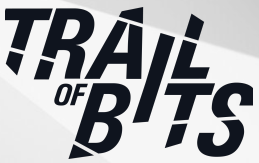

Ref Finance DeFi Protocol Smart Contracts

Security Assessment

February 19, 2025

Prepared for:
Devon Mac Neil
Ref Finance

Prepared by: **Alexander Remie and Bo Henderson**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Jessica Nelson, Project Manager
jessica.nelson@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Alexander Remie, Consultant
alexander.remie@trailofbits.com

Bo Henderson, Consultant
bo.henderson@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
October 9, 2024	Pre-project kickoff call
October 22, 2024	Status update meeting #1
October 29, 2024	Status update meeting #2
November 13, 2024	Delivery of report draft and report readout meeting
February 19, 2025	Delivery of final comprehensive report

Executive Summary

Engagement Overview

Ref Finance engaged Trail of Bits to review the security of its DeFi Protocol smart contracts, including tokens, exchanges, and staking services.

A team of two consultants conducted the review from October 15 to November 8, 2024, for a total of eight engineer-weeks of effort. Our testing efforts focused on validating the correctness of the financial logic and identifying risks in contract interactions. With full access to source code and documentation, we performed static and dynamic testing of the target repos, using automated and manual processes.

Observations and Impact

The Ref Finance smart contracts are well organized, and we did not identify any high-severity issues during this review. However, the inline documentation is limited and should be improved so that it becomes easier to understand the inner workings of the smart contracts. Many tests are present for each of the repos; however, the tests were difficult to run due to outdated dependencies and other build errors, and some of the tests that ran failed. The exchange smart contract has both fuzz tests and unit tests.

Our review identified 10 issues, including a medium-severity issue (TOB-REF-9) related to a lack of input validation when creating new pools that could cause unexpected panics when users use the pool. All other issues are either informational or of undetermined severity.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Ref Finance take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Expand tests.** The existing fuzz tests are naive, relying on pseudorandom numbers for action generation. A more robust approach would test the wasm code directly, generate and execute NEAR transactions, and track coverage.
- **Expand the inline documentation.** The current implementation only sparingly includes inline documentation, which hinders understanding. Adding more inline documentation will make it easier for developers, auditors, and users to understand the implementation.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	1
Low	0
Informational	6
Undetermined	2

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	1
Auditing and Logging	2
Data Validation	4
Error Reporting	1
Patching	1

Project Goals

The engagement was scoped to provide a security assessment of the Ref Finance DeFi smart contracts. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are the dependency versions in use up to date and secure?
- Do the smart contracts conform to the targeted standards?
- Are there any areas of common weakness within the smart contracts (e.g.,reentrancy, integer under/overflows, race conditions, replay attacks)?
- Do the smart contracts properly implement the intended design features and functions?
- Does the system properly calculate fees?
- Are there appropriate access controls on critical functions?
- Is it possible to front-run transactions and negatively affect the system?
- Does the arithmetic produce correct results when given the values that can occur in the protocol?
- Are there areas within ownership and access that may be compromised or altered to cause adverse states, access, or exploitation?
- Are the access controls of each contract sufficient to ensure that it behaves as intended in a hostile environment?
- Are proper governance mechanisms in place to prevent loss of funds?
- Do DEX actions occur accurately and as intended?
- Can the liquidity, farming, or token mechanisms be maliciously manipulated in any way to produce unintended results?
- Could the system experience a denial of service?
- Are all inputs and system parameters properly validated?
- Does the codebase conform to industry best practices?

Project Targets

The engagement involved a review and testing of the targets listed below.

boost-farm

Repository	https://github.com/ref-finance/boost-farm
Version	`62b27445b9c8e66c8a3e58e90a14356977acdaca`
Type	Rust
Platform	NEAR

ref-contracts

Repository	https://github.com/ref-finance/ref-contracts
Version	`85d2842ac316362527a44462ebb2fe76213c94a4`
Type	Rust
Platform	NEAR

ref-dcl

Repository	https://github.com/ref-finance/ref-dcl
Version	`28f0a70a4bcfd9b0d0ae8f79db3098771e1c6070`
Type	Rust
Platform	NEAR

ref-token

Repository	https://github.com/ref-finance/ref-token
Version	`757af65863dd6e835b591ec37d48720f2c418059`
Type	Rust
Platform	NEAR

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- A full manual review of Ref Finance's Rust code
- Automated analysis using the `cargo-audit` tool identified known vulnerabilities in contract dependencies. Additionally, we built all target smart contracts and triaged warnings produced by the rust compiler and the Clippy linter.
- A custom fuzz harness and tests were developed in an effort to validate critical system invariants and identify rounding errors.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- Although we were able to run DCL tests, we were unable to run tests for the other three targeted repositories due to the presence of deprecated and yanked dev dependencies. Failures in the DCL tests prevented us from using automated tooling such as `necessist` to identify insufficiently or incorrectly tested code. The following three tests failed with the message "Exceeded the prepaid gas":
  - `test_batch_liquidity_gas_100_slot1`
  - `test_batch_liquidity_gas_100_slot2`
  - `test_batch_liquidity_gas_1_slot`
- Although we manually reviewed the exchange math and created some additional fuzz tests to target certain sections of the DCL pool arithmetic, our coverage of this aspect of the system was incomplete, and problems may still be present. In particular, our review of math based on the Curve exchange was insufficient.
- We did not review the burrowland-related functions due to time limitations, but we did review the `boost-farming`-related functions.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description	Policy
Clippy	An open-source Rust linter used to catch common mistakes and unidiomatic Rust code	Appendix C
`cargo-audit`	A Cargo plugin for reviewing project dependencies for known vulnerabilities	Appendix C
Custom fuzz harness	A rust unit test that implements a simple fuzz harness and tests the properties shown below	Appendix C

Areas of Focus

Our automated testing and verification work focused on the following system properties:

- Critical arithmetic should always round in favor of the protocol
- Correctness of financial logic in liquidity provision and swaps

Test Results

The results of this focused testing are detailed below.

DCL exchange. We wrote a custom fuzz harness that generates random seeds from which deterministic pseudorandom numbers were generated for function arguments. We created helper functions for executing critical liquidity management and swap operations and orchestrated them in two entrypoint methods that test the properties shown below across 100 runs. To investigate a particular fuzz run, a `debug_seed` helper method allows the execution of one fuzz run with a predetermined, hardcoded seed value.

Property	Tool	Result
Depositing and then withdrawing liquidity should never be profitable.	Custom	**Passed**
Swapping a token and swapping the same amount back should never be profitable.	Custom	**Passed**

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	We uncovered no issues related to the use of arithmetic within the codebase. The cargo configuration enables overflow checks at runtime and manually resizes integer values to preserve maximum precision during, for example, multiply-then-divide operations.	**Satisfactory**
Auditing	Although many critical operations emit sufficient events for effective off-chain monitoring, we identified multiple instances where events emitted are insufficient (TOB-REF-5). In addition, some assertions can fail without providing the user with any information about what went wrong (TOB-REF-3).	**Moderate**
Authentication / Access Controls	For the most part, access controls are present in all functions. However, we did identify one issue related to the `mft_register` function being callable by anyone (TOB-REF-6). Furthermore, although the process for transferring ownership is error prone (TOB-REF-2), authenticated actions are split up between owners and guardians according to the principle of least privilege.	**Moderate**
Complexity Management	The code is well structured and easy to read. Well-designed types are used throughout the system, and code duplication is minimal. The responsibility of each function is clearly indicated by a descriptive name. However, in-line comments could be expanded to improve the readability of the implementation.	**Satisfactory**
Decentralization	The decentralization of the system depends on how the owner accounts are managed. In the `ref-exchange` contracts, for example, the owner can call `retrieve_unmanaged_token` to perform arbitrary	**Further Investigation Required**

	withdrawals. In addition, all contracts besides the `ref-token` are upgradable.	
Documentation	High-level usage documentation is clearly written and publicly available. However, in-line comments are scarce; most functions do not feature NatSpec comments, and many struct fields lack comments describing the developer's intentions.	**Moderate**
Testing and Verification	The `near-sdk-sim` crate is no longer maintained, and other crates, such as `parity-secp256k1`, have been yanked. This prevented us from running most tests and, of the tests we were able to run, not all passed. The suite includes a combination of unit tests, simulated integration tests, and fuzz tests.	**Further Investigation Required**
Transaction Ordering	Callbacks enforce access controls and explicitly verify the required contract state, mitigating a wide range of transaction-ordering risks. Decentralized exchanges are inevitably exposed to front-running risks but, for example, the `min_output_amount` parameter in the DCL pool adequately mitigates such risks.	**Satisfactory**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Outdated Rust dependencies	Patching	**Undetermined**
2	Lack of two-step process for ownership transfers	Data Validation	**Informational**
3	Require calls without error message	Error Reporting	**Informational**
4	Missing UnorderedSet.insert return value check	Data Validation	**Informational**
5	Insufficient event generation	Auditing and Logging	**Informational**
6	Missing access controls on mft_register	Access Controls	**Informational**
7	Value of is_view argument set to false instead of true	Auditing and Logging	**Informational**
8	min_amount_out hardcoded to zero	Data Validation	**Undetermined**
9	Unexpected panics due to missing validation of token_decimals vector	Data Validation	**Medium**

Detailed Findings

1. Outdated Rust dependencies	
Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Patching	Finding ID: TOB-REF-1
Target: `Cargo.toml`	

Description

Ref Finance depends on multiple outdated package versions that contain known vulnerabilities. These vulnerabilities, which we identified using the `cargo-audit` tool, are summarized below.

Because we did not identify any security issues caused by the use of these packages, the severity of this finding is undetermined. Keep in mind that, even if these vulnerabilities are not triggered directly by the Ref Finance smart contracts, they could be triggered by its dependencies.

Repository	Known Vulnerabilities	Warnings
`boost-farm`	5	4
`ref-contracts`	15	18
`ref-dcl`	7	7
`ref-token`	0	2

The vast majority of vulnerable dependencies are transient dependencies of the `near-sdk` or related crates (e.g. `near-crypto`), so upgrading all NEAR packages to the latest versions is critical.

Recommendations

Short term, update the vulnerable and unmaintained dependencies in the Ref Finance codebases. This will prevent the exploitation of a known vulnerability against the system.

Long term, use the `cargo-audit` tool to ensure that the dependency versions used by the Ref Finance smart contracts are free of known vulnerabilities. Additionally, integrate this tool into your CI/CD pipeline for timely, automatic alerts.

2. Lack of two-step process for ownership transfers

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-REF-2
Target: `ref-token/xref-token/src/owner.rs`, `ref-contracts/ref-exchange/src/owner.rs`	

Description

When called, the `set_owner` method immediately sets the contract owner to the provided account ID. The use of a single step to make such a critical change is error-prone; if the function is called with erroneous input, the results could be irrevocable or difficult to recover from.

```
/// Change owner. Only can be called by owner.
pub fn set_owner(&mut self, owner_id: ValidAccountId) {
    self.assert_owner();
    self.owner_id = owner_id.as_ref().clone();
}
```

Figure 2.1: The `set_owner` method of the `xref-token` contract (`owner.rs#L7-L11`)

Even if the owner role is expected to be held by a DAO contract, transfer of ownership is a high-risk operation that will most likely rarely happen. The requirement of a separate acceptance step before role transfer is completed would guarantee that the address of the new owner contract is able to properly execute transactions on the contracts it will be administering.

Exploit Scenario

Alice invokes `set_owner` to change the contract owner but accidentally enters the wrong address. She permanently loses the ability to set important system parameters and upgrade the contract.

Recommendations

Short term, implement a two-step process for all irrecoverable critical operations. Consider splitting the `set_owner` method into two steps: one to propose the new owner, and another in which the new owner accepts ownership of the contract.

Long term, identify and document all possible actions that can be taken by privileged accounts, along with their associated risks. This will facilitate review of the codebase and help prevent future mistakes.

3. Require calls without error message

Severity: **Informational**	Difficulty: **High**
Type: Error Reporting	Finding ID: TOB-REF-3
Target: `boost-farm/contracts/boost-farming/src/shadow_actions.rs`	

Description

The `require` macro has two calls that lack an error message. Without an error message, it will be difficult to figure out which error caused a transaction to fail.

```
26    require!(msg.is_empty());
```

Figure 3.1: The first `require` call without an error message in the `boost-farming` contract (shadow_actions.rs#L26)

```
60    require!(msg.is_empty());
```

Figure 3.2: The second `require` call without an error message in the `boost-farming` contract (shadow_actions.rs#L60)

Recommendations

Short term, declare a new error message inside `errors.rs`:

```
pub const E501_UNEXPECTED_MSG: &str = "E501: unexpected msg";
```

Figure 3.3: Suggested new error to be placed inside `errors.rs`

Additionally, update the mentioned two `require` macro calls to:

```
require!(msg.is_empty(), E501_UNEXPECTED_MSG);
```

Figure 3.4: Updated require macro call that includes an error message

Long term, ensure that all `require` macro calls in all of the repositories have an error message.

4. Missing UnorderedSet.insert return value check

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-REF-4
Target: `boost-farm/contracts/boost-farming/src/owner.rs`	

Description

The `extend_operators` function does not check the return value of the `UnorderedSet.insert` function call. As a result, trying to add duplicates to the `operators` set will not result in a revert. This does not pose an immediate problem, as this does not cause invalid data to be stored inside the `operators` contract state variable.

```
128     self.data_mut().operators.insert(&operator);
```

Figure 4.1: The `insert` function call that is lacking a return value check (owner.rs#L128)

The `operators` variable is of type `UnorderedSet`. The `UnorderedSet` `insert` and `remove` functions return a Boolean value. This value indicates if the insertion or removal was successful. This value will be `false` when trying to insert a value that already exists in the set, or when trying to remove a value that does not exist in the set.

Unlike the `extend_operators` function, the `remove_operators` function does check the return value of the `UnorderedSet.remove` function call.

```
138     let is_success = self.data_mut().operators.remove(&operator);
139     require!(is_success, E007_INVALID_OPERATOR);
```

Figure 4.2: The `remove` function call that does check the return value (owner.rs#L128)

Recommendations

Short term, update the `extend_operators` function to check the return value of the `operators.insert` function call and revert in case it is false, similar to the `remove_operators` implementation.

Long term, always check return values of functions, and in case they should be ignored, explicitly make that clear using the `let _ = <function call>` pattern, in addition to a comment stating why the return value is ignored.

5. Insufficient event generation	
Severity: **Informational**	Difficulty: **High**
Type: Auditing and Logging	Finding ID: TOB-REF-5
Target: `boost-farm/contracts/boost-farming/src/owner.rs`	

Description

Multiple critical operations do not emit events. As a result, it will be difficult to review the correct behavior of the contracts once they have been deployed.

Events generated during contract execution aid in monitoring, baselining of behavior, and detection of suspicious activity. Without events, users and blockchain-monitoring systems cannot easily detect behavior that falls outside the baseline conditions; malfunctioning contracts and attacks could go undetected.

The following functions should emit an event:

- `grant_next_owner`
- `accept_next_owner`
- `confirm_next_owner`
- `cancel_next_owner`
- `extend_operators`
- `remove_operators`

Throughout the `boost-farming` contract, events are emitted using the NEAR `log` macro, or through the abstraction implemented in the `Event` enum (`events.rs`). Either method will work for emitting events related to the changing of the owner and/or operator accounts.

Recommendations

Short term, add events for all operations that could contribute to a higher level of monitoring and alerting. If certain operations are not set up to emit events to optimize gas usage, they should be comprehensively documented.

Long term, consider using a blockchain-monitoring system to track any suspicious behavior in the contracts. The system relies on several contracts to behave as expected. A monitoring mechanism for critical events would quickly detect any compromised system components.

6. Missing access controls on mft_register

Severity: **Informational**	Difficulty: **Low**
Type: Access Controls	Finding ID: TOB-REF-6
Target: `ref-exchange/src/multi_fungible_token.rs`	

Description

There are no access controls on the `mft_register` function, which allows any account to register an LP token for any other account. Although this does not pose an immediate problem, it might make more sense for an account to be allowed to register an LP token only for that account.

```
171     /// Register LP token of given pool for given account.
172     /// Fails if token_id is not a pool.
173     #[payable]
174     pub fn mft_register(&mut self, token_id: String, account_id: ValidAccountId)
{
175         self.assert_contract_running();
176         let prev_storage = env::storage_usage();
177         match parse_token_id(token_id) {
178             TokenOrPool::Token(_) =>
env::panic(ERR110_INVALID_REGISTER.as_bytes()),
179             TokenOrPool::Pool(pool_id) => {
180                 let mut pool = self.pools.get(pool_id).expect(ERR85_NO_POOL);
181                 pool.share_register(account_id.as_ref());
182                 self.pools.replace(pool_id, &pool);
183                 self.internal_check_storage(prev_storage);
184             }
185         }
186     }
```

Figure 6.1: The `mft_register` function call that is lacking access controls (`multi_fungible_token.rs#L171-L186`)

Recommendations

Short term, update the implementation to allow an account to register an LP token for that account.

Long term, add in-line NatSpec comments to all functions that describe the arguments and conditions that need to be met for the function to execute successfully. This will facilitate a review of the codebase and help prevent future mistakes.

7. Value of is_view argument set to false instead of true

Severity: **Informational**	Difficulty: **Low**
Type: Auditing and Logging	Finding ID: TOB-REF-7
Target: `ref-contracts/ref-exchange/src/lib.rs`	

Description

The `internal_pool_swap_by_output_by_cache` function sets the `is_view` argument to `false`. As a result, events will be emitted by the `pool.swap` function. Since this function uses cached values and is called only from view functions (`views.rs`), the `is_view` argument should be set to `true`, so that no events are emitted.

```
  976     fn internal_pool_swap_by_output_by_cache(
  977         &self,
  978         pool_cache: &mut HashMap<u64, Pool>,
  979         pool_id: u64,
  980         token_in: &AccountId,
  981         amount_out: u128,
  982         token_out: &AccountId,
  983         max_amount_in: Option<u128>,
  984         referral_info: &Option<(AccountId, u32)>,
  985     ) -> u128 {
  986         let mut pool =
pool_cache.remove(&pool_id).unwrap_or(self.pools.get(pool_id).expect(ERR85_NO_POOL))
;
  987         let amount_in = pool.swap_by_output(
  988             token_in,
  989             amount_out,
  990             token_out,
  991             max_amount_in,
  992             AdminFees {
  993                 admin_fee_bps: self.admin_fee_bps,
  994                 exchange_id: env::current_account_id(),
  995                 referral_info: referral_info.clone(),
  996             },
  997             false
  998         );
  999         pool_cache.insert(pool_id, pool);
 1000         amount_in
 1001     }
```

Figure 7.1: The `internal_pool_swap_by_cache` function that is passing `true` as `is_view` value (`lib.rs#L949-L974`)

Recommendations

Short term, set the `is_view` value to `true`.

Long term, improve the unit testing suite to check that no events are emitted from `view` functions.

8. min_amount_out hardcoded to zero

Severity: **Undetermined**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-REF-8
Target: `ref-contracts/ref-exchange/src/lib.rs`	

Description

The "swap by input" branch in the `internal_execute_action_by_cache` hard-codes the `min_amount_out` value to zero (line 924 in figure 9.1), thereby performing no slippage check. On the other hand, the "swap by output" branch does set the `max_amount_in` value based on the function arguments (line 941), meaning a slippage check is performed in "swap by output."

```
903     fn internal_execute_action_by_cache(
904         &self,
905         pool_cache: &mut HashMap<u64, Pool>,
906         token_cache: &mut TokenCache,
907         referral_info: &Option<(AccountId, u32)>,
908         action: &Action,
909         prev_result: ActionResult,
910     ) -> ActionResult {
911         match action {
912             Action::Swap(swap_action) => {
913                 let amount_in = swap_action
914                     .amount_in
915                     .map(|value| value.0)
916                     .unwrap_or_else(|| prev_result.to_amount());
917                 token_cache.sub(&swap_action.token_in, amount_in);
918                 let amount_out = self.internal_pool_swap_by_cache(
919                     pool_cache,
920                     swap_action.pool_id,
921                     &swap_action.token_in,
922                     amount_in,
923                     &swap_action.token_out,
924                     0,
925                     referral_info,
926                 );
927                 token_cache.add(&swap_action.token_out, amount_out);
928                 ActionResult::Amount(U128(amount_out))
929             }
930             Action::SwapByOutput(swap_by_output_action) => {
931                 let amount_out = swap_by_output_action
932                     .amount_out
933                     .map(|value| value.0)
934                     .unwrap_or_else(|| prev_result.to_amount());
```

```
935                 let amount_in = self.internal_pool_swap_by_output_by_cache(
936                     pool_cache,
937                     swap_by_output_action.pool_id,
938                     &swap_by_output_action.token_in,
939                     amount_out,
940                     &swap_by_output_action.token_out,
941                     swap_by_output_action.max_amount_in.map(|v| v.0),
942                     referral_info,
943                 );
944                 ActionResult::Amount(U128(amount_in))
945             }
946         }
947     }
```

Figure 8.1: The `internal_execute_action_by_cache` function that is hard-coding the `min_amount_out` value to zero (`lib.rs#L903-L947`)

Recommendations

Short term, instead of hard-coding the `min_amount_out` to zero, set it based on the function arguments, as is similarly done for the `max_amount_in` value in the "swap by output" branch.

Long term, ensure that slippage checks are performed throughout the entire codebase. If no slippage check should be performed, include a comment explaining why it is unnecessary in that specific place.

9. Unexpected panics due to missing validation of token_decimals vector

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-REF-9
Target: `ref-contracts/ref-exchange/src/stable_swap/mod.rs`, `ref-contracts/ref-exchange/src/rated_swap/mod.rs`, `ref-contracts/ref-exchange/src/degen_swap/mod.rs`	

Description

The functions to create a new pool do not validate that the vector of decimals is equal in length to the vector of token IDs. As a result, in case the decimals vector is smaller, panics will occur at specific places in the smart contract execution. This is due to the assumption that the decimals vector size equals the pool's token IDs vector size.

Since pool creation can be performed only by the owner or a guardian, the difficulty of this issue is high.

We will use the `StableSwapPool` as an example, but the same issue applies to a `RatedSwapPool` and `DegenSwapPool`.

Figure 10.1 shows the function to create a new `StableSwapPool`. This passes the `decimals` vector into the `StableSwapPool::new` function. Figure 10.2 shows the `StableSwapPool::new function`, which does not validate the `token_decimals` vector length. Figure 10.3 shows an example where a panic would occur in case the token decimals vector size is less than a pool's token IDs vector size.

```
183     pub fn add_stable_swap_pool(
184         &mut self,
185         tokens: Vec<ValidAccountId>,
186         decimals: Vec<u8>,
187         fee: u32,
188         amp_factor: u64,
189     ) -> u64 {
190         assert!(self.is_owner_or_guardians(), "{}", ERR100_NOT_ALLOWED);
191         check_token_duplicates(&tokens);
192         self.internal_add_pool(Pool::StableSwapPool(StableSwapPool::new(
193             self.pools.len() as u32,
194             tokens,
195             decimals,
196             amp_factor as u128,
197             fee,
198         )))
```

```
199     }
```

Figure 9.1: The `add_stable_swap_pool` function (lib.rs#L183-L199)

```
48      pub fn new(
49          id: u32,
50          token_account_ids: Vec<ValidAccountId>,
51          token_decimals: Vec<u8>,
52          amp_factor: u128,
53          total_fee: u32,
54      ) -> Self {
55          for decimal in token_decimals.clone().into_iter() {
56              assert!(decimal <= MAX_DECIMAL, "{}", ERR60_DECIMAL_ILLEGAL);
57              assert!(decimal >= MIN_DECIMAL, "{}", ERR60_DECIMAL_ILLEGAL);
58          }
59          assert!(
60              amp_factor >= MIN_AMP && amp_factor <= MAX_AMP,
61              "{}",
62              ERR61_AMP_ILLEGAL
63          );
64          assert!(total_fee < FEE_DIVISOR, "{}", ERR62_FEE_ILLEGAL);
65          Self {
66              token_account_ids: token_account_ids.iter().map(|a|
a.clone().into()).collect(),
67              token_decimals,
68              c_amounts: vec![0u128; token_account_ids.len()],
69              volumes: vec![SwapVolume::default(); token_account_ids.len()],
70              total_fee,
71              shares: LookupMap::new(StorageKey::Shares { pool_id: id }),
72              shares_total_supply: 0,
73              init_amp_factor: amp_factor,
74              target_amp_factor: amp_factor,
75              init_amp_time: 0,
76              stop_amp_time: 0,
77          }
78      }
```

Figure 9.2: The constructor of a `StableSwapPool`, which does not validate the `token_decimals` vector size (mod.rs#L183-L199)

```
120     fn amount_to_c_amount(&self, amount: u128, index: usize) -> u128 {
121         let value = self.token_decimals.get(index).unwrap();
```

Figure 9.3: Example where a panic would happen if the token decimals vector size is less than the amount of token IDs in the pool (mod.rs#L120-L121)

Exploit Scenario

Bob, having the owner role, creates a new stable swap pool but mistakenly passes in a decimals vector that is missing one decimal value. As a result, during the execution, unexpected panics will happen, and the pool is unusable.

Recommendations

Short term, add validation of the `token_decimals` vector length in the constructor of the `StableSwapPool`, `RatedSwapPool`, and `DegenSwapPool`.

Long term, perform validation of all input arguments and add tests to ensure that the validation catches all incorrect inputs.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Automated Testing

This section describes the setup of the automated analysis tools used during this audit.

clippy

The Rust linter Clippy can be installed using `rustup` by running the command `rustup component add clippy`. Invoking `cargo clippy` in the root directory of the project runs the tool.

cargo-audit

The `cargo-audit` Cargo plugin identifies known vulnerable dependencies in Rust projects. It can be installed using `cargo install cargo-audit`. To run the tool, run `cargo audit` in the crate root directory.

Custom Fuzz Tests

We added the following file to the `ref-dcl` repository.

```
use std::num::Wrapping;
use rand::Rng;

mod common;
use crate::common::*;

struct FuzzContext {
    worker: Worker<Sandbox>,
    alice: Account,
    ref_token: Contract,
    near_token: Contract,
    dcl: Contract,
    pool_id: String,
}

pub const FUZZ_RUNS: usize = 100; // TODO: increase once everything works

/// A simple pseudo-random number generator (xoshiro256** algorithm)
pub struct PRNG {
    s: [u64; 4]
}

impl PRNG {
    /// Create a new PRNG from a single u64 seed
    pub fn from_seed(seed: u64) -> Self {
        // Initialize state using SplitMix64 algorithm
        let mut state = [0u64; 4];
        let mut x = Wrapping(seed);
        for s in state.iter_mut() {
            x = Wrapping(x.0.wrapping_add(0x9e3779b97f4a7c15));
```

```
            let mut z = Wrapping(x.0);
            z = Wrapping((z.0 ^ (z.0 >> 30)).wrapping_mul(0xbf58476d1ce4e5b9));
            z = Wrapping((z.0 ^ (z.0 >> 27)).wrapping_mul(0x94d049bb133111eb));
            *s = z.0 ^ (z.0 >> 31);
        }
        Self { s: state }
    }

    /// Generate next random u64
    pub fn next_u64(&mut self) -> u64 {
        let result = self.s[1].wrapping_mul(5).rotate_left(7).wrapping_mul(9);
        let t = self.s[1] << 17;
        self.s[2] ^= self.s[0];
        self.s[3] ^= self.s[1];
        self.s[1] ^= self.s[2];
        self.s[0] ^= self.s[3];
        self.s[2] ^= t;
        self.s[3] = self.s[3].rotate_left(45);
        result
    }
}

fn generate_fuzzy_seeds() -> Vec<u64>{
    let mut seeds:Vec<u64> = Vec::new();
    let mut rng = rand::thread_rng();
    for _ in 0..FUZZ_RUNS {
        let seed: u64 = rng.gen();
        seeds.push(seed);
    }
    seeds
}

#[tokio::test]
pub async fn test_fuzzy_math() {
    let seeds = generate_fuzzy_seeds();
    for seed in seeds {
        println!(" ");
        println!("Fuzzing with seed: {}", seed);
        run_invariants(seed).await;
    }
}

#[tokio::test]
#[ignore]
pub async fn debug_seed() {
    let seed: u64 = 15349020051143494433;
    println!("Fuzzing with seed: {}", seed);
    run_invariants(seed).await;
}

async fn run_invariants(seed: u64) {
    match add_and_remove_liquidity(seed).await {
        Ok(_) => println!("add_and_remove_liquidity() worked :)"),
```

```
        Err(e) => println!("add_and_remove_liquidity() failed: {:?}", e)
    }
    match swap_in_and_out(seed).await {
        Ok(_) => println!("swap_in_and_out() worked :)"),
        Err(e) => println!("swap_in_and_out() failed: {:?}", e)
    }
}

async fn add_and_remove_liquidity(seed: u64) -> anyhow::Result<()> {
    let mut prng = PRNG::from_seed(seed);
    let ctx = setup().await?;
    let init_bal_ref = get_ref_bal(&ctx).await;
    let init_bal_near = get_near_bal(&ctx).await;
    let liq_ref: u128 = prng.next_u64().into();
    let liq_near: u128 = prng.next_u64().into();
    let lpt_info = add_liquidity(&ctx, liq_ref, liq_near).await?;
    rm_liquidity(&ctx, lpt_info).await?;
    let done_bal_ref = get_ref_bal(&ctx).await;
    let done_bal_near = get_near_bal(&ctx).await;
    println!("token balances before liq add/remove: ref={} near={}", init_bal_ref,
init_bal_near);
    println!("token balances after  liq add/remove: ref={} near={}", done_bal_ref,
done_bal_near);
    assert!(done_bal_ref <= init_bal_ref);
    assert!(done_bal_near <= init_bal_near);
    Ok(())
}

async fn swap_in_and_out(seed: u64) -> anyhow::Result<()> {
    let mut prng = PRNG::from_seed(seed);
    let ctx = setup().await?;
    let liq_ref: u128 = prng.next_u64().into();
    let liq_near: u128 = prng.next_u64().into();
    add_liquidity(&ctx, liq_ref, liq_near).await?;
    let init_bal_ref = get_ref_bal(&ctx).await;
    let init_bal_near = get_near_bal(&ctx).await;
    let swap_in: u128 = clamp(clamp(prng.next_u64().into(), liq_near, 0),
init_bal_near, 0);
    // swap near for ref, then same amount of ref for near
    let swap_out = swap(&ctx, swap_in, true).await?;
    swap(&ctx, swap_out, false).await?;
    let done_bal_ref = get_ref_bal(&ctx).await;
    let done_bal_near = get_near_bal(&ctx).await;
    println!("token balances before swap in/out: ref={} near={}", init_bal_ref,
init_bal_near);
    println!("token balances after  swap in/out: ref={} near={}", done_bal_ref,
done_bal_near);
    assert!(done_bal_ref <= init_bal_ref);
    assert!(done_bal_near <= init_bal_near);
    Ok(())
}

async fn setup() -> anyhow::Result<FuzzContext> {
```

```
    let worker = workspaces::sandbox().await?;
    let (root, dcl, _) = initialize_contracts_and_users(&worker).await?;
    let near_token = initialize_mock_ft(&worker, "wrap", 24).await?;
    let ref_token = initialize_mock_ft(&worker, "ref", 18).await?;
    call_ft_storage_deposit(&worker, &dcl.id(), &near_token).await?;
    call_ft_storage_deposit(&worker, &dcl.id(), &ref_token).await?;
    let outcome = call_create_pool(
        &worker,
        &root,
        &dcl,
        near_token.id(),
        ref_token.id(),
        10000, // fee
        5000,  // init point
        parse_near!("1 N"), // storage deposit
    )
    .await?;
    let pool_id = outcome.json::<PoolId>().unwrap();
    let alice = tool_create_account(&worker, &root, "alice", None).await;
    call_ft_mint(&worker, &root, &alice, u64::MAX.into(), &near_token).await?;
    call_ft_mint(&worker, &root, &alice, u64::MAX.into(), &ref_token).await?;
    call_storage_deposit(&worker, &alice, None, parse_near!("1 N"), &dcl,
true).await?;
    Ok(FuzzContext {
        worker,
        alice,
        ref_token,
        near_token,
        dcl,
        pool_id,
    })
}

async fn add_liquidity(ctx: &FuzzContext, amt_ref: u128, amt_near: u128) ->
anyhow::Result<UserLiquidity> {
    call_ft_transfer_call(
        &ctx.worker,
        &ctx.alice,
        ctx.dcl.id(),
        amt_ref,
        "\"Deposit\"".to_string(),
        &ctx.ref_token,
    )
    .await?;
    call_ft_transfer_call(
        &ctx.worker,
        &ctx.alice,
        ctx.dcl.id(),
        amt_near,
        "\"Deposit\"".to_string(),
        &ctx.near_token,
    )
    .await?;
```

```
    // add liquitidy to the center of the range
    let outcome = call_add_liquidity(
        &ctx.worker,
        &ctx.alice,
        &ctx.dcl,
        &ctx.pool_id,
        4000, // left point
        6000, // right point
        amt_ref,
        amt_near,
        0, // min x
        0, // min y
    )
    .await;
    let lpt_id = outcome?.json::<LptId>().unwrap();
    let new_lpt_info: UserLiquidity = view_get_liquidity(&ctx.worker, &ctx.dcl,
&lpt_id)
        .await?
        .unwrap();
    Ok(new_lpt_info)
}

async fn rm_liquidity(ctx: &FuzzContext, lpt: UserLiquidity) -> anyhow::Result<()> {
    call_remove_liquidity(
        &ctx.worker,
        &ctx.alice,
        &ctx.dcl,
        &lpt.lpt_id,
        lpt.amount.into(),
        1, // min x
        1, // min y
    )
    .await?;
    Ok(())
}

async fn swap(ctx: &FuzzContext, swap_amt: u128, swap_token_is_near: bool) ->
anyhow::Result<u128> {
    let in_token = if swap_token_is_near { &ctx.near_token } else { &ctx.ref_token
};
    let out_token = if swap_token_is_near { &ctx.ref_token } else { &ctx.near_token
};
    // println!("Swapping amount={} of is_NEAR={}", swap_amt, swap_token_is_near);
    let init_out_balance = get_ft_balance_of(&ctx.worker, &ctx.alice,
out_token).await?.0;
    let quote_result = view_quote(
        &ctx.worker,
        &ctx.dcl,
        &ctx.alice.id(),
        vec![&ctx.pool_id],
        in_token.id(),
        out_token.id(),
        swap_amt,
```

```
        Some("MyTag".to_string()),
    )
    .await?;
    let expected_swap_out = quote_result.amount;
    call_storage_deposit(&ctx.worker, &ctx.alice, None, parse_near!("1 N"),
&ctx.dcl, true).await?;
    call_ft_transfer_call(
        &ctx.worker,
        &ctx.alice,
        ctx.dcl.id(),
        swap_amt,
        format!("{{\"Swap\": {{\"pool_ids\": [\"{}\"], \"output_token\": \"{}\",
\"min_output_amount\": \"{}\"}}}}", ctx.pool_id, out_token.id(), 0),
        in_token,
    )
    .await?;
    let done_out_balance = get_ft_balance_of(&ctx.worker, &ctx.alice,
out_token).await?.0;
    let actual_swap_out = done_out_balance - init_out_balance;
    // println!("Got {} out, expected {}", actual_swap_out, expected_swap_out.0);
    assert!(actual_swap_out == expected_swap_out.into());
    Ok(actual_swap_out)
}

fn clamp(val: u128, max: u128, min: u128) -> u128 {
    min + (val % (max - min + 1))
}

async fn get_ref_bal(ctx: &FuzzContext) -> u128 {
    get_ft_balance_of(&ctx.worker, &ctx.alice, &ctx.ref_token).await.unwrap().0
}

async fn get_near_bal(ctx: &FuzzContext) -> u128 {
    get_ft_balance_of(&ctx.worker, &ctx.alice, &ctx.near_token).await.unwrap().0
}
```

Figure C.1: Fuzz testing harness and invariant tests

D. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From February 12 to February 14, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the Ref Finance team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the nine issues described in this report, the Ref Finance team resolved six and did not resolve the remaining three. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity
1	Outdated Rust dependencies	Unresolved
2	Lack of two-step process for ownership transfers	Unresolved
3	Require calls without error message	Resolved
4	Missing UnorderedSet.insert return value check	Resolved
5	Insufficient event generation	Resolved
6	Missing access controls on mft_register	Unresolved
7	Value of is_view argument set to false instead of true	Resolved
8	min_amount_out hardcoded to zero	Resolved
9	Unexpected panics due to missing validation of token_decimals vector	Resolved

Detailed Fix Review Results

TOB-REF-1: Outdated Rust dependencies
Unresolved. Ref Finance has provided the following context:

> *Currently, no security issues have been found due to project dependencies. Also, upgrading the main version of the SDK carries significant risks, so no changes will be made for now.*

TOB-REF-2: Lack of two-step process for ownership transfers
Unresolved. Ref Finance has provided the following context:

> *The owner_id is already set to the DAO and will not be modified further.*

TOB-REF-3: Require calls without error message
Resolved in PR 14 to the `boost-farm` repository. Error messages have been added to the noted `require` macro calls.

TOB-REF-4: Missing UnorderedSet.insert return value check
Resolved in PR 14 to the `boost-farm` repository. The value returned by this `insert` method is now verified to ensure that the operator being inserted does not already exist.

TOB-REF-5: Insufficient event generation
Resolved in PR 14 to the `boost-farm` repository. Appropriate events have been added to all functions mentioned by this issue.

TOB-REF-6: Missing access controls on mft_register
Unresolved. Ref Finance has provided the following context:

> *By design.*

TOB-REF-7: Value of is_view argument set to false instead of true
Resolved in PR 113 to the `ref-contracts` repository. The `is_view` argument has been set to true.

TOB-REF-8: min_amount_out hardcoded to zero
Resolved in PR 113 to the `ref-contracts` repository. The existing `min_amount_out` field of the `SwapAction` struct is now properly provided to the internal swap method.

TOB-REF-9: Unexpected panics due to missing validation of token_decimals vector
TBD in PR 113 to the `ref-contracts` repository. Decimal array length checks have been added to the `add_stable_swap_pool`, `add_rated_swap_pool`, and `add_degen_swap_pool` methods of the main `ref-exchange` contract.