Security Audit Report for Ref Contract and Ref Dcl

Date: July 5, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Ref Finance
Target	Ref Contract and Ref Dcl

Version History

Version	Date	Description
1.0	July 5, 2024	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2024-07-05 12:25:45+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 Target Contracts

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

The target of this audit is the code repository of Ref Contract [1] and Ref Dcl [2] of Ref Finance. Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include ref-contracts/ref-contracts/ref-exchange/src folder and ref-dcl/contracts/dcl/src contract only. Specifically, the files covered in this audit include:

```
ref-dcl/contracts/dcl/src/api/dcl_api.rs
ref-dcl/contracts/dcl/src/dcl/pool.rs
ref-dcl/contracts/dcl/src/dcl/swap.rs
ref-dcl/contracts/dcl/src/api/token_receiver.rs

ref-contracts/ref-exchange/src/degen_swap/degen.rs
ref-contracts/ref-exchange/src/degen_swap/math.rs
ref-contracts/ref-exchange/src/degen_swap/mod.rs
ref-contracts/ref-exchange/src/degen_swap/price_oracle.rs
ref-contracts/ref-exchange/src/degen_swap/pyth_oracle.rs
ref-contracts/ref-exchange/src/rated_swap/sfrax_rate.rs
ref-contracts/ref-exchange/src/account_deposit.rs
ref-contracts/ref-exchange/src/action.rs
ref-contracts/ref-exchange/src/custom_keys.rs
ref-contracts/ref-exchange/src/errors.rs
ref-contracts/ref-exchange/src/lib.rs
ref-contracts/ref-exchange/src/oracle.rs
ref-contracts/ref-exchange/src/owner.rs
ref-contracts/ref-exchange/src/pool.rs
ref-contracts/ref-exchange/src/simple_pool.rs
ref-contracts/ref-exchange/src/token_receiver.rs
ref-contracts/ref-exchange/src/views.rs
```

Listing 1.1: Audit Scope for this Report

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.

[1] https://github.com/ref-finance/ref-contracts/tree/degen-pool

[2] https://github.com/ref-finance/ref-dcl/tree/open_create_pool

Project	Version	Commit Hash
Ref Contract	Version 1	37150859766902dc123db58066cc64305f259e42
	Version 2	5090a7ad4ec7d333f7c6d1bb0b7ccf3e929098a9
Ref Dcl	Version 1	ac89456c21b825b92bbadc9ba18f82663f240f70
	Version 2	47267c695f8144b8cc0a9ed7dd7624b7d34cd56b

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 smart contracts, 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 smart contracts.

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 smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts 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 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data flow
- Exception handling
- Untrusted external call and control flow

- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Permission management
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 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 [3] and Common Weakness Enumeration [4]. The overall *severity* of the risk is determined by *likelihood* and *impact*. 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.

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

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

Table 1.1: Vulnerability Severity Classification

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

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 four 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.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **two** potential security issues. Besides, we have **two** recommendations.

- High Risk: 2
- Recommendation: 2

ID	Severity	Description	Category	Status
1	High	Inaccurate output amount calculation in function `internal_swap_by_output()`	DeFi Security	Fixed
2	High	Lack of state update in function `internal_quote_by_output()`	DeFi Security	Fixed
3	-	Redundant check in function `swap()`	Recommendation	Fixed
4	-	Duplicated price requests	Recommendation	Fixed

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Inaccurate output amount calculation in function `internal_swap_by_output()`

Severity High

Status Fixed at `Version 2`

Introduced by `Version 1`

Description In the function `internal_swap_by_output()`, the `actual_output_amount` variable, which is calculated and updated during the swap process, is used to determine the amount of output tokens sent to the user. However, this variable does not accurately represent the actual output token amount during the swap.

```
pub fn internal_swap_by_output(
    &mut self,
    account_id: &AccountId,
    pool_ids: Vec<PoolId>,
    input_token: &AccountId,
    max_input_amount: Balance,
    output_token: &AccountId,
    output_amount: Balance,
    skip_unwrap_near: Option<bool>,
    client_echo: Option<String>,
) -> Balance {
    pool_ids.iter().for_each(|pool_id| {
        self.assert_pool_running(&self.internal_unwrap_pool(pool_id));
        let (token_x, token_y, _) = pool_id.parse_pool_id();
        self.assert_no_frozen_tokens(&[token_x, token_y]);
    });

    let protocol_fee_rate = self.data().protocol_fee_rate;
    let vip_info = self.data().vip_users.get(account_id);
    let (actual_input_token, actual_input_amount, actual_output_amount) = {
        let mut next_desire_token = output_token.clone();
        let mut next_desire_amount = output_amount;
        let mut actual_output_amount = output_amount;
        for pool_id in pool_ids.iter() {
            let mut pool = self.internal_unwrap_pool(&pool_id);

            let pool_fee = pool.get_pool_fee_by_user(&vip_info);

            if next_desire_token.eq(&pool.token_x) {
                let (need_amount, acquire_amount, is_finished, total_fee, protocol_fee) = pool.internal_y_swap_x_desire_x(pool_fee, protocol_fee_rate, next_desire_amount, 800001, false);
                if !is_finished {
                    env::panic_str(&format!("ERR_TOKEN_{}_NOT_ENOUGH", pool.token_x.to_string().to_uppercase()));
                }

                pool.total_y += need_amount;
                pool.total_x -= acquire_amount;
                pool.volume_y_in += U256::from(need_amount);
                pool.volume_x_out += U256::from(acquire_amount);

                actual_output_amount = acquire_amount;
                next_desire_token = pool.token_y.clone();
                next_desire_amount = need_amount;

                Event::SwapDesire {
                    swapper: account_id,
                    token_in: &pool.token_y,
                    token_out: &pool.token_x,
                    amount_in: &U128(need_amount),
                    amount_out: &U128(acquire_amount),
                    pool_id: &pool.pool_id,
                    total_fee: &U128(total_fee),
                    protocol_fee: &U128(protocol_fee),
                }
                .emit();
            } else if next_desire_token.eq(&pool.token_y) {
                let (need_amount, acquire_amount, is_finished, total_fee, protocol_fee) = pool.internal_x_swap_y_desire_y(pool_fee, protocol_fee_rate, next_desire_amount, -800001, false);
                if !is_finished {
                    env::panic_str(&format!("ERR_TOKEN_{}_NOT_ENOUGH", pool.token_y.to_string().to_uppercase()));
                }

                pool.total_x += need_amount;
                pool.total_y -= acquire_amount;
                pool.volume_x_in += U256::from(need_amount);
                pool.volume_y_out += U256::from(acquire_amount);

                actual_output_amount = acquire_amount;
                next_desire_token = pool.token_x.clone();
```

```
                    next_desire_amount = need_amount;

                    Event::SwapDesire {
                        swapper: account_id,
                        token_in: &pool.token_x,
                        token_out: &pool.token_y,
                        amount_in: &U128(need_amount),
                        amount_out: &U128(acquire_amount),
                        pool_id: &pool.pool_id,
                        total_fee: &U128(total_fee),
                        protocol_fee: &U128(protocol_fee),
                    }
                    .emit();
                } else {
                    env::panic_str(E404_INVALID_POOL_IDS);
                }
                self.internal_set_pool(&pool_id, pool);
            }
            (next_desire_token, next_desire_amount, actual_output_amount)
        };
        require!(input_token == &actual_input_token, E213_INVALID_INPUT_TOKEN);
        require!(actual_input_amount <= max_input_amount, E204_SLIPPAGE_ERR);

        if actual_output_amount > 0 {
            if let Some(msg) = client_echo {
                self.process_ft_transfer_call(account_id, &output_token, actual_output_amount, msg);
            } else {
                self.process_transfer(account_id, &output_token, actual_output_amount, skip_unwrap_near);
            }
        }

        actual_input_amount
    }
```

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

Impact The inaccurate output amount calculation in function `internal_swap_by_output()` leads to incorrect internal accounting. This allows attackers to receive more tokens than they should be entitled to.

Suggestion Revise the output token amount accordingly.

2.1.2 Lack of state update in function `internal_quote_by_output()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `internal_quote_by_output()` function, the state of the `pool` modified during the quoting process is not written back to the `pool_cache`. Therefore, if the same pool is accessed again, the retrieved state will be incorrect.

```
pub fn internal_quote_by_output(
    &self,
    pool_cache: &mut HashMap<PoolId, Pool>,
    vip_info: Option<HashMap<PoolId, u32>>,
    pool_ids: Vec<PoolId>,
    input_token: AccountId,
    output_token: AccountId,
    output_amount: U128,
    tag: Option<String>,
) -> QuoteResult {
    let quote_failed = QuoteResult {
        amount: 0.into(),
        tag: tag.clone(),
    };
    if self.data().state == RunningState::Paused {
        return quote_failed;
    }

    let protocol_fee_rate = self.data().protocol_fee_rate;

    let (actual_input_token, actual_input_amount) = {
        let mut next_desire_token = output_token;
        let mut next_desire_amount = output_amount.0;
        for pool_id in pool_ids {
            let mut pool = pool_cache.remove(&pool_id).unwrap_or(self.internal_unwrap_pool(&pool_id));
            if pool.state == RunningState::Paused ||
                self.data().frozenlist.contains(&pool.token_x) || self.data().frozenlist.contains(&pool.token_y) {
                return quote_failed;
            }

            let pool_fee = pool.get_pool_fee_by_user(&vip_info);

            let is_finished = if next_desire_token.eq(&pool.token_x) {
                let (need_amount, _, is_finished, _, _) = pool.internal_y_swap_x_desire_x(pool_fee, protocol_fee_rate, next_desire_amount, 800001, true);
                next_desire_token = pool.token_y.clone();
                next_desire_amount = need_amount;
                is_finished
            } else if next_desire_token.eq(&pool.token_y) {
                let (need_amount, _, is_finished, _, _) = pool.internal_x_swap_y_desire_y(pool_fee, protocol_fee_rate, next_desire_amount, -800001, true);
                next_desire_token = pool.token_x.clone();
                next_desire_amount = need_amount;
                is_finished
            } else {
                return quote_failed;
            };
            if !is_finished {
                return quote_failed;
            }
        }
        (next_desire_token, next_desire_amount)
    };
    if input_token != actual_input_token {
        return quote_failed;
    }
    QuoteResult {
        amount: actual_input_amount.into(),
        tag,
    }
}
```

Listing 2.2: ref-dcl/contracts/dcl/src/dcl/swap.rs

Impact This can lead to erroneous results if duplicate pool ids are provided.

Suggestion Write back the updated `pool` state to `pool_cache`.

2.2 Additional Recommendation

2.2.1 Redundant check in function `swap()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `assert_contract_running()` check is redundant in functions `swap()` and `swap_by_output()` since the same check will be performed in the function `execute_actions()`.

```
    /// Execute set of swap actions between pools.
    /// If referrer provided, pays referral_fee to it.
    /// If no attached deposit, outgoing tokens used in swaps must be whitelisted.
    #[payable]
    pub fn swap(&mut self, actions: Vec<SwapAction>, referral_id: Option<ValidAccountId>) -> U128
        {
        self.assert_contract_running();
        U128(
            self.execute_actions(
                actions
                    .into_iter()
                    .map(|swap_action| Action::Swap(swap_action))
                    .collect(),
                referral_id,
            )
            .to_amount(),
        )
    }

    /// Execute set of swap_by_output actions between pools.
    /// If referrer provided, pays referral_fee to it.
    /// If no attached deposit, outgoing tokens used in swaps must be whitelisted.
    #[payable]
```

```
pub fn swap_by_output(&mut self, actions: Vec<SwapByOutputAction>, referral_id: Option<ValidAccountId>) -> U128 {
    self.assert_contract_running();
    U128(
        self.execute_actions(
            actions
                .into_iter()
                .map(|swap_by_output_action| Action::SwapByOutput(swap_by_output_action))
                .collect(),
            referral_id,
        )
        .to_amount(),
    )
}
```

Listing 2.3: ref-contracts/ref-exchange/src/lib.rs

Suggestion Remove the redundant check.

2.2.2 Duplicated price requests

Status Fixed in Version 2

Introduced by Version 1

Description Every time the function swap() of DegenSwapPool is invoked, it requests price synchronization for all tokens in the pool from the oracles. However, since the swap() function can be invoked multiple times within a single transaction, this may result in redundant token price requests, leading to unnecessary gas consumption.

```
pub fn swap(
    &mut self,
    token_in: &AccountId,
    amount_in: Balance,
    token_out: &AccountId,
    min_amount_out: Balance,
    fees: &AdminFees,
    is_view: bool
) -> Balance {

    assert_ne!(token_in, token_out, "{}", ERR71_SWAP_DUP_TOKENS);
    let in_idx = self.token_index(token_in);
    let out_idx = self.token_index(token_out);
    let result = self.internal_get_return(in_idx, amount_in, out_idx, &fees);
    let amount_swapped = self.c_amount_to_amount(result.amount_swapped, out_idx);
    assert!(
        amount_swapped >= min_amount_out,
        "{}",
        ERR68_SLIPPAGE
    );
    if !is_view {
        env::log(
            format!(
                "Swapped {} {} for {} {}, total fee {}, admin fee {}",
                amount_in, token_in, amount_swapped, token_out,
                self.c_amount_to_amount(result.fee, out_idx),
                self.c_amount_to_amount(result.admin_fee, out_idx)
            )
            .as_bytes(),
        );
    }

    self.c_amounts[in_idx] = result.new_source_amount;
    self.c_amounts[out_idx] = result.new_destination_amount;
    self.assert_min_reserve(self.c_amounts[out_idx]);

    // Keeping track of volume per each input traded separately.
    self.volumes[in_idx].input.0 += amount_in;
    self.volumes[out_idx].output.0 += amount_swapped;

    // handle admin fee.
    if fees.admin_fee_bps > 0 && result.admin_fee > 0 {
        let (exchange_share, referral_share) = if let Some((referral_id, referral_fee)) = &fees.referral_info {
            if self.shares.contains_key(referral_id)
            {
                self.distribute_admin_fee(&fees.exchange_id, referral_id, *referral_fee, out_idx, result.admin_fee, is_view)
            } else {
                self.distribute_admin_fee(&fees.exchange_id, referral_id, 0, out_idx, result.admin_fee, is_view)
            }
        } else {
            self.distribute_admin_fee(&fees.exchange_id, &fees.exchange_id, 0, out_idx, result.admin_fee, is_view)
        };
        if !is_view {
            if referral_share > 0 {
                env::log(
                    format!(
                        "Exchange {} got {} shares, Referral {} got {} shares",
                        &fees.exchange_id, exchange_share, &fees.referral_info.as_ref().unwrap().0, referral_share,
                    )
                    .as_bytes(),
                );
            } else {
                env::log(
                    format!(
                        "Exchange {} got {} shares, No referral fee",
                        &fees.exchange_id, exchange_share,
                    )
                    .as_bytes(),
                );
            }
        }
    }

    if !is_view {
        for token_id in self.token_account_ids.iter() {
            let degen = global_get_degen(token_id);
            degen.sync_token_price(token_id);
        }
    }

    amount_swapped
}
```

Listing 2.4: ref-contracts/ref-exchange/src/lib.rs

Suggestion Add checks to verify if the token is currently in the midst of syncing its price, and only request an update from the oracle if it is not already in the sync process.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS