Audited by CRED SHIELDS

CredShields

Smart Contract Audit

June 2nd, 2025 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of DOTLABS LTD. between April 25th, 2025, and May 9th, 2025. A retest was performed on May 29th, 2025.

Author

Shashank (Co-founder, CredShields) shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead), Shreyas Koli(Auditor), Naman Jain (Auditor), Sanket Salavi (Auditor), Yash Shah (Auditor)

Prepared for

DOTLABS LTD.

Table of Contents

1. Executive Summary ----------

DOTLABS LTD. engaged CredShields to perform a smart contract audit from April 25th, 2025, to May 9th, 2025. During this timeframe, 14 vulnerabilities were identified. **A retest was performed on May 29th, 2025, and all the bugs have been addressed.**

During the audit, 4 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "DOTLABS LTD." and should be prioritized for remediation.

The table below shows the in-scope assets and a breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Σ
Mushi V2.0 Contracts	1	3	4	6	0	**14**
	1	**3**	**4**	**6**	**0**	**14**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Mushi V2.0 Contract's scope during the testing window while abiding by the policies set forth by DOTLABS LTD.'s team.

State of Security

To maintain a robust security posture, it is essential to continuously review and improve upon current security processes. Utilizing CredShields' continuous audit feature allows both DOTLABS LTD.'s internal security and development teams to not only identify specific vulnerabilities but also gain a deeper understanding of the current security threat landscape.

To ensure that vulnerabilities are not introduced when new features are added, or code is refactored, we recommend conducting regular security assessments. Additionally, by analyzing the root cause of resolved vulnerabilities, the internal teams at DOTLABS LTD. can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, DOTLABS LTD. can future-proof its security posture and protect its assets.

2. The Methodology ----------

DOTLABS LTD. engaged CredShields to perform a Mushi V2.0 Smart Contract audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation Phase

The CredShields team meticulously reviewed all provided documents and comments in the smart contract code to gain a thorough understanding of the contract's features and functionalities. They meticulously examined all functions and created a mind map to systematically identify potential security vulnerabilities, prioritizing those that were more critical and business-sensitive for the refactored code. To confirm their findings, the team deployed a self-hosted version of the smart contract and performed verifications and validations during the audit phase.

A testing window from April 25th, 2025, to May 9th, 2025, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed upon:

IN SCOPE ASSETS
https://github.com/0xmushidefi/mushi-v0.2/tree/f79827ec7ab57b34bcac99aabbfc8931555f415a

2.1.2 Documentation

Documentation was not required as the code was self-sufficient for understanding the project.

2.1.3 Audit Goals

CredShields employs a combination of in-house tools and manual methodologies to conduct thorough security audits for Rust-based smart contracts. The audit process primarily involves manually reviewing the contract's source code, following best practices for Rust and WebAssembly (Wasm) development, and leveraging an internally developed, industry-aligned checklist. The team focuses on understanding key concepts, creating targeted test cases, and analyzing business logic to identify potential vulnerabilities.

2.2 Retesting Phase

DOTLABS LTD. is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat agents, Vulnerability factors, and Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	None	Low	Medium
		LOW	MEDIUM	HIGH
Likelihood				

Overall, the categories can be defined as described below -

1. **Informational**

   We prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do not pose a direct risk to the contract. Code maintainers should use their own judgment on whether to address them.

2. **Low**

   Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

3. **Medium**

   Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

4. **High**

   High-severity vulnerabilities pose a significant risk to the Smart Contract and the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

5. **Critical**

   Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds and Ether from Smart Contracts or users and put sensitive user information at risk of compromise or modification. The client's reputation and financial stability will be severely impacted if these issues are not addressed immediately.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- Shashank, Co-founder CredShields shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have about the engagement or this document.

3. Findings Summary ----------

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by the asset and severity. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, 14 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	Vulnerability Type
Users can Withdraw Collateral without Repayment	Critical	Logic Error
Daily Liquidation Miscalculates Borrowed and Collateral when Multiple Days Elapse	High	State Accounting Inconsistency
Buyers can Pay Incorrect Fees due to Inconsistent Fee Validation	High	Inconsistent Validation
User can Repay Expired Loan Without Validation	High	Missing Validation
Buyers and Sellers can Suffer Unexpected Losses due to Missing Slippage Protection	Medium	Missing Slippage Protection
Users can Illegitimately Claim Referral Rewards	Medium	Missing Account Validation
Valid Referral Fee Reverts at Minimum Threshold	Medium	Business Logic Issue
Missing MAX_SUPPLY Validation when Minting Base Token	Medium	Business Logic Issue

Once started is set to true, it cannot be changed back to false	Low	State Immutability
Missing toolchain Version In Anchor.toml	Low	Environment Misconfiguration
Protocol State can be Mutated Without Any Effective Change	Low	Unnecessary State Change
Invalid Fee Parameters can Be Set During Initialization	Low	Missing Input Validation
Missing Default Address Validation	Low	Missing Input Validation
Contract Lacks Secure Ownership Transfer Mechanism	Low	Missing Authorization Control

Table: Findings in Smart Contracts

4. Remediation Status ----------

DOTLABS LTD. is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on May 29th, 2025, and all the issues have been addressed.**

Also, the table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Users can Withdraw Collateral without Repayment	Critical	**Fixed** [May 29, 2025]
Daily Liquidation Miscalculates Borrowed and Collateral when Multiple Days Elapse	High	**Fixed** [May 29, 2025]
Buyers can Pay Incorrect Fees due to Inconsistent Fee Validation	High	**Fixed** [May 29, 2025]
User can Repay Expired Loan Without Validation	High	**Fixed** [May 29, 2025]
Buyers and Sellers can Suffer Unexpected Losses due to Missing Slippage Protection	Medium	**Fixed** [May 29, 2025]
Users can Illegitimately Claim Referral Rewards	Medium	**Fixed** [May 29, 2025]
Valid Referral Fee Reverts at Minimum Threshold	Medium	**Fixed** [May 29, 2025]
Missing MAX_SUPPLY Validation when Minting Base Token	Medium	**Fixed** [May 29, 2025]
Once started is set to true, it cannot be changed back to false	Low	**Fixed** [May 29, 2025]
Missing toolchain Version In Anchor.toml	Low	**Fixed** [May 29, 2025]
Protocol State can be Mutated Without Any Effective Change	Low	**Fixed** [May 29, 2025]

Invalid Fee Parameters can Be Set During Initialization	Low	**Fixed** [May 29, 2025]
Missing Default Address Validation	Low	**Fixed** [May 29, 2025]
Contract Lacks Secure Ownership Transfer Mechanism	Low	**Won't Fix** [May 29, 2025]

Table: Summary of findings and status of remediation

5. Bug Reports ----------

Bug ID #1 [Fixed]

Users can Withdraw Collateral without Repayment

Vulnerability Type
Logic Error

Severity
Critical

Description
In the remove_collateral flow, the loan health check is implemented as a require! condition that compares user_loan.borrowed against the converted value of (collateral - amount) * 99, using ctx.accounts.common.mushi_to_eclipse. The issue arises because this multiplication by 99 inflates the collateral amount before conversion, making it appear 99 times more valuable than it actually is. This causes the check to always succeed, even when the real collateral is insufficient. As a result, users can call remove_collateral and withdraw more than allowed while still owing on their loan.

Affected Code

- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/remove_collateral.rs#L31

Impacts
Users can withdraw nearly all of their collateral without proper checks, leaving loans severely under-collateralized and exposing the protocol to systemic insolvency.

Remediation
Apply the 99/100 reduction after the mushi_to_eclipse conversion, ensuring accurate comparison between actual collateral and borrowed value.

```
- ctx.accounts.common.mushi_to_eclipse((collateral - amount) * 99)? / 100
+ ctx.accounts.common.mushi_to_eclipse(collateral - amount)? * 99 / 100
```

Retest

This issue has been fixed as per suggested remediation.

Bug ID #2 [Fixed]

Daily Liquidation Miscalculates Borrowed and Collateral when Multiple Days Elapse

Vulnerability Type
State Accounting Inconsistency

Severity
High

Description
The liquidate function is intended to be called daily and performs a loop over days between the last liquidation date and the current timestamp. However, the function receives only a single instance of DailyStats(last_liquidation_date_state), which corresponds to the PDA for only one specific date — typically the current last_liquidation_date.
If multiple days have elapsed without calling liquidate, the while-loop continues updating global_state.last_liquidation_date while repeatedly reading the same last_liquidation_date_state values. This means the function aggregates the same borrowed and collateral values across all pending days, instead of retrieving each unique day's DailyStats. As a result:

- The total borrowed and collateral values are multiplied by the number of days missed.
- The global_state.total_borrowed and global_state.total_collateral are reduced excessively, leading to underflow or inaccurate protocol accounting.
- Since DailyStats is stored in a PDA using a derived address tied to the liquidation date, correct behavior would require loading and handling a different PDA account for each day — but the function currently receives only one.

Affected Code

- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/utils.rs#L155-L191

Impacts
If the liquidate function is skipped for multiple days, the system inaccurately processes liquidation by reusing the same day's statistics. This results in the repeated subtraction of identical collateral and borrowed values from the global state, leading to incorrect token burning, misreported debt levels, and potential underflow conditions.

Remediation
Modify the liquidation logic to increment last_liquidation_date_state by one day and ensure that last_liquidation_date corresponds to that exact day. This adjustment enforces liquidation in daily increments and avoids repeated application of the same DailyStats values.

Retest
This issue has been fixed by liquidating one day at a time.

Client's comment: I already suggested to use small service (node.js backend) can call liquidate function every day.

Bug ID #3 [Fixed]

Buyers can Pay Incorrect Fees due to Inconsistent Fee Validation

Vulnerability Type
Inconsistent Validation

Severity
High

Description
The buy and buy_with_referral functions in the smart contract handle fee calculations differently, leading to inconsistent validation. In the buy function, the total fee (FEES_BUY + FEES_BUY_REFERRAL) is validated once, ensuring that the combined fee exceeds the minimum threshold (MIN). However, in buy_with_referral, the treasury fee (FEES_BUY) and referral fee (FEES_BUY_REFERRAL) are validated separately, potentially allowing transactions where one fee is below MIN while the other is not.

The issue arises because the buy function checks the total fee (fee > MIN) but does not enforce individual minimums for the treasury and referral portions. Meanwhile, buy_with_referral enforces fee_treasury > MIN and fee_referral > MIN separately, which could lead to discrepancies in fee enforcement.

Affected Code
- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/buy_sell.rs#L48
- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/buy_sell.rs#L134
- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/buy_sell.rs#L135

Impacts
This inconsistency may result in buyers paying incorrect fees, either underpaying or overpaying due to improper validation. If the referral fee (FEES_BUY_REFERRAL) is too small but the treasury fee (FEES_BUY) is valid, the buy_with_referral function will revert, while the buy function would allow the transaction. This could lead to unfair fee distribution, loss of expected revenue for referrers, or unintended transaction failures when switching between the two functions.

Remediation

The remediation involves standardizing the fee validation logic between the buy and buy_with_referral functions to ensure consistent enforcement of minimum fees. The primary issue stems from the buy function validating only the combined fee (FEES_BUY + FEES_BUY_REFERRAL), while buy_with_referral checks the treasury and referral fees separately. To resolve this, both functions should first compute the total fee and verify that it meets the minimum threshold (MIN), ensuring no underpayment occurs.

Retest

This issue has been fixed by combining both fees into one total fee.

Bug ID #4 [Fixed]

User can Repay Expired Loan Without Validation

Vulnerability Type
Missing Validation

Severity
High

Description
In the repay function, the contract allows repayment of any loan by accessing user_loan.borrowed without validating whether the loan has expired. This omission permits users to bypass consequences associated with overdue loans, such as penalties or liquidation, undermining enforcement of loan terms.

Affected Code

- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/repay.rs#L13-L40

Impacts
Users can continue repaying expired loans without penalty, weakening the protocol's lending discipline and exposing it to potential bad debt.

Remediation
Add a check to ensure the loan is not expired.

Retest
This issue has been fixed by validating if loan is expired or not.

Bug ID #5 [Fixed]

Buyers and Sellers can Suffer Unexpected Losses due to Missing Slippage Protection

Vulnerability Type
Missing Slippage Protection

Severity
Medium

Description
The contract lacks slippage protection mechanisms in the buy, buy_with_referral, and sell functions, which are critical for ensuring users receive a fair exchange rate when trading tokens.
In the buy and buy_with_referral functions, the contract calculates the amount of mushi tokens a user receives based on the current exchange rate (eclipse_to_mushi), but there is no check to ensure that the final amount does not deviate beyond an acceptable threshold due to price fluctuations. Similarly, in the sell function, the contract converts mushi tokens to eclipse (mushi_to_eclipse) without enforcing a minimum expected output.
Without slippage checks, users may receive significantly fewer tokens than anticipated if the price changes unfavorably between transaction submission and execution, particularly in volatile market conditions or when liquidity is low.

Affected Code
- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/buy_sell.rs#L15
- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/buy_sell.rs#L90
- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/buy_sell.rs#L185

Impacts
Users may suffer unexpected and substantial financial losses due to unfavorable price movements during trade execution.

Remediation
To mitigate this issue, the contract should implement slippage protection by allowing users to specify minimum expected amounts when executing trades.

For buy and buy_with_referral, users should provide a min_mushi_out parameter, and the contract should verify that the received amount meets or exceeds this threshold before completing the transaction. Similarly, for sell, users should provide a min_eclipse_out parameter to ensure they receive an acceptable amount of the output token.

Retest
This issue has been fixed by adding minimum output validation.

Bug ID #6 [Fixed]

Users can Illegitimately Claim Referral Rewards

Vulnerability Type
Missing Account Validation

Severity
Medium

Description
In the buy_with_referral function, the referral fee is transferred to referral_quote_ata without verifying whether the referral address is distinct from the transaction initiator. This allows a user to pass their own address as the referral_address, effectively redirecting the referral reward back to themselves. The function lacks a critical validation step to prevent this self-referral loophole, thereby allowing users to exploit the system by earning unearned rewards.

Affected Code

- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/context/common.rs#L250

Impacts
Users can repeatedly claim referral incentives without referring others, draining protocol funds, bypassing intended reward mechanics, and undermining the integrity of the referral system.

Remediation
Ensure the referral address is not equal to the signer's address.

Retest
This issue has been fixed adding a validation for signer is not referral.

Bug ID #7 [Fixed]

Valid Referral Fee Reverts at Minimum Threshold

Vulnerability Type
Business Logic Issue

Severity
Medium

Description
In the buy_with_referral function, the referral fee is validated using the condition if fee_referral <= MIN, which causes the function to revert even when the calculated fee_referral is exactly equal to MIN. This condition incorrectly treats a valid minimum fee as invalid. As a result, legitimate referral transactions where the fee meets the exact minimum requirement will fail, disrupting user experience and referral incentives.

Affected Code

- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/buy_sell.rs#L135

Impacts
Users whose referral fee equals the configured minimum are blocked from completing valid purchases, potentially losing rewards and damaging trust in the referral program.

Remediation
Adjust the condition to reject only fees strictly below the minimum. Use '=>' instead of '>'.

Retest
This issue has been fixed by updating the strict condition with non strict conditional check.

Bug ID #8 [Fixed]

Missing MAX_SUPPLY Validation when Minting Base Token

Vulnerability Type
Business Logic Issue

Severity
Medium

Description
The system defines a fixed MAX_SUPPLY of 10e28 tokens, but the minting logic does not enforce this constraint during token issuance. Specifically, none of the minting functions validate that total_supply + mint_amount <= MAX_SUPPLY before minting. This omission permits exceeding the intended token cap, especially when minting occurs in multiple stages or via composable calls, undermining the tokenomics model.

Affected Code

- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/utils.rs#L19-L40

Impacts
An actor with minting privileges can inflate the total token supply beyond MAX_SUPPLY, leading to economic instability, loss of stakeholder trust, and deviation from the platform's tokenomics.

Remediation
Enforce a check that prevents minting if the new total would exceed MAX_SUPPLY.

Retest
This issue has been fixed in every file by adding MAX_SUPPLY checks.

Bug ID #9 [Fixed]

Once started is set to true, it cannot be changed back to false

Vulnerability Type
State Immutability

Severity
Low

Description
The start function in the smart contract permanently sets global_state.started = true without any mechanism to reverse or pause the contract's operational state. Once initialized, the contract remains in a "started" state indefinitely, preventing administrators from halting operations even in emergencies, such as detected vulnerabilities, market manipulation, or critical failures.

Affected Code

- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/start.rs#L59

Impacts
The inability to pause or deactivate the contract poses significant risks. If a bug is discovered after initialization, the protocol cannot be frozen to prevent further damage, potentially leading to unchecked financial losses, exploitation, or governance failures.

Remediation
The contract should implement a way to toggle the started state between true and false. This can be achieved by adding a new privileged function like set_started(bool) that allows the admin to change the contract's operational state.

Retest
This issue has been fixed by adding a toggle feature for start.

Bug ID #10 [Fixed]

Missing toolchain Version In Anchor.toml

Vulnerability Type
Environment Misconfiguration

Severity
Low

Description
The Anchor.toml file lacks explicit anchor_version and solana_version declarations under the [toolchain] section. This omission can lead to version drift across different development environments, potentially introducing unexpected behavior, compilation errors, or inconsistencies during deployment. Without a fixed toolchain version, CI/CD pipelines and team members may inadvertently use incompatible versions of Anchor or Solana.

Affected Code
- Anchor.toml

Impacts
Builds and deployments may fail or behave inconsistently across machines, reducing reliability and increasing debugging overhead.

Remediation
Explicitly define the Anchor and Solana versions in the [toolchain] section of Anchor.toml.

Retest
This issue has been fixed by adding a toolchain version.

Bug ID #11 [Fixed]

Protocol State can be Mutated Without Any Effective Change

Vulnerability Type
Unnecessary State Change

Severity
Low

Description
In the update_main_state function, state fields are reassigned even when new values match the current ones. This results in a write operation to the main_state account even when no change occurs. Such redundant writes increase transaction costs and consume compute budget without delivering any functional benefit. In worst cases, this behavior can inadvertently trigger downstream state-change-dependent logic (e.g., indexers, watchers, hooks) even though the state remains semantically unchanged.

Affected Code

- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/update_main_state.rs#L18-L57

Impacts
Redundant state writes increase transaction cost and may trigger unnecessary events or off-chain effects, reducing overall protocol efficiency.

Remediation
Update each field only if the incoming value differs from the current one.

Retest
This issue has been fixed by updating the update_main_state function to prevent redundant writes.

Bug ID #12 [Fixed]

Invalid Fee Parameters can Be Set During Initialization

Vulnerability Type
Missing Input Validation

Severity
Low

Description
The init_main_state function lacks validation for buy_fee, sell_fee, and buy_fee_leverage, allowing arbitrary values to be set during contract initialization. This is inconsistent with the update_main_state function, which enforces constraints (e.g., 975 ≤ fee ≤ 992, buy_fee_leverage ≤ 25). Without equivalent checks, the protocol may be initialized in an invalid or dangerous configuration, potentially bypassing safeguards intended to protect users and economic assumptions.

Affected Code
- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/init_main_state.rs#L20-L22

Impacts
The contract can be deployed with excessive fees or leverage parameters, harming users and destabilizing protocol behavior.

Remediation
Add the same fee range checks to init_main_state as used in update_main_state.

Retest
This issue has been fixed by adding fee range checks in init_main_state.

Bug ID #13 [Fixed]

Missing Default Address Validation

Vulnerability Type
Missing Input Validation

Severity
Low

Description
In the init_main_state function, the fee_receiver field is set directly from user input without validating that the address is non-default (i.e., not Pubkey::default()). If left unset or intentionally set to the default address, transaction fees could be irreversibly sent to an unusable or incorrect destination, causing loss of protocol revenue.

Affected Code

- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/init_main_state.rs#L18

Impacts
Fees collected by the protocol may be burned or redirected to an inaccessible address, resulting in financial loss and misallocation of protocol income.

Remediation
Ensure the fee_receiver is not the default address.

Retest
This issue has been fixed by validating the fee_receiver's address.

Bug ID #14 [Won't Fix]

Contract Lacks Secure Ownership Transfer Mechanism

Vulnerability Type
Missing Authorization Control

Severity
Low

Description
The contract does not implement a two-step ownership transfer process, preventing the current admin from securely delegating control to a new address. A safe transfer pattern requires the current owner to nominate a pending_admin, which the new address must explicitly accept to finalize the transition. Without this mechanism, accidental or unauthorized ownership changes may occur, or future admin rotations may require redeployments.

Affected Code

- https://github.com/0xmushidefi/mushi-v0.2/blob/f79827ec7ab57b34bcac99aabbfc8931555f415a/programs/mushi_program/src/ixs/update_main_state.rs#L23

Impacts
The protocol lacks flexibility in administrative management, and ownership transitions are prone to misconfiguration or require manual intervention outside of protocol logic.

Remediation
Introduce a two-step admin transfer flow using pending_admin and explicit acceptance.

Retest
The code still allows insecure, immediate transfer of admin rights without requiring acceptance from the new admin.

Client's comment: We are good to go with one-step ownership transfer

6. The Disclosure ----------

The Reports provided by CredShields are not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields' Audit team is not responsible for any decisions or actions taken by any third party based on the report.

YOUR SECURE FUTURE STARTS HERE

CRED SHIELDS

At CredShields, we're more than just auditors. We're your strategic partner in ensuring a secure Web3 future. Our commitment to your success extends beyond the report, offering ongoing support and guidance to protect your digital assets

Audited by CRED SHIELDS

20A TANJONG PAGAR ROAD, SINGAPORE (088443)

info@credshields.com