BLOCKSEC

Security Audit
Report for Ref-Dcl

Date: October 26, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**
1.1 About the Audit Target . . . 1
1.2 Disclaimer . . . 2
1.3 Procedure of Auditing . . . 2
1.3.1 Security Issues . . . 3
1.3.2 Additional Recommendation . . . 3
1.4 Security Model . . . 3

Chapter 2 Findings **5**
2.1 Security Issue . . . 5
2.1.1 Incorrect handling of empty swap routes allows unbacked asset records . 5
2.1.2 Incorrect sender authentication allows unauthorized access to restricted pools . . . 6
2.1.3 Incorrect referral fee accounting overstates recorded pool balances . . . 7
2.1.4 Incorrect storage deposit validation rejects sufficiently funded token registrations . . . 8
2.2 Note . . . 9
2.2.1 Potential centralization risks . . . 9

Report Manifest

Item	Description
Client	Rhea Finance
Target	Ref-Dcl

Version History

Version	Date	Description
1.0	October 26, 2026	First release

Signature



Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-10-26 05:40:50+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About the Audit Target

Information	Description
Type	Smart Contract
Language	Rust
Approach	Semi-automatic and manual verification

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of Ref-Dcl of Rhea Finance.

This update extends Ref-Dcl with exact-output swaps, batch liquidity operations, liquidity position merging, and market-depth queries. It also introduces pool whitelists, configurable pool taxes, referral fee distribution, and enhanced client-echo controls. Additional changes cover token registration, failed-transfer recovery, and reserve-limited settlement for swaps, liquidity withdrawals, and order cancellations.

Note this audit focuses on the codes located in the following directories/files:

- contracts/dcl/src/api/dcl_api.rs
- contracts/dcl/src/api/dcl_liquidity_api.rs
- contracts/dcl/src/api/dcl_order_api.rs
- contracts/dcl/src/api/deposit_tokens.rs
- contracts/dcl/src/api/lostfound.rs
- contracts/dcl/src/api/management.rs
- contracts/dcl/src/api/mod.rs
- contracts/dcl/src/api/nft.rs
- contracts/dcl/src/api/storage_api.rs
- contracts/dcl/src/api/token_receiver.rs
- contracts/dcl/src/api/user_asset_api.rs
- contracts/dcl/src/api/view.rs
- contracts/dcl/src/client_echo_limit.rs
- contracts/dcl/src/dcl/common_math.rs
- contracts/dcl/src/dcl/dcl_md.rs
- contracts/dcl/src/dcl/pool.rs
- contracts/dcl/src/dcl/slot_bitmap.rs
- contracts/dcl/src/dcl/swap.rs
- contracts/dcl/src/dcl/swap_math.rs
- contracts/dcl/src/dcl/user_liquidity.rs
- contracts/dcl/src/dcl/user_order.rs
- contracts/dcl/src/errors.rs
- contracts/dcl/src/event.rs
- contracts/dcl/src/legacy.rs

[1] https://github.com/ref-finance/ref-dcl

- contracts/dcl/src/lib.rs
- contracts/dcl/src/owner.rs
- contracts/dcl/src/referral.rs
- contracts/dcl/src/user.rs
- contracts/dcl/src/user_asset.rs
- contracts/dcl/src/utils.rs

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
Ref-Dcl	`Version 1`	`fd10c19967bc541c6656bb95af718cb2131c2322`
	`Version 2`	`fa76b465f9e9c173eee2c6b3476e20ed2e9fc15b`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the Target, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of the Target.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.

- **Semantic Analysis** We study the business logic of the Target and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness
* Oracle security
* Business logic correctness
* Semantic and functional consistency
* Emergency mechanism
* Economic and incentive impact

1.3.2 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **four** potential security issues. Besides, we have **one** note.

- Medium Risk: 2
- Low Risk: 2
- Note: 1

ID	Severity	Description	Category	Status
1	Medium	Incorrect handling of empty swap routes allows unbacked asset records	Security Issue	Fixed
2	Medium	Incorrect sender authentication allows unauthorized access to restricted pools	Security Issue	Fixed
3	Low	Incorrect referral fee accounting overstates recorded pool balances	Security Issue	Fixed
4	Low	Incorrect storage deposit validation rejects sufficiently funded token registrations	Security Issue	Fixed
5	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Incorrect handling of empty swap routes allows unbacked asset records

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `swap.rs`, the functions `internal_swap()` and `internal_swap_by_output()` do not check that `pool_ids` is non-empty. All pool access and frozen-token checks run inside the route loop, so an empty route skips every check and performs no exchange. Nevertheless, when the output token specified in the message equals the input token, the request still passes validation, and the caller schedules an outgoing transfer of the full supplied amount.

An attacker can exploit this path through the swap entry point. In the file `token_receiver.rs`, the function `ft_on_transfer()` identifies the incoming token through `env::predecessor_account_id()`, so an account without an FT implementation can call this function directly with an empty route, specify its own account ID as the output token, and supply an arbitrary amount without transferring any tokens. The outgoing call to the function `ft_transfer()` then fails, and the function `callback_post_withdraw_asset()` routes the amount to the function `refund_or_record_lostfound()`, which adds the amount to the designated recipient's internal balance or, if the recipient is unregistered, records it in `lostfound`. In either case, no token registration or storage deposit is required for the new entry.

```
152         pool_ids.iter().for_each(|pool_id| {
153             let pool = self.internal_unwrap_pool(pool_id);
```

```
154             self.assert_pool_running(&pool);
155             self.assert_pool_account_allowed(&pool, account_id);
156             if let Some(client_echo_receiver_id) = client_echo_receiver_id.as_ref() {
157                 self.assert_pool_account_allowed(&pool, client_echo_receiver_id);
158             }
159             let (token_x, token_y, _) = pool_id.parse_pool_id();
160             self.assert_no_frozen_tokens(&[token_x, token_y]);
161         });
```

Listing 2.1: contracts/dcl/src/dcl/swap.rs

```
91  fn ft_on_transfer(
92      &mut self,
93      sender_id: AccountId,
94      amount: U128,
95      msg: String,
96  ) -> PromiseOrValue<U128> {
```

Listing 2.2: contracts/dcl/src/api/token_receiver.rs

Impact Attackers can create unbacked asset records in registered users' balances or the `lostfound` ledger, resulting in incorrect accounting and additional storage costs borne by the protocol.

Suggestion Reject empty swap routes and ensure that failed-transfer refunds only credit amounts backed by an incoming transfer or a prior balance deduction.

2.1.2 Incorrect sender authentication allows unauthorized access to restricted pools

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `set_pool_whitelist()` allows the owner to restrict pool access to specified accounts. However, the swap flow checks each pool's whitelist against the `sender_id` passed to the function `ft_on_transfer()` in the file `token_receiver.rs`. This identity is supplied by the calling input token contract, and unlike the deposit flow, the swap flow does not verify that the input token is approved for the claimed sender. An attacker-controlled token contract can therefore pass any whitelisted account as the `sender_id`.

Pool creation is permissionless, so an attacker can stage the attack with two pools. The attacker creates an unrestricted `M/A` pool, where `M` is a malicious token, and provides liquidity in genuine token `A`. The malicious contract then submits one swap routed through this pool followed by the restricted `A/B` pool under the forged `sender_id`. The first swap supplies the genuine token `A` consumed by the second, and both whitelist checks use the impersonated identity. Setting `swap_out_recipient` to a controlled account delivers token `B` to the attacker.

```
152         pool_ids.iter().for_each(|pool_id| {
153             let pool = self.internal_unwrap_pool(pool_id);
154             self.assert_pool_running(&pool);
```

```
155             self.assert_pool_account_allowed(&pool, account_id);
156             if let Some(client_echo_receiver_id) = client_echo_receiver_id.as_ref() {
157                 self.assert_pool_account_allowed(&pool, client_echo_receiver_id);
158             }
159             let (token_x, token_y, _) = pool_id.parse_pool_id();
160             self.assert_no_frozen_tokens(&[token_x, token_y]);
161         });
```

Listing 2.3: contracts/dcl/src/dcl/swap.rs

```
91  fn ft_on_transfer(
92      &mut self,
93      sender_id: AccountId,
94      amount: U128,
95      msg: String,
96  ) -> PromiseOrValue<U128> {
```

Listing 2.4: contracts/dcl/src/api/token_receiver.rs

Impact An attacker can trade through a whitelisted pool without being an authorized participant.

Suggestion Validate input token eligibility for the claimed sender before applying pool whitelist checks, and ensure that authorization relies on sender identities provided by trusted sources.

2.1.3 Incorrect referral fee accounting overstates recorded pool balances

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Referral fees are allocated from the protocol's share of swap fees and credited to the referrer's internal token balance through the function `internal_try_add_referral_fee()`. However, in the function `internal_swap()`, this allocation only reduces `pool.total_fee_x_charged` or `pool.total_fee_y_charged`, without reducing the corresponding pool token balance, `pool.total_x` or `pool.total_y`. Consequently, the credited amount remains included in the pool's recorded reserves even though it has already been allocated to the referrer. This discrepancy accumulates as additional referral fees are credited. The same accounting error also exists in the functions `internal_swap_by_output()` and `internal_swap_by_stop_point()`, as well as the referral fee distribution during limit-order swaps.

```
194                 let referral_fee = if let Some(rate) = referral_fee_rate {
195                     let ref_fee = protocol_fee * rate as u128 / BP_DENOM;
196                     if ref_fee > 0 {
197                         // Try to credit referral's inner account
198                         let credited = self.internal_try_add_referral_fee(referral_id.as_ref().
                                unwrap(), &pool.token_x, ref_fee);
199                         if credited > 0 {
200                             // Only deduct from pool's tracked protocol fee if referral was
                                    credited
201                             pool.total_fee_x_charged -= ref_fee;
```

```
202                         }
203                         credited
204                     } else {
205                         0
206                     }
207                 } else {
208                     0
209                 };
```

Listing 2.5: contracts/dcl/src/dcl/swap.rs

```
636   pub(crate) fn internal_try_add_referral_fee(&mut self, referral_id: &AccountId, token_id: &
          AccountId, amount: Balance) -> Balance {
637       if let Some(mut user) = self.internal_get_user(referral_id) {
638           user.add_asset(token_id, amount);
639           self.internal_set_user(referral_id, user);
640           amount
641       } else {
642           // Referral account doesn't exist, fee falls back to protocol
643           0
644       }
645   }
```

Listing 2.6: contracts/dcl/src/dcl/swap.rs

Impact The pool's recorded reserves are overstated by the cumulative referral fees credited to referrers.

Suggestion Deduct credited referral fees from the corresponding pool token balance and apply consistent accounting across all affected execution paths.

2.1.4 Incorrect storage deposit validation rejects sufficiently funded token registrations

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `deposit_tokens.rs`, the function `register_tokens()` calculates the storage deposit required to register the specified tokens. However, the deposit check requires the attached amount to be strictly greater than `storage_cost`. Consequently, a request attaching exactly the calculated amount fails with `E104_INSUFFICIENT_DEPOSIT`, even though the storage requirement is fully covered. This prevents the registration from completing unless the user attaches an additional surplus.

```
47   pub fn register_tokens(&mut self, token_ids: Vec<AccountId>) {
48       self.assert_contract_running();
49       let attached = env::attached_deposit();
50       let account_id = env::predecessor_account_id();
51       let mut user = self.internal_unwrap_user(&account_id);
52
53       let before = env::storage_usage();
```

```
54      for token_id in token_ids {
55          require!(user.deposit_tokens.insert(&token_id), E014_DUPLICATE_DEPOSIT_TOKEN);
56      }
57      let storage_cost = (env::storage_usage() - before) as u128
58          * env::storage_byte_cost()
59          * TOKEN_REGISTRATION_STORAGE_MULTIPLIER;
60      require!(attached > storage_cost, E104_INSUFFICIENT_DEPOSIT);
61      self.internal_set_user(&account_id, user);
62      let refund = attached - storage_cost;
63      Promise::new(account_id).transfer(refund);
64  }
```

Listing 2.7: contracts/dcl/src/api/deposit_tokens.rs

Impact Token registration requests revert when the attached deposit equals the calculated storage cost, despite sufficient funding.

Suggestion Accept deposits greater than or equal to the calculated storage cost.

2.2 Note

2.2.1 Potential centralization risks

Introduced by `Version 1`

Description In this project, privileged accounts can perform sensitive operations, introducing centralization risks. The `owner` can upgrade the DCL contract, configure pool taxes and recipients, restrict pool access, and withdraw `lostfound` assets. The `owner` and `operator`s can also pause pools, freeze tokens, modify protocol fee rates, and configure referral rewards and client-echo authorizations. Compromise or misuse of these privileges could threaten user assets, disrupt operations, or redirect fee revenue.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS