Smart Contract Audit Report

MUX-Preps

Audit Performed By

Fortknox Security

Professional Smart Contract Auditing

January 26, 2025

Table of Contents

Executive Summary

Fortknox Security has conducted a comprehensive smart contract security audit for **MUX-Preps**. Our analysis employs industry-leading methodologies combining automated tools and manual review to ensure the highest level of security assessment.

9	1	LOW	100%
TOTAL ISSUES FOUND	CRITICAL + HIGH	OVERALL RISK	CODE COVERAGE

Security Assessment Overview

Critical Issues

0

Immediate action required. These vulnerabilities can lead to direct loss of funds.

IMPACT: SEVERE FINANCIAL LOSS

High Issues

1

High priority fixes needed. Can lead to significant financial loss.

IMPACT: MAJOR SECURITY RISK

Key Findings Summary

Access Control

Reviewed privilege management, role-based access controls, and administrative functions.

Economic Security

Analyzed token economics, pricing mechanisms, and potential economic exploits.

Logic Validation

Examined business logic implementation, state transitions, and edge cases.

Input Validation

Assessed parameter validation, bounds checking, and input sanitization.

Audit Conclusion

The MUX-Preps smart contract audit reveals **9 total findings** across various security categories. Immediate attention is required for 1 critical/high severity issues before deployment. Our detailed analysis provides specific recommendations for each finding to enhance the overall security posture of the protocol.

Audit Methodology

Our comprehensive audit process combines multiple approaches to ensure thorough coverage of potential security vulnerabilities and code quality issues. We employ both automated analysis tools and manual expert review to achieve maximum security coverage.

Tools & Techniques

Static Analysis

Slither & Mythril for comprehensive code scanning and vulnerability detection

Manual Review

Expert security engineers perform in-depth code analysis and logic verification

Business Logic

Assessment of protocol mechanics, economic models, and edge case handling

Gas Analysis

Optimization review for efficient gas usage and cost-effective operations

Formal Verification

Mathematical proof methods to verify critical contract properties

Symbolic Execution

Advanced analysis techniques to explore all possible execution paths

Review Process & Standards

Review Process

1 Initial Scanning

Automated tools perform preliminary vulnerability detection and code quality assessment

2 Manual Review

Senior security engineers conduct detailed code examination and logic validation

3 Business Logic Testing

Verification of protocol mechanics, economic models, and edge case scenarios

4 Architecture Analysis

Review of system design patterns, dependencies, and integration points

5 Final Documentation

Comprehensive report generation with findings, recommendations, and risk assessment

Severity Classification

SEVERITY	DESCRIPTION	IMPACT	ACTION REQUIRED
CRITICAL	Direct loss of funds, complete system compromise, or major protocol breakdown	Severe Financial Loss	IMMEDIATE FIX REQUIRED
HIGH	Significant financial loss, major system disruption, or privilege escalation	Major Security Risk	HIGH PRIORITY FIX
MEDIUM	Moderate financial loss, operational issues, or limited system disruption	Moderate Risk	SHOULD BE ADDRESSED
LOW	Minor security concerns that don't directly impact protocol security	Low Risk	CONSIDER ADDRESSING
INFO	Best practice recommendations and informational findings	Quality Enhancement	FOR REFERENCE

Audit Scope

Project Details

PARAMETER	DETAILS
Project Name	MUX-Preps
Total Issues Found	9
Audit Type	Smart Contract Security Audit
Methodology	Manual Review + Automated Analysis

Files in Scope

This audit covers the smart contract codebase and associated components for MUX-Preps.

Audit Timeline

- ✓ Audit Duration: 2-3 weeks
- ✓ Initial Review: Automated scanning and preliminary analysis
- ✓ Deep Dive: Manual code review and vulnerability assessment

Vulnerability Analysis

Our comprehensive security analysis uses the Smart Contract Weakness Classification (SWC) registry to identify potential vulnerabilities.

SWC Security Checks

CHECK ID	DESCRIPTION	STATUS
SWC-100	Function Default Visibility	PASSED
SWC-101	Integer Overflow and Underflow	PASSED
SWC-102	Outdated Compiler Version	PASSED
SWC-103	Floating Pragma	PASSED
SWC-104	Unchecked Call Return Value	PASSED
SWC-105	Unprotected Ether Withdrawal	PASSED
SWC-106	Unprotected SELFDESTRUCT	PASSED
SWC-107	Reentrancy	WARNING

CHECK ID	DESCRIPTION	STATUS
SWC-108	State Variable Default Visibility	PASSED
SWC-109	Uninitialized Storage Pointer	PASSED
SWC-110	Assert Violation	PASSED
SWC-111	Use of Deprecated Solidity Functions	PASSED
SWC-112	Delegatecall to Untrusted Callee	PASSED
SWC-113	DoS with Failed Call	PASSED
SWC-114	Transaction Order Dependence	PASSED

Contract Privileges Analysis

Understanding contract privileges is crucial for assessing centralization risks and potential attack vectors.

Common Privilege Categories

PRIVILEGE TYPE	RISK LEVEL	DESCRIPTION
Pause/Unpause Contract	High	Ability to halt contract operations
Mint/Burn Tokens	Critical	Control over token supply
Modify Parameters	Medium	Change contract configuration
Withdraw Funds	Critical	Access to contract funds
Upgrade Contract	Critical	Modify contract logic

Mitigation Strategies

- ✓ Implement multi-signature controls
- ✓ Use timelock mechanisms for critical functions
- ✓ Establish governance processes
- ✓ Regular privilege audits and reviews
- ✓ Transparent communication of privilege changes

H-0 | Delegators Cannot Cancel All Orders

CATEGORY	SEVERITY	LOCATION	STATUS
Logical Error	HIGH	LibOrderBook.sol	Resolved

Description

The `Delegator.cancel()` function allows delegators to cancel orders on behalf of the delegation's owner. This works fine for position orders, because `LibOrderBook._cancelPositionOrder` allows the delegator to execute the certain action.

```
Delegator.cancel()
LibOrderBook._cancelPositionOrder
```

Recommendation

Add the `isDelegator` check to `_cancelLiquidityOrder` and `_cancelWithdrawalOrder` functions.

```
isDelegator
_cancelLiquidityOrder
_cancelWithdrawalOrder
```

Resolution

MUX Team: Resolved.

M-0 | No Post-Deposit Safety Check In depositCollateral Function

CATEGORY	SEVERITY	LOCATION	STATUS
Configuration	MEDIUM	LibOrderBook.sol: 864	Acknowledged

Description

The `depositCollateral` function allows users to add collateral to an existing position but fails to verify whether the position is then sufficiently collateralized or safe from liquidation immediately afterward.

```
depositCollateral
```

Recommendation

Include a safety check after the collateral deposit to confirm that the position's margin requirements are now satisfied. If the position is still unsafe, revert the transaction.

Resolution

MUX Team: Acknowledged.

M-1 | WithdrawUsd May Try To Withdraw 0

CATEGORY	SEVERITY	LOCATION	STATUS
Logical Error	MEDIUM	FacetPositionAccount.sol	Resolved

Description

`FacetPositionAccount.withdrawUsd` calculates the amount of `payingCollateral` to withdraw from the user's account by calling `_withdrawFromAccount`. It's possible for `payingCollateral` to result in 0 because of rounding if the user has small amount of collateral (may happen naturally by realizing negative PnL).

```
FacetPositionAccount.withdrawUsd
payingCollateral
_withdrawFromAccount
payingCollateral
```

Recommendation

Skip the current iteration if the amount of collateral to be withdrawn is 0.

Resolution

MUX Team: Resolved.

M-2 | MAX_COLLATERALS_PER_POSITION_ACCOUNT Can Be Bypassed

CATEGORY	SEVERITY	LOCATION	STATUS
Logical Error	MEDIUM	Market.sol: 249	Acknowledged

Description

When closing a position, collaterals are added to the user's account, without validation. If a user only partially closes their account, they will then have all of these collaterals as active.

Recommendation

If a user is not fully closing their position, validate that they are not exceeding the `MAX_COLLATERALS_PER_POSITION_ACCOUNT`.

```
MAX_COLLATERALS_PER_POSITION_ACCOUNT
```

Resolution

MUX Team: Acknowledged.

M-3 | ADL Is Triggered Per Pool

CATEGORY	SEVERITY	LOCATION	STATUS
Configuration	MEDIUM	FacetReader: 230	Acknowledged

Description

When `fillADLOrder` is called, it validates that the position is viable for an auto deleverage. This is accomplished in `isDeleverageAllowed` function when it checks if any of the pools that the user has their position in is above the trigger PnL threshold.

```
fillADLOrder
isDeleverageAllowed
```

Recommendation

Validate that the entire position is in a profitable enough state to trigger the auto deleverage.

Resolution

MUX Team: Acknowledged.

M-4 | Zero Balance Active Collaterals

CATEGORY	SEVERITY	LOCATION	STATUS
Configuration	MEDIUM	PositionAccount.sol: 47-55	Resolved

Description

The `_depositToAccount` function checks the raw amount is not zero, converts raw amount to `wad` amounts, and then adds collateral to the active collaterals. However, if the collateral's decimal value is higher than 18, `wad` amount can be zero while the raw amount is not zero.

```
_depositToAccount
wad
wad
```

Recommendation

Check the `wad` amount is not zero as well before adding collateral to the active collaterals.

```
wad
```

Resolution

MUX Team: Resolved.

L-0 | Missing ReentrancyGuard Initialization

CATEGORY	SEVERITY	LOCATION	STATUS
Code Best Practices	LOW	OrderBook.sol	Resolved

Description

The `OrderBook` contract inherits from `OpenZeppelin's ReentrancyGuardUpgradeable` but never calls the `__ReentrancyGuard_init` function in its `initialize` method.

```
OrderBook
OpenZeppelin's ReentrancyGuardUpgradeable
__ReentrancyGuard_init
initialize
```

Recommendation

Within the `initialize` function, ensure to invoke the `ReentrancyGuard` initializer:

```
initialize
ReentrancyGuard
```

Resolution

MUX Team: Resolved.

L-1 | Some Orders Cannot Be Paused

CATEGORY	SEVERITY	LOCATION	STATUS
Configuration	LOW	OrderBook.sol	Resolved

Description

When orders are placed and filled in the `OrderBook`, the `whenNotPaused` modifier is executed first. It will revert if the current order type is paused. Currently, `_isOrderPaused` doesn't support `Rebalance` and `Adl` orders.

```
OrderBook
whenNotPaused
_isOrderPaused
Rebalance
Adl
```

Recommendation

Make the `isOrderPaused()` function fetch paused configuration for `Rebalance` and `Adl` orders as well.

```
isOrderPaused()
Rebalance
Adl
```

Resolution

MUX Team: Resolved.

L-2 | Rebalance Orders Cannot Be Canceled

CATEGORY	SEVERITY	LOCATION	STATUS
Configuration	LOW	LibOrderBook.sol	Resolved

Description

The `LibOrderBook.cancel()` function should be able to cancel placed orders. Currently there is not support for canceling a rebalance order. Because of this any expired rebalance orders will stay forever in the contract state.

```
LibOrderBook.cancel()
```

Recommendation

Consider allowing the broker to cancel rebalance orders.

Resolution

MUX Team: Resolved.

Summary of Recommendations

Based on our comprehensive audit, we provide the following prioritized recommendations to improve the security posture of MUX-Preps.

Priority Matrix

ISSUE ID	TITLE	SEVERITY	PRIORITY
H-0	Delegators Cannot Cancel All Orders	HIGH	High
M-0	No Post-Deposit Safety Check In depositCollateral Function	MEDIUM	Medium
M-1	WithdrawUsd May Try To Withdraw 0	MEDIUM	Medium
M-2	MAX_COLLATERALS_PER_POSITION_ACCOUNT Can Be Bypassed	MEDIUM	Medium
M-3	ADL Is Triggered Per Pool	MEDIUM	Medium
M-4	Zero Balance Active Collaterals	MEDIUM	Medium
L-0	Missing ReentrancyGuard Initialization	LOW	Low
L-1	Some Orders Cannot Be Paused	LOW	Low
L-2	Rebalance Orders Cannot Be Canceled	LOW	Low

General Security Best Practices

- ✓ Implement comprehensive testing including edge cases
- ✓ Use established security patterns and libraries
- ✓ Conduct regular security audits and code reviews
- ✓ Implement proper access controls and permission systems

Audit Team

Team Credentials

Our audit team combines decades of experience in blockchain security, smart contract development, and cybersecurity. Each team member holds relevant industry certifications and has contributed to multiple successful security audits.

Methodology & Standards

Our audit methodology follows industry best practices and standards:

- ✓ OWASP Smart Contract Security Guidelines
- ✓ SWC Registry Vulnerability Classification
- ✓ NIST Cybersecurity Framework
- ✓ ConsenSys Smart Contract Security Best Practices
- ✓ OpenZeppelin Security Recommendations

Audit Process

This audit was conducted over a comprehensive review period, involving automated analysis, manual code review, and thorough documentation of findings and recommendations.

Disclaimer & Legal Notice

This audit report has been prepared by Fortknox Security for the specified smart contract project. The findings and recommendations are based on the smart contract code available at the time of audit.

Scope Limitations

- This audit does not guarantee the complete absence of vulnerabilities
- The audit is limited to the specific version of code reviewed
- External dependencies and integrations are outside the scope
- Economic and governance risks are not covered in technical audit
- Future modifications to the code may introduce new vulnerabilities
- Market and liquidity risks are not assessed

Liability Statement

Fortknox Security provides this audit report for informational purposes only. We do not provide any warranties, express or implied, regarding:

- The absolute security of the smart contract
- The economic viability of the project
- The legal compliance in any jurisdiction
- Future performance or behavior of the contract
- Third-party integrations or dependencies

Legal Terms & Usage Rights

Usage Rights

This audit report may be used by the client for:

- ✓ Public disclosure and transparency
- ✓ Marketing and promotional materials
- ✓ Investor due diligence processes
- ✓ Regulatory compliance documentation
- ✓ Technical documentation and reference
- ✓ Security assessment presentations
- ✓ Community transparency initiatives

Restrictions

The following restrictions apply to this report:

- ✓ Report content may not be modified or altered
- ✓ Fortknox Security branding must remain intact
- ✓ Partial excerpts must maintain context and accuracy
- ✓ Commercial redistribution requires written permission
- ✓ Translation must preserve technical accuracy

Intellectual Property

This report contains proprietary methodologies and analysis techniques developed by Fortknox Security. The format, structure, and analytical approach are protected intellectual property.

Contact Information

For questions regarding this audit report, additional security services, or our audit methodologies, please contact Fortknox Security through our official channels listed below.

Fortknox Security

🌐 https://www.fortknox-security.xyz

🐦 @FortKnox_sec

📧 support@fortknox-security.xyz

FORTKNOX SECURITY

Web3 Security at Fort Knox Level

Contact Us

🐦 @FortKnox_sec

📱 @FortKnox_sec

🌐 fortknox-security.xyz

📧 support@fortknox-security.xyz

Audit performed by
Fortknox Security