` are not checking for overflow and not asserting that the amount to be funded, defunded, deposited or withdrawn is greater than `zero`. The functions are not checking for broken invariants (such as negative balance). This is recurrent for all fund contracts. Below we present commented code snippets indicating the validations that could be introduced in the code.

```
@external
func fund{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    asset_id_ : felt, amount : felt
):
    alloc_locals
    ...
    ###################################################################
    # @audit-issue Not checking if current_amount >= 0
    # If current_amount < 0 we have a broken invariant that must be
    # investigated.
    ###################################################################
    let current_amount : felt = balance_mapping.read(asset_id=asset_id_)
    ###################################################################
    # @audit-issue Not checking if amount > 0
    # @audit-issue Not checking for overflow
    # @audit-issue Not emitting event
    ###################################################################
    balance_mapping.write(asset_id=asset_id_, value=current_amount + amount)

    return ()
end
```

```
@external
func defund{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    asset_id_ : felt, amount : felt
):
    alloc_locals
    ...
    #################################################################
    # @audit-issue Not checking if current_amount >= 0
    # If current_amount < 0 we have a broken invariant that must be
    # investigated.
    #################################################################
    let current_amount : felt = balance_mapping.read(asset_id=asset_id_)
    with_attr error_message("Amount to be deducted is more than asset's balance"):
        assert_le(amount, current_amount)
    end
    #################################################################
    # @audit-issue Not checking if amount > 0
    # @audit-issue Not emitting event
    #################################################################
    balance_mapping.write(asset_id=asset_id_, value=current_amount - amount)

    return ()
end
```

```
@external
func deposit{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    asset_id_ : felt, amount : felt, position_id_ : felt
):
    alloc_locals
    ...
    #################################################################
    # @audit-issue Not checking if current_amount >= 0
    # If current_amount < 0 we have a broken invariant that must be
    # investigated.
    #################################################################
    let current_amount : felt = balance_mapping.read(asset_id=asset_id_)
    balance_mapping.write(asset_id=asset_id_, value=current_amount + amount)

    #################################################################
    # @audit-issue Not checking if current_liq_amount >= 0
    # If current_liq_amount < 0 we have a broken invariant that must be
    # investigated.
    #################################################################
    let current_liq_amount : felt = asset_liq_position.read(
        asset_id=asset_id_, position_id=position_id_
    )
    #################################################################
    # @audit-issue Not checking for overflow
    # @audit-issue Not checking if amount > 0
    # @audit-issue Not emitting event
    #################################################################
    asset_liq_position.write(
        asset_id=asset_id_, position_id=position_id_, value=current_liq_amount + amount
    )
    return ()
end
```

```
@external
func withdraw{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    asset_id_ : felt, amount : felt, position_id_ : felt
):
    alloc_locals
    ...
    #################################################################
    # @audit-issue Not checking if current_amount >= 0
    # If current_amount < 0 we have a broken invariant that must be
    # investigated.
    #################################################################
    let current_amount : felt = balance_mapping.read(asset_id=asset_id_)
    with_attr error_message("Amount to be deducted is more than asset's balance"):
        assert_le(amount, current_amount)
    end
    #################################################################
    # @audit-issue Not checking if amount > 0
    #################################################################
    balance_mapping.write(asset_id=asset_id_, value=current_amount - amount)

    #################################################################
    # @audit-issue Not checking if current_liq_amount >= 0
    # If current_lid_amount < 0 we have a broken invariant that must be
    # investigated.
    #################################################################
    let current_liq_amount : felt = asset_liq_position.read(
        asset_id=asset_id_, position_id=position_id_
    )
    ###############################################
    # @audit-issue Not checking if amount > 0
    # @audit-issue Not emitting event
    ###############################################
    asset_liq_position.write(
        asset_id=asset_id_, position_id=position_id_, value=current_liq_amount - amount
    )
    return ()
end
```

Recommendation(s): Implement all the required validation for the proper operation of the contract.

Status: Fixed

Update from the client: Added validations

PR Link: https://github.com/zkxteam/zkxprotocol/pull/121

5.22.2 [Best Practices] Repeated code in fund contracts

File(s): `L2/contracts/InsuranceFund.cairo`, `L2/contracts/ABRFund.cairo`, `L2/contracts/LiquidityFund.cairo`, `L2/contracts/Holding.cairo`, `L2/contracts/EmergencyFund.cairo`

Description: All fund related contracts (`LiquidityFund`, `ABRFund`, `InsuranceFund`, `Holding`, `EmergencyFund`) have common storage variables `balance_mapping` and common functions `fund(...)` and `defund(...)`. This repeated logic could be placed in a "fund" contract which all other fund related contracts could inherit from. This would improve readability and maintainability of the code. In the event that a change to fund logic needs to happen, only one file needs to be changed.

Recommendation(s): Consider placing common fund logic into a dedicated "fund" contract and have other contracts inherit it.

Status: Fixed

Update from the client: Added fund library which holds common logic which will be inherited by all other fund contracts

PR Link: https://github.com/zkxteam/zkxprotocol/pull/121

5.23 `L2/contracts/MarketPrices.cairo`

5.23.1 [High] Function `update_market_price(...)` not checking for market existence

File(s): `L2/contracts/MarketPrices.cairo`

Description: The function `update_market_price(...)` is not validating if the market exists. In case the market does not exist, it won't be possible to get the fields `market.asset` and `market.assetCollateral`. The code snippet is produced below.

```
@external
func update_market_price{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    market_id_ : felt,
    price_ : felt
):
    ...
    ################################################################
    # @audit-issue Code not handling inexistent markets
    ################################################################
    # Get Market from the corresponding Id
    let (market : Market) = IMarkets.getMarket(
        contract_address=market_contract_address,
        id=market_id_
    )

    ################################################################
    # @audit-issue Not checking price for zero or negative values
    ################################################################
    # Create a struct object for the market prices
    tempvar new_market_price : MarketPrice = MarketPrice(
        asset_id=market.asset,
        collateral_id=market.assetCollateral,
        timestamp=timestamp_,
        price=price_,
    )

    ################################################################
    # @audit-issue Not emitting event
    ################################################################
    market_prices.write(id=market_id_, value=new_market_price)
    return ()
end
```

Recommendation(s): Check if the market exists and revert otherwise.

Status: Fixed

Update from the client: Added necessary checks

PR Link: https://github.com/zkxteam/zkxprotocol/pull/130

5.23.2 [High] Market prices can be set to arbitrary values (including zero and negative values)

File(s): `L2/contracts/MarketPrices.cairo`

Description: It is possible to have a market price of zero or even negative. This may have unexpected effects on the protocol where `get_market_price(...)` is used.

Recommendation(s): Discuss with the team whether a market price of zero should be allowed on the system. If zero is not an intended price then consider adding a check to `update_market_price(...)` to prevent `price_` from being zero.

Status: Fixed

Update from the client: Added necessary checks

PR Link: https://github.com/zkxteam/zkxprotocol/pull/130

5.23.3 [High] Standard collateral can be set negative or zero

File(s): `L2/contracts/MarketPrices.cairo`

Description: The function `set_standard_collateral(...)` does not validate the `collateral_id_` for `zero` or negative values. The code snippet is reproduced below.

```
@external
func set_standard_collateral{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    collateral_id_ : felt
):
    ...
    ################################################################
    # @audit-issue Collateral can be set to zero or negative values
    ################################################################
    standard_collateral.write(value=collateral_id_)
    return ()
end
```

Recommendation(s): Assert that the standard collateral is within a valid range. Revert otherwise.

Status: Fixed

Update from the client: This function has been removed as we no longer make use of standard collateral to perform calculations

PR Link: https://github.com/zkxteam/zkxprotocol/pull/130

5.24 L2/contracts/Account.cairo

5.24.1 [Best Practices] Test function present in non-test contract

File(s): L2/contracts/Account.cairo

Description: A test function set_balance(...) is in Account.cairo. Any testing related code should not be in the same file as code that could potentially be deployed. We acknowledge the comment stating that it is to-be-removed so this is just a best practices finding.

Recommendation(s): Since this function is used for testing, consider creating a contract TestAccount.cairo where Account.cairo is inherited and the test function set_balance(...) can be added on top. The TestAccount.cairo contract can be used for testing purposes.

Status: Fixed

Update from the client: Isolated test functions from the contracts to be deployed

PR Link: https://github.com/zkxteam/zkxprotocol/pull/125

5.24.2 [Best Practices] Withdrawal status values could be constants

File(s): L2/contracts/Account.cairo

Description: When creating WithdrawalHistory struct objects the status of a withdrawal is written to WithdrawalHistory.status. This value can be set to 0 (representing an initiated withdrawal) or 1 (representing a successful withdrawal). This is explained in DataTypes.cairo however the use of named constants would provide better code readability. A relevant code snippet is shown below:

```
# Create a withdrawal history object
local withdrawal_history_ : WithdrawalHistory = WithdrawalHistory(
    request_id=request_id_,
    collateral_id=collateral_id_,
    amount=amount_in_felt,
    timestamp=timestamp_,
    node_operator_L2_address=node_operator_L2_address_,
    status=0
)
```

An example of using a constant for readability:

```
# Create a withdrawal history object
local withdrawal_history_ : WithdrawalHistory = WithdrawalHistory(
    request_id=request_id_,
    collateral_id=collateral_id_,
    amount=amount_in_felt,
    timestamp=timestamp_,
    node_operator_L2_address=node_operator_L2_address_,
    status=WITHDRAWAL_INITIATED
)
```

Recommendation(s): Consider using constants for the values representing an initiated or successful withdrawal to improve readability.

Status: Fixed

Update from the client: Added named constants for withdrawal status. **PR Link**: https://github.com/zkxteam/zkxprotocol/pull/132

6 Documentation Evaluation

Documenting the code is adding enough information to explain what it does so that it is easy to understand the purpose and the underlying functionality of each file/function/line. Documentation can come not only in the form of a `README.md` but also through inline comments, web-sites, research papers, videos and external documentation. Besides being a good programming practice, providing proper documentation improves the efficiency of audits. Less time can be spent understanding the protocol and more time can be put towards auditing which improves the efficiency and overall output of the audit. Along the audit we received an internal protocol specification composed of 101 pages as well as eight diagrams explaining the execution flow of certain user/protocol actions. Since this documentation is confidential at the moment of this audit, it is not linked in this document. The provided written documentation was based on internal communication though ticketing software. While it acted as a substitute for formal developer documentation throughout this audit, it is highly recommended to create formal developer documentation to make the protocol easier to understand. **All points of attention raised in relation to the documentation have been resolved.**

7 Test Suite Evaluation

In this section, we present our evaluation of the test suite provided with the application. We check compilation, tests, and code coverage. Tests for L1 and L2 are separate, and they are analyzed along the following subsections. The ZKX protocol was still in development at the time of the audit, with a primary focus on completing the more complex L2 implementation before L1. The L1 contracts and test suite is a simple implementation to demonstrate the intended protocol functionality. The L1 test suite is limited and we acknowledge that this will be improved in the future.

The L2 test suite covers the majority of use cases, however it does not check edge cases. It is important to improve testing to cover for edge cases to identify unexpected behaviors that otherwise would have remained when deployed. The L2 test suite should consider unexpected interactions with the protocol, such as requesting negative deposits and withdrawals, passing very large data arrays to functions, passing invalid and malicious addresses trying to break data integrity between L1 and L2, as well as boundary testing on argument inputs.

The test suite has no communication between L1 and L2. It is highly recommended to develop tests where both layers interact with each other to ensure that cross chain communication works as expected. Since we are dealing with two distinct layers, communication must be fully tested as any incorrect assumptions related to cross chain communication may cause unexpected behavior.

7.1 Layer 1 Contract Compilation

Below, we present the output of the compilation process. We notice some warnings that must be properly handled.

```
> npx hardhat compile
Downloading compiler 0.8.14
Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/mocks/StarknetCoreMock.sol:64:9:
   |
64 |         uint256 nonce
   |         ^^^^^^^^^^^^^


Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/mocks/StarknetCoreMock.sol:85:9:
   |
85 |         uint256 nonce
   |         ^^^^^^^^^^^^^
```

7.2 Layer 1 Tests Output

```
> npx hardhat test

  Asset management
     Add ETH (51ms)
     Add ZKXToken (51ms)
     Add both ETH & ZKXToken (71ms)
     Add ETH & ZKXToken, then remove ETH (97ms)
     Add ETH & ZKXToken, then remove ZKXToken (94ms)
     NOT possible to add ETH twice (93ms)
     NOT possible to add ZKXToken twice (72ms)
     NOT possible to set token address for non-existing asset
     NOT possible to set token address to ZERO (52ms)
     NOT possible to change already set token address
     Add 3 assets, then remove first (130ms)
     Add 3 assets, then remove middle (119ms)
     Add 3 assets, then remove last (114ms)

  Deployment
     Constructor event emission  (44ms)
     State after deployment (51ms)
     Unable to deploy with zero StarknetCore address
     Change Withdrawal Request Address (85ms)
     Change Asset Address (69ms)

  Deposits
     Not possible to deposit ETH before asset added
     Not possible to deposit 0 ETH
     Successful ETH deposit (39ms)
     Deposit and then withdraw ETH (87ms)
     Not possible to deposit Tokens before asset added
     Not possible to deposit Tokens is address not set
     Not possible to deposit more Tokens than user has (43ms)
     Successful Token deposit (64ms)
     Deposit and then withdraw tokens (125ms)
     Multiple token deposits (115ms)
     Multiple ETH deposits (57ms)

  Deposit Cancellation
     deposit by one user, cancellation by another user (72ms)
     Trying to finalize cancel without initiating (71ms)
     Initiating cancel before Message Cancellation delay is complete (83ms)
     Successful Cancellation of deposit (87ms)

  Multisig
     Initial state
     Withdraw ETH (102ms)
     Add assets (ZKX token & ETH), then remove ZKX token (300ms)
     Atomicly add ZKX token and set its contract address (91ms)
     Change asset contract address (48ms)
     Change withdrawal contract address (47ms)
     Atomicly Change asset & withdrawal contract addresses (61ms)


  40 passing (6s)
```

7.3 Layer 1 Code Coverage

The audited code has not been instrumented with the `Solidity Coverage` tool. In order to use the Solidity Coverage, we must install it using the command line below:

```
npm install --save-dev solidity-coverage
```

After this, we must require the plugin in the `hardhat.config.js` file, as shown below.

```
require('solidity-coverage');
```

After doing these steps, we can check the coverage by running the command:

```
npx hardhat coverage
```

The output is presented below.

```
-----------------------|----------|----------|----------|----------|----------------|
File                   |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-----------------------|----------|----------|----------|----------|----------------|
 contracts/            |    89.25 |    55.68 |     87.5 |     91.6 |                |
  Constants.sol        |      100 |      100 |      100 |      100 |                |
  IStarknetCore.sol    |      100 |      100 |      100 |      100 |                |
  L1ZKXContract.sol    |    90.11 |    55.68 |    90.48 |    92.31 |... 569,570,572 |
  ZKXToken.sol         |       50 |      100 |    66.67 |       50 |             23 |
 contracts/Multisig/   |    82.35 |    41.38 |    64.29 |    80.49 |                |
  IMutisig.sol         |      100 |      100 |      100 |      100 |                |
  MultisigAdmin.sol    |    82.35 |    41.38 |    64.29 |    80.49 |... 210,213,214 |
 contracts/mocks/      |      100 |       90 |      100 |      100 |                |
  StarknetCoreMock.sol |      100 |       90 |      100 |      100 |                |
-----------------------|----------|----------|----------|----------|----------------|
All files              |    87.71 |    52.56 |    82.22 |     88.7 |                |
-----------------------|----------|----------|----------|----------|----------------|
```

The contract `ZKXToken.sol` is just a moch contract emulating a stable coin, it is out of the scope of this audit.

7.4 Layer 2 Contract Compilation

The L2 contracts can be compiled using the command below.

```
nile compile
```

After running this commands, all contracts were compiled without warnings or error messages.

7.5 Layer 2 Tests Output

Before running the test suite, we had to install the OpenZeppelin Cairo Contracts using the command line below.

```
pip install openzeppelin-cairo-contracts
```

The tests output are composed of 6,167 lines. For the purposes of this document, we are only showing the tests summary below.

```
> pytest
platform linux -- Python 3.8.10, pytest-7.1.2, pluggy-1.0.0
rootdir: /home/cris/cairo/zkxprotocol-audit-fixes-L2/L2, configfile: tox.ini
plugins: web3-5.30.0, typeguard-2.13.3, asyncio-0.19.0
asyncio: mode=auto
collected 291 items                                                              ...

291 passed, 420 warnings in 1814.77s (0:30:14)
```

8 About Nethermind

Founded in 2017 by a small team of world-class technologists, Nethermind builds Ethereum solutions for developers and enterprises. Boosted by a grant from the Ethereum Foundation in August 2018, our team has worked tirelessly to deliver the fastest Ethereum client in the market. Our flagship Ethereum client is all about performance and flexibility. Built on .NET core, a widespread, enterprise-friendly platform, Nethermind makes integration with existing infrastructures simple, without losing sight of stability, reliability, data integrity, and security

Nethermind is made up of several engineering teams across various disciplines, all collaborating to realize the Ethereum roadmap, by conducting research and building high-quality tools. Teams focus on specific areas of the Ethereum problem space. Each consists of specialists and experienced developers working alongside interns, learning the ropes in the Nethermind Internship Program.

Our mission is to gather passionate talent from around the world, and to tackle some of the blockchain's most complex problems. Nethermind provides software solutions and services for developers and enterprises building the Ethereum ecosystem. We offer security reviews to projects built on EVM compatible chains and StarkNet. We have expertise in multiple areas of the Ethereum ecosystem, including protocol design, smart contracts (written in Solidity and Cairo), MEV, etc. We develop some of the most used tools on Starknet and one of the most used Ethereum clients. Learn more about us at `https://nethermind.io`.