ARMUR Audit Report

Prepared for FXDX

PRESENTED TO

FXDX

PRESENTED BY

Akhil Sharma
Amritansh

Mar 13, 2023

Contents

POSITIONMANAGER.SOL DOCS

This smart contract is an implementation of a Position Manager. It provides functions for increasing and decreasing positions with a range of tokens, including ETH, using a provided router and an order book. It also allows the setting of order keepers, liquidators, and partners.

Here is a more detailed explanation of the functions and variables in the contract:

Variables:

ARMUR

- **orderBook:** The address of the order book contract.
- **inLegacyMode:** A boolean indicating whether the contract is in legacy mode.
- **shouldValidateIncreaseOrder:** A boolean indicating whether to validate an increase order.
- **isOrderKeeper**: A mapping of addresses to booleans indicating whether an address is an order keeper.
- **isPartner:** A mapping of addresses to booleans indicating whether an address is a partner.
- **isLiquidator**: A mapping of addresses to booleans indicating whether an address is a liquidator.

POSITIONMANAGER.SOL DOCS

Events -

- SetLiquidator(address indexed account, bool isActive): Emitted when an account is added or removed as a liquidator.
- SetPartner(address account, bool isActive): Emitted when an account is added or removed as a partner.
- SetInLegacyMode(bool inLegacyMode): Emitted when the contract switches between legacy mode and non-legacy mode.
- SetShouldValidateIncreaseOrder(bool shouldValidateIncreaseOrder): Emitted when the shouldValidateIncreaseOrder variable is set.

ARMUR

Modifiers -

- onlyOrderKeeper(): Requires the caller to be an order keeper.
- onlyLiquidator(): Requires the caller to be a liquidator.
- onlyPartnersOrLegacyMode(): Requires the caller to be a partner or for the contract to be in legacy mode.
- Functions:

- constructor(): Initializes the contract with the provided arguments.

POSITIONMANAGER.SOL

DOCS

- setOrderKeeper(address _account, bool _isActive): Sets an account as an order keeper or removes them.
- setLiquidator(address _account, bool _isActive): Sets an account as a liquidator or removes them.
- setPartner(address _account, bool _isActive): Sets an account as a partner or removes them.
- setInLegacyMode(bool _inLegacyMode): Sets the inLegacyMode variable to true or false.
- setShouldValidateIncreaseOrder(bool _shouldValidateIncreaseOrder): Sets the shouldValidateIncreaseOrder variable to true or false.
- increasePosition(): Increases a position with a specified token and size delta.
- increasePositionETH(): Increases a position with ETH and a specified size delta.
- decreasePosition(): Decreases a position with a specified token, collateral delta, and size delta.

decreasePositionETH(): Decreases a position with ETH, a specified collateral delta, and size delta.

ARMUR

POSITIONMANAGER.SOL ISSUES

- **Access control:** The contract uses a modifier to restrict access to certain functions, but the access control mechanism relies on mappings that are only updated by the contract owner. There is a risk that the owner could be compromised, or the mappings could be modified by an attacker with access to the owner's private key, resulting in unauthorized access to critical functions.

- **Denial of service (DoS) attacks:** The contract uses the nonReentrant modifier to prevent reentrancy attacks, but there is a risk of DoS attacks if an attacker can trigger a function that requires a lot of gas to execute. This could cause the function to run out of gas and fail, preventing legitimate users from using the contract.

- **Integer overflow/underflow:** There are several arithmetic operations in the contract that could potentially result in integer overflow or underflow if the inputs are not properly validated. This could allow an attacker to manipulate the contract's state or steal funds from the contract.

- **Dependency risks:** The contract imports several other contracts, including interfaces and a base contract. There is a risk that these contracts could contain vulnerabilities that could be exploited to attack this contract.

ARMUR

POSITIONMANAGER.SOL ISSUES

- **Lack of validation**: Some input parameters are not sufficiently validated, such as the _path parameter in the increasePosition and increasePositionETH functions. If this parameter is manipulated by an attacker, it could cause the contract to behave unexpectedly or result in loss of funds.

- **Lack of event checking**: There are several events emitted in the contract, but there is no mechanism to check if these events have been emitted correctly or if they have been emitted at all. This could make it difficult to detect certain types of attacks or errors.

- **Lack of documentation:** The contract's functionality is not well-documented, which could make it difficult for users to understand how to use the contract or for auditors to review the contract for security issues.

- **Lack of testing**: There is no evidence in the contract's source code that the contract has been extensively tested for security issues. This increases the risk of undiscovered vulnerabilities that could be exploited by attackers.

ARMUR

ROUTER.SOL

ISSUES

- **Access control vulnerabilities:** The onlyGov modifier is used to restrict access to certain functions. However, this implementation could be problematic if the gov address is compromised or if the owner loses their private key. Additionally, there are no other roles with different access levels.

- **Plugin vulnerabilities:** There is a plugins mapping that can be modified by the gov address. It allows plugins to be added or removed from the contract. This could be exploited by a malicious plugin to alter the state of the contract or perform unauthorized actions.

- **Approval vulnerabilities:** There is a approvedPlugins mapping that allows plugins to be approved or denied by other contracts. However, there is no mechanism in place to revoke approval once it is given. This could lead to unauthorized actions being performed by approved plugins in the future.

ARMUR

ROUTER.SOL

ISSUES

- **Reentrancy vulnerabilities**: There are several external function calls to other contracts, which could potentially introduce reentrancy attacks. The SafeERC20 library is used to avoid reentrancy issues with token transfers, but this does not protect against all reentrancy attacks.

- **Input validation vulnerabilities:** Some functions do not validate their inputs correctly. For example, the swap function assumes that the input path contains at least two elements, which could lead to an out-of-bounds array access if the path is empty. Additionally, the directPoolDeposit function does not check if the sender has approved the contract to transfer tokens on their behalf.

- **Function visibility vulnerabilities:** Some functions are marked as external when they should be marked as public. This could make it difficult for other contracts to interact with the contract properly.

- **Lack of exception handling:** There are no try-catch blocks in the contract to handle exceptions that might be thrown by external function calls. This could cause the contract to become stuck or behave unpredictably in the event of an error.

- **Naming and documentation issues:** Some function names and comments are unclear, which could make it difficult for other developers to understand the purpose of the contract or how to use it correctly.

ARMUR

SHORTSTRACKER.SOL ISSUES

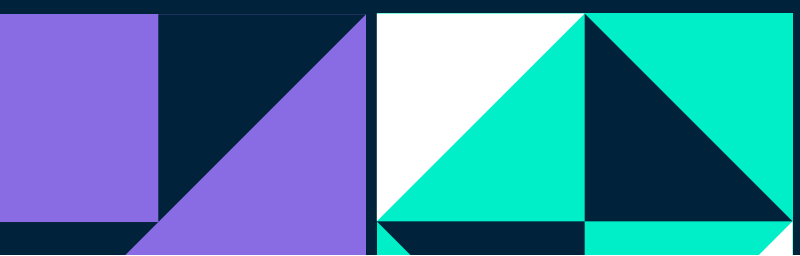
- **Access control**: The contract uses the **Governable** contract for access control, which implies that only the contract owner (governor) can modify certain parameters such as setting handlers, updating global short data, and setting initial data. However, it's possible that the owner account is compromised, or the contract is deployed with a faulty implementation of the **Governable** contract that allows unauthorized access.

- **Integer overflow/underflow:** The contract uses the **SafeMath** library to prevent integer overflow/underflow, but it's still possible that the arithmetic operations in the contract could exceed the maximum or minimum values for the integer types used. For example, the **MAX_INT256** constant is defined as the maximum value for **int256**, but the contract uses **uint256** instead.

ARMUR

- **Uninitialized variables:** The **globalShortAveragePrices** and **isGlobalShortDataReady** variables are not initialized in the constructor, which could lead to unexpected behavior if they are used before being set.

- Denial-of-service (DoS) attacks: The **getNextGlobalShortData** function could potentially be used to launch a DoS attack by passing in invalid parameters that cause the function to consume exce gas or throw an exception. For example, passing in value for **_sizeDelta** could cause the function to run gas.



SHORTSTRACKER.SOL ISSUES

- **Reentrancy attacks:** The contract could be vulnerable to reentrancy attacks if any of the functions that interact with external contracts (such as **vault**) are not implemented correctly. For example, the **getRealisedPnl** function calls **vault.getPosition**, which could potentially be reentered if the **getPosition** function also calls back into the **ShortsTracker** contract.

- I**nformation leakage:** The **data** mapping is public, which means that anyone can read the values stored in the mapping. Depending on what data is stored in the mapping, this could potentially reveal sensitive information about the contract or its users.

- **Logic errors:** There are several places in the contract where the logic could be incorrect or lead to unexpected behavior. For example, the **updateGlobalShortData** function checks if **_isLong** or **_sizeDelta** is zero, but it's not clear why this check is necessary. Additionally, the **getGlobalShortDelta** function calculates the **delta** variable using integer division, which could potentially result in rounding errors.

ARMUR

- **Oracle Manipulation:** The contract relies on oracle price feeds to determine the value of various assets, which means that if the oracle can be manipulated, then the contract's functionality can be compromised. An attacker could manipulate the price feed by providing incorrect data, resulting in a false asset value, and potentially enabling the attacker to profit from it.

- **Arbitrary Contract Interaction**: The contract uses external contract interfaces to interact with other contracts, such as AMM, oracle, and flags. An attacker could potentially deploy a malicious contract with the same interface, which would be invoked by the target contract, allowing them to exploit vulnerabilities in the malicious contract and execute arbitrary code.

- **Insecure contract design:** The contract does not follow the latest best practices for secure contract design. For example, it uses a version of Solidity that is not the latest, which could leave it vulnerable to known vulnerabilities. The contract also lacks a fallback function, which could make it vulnerable to reentrancy attacks.

- **Authorization Issues:** The contract uses a governor address to manage its configuration, which means that the governor has the power to change critical parameters that impact the contract's security. However, there is no mechanism in place to prevent the governor from making unauthorized changes, which could lead to the governor abusing their power t compromise the contract's functionality or steal funds.

ARMUR

- **Reentrancy Attacks:** The contract interacts with other contracts, which could leave it vulnerable to reentrancy attacks. An attacker could call a function on the target contract that calls another contract, and then have that contract call back to the target contract before it has finished executing, potentially allowing the attacker to manipulate the target contract's state.

- **Integer Overflow/Underflow**: The contract uses SafeMath library to mitigate the risk of integer overflow/underflow. However, it is still possible for an attacker to craft inputs that cause the arithmetic operations to result in an overflow/underflow, potentially causing the contract to behave in unexpected ways.

- **Denial of Service:** The contract has several functions that can be used to change critical parameters, such as the adjustment basis points and spread basis points, which could be used to deny service to other users or manipulate the contract's state.

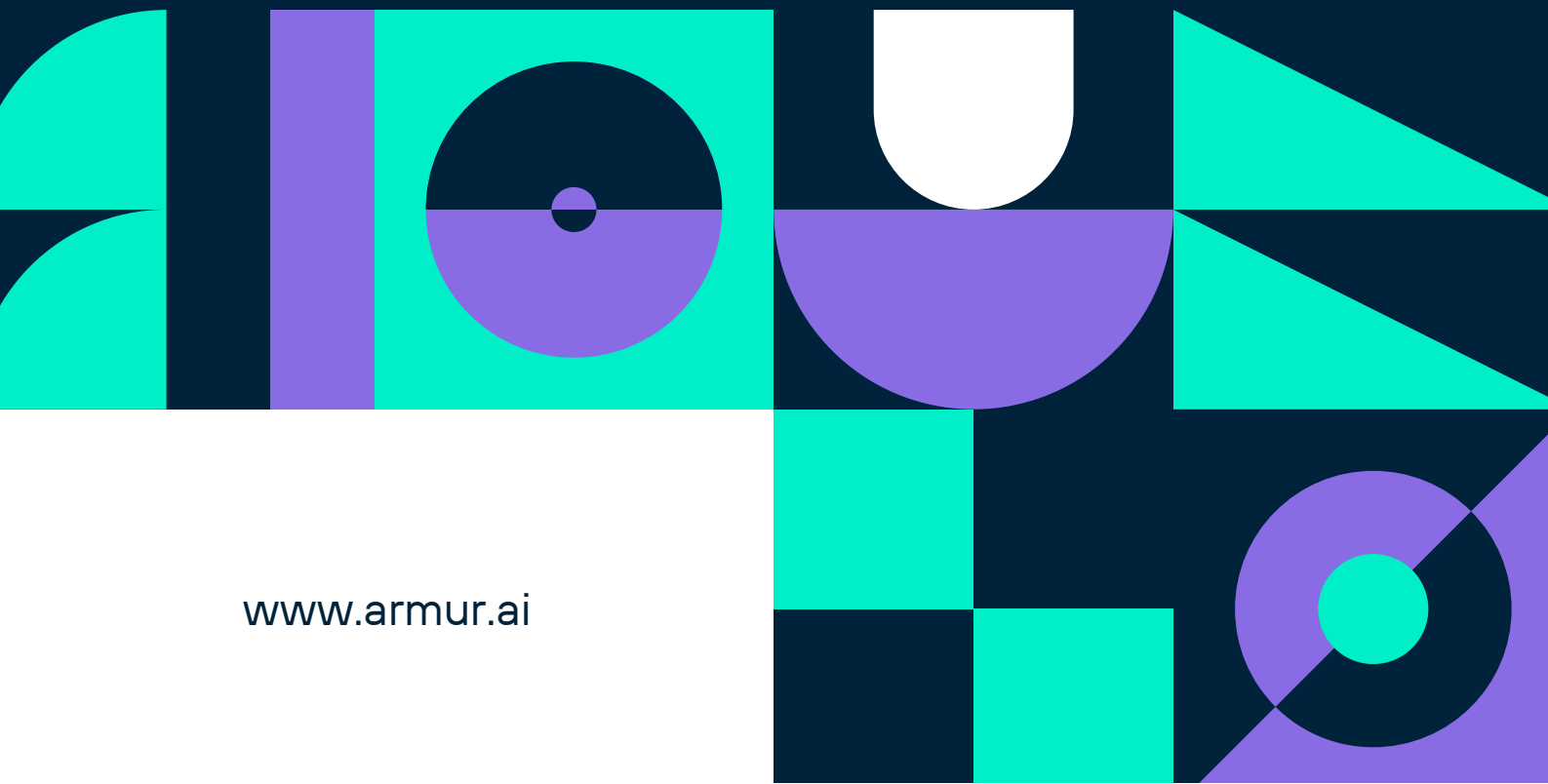
ARMUR

- **Lack of Access Controls:** The contract does not have any access control mechanisms, which means that any user can call any function on the contract, potentially leading to unauthorized changes in the contract's state.

- **Gas Limitations**: The contract does not consider the gas limit for transactions, which could lead to out-of-gas errors if a transaction consumes too much gas. This could cause the transaction to fail, resulting in a loss of funds or other undesirable behavior.

- **Lack of Comprehensive Testing:** The contract lacks a comprehensive testing suite, which could leave undiscovered vulnerabilities that could be exploited by attackers.

Thanks.



www.armur.ai