Security Audit Report for HaloTrade Contracts

Date: June 3, 2023

Version: 1.0

Contact: contact@blocksec.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2023.06.06 10:23:56 +08'00'

Contents

Report Manifest

Item	Description
Client	HaloTrade
Target	HaloTrade Contracts

Version History

Version	Date	Description
1.0	June 3, 2023	First Release

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 5 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 repo of HaloTrade Contracts [1] of the HaloTrade project. The HaloTrade project is an AMM market on Aura network that allows users to trade native and CW20 (a token standard like ERC20) tokens.

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.

Project	Version	Commit Hash
HaloTrade Contracts	`Version 1`	`e26a0321d7c27f0467cb85c50842a398772dea95`
	`Version 2`	`f56934c91712d35ab113cb602de19fe95c3165b9`

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 (i.e., the Rust 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.

[1] https://github.com/halotrade-zone/halotrade-contracts/tree/v1.0.0-beta

- **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 [2] and Common Weakness Enumeration [3]. 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.

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 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.

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

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

Chapter 2 Findings

In total, we find **four** potential issues. Besides, we have **two** recommendations and **three** notes.

- High Risk: 1
- Medium Risk: 1
- Low Risk: 2
- Recommendation: 2
- Note: 3

ID	Severity	Description	Category	Status
1	High	Unverified input amount in the CW20 callback	Software Security	Fixed
2	Low	Potential front-running in pair creation	Software Security	Fixed
3	Medium	Non-updatable native token decimals	DeFi Security	Fixed
4	Low	Incorrect amount of liquidity calculation	DeFi Security	Fixed
5	-	Implement access control for `AssertMinimumReceive`	Recommendation	Fixed
6	-	Verify parameters for the commission rate in the pair	Recommendation	Fixed
7	-	Deflation tokens are not supported	Note	-
8	-	Alteration of native token decimals	Note	-
9	-	Potential dust draining in the router	Note	-

The details are provided in the following sections.

2.1 Software Security

2.1.1 Unverified input amount in the CW20 callback

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `halo-pair` contract, the `receive_cw20` function is invoked by the `Receive` callback of the CW20 implementation. This enables users to first transfer CW20 tokens to the pair contract and subsequently initiate the swap logic in the callback. However, the CW20 standard specifies that the transferred token amount is encoded in the `cw20_msg` struct, but the pair contract does not utilize that field as the swap input amount. Instead, the swap logic relies on the amount in the `Cw20HookMsg::Swap` message, which is wrapped in `cw20_msg` and controlled by the user.

As a consequence, malicious users can exploit this to fabricate a false input amount for the swap. For example, an attacker could send one token to the pair while indicating the `offer_asset` amount as 1000 in the `Cw20HookMsg::Swap` message. In the most severe case, this vulnerability could result in the total loss of all funds in the pair.

```
121 pub fn receive_cw20(
122     deps: DepsMut,
123     env: Env,
```

```
124     info: MessageInfo,
125     cw20_msg: Cw20ReceiveMsg,
126 ) -> Result<Response, ContractError> {
127     // let contract_addr = info.sender.clone();
128
129     match from_binary(&cw20_msg.msg) {
130         Ok(Cw20HookMsg::Swap {
131             offer_asset,
132             belief_price,
133             max_spread,
134             to,
135         }) => {
136             // only asset contract can execute this message
137             let mut authorized: bool = false;
138             let config: PairInfoRaw = PAIR_INFO.load(deps.storage)?;
139             let pools: [Asset; 2] =
140                 config.query_pools(&deps.querier, deps.api, env.contract.address.clone())?;
141             for pool in pools.iter() {
142                 if let AssetInfo::Token { contract_addr, .. } = &pool.info {
143                     if contract_addr == &info.sender {
144                         authorized = true;
145                     }
146                 }
147             }
148             if !authorized {
149                 return Err(ContractError::Unauthorized {});
150             }
151
152             let to_addr = if let Some(to_addr) = to {
153                 Some(deps.api.addr_validate(to_addr.as_str())?)
154             } else {
155                 None
156             };
157
158             swap(
159                 deps,
160                 env,
161                 info,
162                 Addr::unchecked(cw20_msg.sender),
163                 offer_asset,
164                 belief_price,
165                 max_spread,
166                 to_addr,
167             )
168         }
```

Listing 2.1: contracts/halo-pair/src/contract.rs

Impact The exploiter has the capability to drain all the funds from any given pair.

Suggestion Verify the amounts sent to the pair in the `Receive` callback.

2.1.2 Potential front-running in pair creation

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The pair creator, being the sole entity whitelisted to supply initial liquidity for a pair, introduces a vulnerability to front-running attacks. Malicious users can exploit this by front-running a pair creation message, thereby hindering others from offering the initial liquidity.

```
19pub fn calculate_lp_token_amount_to_user(
20    info: &MessageInfo,
21    pair_info: &PairInfoRaw,
22    lp_total_supply: Uint128,
23    deposits: [Uint128; 2],
24    pools: [Asset; 2],
25) -> Result<Uint128, ContractError> {
26    if lp_total_supply == Uint128::zero() {
27        // when pool is empty
28        // if the sender is not in whitelist of requirements, then return error
29        if !pair_info.requirements.whitelist.contains(&info.sender) {
30            return Err(ContractError::Std(StdError::generic_err(
31                "the sender is not in whitelist",
32            )));
33        }
34
35        // if the minimum amount of deposit is not satisfied, then return error
36        if deposits[0] < pair_info.requirements.first_asset_minimum
37            || deposits[1] < pair_info.requirements.second_asset_minimum
38        {
39            return Err(ContractError::Std(StdError::generic_err(
40                "the minimum deposit is not satisfied",
41            )));
42        }
```

Listing 2.2: packages/haloswap/src/formulas.rs

Impact Pair creation messages are vulnerable to front-running attacks, which can prevent other users from providing the initial liquidity.

Suggestion Enforce appropriate countermeasures.

2.2 DeFi Security

2.2.1 Non-updatable native token decimals

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The factory stores the decimals of native tokens, which can be set and modified by the factory owner (i.e., the deployer of the factory). Nonetheless, modifying the decimals of native tokens in the factory does not affect the decimals stored in pairs that were created earlier.

```
197    pub fn execute_add_native_token_decimals(
198        deps: DepsMut,
199        env: Env,
200        info: MessageInfo,
201        denom: String,
202        decimals: u8,
203    ) -> StdResult<Response> {
204        let config: Config = CONFIG.load(deps.storage)?;
205
206        // permission check
207        if deps.api.addr_canonicalize(info.sender.as_str())? != config.owner {
208            return Err(StdError::generic_err("unauthorized"));
209        }
210
211        let balance = query_balance(&deps.querier, env.contract.address, denom.to_string())?;
212        if balance.is_zero() {
213            return Err(StdError::generic_err(
214                "a balance greater than zero is required by the factory for verification",
215            ));
216        }
217
218        add_allow_native_token(deps.storage, denom.to_string(), decimals)?;
219
220        Ok(Response::new().add_attributes(vec![
221            ("action", "add_allow_native_token"),
222            ("denom", &denom),
223            ("decimals", &decimals.to_string()),
224        ]))
225    }
```

Listing 2.3: contracts/halo-pair/src/contract.rs

Impact Updating the decimal of a native token after the creation of pairs may result in malfunctioning of the associated pairs.

Suggestion N/A

2.2.2 Incorrect amount of liquidity calculation

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `provide_liquidity` function of the `halo-pair` contract, all the liquidity minted would be reduced by 1 (i.e., the `LP_TOKEN_RESERVED_AMOUNT` constant). The related logic is shown in the following code snippet.

```
297    // mint amount of 'share' LP token to the receiver
298    messages.push(CosmosMsg::Wasm(WasmMsg::Execute {
```

```
299         contract_addr: deps
300             .api
301             .addr_humanize(&pair_info.liquidity_token)?
302             .to_string(),
303         msg: to_binary(&Cw20ExecuteMsg::Mint {
304             recipient: receiver.to_string(),
305             amount: share - Uint128::from(LP_TOKEN_RESERVED_AMOUNT),
306         })?,
307         funds: vec![],
308     }));
```

Listing 2.4: contracts/halo-pair/src/contract.rs

Impact Each invocation of the `provide_liquidity` function, when supplying liquidity, would lead to an insufficient amount of LP tokens being minted for the user.

Suggestion Revise the shares calculation logic.

2.3 Additional Recommendation

2.3.1 Implement access control for `AssertMinimumReceive`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `ExecuteSwapOperation` and `AssertMinimumReceive` messages in the `halo-router` contract are intended for internal use only. Although access control has been applied to the `ExecuteSwapOperation` message to restrict the caller, no access control is currently in place for the `AssertMinimumReceive` message.

```
 6pub fn assert_minium_receive(
 7    deps: Deps,
 8    asset_info: AssetInfo,
 9    prev_balance: Uint128,
10    minium_receive: Uint128,
11    receiver: Addr,
12) -> StdResult<Response> {
13    let receiver_balance = asset_info.query_pool(&deps.querier, deps.api, receiver)?;
14    let swap_amount = receiver_balance.checked_sub(prev_balance)?;
15
16    if swap_amount < minium_receive {
17        return Err(StdError::generic_err(format!(
18            "assertion failed; minimum receive amount: {}, swap amount: {}",
19            minium_receive, swap_amount
20        )));
21    }
22
23    Ok(Response::default())
24}
```

Listing 2.5: contracts/halo-router/src/assert.rs

Impact N/A

Suggestion Implement access control checks for the `AssertMinimumReceive` message.

2.3.2 Verify parameters for the commission rate in the pair

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The commission rate represents the fee applied to swaps in the `halo-pair` contract and is calculated based on a ratio. However, during instantiation, the contract does not currently verify if the commission rate is strictly greater than zero. Furthermore, an upper limit should be established for the commission rate to guarantee that the pair remains functional.

```
28    #[cfg_attr(not(feature = "library"), entry_point)]
29    pub fn instantiate(
30        deps: DepsMut,
31        env: Env,
32        _info: MessageInfo,
33        msg: InstantiateMsg,
34    ) -> StdResult<Response> {
35        set_contract_version(deps.storage, CONTRACT_NAME, CONTRACT_VERSION)?;
36
37        let pair_info: &PairInfoRaw = &PairInfoRaw {
38            contract_addr: deps.api.addr_canonicalize(env.contract.address.as_str())?,
39            liquidity_token: CanonicalAddr::from(vec![]),
40            asset_infos: [
41                msg.asset_infos[0].to_raw(deps.api)?,
42                msg.asset_infos[1].to_raw(deps.api)?,
43            ],
44            asset_decimals: msg.asset_decimals,
45            requirements: msg.requirements,
46            commission_rate: msg.commission_rate,
47        };
48
49        PAIR_INFO.save(deps.storage, pair_info)?;
50
51        COMMISSION_RATE_INFO.save(deps.storage, &msg.commission_rate)?;
```

Listing 2.6: contracts/halo-pair/src/contract.rs

Impact An improperly configured commission rate could make the pair non-functional.

Suggestion Verify the commission rate in the pair creation function.

2.4 Note

2.4.1 Deflation tokens are not supported

Description In the Ethereum DeFi context, there exists a category of tokens called “deflation token” which impose fees during token transfers. For example, when `A` is a deflation token and Alice transfers 100 A tokens to Bob, Bob only receives 95 A tokens due to the fee.

Currently, custom contract code deployment is disallowed on the Aura network, so there is no risk of deflation tokens (standard CW20 tokens are non-deflationary). However, if custom contract code deployment becomes permissible, HaloTrade would not be able to handle deflation tokens.

2.4.2 Alteration of native token decimals

Description HaloTrade contracts allow the project (i.e., the factory deployer) to set, store, and alter native token information, specifically the decimals of native token denominations. Nonetheless, this centralized approach introduces a risk, as changing the decimals of native tokens can considerably affect the pricing of related pairs.

Feedback from the Project Agreed, normally the native token decimal should be queried from the base chain through here: `https://lcd.aura.network/#/Query/CosmosBankV1Beta1DenomMetadata`, this is OK for Aura native token. However, we don't have any way to query ibc tokens because they are also treated as native. Token decimals only affect visualization of the app, we don't use it in calculation so at the moment we haven't figured out a way to do this elegantly yet. Would love to have some recommendations.

2.4.3 Potential dust draining in the router

Description HaloTrade contracts presume that tokens involved in a swap are initially transferred to the router and then forwarded to the specific pairs. In practice, however, some funds might remain in the router due to errors or other factors. Under such circumstances, a user could send a minimal unit of the token to the router, deceiving it into executing a swap for the remaining tokens and generating a profit.