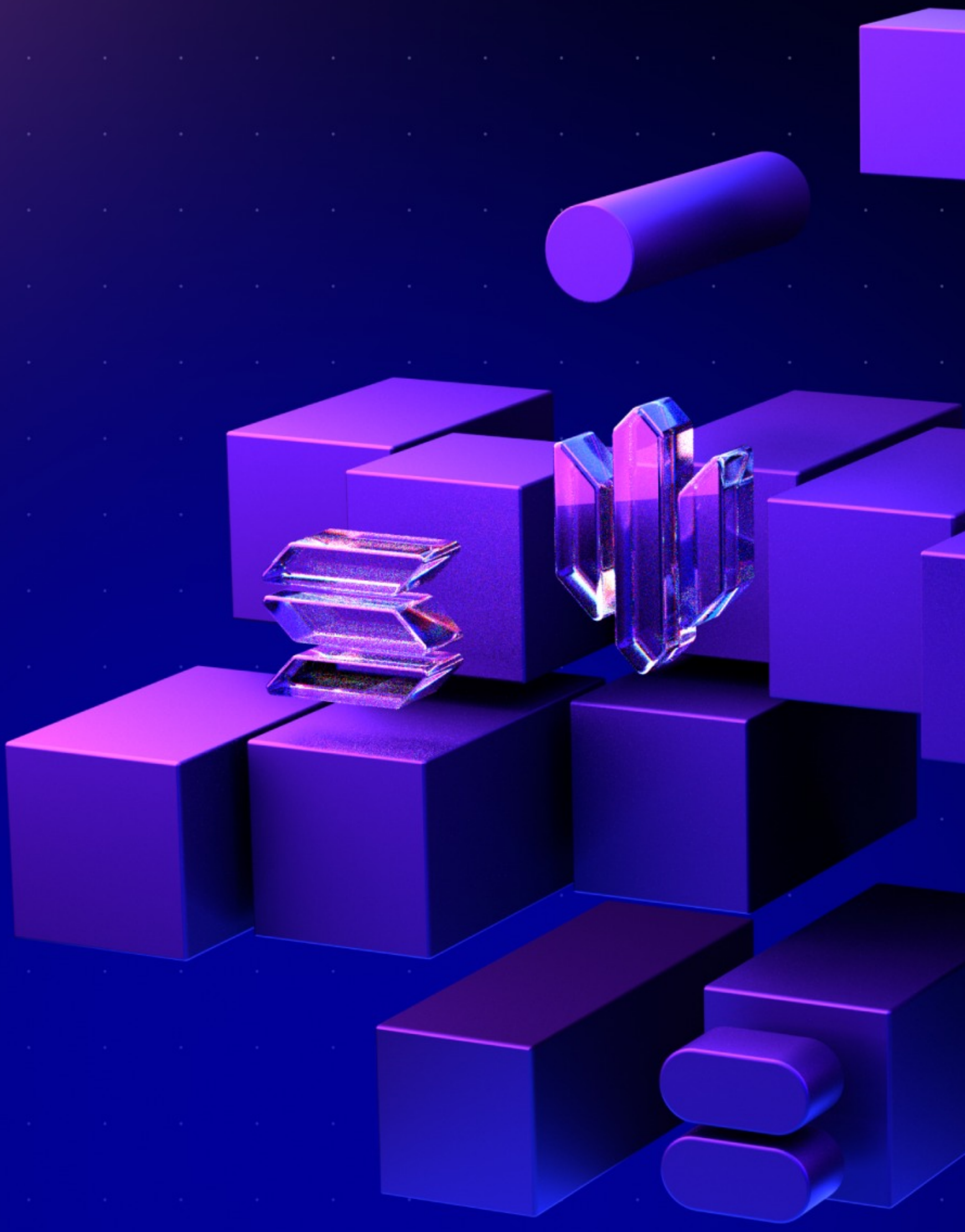
ackee
blockchain security

Onre

Tokenized (re)Insurance Pools

25.11.2025



Audit Report

Contents

1. Document Revisions

1.0	Final Report	03.11.2025
1.1	Fix Review	20.11.2025
1.2	Fix Review	25.11.2025

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain Security

Ackee Blockchain Security is an in-house team of security researchers performing security audits focusing on manual code reviews with extensive fuzz testing for Ethereum and Solana. Ackee is trusted by top-tier organizations in web3, securing protocols including Lido, Safe, and Axelar.

We develop open-source security and developer tooling Wake for Ethereum and Trident for Solana, supported by grants from Coinbase and the Solana Foundation. Wake and Trident help auditors in the manual review process to discover hardly recognizable edge-case vulnerabilities.

Our team teaches about blockchain security at the Czech Technical University in Prague, led by our co-founder and CEO, Josef Gattermayer, Ph.D. As the official educational partners of the Solana Foundation, we run the School of Solana and the Solana Auditors Bootcamp.

Ackee's mission is to build a stronger blockchain community by sharing our knowledge.

Ackee Blockchain a.s.

Rohanske nabrezi 717/4

186 00 Prague, Czech Republic

https://ackee.xyz

hello@ackee.xyz

2.2. Audit Methodology

The Ackee Blockchain Security auditing process follows a routine series of steps:

1. **Code review**
    a. High-level review of the specifications, sources, and instructions provided to us to make sure we understand the project's size, scope, and functionality.
    b. Detailed manual code review, which is the process of reading the source code line-by-line to identify potential vulnerabilities. We focus mainly on common classes of Solana program vulnerabilities, such as:

        missing ownership checks, missing signer authorization, signed CPI of unverified programs, cosplay of Solana accounts, missing rent exemption assertion, bump seed canonicalization, incorrect accounts closing, casting truncation, numerical precision errors, arithmetic overflows or underflows.

    c. Comparison of the code and given specifications, ensuring that the program logic correctly implements everything intended.
    d. Review of best practices to improve efficiency, clarity, and maintainability.
2. **Testing and automated analysis**
    a. Run client's tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests using our testing framework Trident.
3. **Local deployment + hacking**
    a. The programs are deployed locally, and we try to attack the system and break it. There is no specific strategy here, and each project's attack attempts are unique to its implementation.

2.3. Finding Classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in *configuration* (system settings or parameters, such as deployment scripts, compiler configurations, using multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**N/A**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Low	-
	Low	Medium	Low	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or *configuration*, but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or *configuration* was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review Team

The following table lists all contributors to this report. For authors of the specific revision, see the “Revision team” section in the respective “Report revision” chapter.

Member’s Name	Position
Andrej Lukačovič	Lead Auditor
Felipe Donato	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We’ve put our best effort to find all vulnerabilities in the system, however our findings shouldn’t be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Onre Protocol is a token exchange platform built on Solana that facilitates automated market-making through deterministic pricing mechanisms.

Tokenized (re)Insurance Pools is a subcomponent of Onre which implements unique offer-based architecture where the protocol owner (Boss) creates and manages token exchange offers with time-based pricing vectors that simulate APR-based growth.

Revision 1.0

Onre engaged Ackee Blockchain Security to perform a security review of Onre Tokenized (re)Insurance Pools with a total time donation of 13 engineering days in a period between October 15 and November 3, 2025, with Andrej Lukačovič as the lead auditor.

The audit was performed on the commit `27e9fe7`[1] and the scope was the following:

- OnreApp

We began our review by analyzing the protocol architecture and documentation to understand the offer-based token exchange mechanism and pricing vector implementation. The initial phase focused on mapping the trust model, identifying critical functions, and understanding the dual settlement paths (burn/mint vs vault transfer).

In the second phase, we conducted systematic function-by-function analysis of all core protocol functions. We developed proof-of-concept scenarios for critical vulnerabilities while maintaining active communication with the client to clarify design intentions and discuss findings. During this phase, we paid special attention to:

- ensuring the fee collection mechanism operated consistently across both settlement paths;
- validating pricing vector calculations and APR-based growth mathematics;
- analyzing the approval system for replay attacks and binding issues;
- detecting Time-of-Check-Time-of-Use (TOCTOU) vulnerabilities in state transitions;
- ensuring Token-2022 extension compatibility did not introduce attack vectors;
- validating access controls and RBAC privilege boundaries;
- examining vault management for potential rug pull vectors; and
- ensuring proper validation of user inputs and slippage protection.

In the final phase, we categorized findings by severity, documented exploit scenarios, and provided actionable remediation recommendations.

For the fuzzing phase, we used the Trident framework to test the protocol's behavior under various conditions. We implemented a set of invariants and flows to ensure the protocol's behavior was correct. During fuzzing, we identified one logic error issues (M1) that prevented the Boss from withdrawing from Token-2022 vault accounts.

Our review resulted in 15 findings, ranging from Info to Medium severity.

The M2 issue instantly invalidates all existing approvals when changing the approver, with no grace period or migration path, locking multiple users out of regulated offers simultaneously.

The M4 issue exposes users to attack vectors through malicious Token-2022 extensions: permanent delegate extensions enable token theft, transfer fees break price calculations, and freeze authority allows indefinite fund lockup.

The M3 issue grants unlimited token minting without supply caps, frequency limits, or governance controls, enabling arbitrary inflation that devalues all existing token holders.

The W8 issue enables Time-of-Check-Time-of-Use vulnerability where the protocol owner can modify fees up to 100% after a user checks the price. Users submitting transactions expecting 1% fees can be forced to pay 99%, resulting in direct loss of funds without timelock or protection mechanisms.

The M1 issue prevents the protocol owner from withdrawing from Token-2022 vault accounts. The withdrawal instruction's incomplete constraints default to legacy SPL Token, causing transaction failures when accessing Token-2022 vaults due to PDA address mismatch.

We noticed that the approval system is prone to double-spend attacks with no implemented limits. Based on the client's response, this behavior is intentional, allowing approved users unlimited approvals.

Additionally, the audit identified design flaws including inconsistent APY displays (correct compound returns displayed but linear growth applied in `calculate_vector_price` for `take_offer`), vector timing ambiguities enabling backdated price manipulation, and denial-of-service vectors through improper validation logic.

Ackee Blockchain Security recommends Onre:

- implement slippage protection and timelocks for all parameter changes to prevent TOCTOU risk;
- add strict validation for Token-2022 extensions, rejecting tokens with dangerous extensions such as permanent delegate, transfer fees, and freeze authority;
- clarify the design intent for vector timing mechanics and implement consistent validation logic;

- implement proper two-step ownership transfers with acceptance requirements;
- introduce counter-balance to Boss role to mitigate single point of failure and issues caused by centralization; and
- address all remaining identified issues.

See Report Revision 1.0 for the system overview and trust model.

Revision 1.1

Onre engaged Ackee Blockchain Security to perform a fix review of the findings from the previous revision.

The review was performed between November 13 and November 13, 2025 on the commit `233b005`[2].

The M2 was partially fixed. The fix limited the approver count to two keys and updated the validation logic to check against both approvers, ensuring the addition of a second approver would not break existing approvals. However, removing both approvers and appointing new ones would still cause existing approvals to become invalid, which can happen if the rotation is executed in a wrong manner.

The M3 was fixed by implementing a supply cap set via the boss-only function. However, the function allowed the maximum supply to be set to 0, removing the cap.

The L1 was fixed by implementing a two-step ownership transfer pattern requiring the proposed boss to sign the acceptance transaction to complete the transfer.

The W6 was fixed by placing the old vector cleanup call before attempting to find empty slots.

The L2 was fixed by implementing upgrade authority validation, ensuring only the designated deployer could call initialize.

The W7 was fixed by requiring the vector start time to be greater than the current time for deletion operations.

The W8 was acknowledged. The Onre team considered the current multisig-based trust model sufficient for their intended use case, while recognizing that timelock or slippage mechanisms would enhance security.

The M4 was acknowledged. The Onre team committed to validating Token-2022 extensions and transfer behaviors before listing new tokens.

The W1 was acknowledged. The Onre team stated that the APY getter intentionally displayed compound interest calculations for informational purposes and served different functions in their system.

The W2 was fixed by correcting the validation to compare computed start time values rather than raw base time inputs when ordering vectors.

The W3 was fixed by implementing separate fee accounting, ensuring fees were transferred to designated recipients rather than destroyed.

The W4 was acknowledged. The Onre team clarified that backdated vectors were a design feature and that they had removed safeguards limiting how far the base time could be set in the past, allowing price changes at vector activation without user protection mechanisms.

The I1 was fixed by removing the unnecessary and unused code.

The M1 was fixed by adding token program constraints, ensuring the system properly handled both SPL and Token-2022 vault accounts.

The W5 was partially fixed by improvements including max supply caps and enhanced event logging. The Onre team asserted that boss privileges were controlled through Squads multisig, though this restriction was not enforced at the contract level.

Even though the fix review did not result in any new findings, we want to emphasize two warnings that could arise from the unexpected usage of Token-2022 mints.

The W9 where Token-2022 transfer fees were not accounted for when the program lacked mint authority, causing users to receive fewer tokens than calculated while events reported incorrect amounts.

The W10 where transfer-fee tokens broke the protocol's burn mechanism, causing all trades to fail for the affected token pairs.

Revision 1.2

Onre engaged Ackee Blockchain Security to perform a fix review of the findings from the previous revision.

The review was performed between November 21 and November 21, 2025 on the commit `8b5b78e`[3].

The W9 and W10 were fixed by preventing take offer from succeeding if the token out or token in mints have non-zero transfer fees.

The fix, however, introduced an issue, if the fee is increased from 0 to some number, users will not be able to take the offer. The client is aware of this and they acknowledged this potential Denial of Service.

See Report Revision 1.2 for the ...

[1] full commit hash: `27e9fe76385102c4289ca712636f5099a0556106`

[2] full commit hash: `233b005b33690e3e9f6ffe866d7ee9d61e9b5679`

[3] full commit hash: `8b5b78e4abf99c154df50a8fac60ae2b04aa3ec9`

4. Findings Summary

The following section summarizes findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- *Description*
- *Exploit scenario* (if severity is low or higher)
- *Recommendation*
- *Fix* (if applicable).

Summary of findings:

Critical	High	Medium	Low	Warning	Info	Total
0	0	4	2	10	1	17

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
M1: Missing token_program constraint prevents Token-2022 vault withdrawals	Medium	1.0	Fixed
M2: Global Approver Key Rotation can cause system-wide approval lockout	Medium	1.0	Partially fixed
M3: Boss Unbounded ONyc Token Minting	Medium	1.0	Fixed
M4: Token 2022 is allowed but there is no validation for its extensions in place	Medium	1.0	Acknowledged

Finding title	Severity	Reported	Status
L1: Unsafe Single-Step Ownership Transfer	Low	1.0	Fixed
L2: Boss role hijack via Initialize call frontrun	Low	1.0	Fixed
W1: Inconsistent APY-take_offer model	Warning	1.0	Acknowledged
W2: Vector Addition Blocked Due to Incorrect Validation Logic	Warning	1.0	Fixed
W3: Fees Are Burned Instead of Being Collected	Warning	1.0	Fixed
W4: Backdated Vector Price	Warning	1.0	Acknowledged
W5: Centralization and Absence of Standard DeFi Safeguards	Warning	1.0	Partially fixed
W6: Vector Cleanup Executes After Empty Slot Check	Warning	1.0	Fixed
W7: Deleting Active Single Vector Causes DoS	Warning	1.0	Fixed
W8: Immediate Fee Changes Enable TOCTOU Attacks	Warning	1.0	Acknowledged
I1: Unnecessary code logic	Info	1.0	Fixed

Finding title	Severity	Reported	Status
W9: Token-2022 Transfer Fee Causes User to Receive Less token_out	Warning	1.1	Fixed
W10: Token-2022 Transfer Fee on token_in Breaks Burn Path leading to potential DoS	Warning	1.1	Fixed

Table 3. Table of Findings

Report Revision 1.0

Revision Team

Member's Name	Position
Andrej Lukačovič	Lead Auditor
Felipe Donato	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

Onre implements deterministic token exchange through time-based pricing vectors on Solana. Offers are uniquely defined by token pairs with configurable APR-based price growth calculated at discrete intervals. The system employs dual settlement paths: burn/mint for program-controlled tokens and vault transfers for external tokens.

Core components include pricing vectors for time-based valuation, optional cryptographic approvals for regulated access, vault management for token custody, and centralized administrative controls. Tokenized (re)Insurance Pools uses PDA-controlled intermediary accounts for permissionless operations and supports both direct and routed token flows.

Integration with Token-2022 enables advanced token features while introducing extension-based attack vectors. All operations flow through the Boss authority who maintains unrestricted control over minting, fees, vaults, and offer parameters.

Trust Model

Onre operates under a fully centralized trust model with the Boss role holding absolute authority over all critical functions.

Boss Privileges:

- Unlimited token minting without caps or restrictions
- Immediate fee modifications up to 100% without user notification
- Unrestricted vault withdrawals at any time
- Instant offer closure and parameter changes
- Complete control over approval requirements and approver rotation
- Authority to enable/disable protocol operations via kill switch
- Ability to add and remove admins

Admin Privileges:

- Activate the kill switch

Users must trust that:

- The Boss will not mint tokens to devalue holdings
- Fees will not be manipulated during transaction execution
- Vault funds will not be withdrawn while offers are active
- Offer parameters will remain stable during interactions
- The approver key will not be rotated to invalidate existing approvals

Fuzzing

During the audit, we developed manually-guided fuzz tests to evaluate the protocol’s correctness, security, and robustness. The fuzz test templates are generated from the IDL created by Anchor and implemented according to specific requirements. Trident notably supports the specification of flows or invariant checks.

Flows help the fuzzer achieve better coverage of valid instruction sequences.

Invariant checks, meanwhile, detect unwanted changes during instruction execution. After successful instruction invocation, multiple invariant checks can be specified to verify that account contents were updated as expected during execution.

Throughout the fuzzing process, two common failure types may occur: a panic during instruction execution or a failure of the specified invariant check. The former might happen when an unchecked arithmetic overflow is detected, while the latter occurs when behavior defined as unwanted is identified (e.g., unauthorized balance changes).

The complete list of implemented flows and invariants is available in Appendix B.

Findings

The following section presents the list of findings discovered in this revision. For the complete list of all findings, Go back to Findings Summary

M1: Missing token_program constraint prevents Token-2022 vault withdrawals

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	-	Type:	Logic error

Description

The Onre vault system enables the Boss to deposit and withdraw tokens through dedicated instructions.

The `OfferVaultDeposit` instruction creates vault token accounts that store tokens for later distribution, while `OfferVaultWithdraw` enables the Boss to recover these funds.

The issue arises when attempting to withdraw from Token-2022 vault accounts. The `OfferVaultWithdraw` instruction's constraint validation incorrectly defaults to the legacy Token Program:

Listing 1. Excerpt from offer_withdraw

```
#[account(
    mut,
    associated_token::mint = token_mint,
    associated_token::authority = vault_authority,
)]
pub vault_token_account: Box<InterfaceAccount<'info, TokenAccount>>,
```

Without specifying `token_program`, the constraint defaults to legacy SPL Token. Since Associated Token Accounts derive different addresses for Token-2022 versus legacy tokens, the constraint fails for Token-2022 vaults. This results in tokens becoming locked and unable to be withdrawn.

Exploit scenario

1. Alice, the boss, calls `OfferVaultDeposit` to deposit tokens into the vault. Due to the `init_if_needed` constraint, the `vault_token_account` initializes with Token-2022 as the `token_program`.
2. Protocol operates normally.
3. Alice calls `OfferVaultWithdraw` to withdraw tokens from the vault.
4. The withdrawal fails because the `token_program` constraint defaults to the legacy SPL Token program, which results in a different Associated Token Account address.
5. The tokens remain locked in the vault until the program is redeployed with the fix.

Recommendation

Add the `token_program` constraint to the `vault_token_account` account in the `OfferVaultWithdraw` instruction.

Fix 1.1

The issue was fixed by adding the `token_program` constraint to the `vault_token_account` in the `OfferVaultWithdraw` instruction, which ensured compatibility with Token-2022 vault accounts and enabled successful withdrawals.

Go back to Findings Summary

M2: Global Approver Key Rotation can cause system-wide approval lockout

Medium severity issue

Impact:	Low	Likelihood:	High
Target:	-	Type:	Logic error

Description

When the boss changes the approver in State, all existing approvals immediately become invalid because the verification checks against the current approver, not the one who originally signed the approval.

Listing 2. Excerpt from approver_utils

```
require!(parsed.pubkey == trusted_pubkey.to_bytes(),
ErrorCode::WrongAuthority);
```

Exploit scenario

Alice is the boss.

Bob is a user.

1. Alice updates the approver.
2. Bob who was granted permission for 7 days calls `take_offer` or `take_offer_permissionless`.
3. Bob is unable to complete the call as his approval message fails the validation step.

Recommendation

Store the approver key used for each approval at the time of signing. Modify the verification logic to check approvals against their original approver key rather than the current global approver.

Partial solution 1.1

The issue was partially fixed. OnreApp limited the approver count to two approver keys, and the validation logic was updated to check against both approver keys (1 and 2), which made the initial addition of a second approver safe without breaking existing approvals. However, the logic still allows removing both approvers and appointing new different approvers, which would again cause existing approvals to become invalid, which can happen if the rotation is executed in a wrong manner.

Go back to Findings Summary

M3: Boss Unbounded ONyc Token Minting

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	-	Type:	Standards violation

Description

The boss can mint unlimited ONyc tokens without restrictions.

No safeguards prevent arbitrary inflation via unbounded minting, which can inflate supply, dilute holders, and manipulate any ONyc-dependent mechanics:

- no max supply check;
- no rate limiting; and
- no time locks or vesting.

Exploit scenario

1. Alice, the boss, decides to mint a large quantity of ONyc tokens.
2. Alice inflates the supply beyond intended limits.
3. Bob, a legitimate holder, loses value in his ONyc holdings due to inflation.

Recommendation

Implement minting restrictions such as maximum supply caps, rate limiting mechanisms, or time-locked governance processes for minting decisions.

Fix 1.1

The issue was fixed by introducing an on-chain supply cap mechanism, which included a configurable `max_supply` parameter in the protocol state, an administrative function to set this cap, and enforcement logic within the minting functions that validated the total supply remained within limits before executing any mint operation. However, if 0 was passed to `configure_max_supply` as `max_supply`, this would remove the cap.

Go back to Findings Summary

M4: Token 2022 is allowed but there is no validation for its extensions in place

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	-	Type:	Code quality

Description

The protocol is Token-2022 compatible through `InterfaceAccount<Mint>` and `Interface<TokenInterface>`. While this architectural decision enables advanced token functionality, it introduces critical security risks as the protocol lacks proper validation mechanisms for potential misuse of Token-2022 extensions.

Permanent Delegate Extension: Grants transfer authority to a designated account. Once activated, the delegate can drain tokens from any holder at will, bypassing normal ownership controls. This completely undermines the protocol's custody assumptions.

Transfer Fee Extension: Enables automatic fee deduction on every transfer. If misused, can set fees up to 100%, effectively stealing all transferred tokens. In the offer system, this breaks price calculations and fee assumptions as users receive less than expected due to hidden fees.

Freeze Authority Extension: Allows freezing of token accounts, preventing any transfers. The misuse of mint authority can lock users' funds indefinitely, creating a denial-of-service condition for all offer operations involving frozen tokens.

Transfer Hook Extension: Executes arbitrary CPI calls during transfers and can lead to severe security issues.

The protocol's burn/mint and vault transfer mechanisms assume standard token behavior. These extensions violate those assumptions, enabling scenarios where tokens become unmovable in vaults, or the offer mathematical model produces incorrect outputs due to incorrect or incomplete assumptions.

Exploit scenario

1. Alice, the Boss, creates a Token-2022 mint with the Permanent Delegate extension, setting herself as the irrevocable delegate authority.
2. Alice creates an attractive offer in the protocol: her token (with hidden delegate) for USDC at below-market rates.
3. Bob, a user, sees the favorable pricing and executes multiple trades, accumulating 100,000 of Alice's tokens, believing he owns them securely.
4. Alice exercises her permanent delegate authority to transfer all 100,000 tokens from Bob's account back to herself, bypassing any approval requirements.
5. Bob loses his entire position with no recourse.
6. Alice turns the program into a honeypot.

Recommendation

Implement validation mechanisms for Token-2022 extensions to ensure only safe and expected extensions are allowed, combined with whitelisting of trustworthy addresses that should be allowed to use critical extensions. Alternatively, restrict the program to work only with standard SPL tokens if Token-2022 extensions are not required.

Acknowledgment 1.1

The Onre team acknowledged the issue and stated that they would perform due diligence and validation before listing any new offers to ensure tokens behaved as expected.

Go back to Findings Summary

L1: Unsafe Single-Step Ownership Transfer

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	-	Type:	Standards violation

Description

Boss can instantly transfer complete protocol control to any address without validation or confirmation.

Listing 3. Excerpt from set_boss

```rust
pub fn set_boss(ctx: Context<SetBoss>, new_boss: Pubkey) -> Result<()> {
    require!(
        new_boss != Pubkey::default(),
        SetBossErrorCode::InvalidBossAddress
    );

    let state = &mut ctx.accounts.state;
    let old_boss = ctx.accounts.boss.key(); // Capture old boss before update
    state.boss = new_boss;
    emit!(BossUpdatedEvent { old_boss, new_boss });
    Ok(())
}
```

Exploit scenario

1. Alice, the boss, calls `set_boss` and supplies the wrong address to the function.
2. OnreApp needs to be re-deployed as the higher entity of the RBAC has been compromised.

Recommendation

Implement a 2-step process of role transfer for critical roles:

1. The current boss proposes a new boss address.
2. The proposed new boss must accept the role by signing the accept instruction implemented in the program to complete the transfer.

Fix 1.1

The issue was fixed by implementing the recommended two-step process for transferring the boss role, which required both the current boss to propose a new boss and the proposed new boss to accept the role.

Go back to Findings Summary

L2: Boss role hijack via Initialize call frontrun

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	-	Type:	Standards violation

Description

The boss role can be hijacked via a frontrun attack on the Initialize function, which has no proper access control to prevent a malicious caller from frontrunning the intended boss. The function checks if the boss field has already been set in the State PDA. If not, it proceeds to initialize the State PDA's boss field with whatever address was supplied during the function call.

Listing 4. Excerpt from initialize

```rust
if state.boss != Pubkey::default() {
    return err!(InitializeErrorCode::BossAlreadySet);
}
```

Exploit scenario

1. Alice, the legitimate deployer, deploys OnreApp and prepares to call the Initialize function.
2. Bob, a malicious actor, frontruns Alice's transaction by submitting an Initialize call before Alice does.
3. Bob's transaction executes first, setting him as the boss of the protocol.
4. Alice's transaction fails or has no effect, leaving Bob in control of the entire protocol until OnreApp is re-deployed.

Recommendation

Implement proper access control for the Initialize function by restricting who can call it, such as using a predetermined deployer address or requiring a signature from an authorized party. Alternatively, use the upgrade_authority address safe pattern.

Deployer Restriction Pattern: Add a deployer check in the initialize function:

```
require!(ctx.accounts.boss.key() == EXPECTED_DEPLOYER,
ErrorCode::UnauthorizedInitializer);
```

Where EXPECTED_DEPLOYER is a constant pubkey set at compile time.

Program Upgrade Authority Pattern: Leverage Solana's upgrade authority as the initializer:

```
let program_data = ctx.program_id.program_data_address()?;
require!(ctx.accounts.boss.key() == program_data.upgrade_authority,
ErrorCode::UnauthorizedInitializer);
```

Fix 1.1

The issue was fixed by implementing the recommended deployer update authority pattern, which restricted the `initialize` function to only be callable by a designated deployer authority and prevented unauthorized role hijacks.

Go back to Findings Summary

W1: Inconsistent APY-take_offer model

Impact:	Warning	Likelihood:	N/A
Target:	-	Type:	Logic error

Description

The function `get_apy` displays the APY using daily compound interest. However, when users call the functions `take_offer` and `take_offer_permissionless`, the code calls the function `calculate_vector_price`. Despite the comment ”/// Calculates continuous price growth using APR-based compound interest”, the formula in `calculate_vector_price` implements linear price growth, not compound interest.

Recommendation

Be consistent with the chosen mathematical model throughout the code. Either update the function `calculate_vector_price` to implement compound interest instead of linear price growth or update `get_apy` to reflect the linear growth model.

Acknowledgment 1.1

The issue was acknowledged, and the Onre team claimed that `get_apy` displayed an APY calculation with compound interest for informational purposes only and served different functions in their system.

Go back to Findings Summary

W2: Vector Addition Blocked Due to Incorrect Validation Logic

Impact:	Warning	Likelihood:	N/A
Target:	-	Type:	Logic error

Description

The protocol incorrectly validates new pricing vectors against historical timestamps rather than effective timestamps, preventing legitimate vector additions.

The validation logic compares the input `base_time` parameter against existing vector `start_time` values, rather than comparing the computed `start_time` (which equals `max(base_time, current_time)`). This creates situations where valid vectors are incorrectly rejected.

Listing 5. Excerpt from add_offer_vector

```rust
let start_time = if base_time > current_time {
    base_time
} else {
    current_time
};
```

Listing 6. Excerpt from add_offer_vector

```rust
if let Some(latest_start_time) = existing_start_times.iter().max() {
    require!(
        &base_time > latest_start_time,
        AddOfferVectorErrorCode::InvalidTimeRange
    );
}
```

Exploit scenario

1. An existing vector was activated at timestamp 1000.
2. The current time is 2000.
3. Alice, the protocol Boss, attempts to add a vector with `base_time` 900 (which would activate immediately at 2000).
4. The system rejects the valid configuration because 900 < 1000.

Recommendation

Validate new vectors using their computed `start_time` value rather than the input `base_time` parameter. Compare `max(base_time, current_time)` against existing vector `start_time` values to ensure proper chronological ordering of effective activation times.

Change the validation from:

```
require!(base_time > latest_start_time, ErrorCode::InvalidTimeRange);
```

To:

```
let computed_start_time = max(base_time, current_time);
require!(computed_start_time > latest_start_time, ErrorCode::InvalidTimeRange);
```

Fix 1.1

The issue was fixed by updating the validation logic to compare `start_time` instead of `base_time` when checking against existing vectors.

[Go back to Findings Summary]

W3: Fees Are Burned Instead of Being Collected

Impact:	Warning	Likelihood:	N/A
Target:	-	Type:	Logic error

Description

The issue occurs when the program is the mint authority of `token_in_mint`.

The protocol charges a fee (`offer.fee_basis_points`) and reports that fee in logs/events as if it is revenue captured by the boss. However, in the burn/mint settlement path (when the program is the mint authority of `token_in_mint`), the entire `token_in_amount` sent by the user—including the supposed ”fee” portion—is burned, and no party receives it.

Listing 7. Excerpt from token_utils

```
if controls_token_in_mint {
    burn_tokens(
        params.token_in_program,
        params.token_in_mint,
        params.token_in_burn_account,
        params.token_in_burn_authority,
        params.vault_authority_signer_seeds.unwrap(),
        params.token_in_amount,
    )?;
}
```

Recommendation

Consistently implement fee collection through either a specialized fee vault for later withdrawal or send the fee directly to the boss address.

Fix 1.1

The issue was fixed by updating the fee handling logic to ensure that fees were transferred to a designated address instead of being burned when the program was the mint authority of `token_in_mint`. This was achieved by adding new fields `token_in_net_amount` and `token_in_fee_amount` to the Offer struct.

Go back to Findings Summary

W4: Backdated Vector Price

Impact:	Warning	Likelihood:	N/A
Target:	-	Type:	Logic error

Description

The boss can add pricing vectors with past `base_time` values, causing immediate retroactive price increases through APR calculation.

The validation logic helps mitigate but does not prevent past `base_time` values. The protocol checks whether the new `base_time` is non-zero:

Listing 8. Excerpt from add_offer_vector

```rust
fn validate_inputs(base_time: u64, base_price: u64, price_fix_duration: u64)
    -> Result<()> {
    // Validate input parameters
    require!(base_time > 0, AddOfferVectorErrorCode::ZeroValue);
```

Listing 9. Excerpt from add_offer_vector

```rust
if let Some(latest_start_time) = existing_start_times.iter().max() {
    require!(
        &base_time > latest_start_time,
        AddOfferVectorErrorCode::InvalidTimeRange
    );
}
```

The boss cannot set `base_time` too far back when calling `add_offer_vector` from the second time onwards. The backdating occurs when:

`latest_start_time < base_time < current_time`

The check only ensures `base_time > latest_start_time`, not `base_time > current_time`.

Exploit scenario

1. At timestamp 100, Alice, the boss, adds the first vector with `base_time = 100`, resulting in `start_time = max(100, 100) = 100`.
2. At timestamp 200, Alice adds a second vector with `base_time = 150`, which passes the validation check since `150 > 100` (the existing `latest_start_time`).
3. The current time is 200, so `start_time = max(150, 200) = 200`.
4. This results in `start_time = 200` and `base_time = 150`, creating a backdated vector.

The vector activates at timestamp 200 (start_time) but calculates prices from timestamp 150 (base_time), gaining 50 time units of retroactive APR growth. The constraint limits how far back the boss can backdate (cannot go before the latest vector) but does not eliminate backdating entirely.

Recommendation

Validate that `base_time` is not less than the current timestamp to prevent retroactive price calculations.

Implement the check:

```
require!(base_time >= current_time, AddOfferVectorErrorCode::BackdatedVector);
```

This ensures all pricing vectors start from the current time onward, eliminating the ability to gain retroactive APR growth through backdating. If price continuity is needed between vectors, achieve it through proper `base_price` adjustment rather than time manipulation.

Acknowledgment 1.1

The issue was acknowledged regarding the backdated vector prices, and the

Onre team claimed that the ability to set `base_time` in the past relative to `start_time` was an intentional design feature. However, the team had removed constraints that would have prevented `base_time` from being set arbitrarily far in the past relative to `current_time`, which enabled sudden and unpredictable price changes for users. While the mathematical model might have functioned as designed, the lack of bounds on historical `base_time` values combined with the absence of slippage protection exposed users to unexpected price impacts at vector activation.

Go back to Findings Summary

W5: Centralization and Absence of Standard DeFi Safeguards

Impact:	Warning	Likelihood:	N/A
Target:	-	Type:	Logic error

Description

Tokenized (re)Insurance Pools lacks standard DeFi safeguards that users typically expect from DeFi protocols.

While the centralized design may be intentional, users should be aware that the protocol operates without timelock delays, multisig controls, complete parameter change notifications, or on-chain limits. This creates a trusted setup where users must have faith in the Boss's good intentions at all times, rather than relying on smart contract guarantees.

Recommendation

Implement standard DeFi safeguards to reduce trust requirements:

- add timelock delays for critical parameter changes;
- implement multisig controls for administrative functions;
- emit events consistently for all parameter changes to notify users; and
- establish on-chain limits for critical operations such as mint operations.

Partial solution 1.1

The issue was partially fixed, and improvements were made, such as implementing a Max Supply Cap and comprehensive Event Emission. The Onre team also claimed to have Multisig Controls, in which the Boss role was controlled via Squads multisig. However, it is worth noting that nothing on-chain seemed to enforce multisig.

Go back to Findings Summary

W6: Vector Cleanup Executes After Empty Slot Check

Impact:	Warning	Likelihood:	N/A
Target:	-	Type:	Code quality

Description

The `add_offer_vector` function searches for empty slots before running cleanup, causing ”TooManyVectors” errors even when old inactive vectors could be removed to free space.

Listing 10. Excerpt from add_offer_vector

```
offer.vectors[empty_slot_index] = new_vector;

// Clean up old vectors before emitting success message
clean_old_vectors(offer, current_time)?;
```

Recommendation

Change the order of operation, calling cleanup prior for adding the vector to the offer.

Fix 1.1

The issue was fixed by updating the `add_offer_vector` function to perform cleanup before attempting to find empty slots, preventing ”TooManyVectors” errors when old inactive vectors could be removed.

Go back to Findings Summary

W7: Deleting Active Single Vector Causes DoS

Impact:	Warning	Likelihood:	N/A
Target:	-	Type:	Code quality

Description

The code prevents deleting the previous vector but allows deleting the current active vector. The `find_active_vector_at` call will default to the previous vector. However, the code does not prevent the Boss from deleting the first vector. When deleting the only active vector, `find_active_vector_at` will return `OfferCoreError::NoActiveVector`, and `take_offer` and `take_offer_permissionless` will be denied service until new vectors are added.

Listing 11. Excerpt from delete_offer_vector

```rust
if current_vector.is_ok() {
    let prev_vector = find_active_vector_at(offer,
current_vector?.start_time - 1);

    if prev_vector.is_ok() {
        require!(
            prev_vector?.start_time != vector_start_time,
            DeleteOfferVectorErrorCode::CannotDeletePreviousVector
        );
    }
}
```

Recommendation

Check if the vector being deleted is the only active vector before allowing deletion. Prevent deletion of the sole active vector to maintain system functionality.

Fix 1.1

The issue was fixed by restricting deletions to only future vectors that had not activated yet. This maintained offer availability while allowing the boss to manage scheduled vectors.

Go back to Findings Summary

W8: Immediate Fee Changes Enable TOCTOU Attacks

Impact:	Warning	Likelihood:	N/A
Target:	-	Type:	Code quality

Description

The `update_offer_fee` function applies fee changes immediately without any timelock, grace period, or user slippage protection mechanisms. This results in users experiencing different fees between the time they initiate a transaction and when the `take_offer` and `take_offer_permissionless` instruction calls execute.

The issue is not meant in the common MEV terms, as the price is not based on supply and demand but on time and initial setup. In that case, sandwich attacks cannot happen, but the fee setting can be potentially exploited.

Exploit scenario

1. Alice, a user, observes the current fee rate and prepares a transaction to take an offer.
2. Bob, the boss, updates the offer fee to a higher rate just before Alice's transaction executes.
3. Alice's transaction executes with the new, higher fee rate instead of the expected rate, resulting in unexpected costs.

Recommendation

Implement a timelock mechanism for fee changes or add slippage protection parameters to allow users to specify maximum acceptable fee rates for their transactions.

Acknowledgment 1.1

The issue was acknowledged, and the Onre team stated that while they recognized that a timelock or slippage protection mechanism would provide additional security guarantees, the current trust model with multisig control was appropriate for the protocol's design and intended use case.

Go back to Findings Summary

I1: Unnecessary code logic

Impact:	Info	Likelihood:	N/A
Target:	-	Type:	Logic error

Description

The source code contains unnecessary logic that should be removed.

The `OfferVaultAuthority` and `MintAuthority` accounts are initialized with allocated space to store bump seeds. Both accounts use Program Derived Addresses. If authorization is required, the PDAs sign using `invoke_signed` from the onre program. The program ID is automatically part of the PDA derivation process. Storing the bump is unnecessary because PDA derivation is canonical—the highest possible bump is always used.

The `boss` account has the writable flag set in multiple unnecessary places. Multiple instruction contexts, such as `TransferMintAuthorityToBoss`, `TransferMintAuthorityToProgram`, `AddOfferVector`, and `DeleteOfferVector`, contain the `boss` account as mutable. However, the `boss` account is not mutated in these instructions.

The code defines a constant `pub const MAX_OFFERS: usize = 10;`. This constant is never used throughout the codebase. The code currently has no offer amount limitation implementation.

Recommendation

Remove the unnecessary allocated space for bump seeds, the writable flags on immutable accounts, and the unused `MAX_OFFERS` constant.

Fix 1.1

The issue was fixed by removing the unnecessary code logic such as the allocated space for bump seeds, the writable flags on immutable accounts, and the unused `MAX_OFFERS` constant.

Go back to Findings Summary

Report Revision 1.1

Revision Team

Revision team is the same as in Report Revision 1.0.

Overview

Since there were no comprehensive changes in this revision, the complete overview is listed in the Executive Summary section Revision 1.1.

Findings

The following section presents the list of findings discovered in this revision. For the complete list of all findings, Go back to Findings Summary

W9: Token-2022 Transfer Fee Causes User to Receive Less token_out

Impact:	Warning	Likelihood:	N/A
Target:	-	Type:	Logic error

Description

When `token_out` is a Token-2022 mint with the `TransferFeeConfig` extension and the program does not control the mint authority, the protocol computes `token_out_amount` assuming vanilla SPL semantics without transfer fees. The protocol transfers that amount from the vault to the user using `transfer_checked` and emits events and logs that report the full `token_out_amount`. However, Token-2022 transfer fee logic reduces the actual amount that lands in the user's account. The protocol fails to account for this reduction, so users consistently receive fewer `token_out` tokens than the price math and events indicate.

Exploit scenario

1. Bob, the protocol creator, creates a Token-2022 token `SOL_T22` with a 2% transfer fee.
2. The protocol offers: pay `USDC` and receive `SOL_T22`.
3. The program does not control the `SOL_T22` mint, so it uses a transfer from a vault.
4. Alice, a user, pays 1000 `USDC` into the offer.
5. The protocol math calculates that Alice should receive 2000 `SOL_T22`.
6. The event logs: `token_out_amount = 2000`.
7. The vault transfers 2000 `SOL_T22` to Alice.
8. Token-2022 takes a 2% fee, which equals 40 `SOL_T22`.

9. Alice’s wallet receives only 1960 `SOL_T22`.
10. The protocol and events claim Alice receives 2000 `SOL_T22`, but her account balance only increases by 1960 `SOL_T22`. The missing 40 `SOL_T22` is routed to the Token-2022 fee receiver as an implicit extra fee.

Recommendation

Choose one of the following approaches:

1. Disallow Token-2022 transfer fee mints: At offer initialization, check whether `token_out_mint` has the `TransferFeeConfig` extension. If it does and the program lacks mint authority (requiring vault transfers), reject the offer creation or mark it as unsupported.
2. Support Token-2022 transfer fee mints: Read the transfer fee configuration and calculate the actual amount the user receives after fees. Then choose one of these implementation paths:
    - increase the transfer amount to compensate for the fee, ensuring the user receives the full quoted amount; or
    - track and emit both the gross amount (before fees) and net amount (after fees) in events, and ensure all UI displays and price calculations use the net amount that the user actually receives.

Fix 1.2

The issue was fixed by preventing take offer from succeeding if the token out or token in mints have non-zero transfer fees.

The fix, however, introduced an issue, if the fee is increased from 0 to some number, users will not be able to take the offer. The client is aware of this and they acknowledged this potential Denial of Service.

Go back to Findings Summary

W10: Token-2022 Transfer Fee on token_in Breaks Burn Path leading to potential DoS

Impact:	Warning	Likelihood:	N/A
Target:	-	Type:	Logic error

Description

If `token_in` is a Token-2022 mint with a transfer fee and the program uses the burn path (i.e., it controls the mint authority), `take_offer` can fail. The protocol:

- transfers the net input amount from the user to the vault
- attempts to burn that same net amount from the vault.

However, Token-2022 transfer fees mean the vault actually receives less than the amount being burned, creating two failure scenarios:

1. **Without pre-funding**: Every transaction fails due to insufficient balance.
2. **With pre-funding**: Each transaction generates a negative delta, creating a cumulative deficit requiring periodic liquidity provisioning to maintain operational solvency and prevent the burn operation from failing.

Exploit scenario

Scenario A - Immediate Failure (No Pre-funding):

1. Bob, the protocol creator, creates a Token-2022 token `USDC_T22` with a 2% transfer fee.
2. The protocol controls the `USDC_T22` mint authority, so it uses the burn path for `token_in`.
3. The protocol offers: pay `USDC_T22` and receive `SOL`.

4. Alice, a user, attempts to take the offer by paying 1000 `USDC_T22`.
5. The protocol calculates the net input amount after its own 1% protocol fee: 990 `USDC_T22` net, 10 `USDC_T22` protocol fee.
6. The protocol calls `transfer_checked` to move 990 `USDC_T22` from Alice's account to the vault.
7. Because `USDC_T22` has a 2% Token-2022 transfer fee, the transfer charges a fee of 19 `USDC_T22`, so only 971 `USDC_T22` actually arrive in the vault.
8. The protocol calls `burn_checked` to burn 990 `USDC_T22` from the vault, assuming the vault received the full 990.
9. The transaction reverts, causing DoS for subsequent transactions unless pre-funding of the vault is ensured.

Scenario B - Gradual Potential Failure (With Pre-funding):

1. Bob, the protocol creator, creates a Token-2022 token `USDC_T22` with a 5% transfer fee.
2. Bob pre-funds the burn vault with 10,000 `USDC_T22` as a buffer.
3. The protocol controls the `USDC_T22` mint authority, so it uses the burn path for `token_in`.
4. The protocol offers: pay `USDC_T22` and receive `SOL`.
5. Alice, a user, attempts to take the offer by paying 1000 `USDC_T22`.
6. The protocol calls `transfer_checked` to move 1000 `USDC_T22` from Alice's account to the vault.
7. Because `USDC_T22` has a 5% Token-2022 transfer fee, only 950 `USDC_T22` arrive in the vault.
8. The protocol calls `burn_checked` to burn 1000 `USDC_T22` from the vault (using the buffer).
9. The burn succeeds but depletes the buffer by 50 `USDC_T22` per trade.

10. After 200 trades of burning more than received (systematic 50 token deficit per trade), the buffer is exhausted and subsequent trades fail, causing DoS unless continuous monitoring and liquidity provisioning is maintained.

Recommendation

Ensure that Token-2022 transfer-fee mints are not used as `token_in` in the burn path. Either block them on-chain or force them into the ”external / transfer-only” path.

Fix 1.2

The issue was fixed by preventing take offer from succeeding if the token in mint has a non-zero transfer fee.

The fix, however, introduced an issue, if the fee is increased from 0 to some number, users will not be able to take the offer. The client is aware of this and they acknowledged this potential Denial of Service.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain Security, Audit Report | Onre: Tokenized (re)Insurance Pools, 25.11.2025.

Appendix B: Trident Findings

This section lists the outputs from the Trident framework used for fuzz testing during the audit.

B.1. Implementation Details

Fuzz testing with Trident executes the actual Solana program logic without approximations or simplifications. The programs are compiled in their production form and deployed to TridentSVM, ensuring that test results accurately reflect real-world behavior.

B.2. Fuzzing

The following table lists all implemented execution flows in the Trident fuzzing framework.

ID	Flow	Added	Status
F1	Test adding offer vectors and taking offers with pricing calculations	1.0	Success
F2	Test boss authority transfer: propose and accept boss operations	1.0	Success
F3	Test adding admin to the admin list	1.0	Success
F4	Test removing admin from the admin list	1.0	Success

Table 4. Trident fuzzing flows

The following table lists all implemented invariant checks in the Trident fuzzing framework.

ID	Invariant	Added	Status
IV1	Instruction reverts unexpectedly	1.0	**Fail** (M1)

ID	Invariant	Added	Status
IV2	Initialize instruction must set correct boss, onyc_mint, and default state values	1.0	Success
IV3	Initialize permissionless authority instruction must set correct name	1.0	Success
IV4	Vault deposit operations must preserve token conservation between boss and vault accounts	1.0	**Fail** (M4)
IV5	Vault withdrawal operations must preserve token conservation between vault and boss accounts	1.0	**Fail** (M4)
IV6	Make offer instruction must initialize offer with correct token mints and parameters	1.0	Success
IV7	Add offer vector instruction must store correct pricing parameters	1.0	Success
IV8	Take offer instruction ensures user pays exactly token_in_amount	1.0	**Fail** (M4)
IV9	Take offer instruction ensures boss receives exactly token_in_amount	1.0	**Fail** (M4)
IV10	Take offer instruction ensures user receives correctly calculated token_out based on pricing vector	1.0	**Fail** (M4)
IV11	Set admin instruction must add new admin to first empty slot	1.0	Success
IV12	Remove admin instruction must clear admin from admin list	1.0	Success
IV13	Propose boss instruction must set proposed_boss field correctly	1.0	Success

ID	Invariant	Added	Status
IV14	Accept boss instruction must transfer boss authority and clear proposed_boss	1.0	Success

Table 5. Trident fuzzing invariants

ackee
blockchain security

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz