Brokkr Protocol Delta-neutral

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 4th, 2022 - April 15th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/04/2022	Jakub Heba
0.2	Document Update	4/13/2022	Jose C. Ramirez
0.3	Draft Version	04/15/2022	Jakub Heba
0.4	Draft Review	04/22/2022	Gabi Urrutia
1.0	Remediation Plan	04/28/2022	Jakub Heba
1.1	Remediation Plan Review	04/29/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Jakub Heba	Halborn	Jakub.Heba@halborn.com
Jose Ramirez	Halborn	Jose.Ramirez@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Brokkr Protocol engaged Halborn to conduct a security audit on their smart contracts beginning on April 4th, 2022 and ending on April 15th, 2022. The security assessment was scoped to the Delta-neutral smart contract provided in the GitHub repository Brotocol Strategies, commit hashes and further details can be found in the Scope section of this report.
The contract is part of a set that implement sophisticated DeFi investment strategies, in particular the one named delta neutral.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned two full-time security engineers to audit the security of the smart contract. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which has been partially addressed by Brokkr team. The main ones are the following:

- Normalize all addresses provided by users in contract's logic.
- Protect configuration parameters against abnormally low/high values by setting proper bounds.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

The following phases and associated tools were used throughout the term of the audit:

- Research into the architecture, purpose, and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could led to arithmetic vulnerabilities.
- Finding unsafe Rust code usage (cargo-geiger)
- Active Fuzz testing (Halborn custom fuzzing tool).
- Test coverage review (cargo tarpaulin).
- Scanning of Rust files for common vulnerabilities (cargo audit).
- Local or public Testnet deployment (LocalTerra or bombay-12)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

1. CosmWasm Smart Contracts
   (a) Repository: brotocol-strategies
   (b) Commit ID: 0676efcdeba1cb303cdd77e87139dbbe2e6210d8
   (c) Contracts in scope:
      - delta_neutral

Out-of-scope: External libraries and financial related attacks.

EXECUTIVE OVERVIEW

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	4

LIKELIHOOD

IMPACT

🟨	🟨	🟥	🟥	🟪
🟩 (HAL-01)	🟨	🟨	🟥	🟥
🟩	🟩	🟨	🟨	🟥
⬜ (HAL-03)	🟩	🟩	🟨	🟨
⬜ (HAL-05) (HAL-06)	⬜ (HAL-04)	🟩	🟩 (HAL-02)	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) - LACK OF ADDRESS NORMALIZATION	Low	SOLVED - 04/28/2022
(HAL-02) - MISSING BOUNDS ON CONFIGURATION VARIABLES	Low	RISK ACCEPTED
(HAL-03) - MISSING DEDICATED ROLES TO MANAGE CONTRACT STATUS	Informational	ACKNOWLEDGED
(HAL-04) - UNUSED CONFIG VARIABLE	Informational	ACKNOWLEDGED
(HAL-05) - NO CONVENTION IN VARIABLE TYPES	Informational	ACKNOWLEDGED
(HAL-06) - UNMANTAINED DEPENDENCY	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) LACK OF ADDRESS NORMALIZATION - LOW

Description:

The multiple features of the delta_neutral contract do not consider that Terra addresses are valid both upper and all lower case. Although valid, a strict comparison between the same address in its all uppercase version (e.g.: **TERRA1KG...XNL8**) and its all lowercase version (e.g.: **terra1kg...xnl8**) failure.

The likelihood of this issue was reduced as the affected functions were owner-only functionalities, therefore much less prone to error, or queries. Queries affected by this issue will only cause inconvenience rather than a security issue.

Undesired situations could occur, such as loss of control over the owner or dependent contract addresses in case of instantiation with a misvalidated address made up of capital letters. Because update_config does not provide an option for changing the owner, as well as some addresses of the contracts, the administrator will lose access to contract management.

Code Location:

Listing 1: delta_neutral/src/contract.rs (Lines 57,62,63,64,65,66,67,68,69,72,73,76)

```
51 let config = Config {
52         minimum_investment_in_uusd: msg.minimum_investment_in_uusd
   ↳ ,
53         maximum_cdps_per_user_per_asset: msg.
   ↳ maximum_cdps_per_user_per_asset,
54         owner: deps.api.addr_canonicalize(&msg.owner)?,
55         is_test_contract: msg.is_test_contract,
56         is_paused: msg.is_paused,
57         fee_recipient: deps.api.addr_validate(&msg.fee_recipient)
   ↳ ?,
58         fee_pct: Decimal::from_ratio(msg.fee_pct.numerator, msg.
fee_pct.denominator),
        maximum_total_investment_in_uusd: msg.maximum_total_investment_in_uusd,
        maximum_user_investment_in_uusd: msg.maximum_user_investment_in_uusd,
        is_reward_distribution_paused: msg.is_reward_distribution_paused,
        aust_token_address: deps.api.addr_validate(&msg.aust_token_address)?,
        mir_token_address: deps.api.addr_validate(&msg.mir_token_address)?,
        bro_token_address: deps.api.addr_validate(&msg.bro_token_address)?,
        ts_factory_contract_address: deps.api.addr_validate(&msg.ts_factory_contract_address)?,
        a_market_contract_address: deps.api.addr_validate(&msg.a_market_contract_address)?,
        mirr_mint_contract_address: deps.api.addr_validate(&msg.mirr_mint_contract_address)?,
        mirr_lock_contract_address: deps.api.addr_validate(&msg.mirr_lock_contract_address)?,
        mirr_staking_contract_address: deps
            .api
            .addr_validate(&msg.mirr_staking_contract_address)?,
        mirr_masset_oracle: deps.api.addr_validate(&msg.mirr_masset_oracle)?,
        mirr_factory_contract_address: deps
            .api
            .addr_validate(&msg.mirr_factory_contract_address)?,
        astro_router_contract_address: deps
            .api
            .addr_validate(&msg.astro_router_contract_address)?,
    };
```

The above-mentioned lack of normalization was also noted in the following lines of the contracts:

Listing 2: Affected resources

```
delta_neutral/src/contract.rs: #284
delta_neutral/src/queries.rs: #122, 136, 175
delta_neutral/src/conversions.rs: #291, 292
delta_neutral/src/commands.rs: #294, 448, 563
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

One of the two approaches detailed below should be used:

- Update the cosmwasm-vm and use cosmwasm_std::Api::addr_validate (reference CWA-2022-002).
- If the update mentioned is not possible, addresses could be stored in canonical format by using the cosmwasm_std::Api::addr_canonicalize utility function.

The following considerations should be considered when implementing the second option:

- To successfully compare a canonical address, both ends should be in canonical format. For example, when performing access controls, the sender (e.g.: info.sender or env.message.sender) should be canonicalized beforehand too.
- To send funds to a canonicalized address or include them into a message to a different contract, they should be first turn into its human-readable format via the cosmwasm_std::Api::addr_humanize utility function

Remediation plan:

SOLVED: The issue was fixed in commit ccd453b35f50f7f1b6638389aa0284153b329e02.

The Brokkr team solved the issue by validating all specified addresses with the is_lower_alpha() custom function, which is a kind of own implementation for the to_lower() method.

3.2 (HAL-02) MISSING BOUNDS ON CONFIGURATION VARIABLES - LOW

Description:

The **delta_neutral** contract has missing bounds on the maximum_cdps_per_user_per_asset, maximum_total_investment_in_uusd and maximum_user_investment_in_uusd variables. This can lead to unexpected contract behavior, such as preventing users from investing in a strategy.

The bounds of these parameters should be enforced to avoid potential errors in the current or future uses of these variables.

Code Location:

Listing 3: delta_neutral/src/contract.rs (Lines 53,59,60)

```
51      let config = Config {
52          minimum_investment_in_uusd: msg.minimum_investment_in_uusd
 ↳ ,
53          maximum_cdps_per_user_per_asset: msg.
 ↳ maximum_cdps_per_user_per_asset,
54          owner: deps.api.addr_canonicalize(&msg.owner)?,
55          is_test_contract: msg.is_test_contract,
56          is_paused: msg.is_paused,
57          fee_recipient: deps.api.addr_validate(&msg.fee_recipient)
 ↳ ?,
58          fee_pct: Decimal::from_ratio(msg.fee_pct.numerator, msg.
 ↳ fee_pct.denominator),
59          maximum_total_investment_in_uusd: msg.
 ↳ maximum_total_investment_in_uusd,
60          maximum_user_investment_in_uusd: msg.
 ↳ maximum_user_investment_in_uusd,
61          is_reward_distribution_paused: msg.
 ↳ is_reward_distribution_paused,
62          aust_token_address: deps.api.addr_validate(&msg.
 ↳ aust_token_address)?,
63          mir_token_address: deps.api.addr_validate(&msg.
 ↳ mir_token_address)?,
64          bro_token_address: deps.api.addr_validate(&msg.
↳ bro_token_address)?,
        ts_factory_contract_address: deps.api.addr_validate(&msg.
↳ ts_factory_contract_address)?,
        a_market_contract_address: deps.api.addr_validate(&msg.
↳ a_market_contract_address)?,
        mirr_mint_contract_address: deps.api.addr_validate(&msg.
↳ mirr_mint_contract_address)?,
        mirr_lock_contract_address: deps.api.addr_validate(&msg.
↳ mirr_lock_contract_address)?,
        mirr_staking_contract_address: deps
            .api
            .addr_validate(&msg.mirr_staking_contract_address)?,
        mirr_masset_oracle: deps.api.addr_validate(&msg.
↳ mirr_masset_oracle)?,
        mirr_factory_contract_address: deps
            .api
            .addr_validate(&msg.mirr_factory_contract_address)?,
        astro_router_contract_address: deps
            .api
            .addr_validate(&msg.astro_router_contract_address)?,
    };
```

Moreover, these variables do not implement their bounds also when modifying them with the update_config function.

Listing 4: delta_neutral/src/commands.rs

```
    if let Some(maximum_cdps_per_user_per_asset) =
↳ maximum_cdps_per_user_per_asset {
        attributes.push(Attribute::new(
            "maximum_cdps_per_user_per_asset_changed",
            maximum_cdps_per_user_per_asset.to_string(),
        ));

        config.maximum_cdps_per_user_per_asset =
↳ maximum_cdps_per_user_per_asset;
    }

    if let Some(maximum_total_investment_in_uusd) =
↳ maximum_total_investment_in_uusd {
        attributes.push(Attribute::new(
            "maximum_total_investment_in_uusd_changed",
            maximum_total_investment_in_uusd.to_string(),
        ));
        config.maximum_total_investment_in_uusd = maximum_total_investment_in_uusd;
    }

    if let Some(maximum_user_investment_in_uusd) = maximum_user_investment_in_uusd {
        attributes.push(Attribute::new(
            "maximum_user_investment_in_uusd_changed",
            maximum_user_investment_in_uusd.to_string(),
        ));
        config.maximum_user_investment_in_uusd = maximum_user_investment_in_uusd;
    }
```

Risk Level:

Likelihood - 4
Impact - 1

Recommendation:

Upper and lower bounds for the maximum_cdps_per_user_per_asset, maximum_user_investment_in_uusd and maximum_total_investment_in_uusd variables should be applied when they are updated during the execution of the update_config function and at instantiation.

Remediation plan:

RISK ACCEPTED: The Brokkr team claimed that to optimize the size of the code in the contract, the bounds mentioned above will not be introduced.

3.3 (HAL-03) MISSING DEDICATED ROLES TO MANAGE CONTRACT STATUS - INFORMATIONAL

Description:

The delta_neutral contract supports several types of pauses that may occur in the logic of an investment strategy. Nevertheless, all of them can only be invoked by the contract owner, which therefore becomes single-point-of-failure.

This problem could be solved by creating a special role that has access only to specific contract functionalities, such as changing the value of is_paused or is_reward_distribution_paused.

Code Location:

Listing 5: lockdrop/src/commands.rs (Lines 206,211,236,237,239,280,281,282,283,284,285)

```
202 pub fn update_config(
203     deps: DepsMut,
204     minimum_investment_in_uusd: Option<Uint128>,
205     maximum_cdps_per_user_per_asset: Option<u32>,
206     is_paused: Option<bool>,
207     fee_recipient: Option<String>,
208     fee_pct: Option<DecimalRatio>,
209     maximum_total_investment_in_uusd: Option<Uint128>,
210     maximum_user_investment_in_uusd: Option<Uint128>,
211     is_reward_distribution_paused: Option<bool>,
212     bro_token_address: Option<String>,
213 ) -> Result<Response, ContractError> {
214     let mut config = load_config(deps.storage)?;
215
216     let mut attributes: Vec<Attribute> = vec![Attribute::new("
 ↳ action", "update_config")];
217
218     if let Some(minimum_investment_in_uusd) =
minimum_investment_in_uusd {
        attributes.push(Attribute::new(
            "minimum_investment_in_uusd_changed",
            minimum_investment_in_uusd.to_string(),
        ));

        config.minimum_investment_in_uusd = minimum_investment_in_uusd;
    }

    if let Some(maximum_cdps_per_user_per_asset) = maximum_cdps_per_user_per_asset {
        attributes.push(Attribute::new(
            "maximum_cdps_per_user_per_asset_changed",
            maximum_cdps_per_user_per_asset.to_string(),
        ));

        config.maximum_cdps_per_user_per_asset = maximum_cdps_per_user_per_asset;
    }

    if let Some(is_paused) = is_paused {
        attributes.push(Attribute::new("is_paused_changed", is_paused.to_string()));

        config.is_paused = is_paused;
    }

    if let Some(fee_recipient) = fee_recipient {
        attributes.push(Attribute::new(
            "fee_recipient_changed",
            fee_recipient.to_string(),
        ));

        config.fee_recipient = deps.api.addr_validate(&fee_recipient)?;
    }

    if let Some(fee_pct) = fee_pct {
        if fee_pct.denominator == 0
            || WDecimal::from_ratio(fee_pct.numerator, fee_pct.denominator)
                > WDecimal::from_ratio(100u128, 1u128)
        {
            return Err(ContractError::InvalidFeeError {});
        }

        attributes.push(Attribute::new("fee_pct_changed", fee_pct.to_string()));

        config.fee_pct = WDecimal::from_ratio(fee_pct.numerator, fee_pct.denominator);
    }

    if let Some(maximum_total_investment_in_uusd) = maximum_total_investment_in_uusd {
        attributes.push(Attribute::new(
            "maximum_total_investment_in_uusd_changed",
            maximum_total_investment_in_uusd.to_string(),
        ));
        config.maximum_total_investment_in_uusd = maximum_total_investment_in_uusd;
    }

    if let Some(maximum_user_investment_in_uusd) = maximum_user_investment_in_uusd {
        attributes.push(Attribute::new(
            "maximum_user_investment_in_uusd_changed",
            maximum_user_investment_in_uusd.to_string(),
        ));
        config.maximum_user_investment_in_uusd = maximum_user_investment_in_uusd;
    }

    if let Some(is_reward_distribution_paused) = is_reward_distribution_paused {
        attributes.push(Attribute::new(
            "is_reward_distribution_paused_changed",
            is_reward_distribution_paused.to_string(),
        ));
        config.is_reward_distribution_paused = is_reward_distribution_paused;
    }

    if let Some(bro_token_address) = bro_token_address {
        attributes.push(Attribute::new(
            "bro_token_address_changed",
            bro_token_address.to_string(),
        ));

        config.bro_token_address = deps.api.addr_validate(&bro_token_address)?;
    }

    store_config(deps.storage, &config)?;
    Ok(Response::new().add_attributes(attributes))
}
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

We suggest exploring the possibility of introducing an additional role responsible for contract management and potential pauses in its logic, for example Pauser.

Remediation plan:

ACKNOWLEDGED: The Brokkr team acknowledged this finding.

3.4 (HAL-04) UNUSED CONFIG VARIABLE - INFORMATIONAL

Description:

The instantiate function implements the is_test_contract variable, which is not used anywhere in the contract.

It is a good practice to eliminate the so-called “dead code”, which in no way affects the logic of the program being executed.

Code Location:

Listing 6: lockdrop/src/contract.rs (Line 55)

```
51 let config = Config {
52         minimum_investment_in_uusd: msg.minimum_investment_in_uusd,
53         maximum_cdps_per_user_per_asset: msg.maximum_cdps_per_user_per_asset,
54         owner: deps.api.addr_canonicalize(&msg.owner)?,
55         is_test_contract: msg.is_test_contract,
56         is_paused: msg.is_paused,
57         fee_recipient: deps.api.addr_validate(&msg.fee_recipient)?,
58         fee_pct: Decimal::from_ratio(msg.fee_pct.numerator, msg.fee_pct.denominator),
59         maximum_total_investment_in_uusd: msg.maximum_total_investment_in_uusd,
60         maximum_user_investment_in_uusd: msg.maximum_user_investment_in_uusd,
61         is_reward_distribution_paused: msg.is_reward_distribution_paused,
62         aust_token_address: deps.api.addr_validate(&msg.aust_token_address)?,
63         mir_token_address: deps.api.addr_validate(&msg.mir_token_address)?,
64         bro_token_address: deps.api.addr_validate(&msg.bro_token_address)?,
65         ts_factory_contract_address: deps.api.addr_validate(&msg.
ts_factory_contract_address)?,
        a_market_contract_address: deps.api.addr_validate(&msg.a_market_contract_address)?,
        mirr_mint_contract_address: deps.api.addr_validate(&msg.mirr_mint_contract_address)?,
        mirr_lock_contract_address: deps.api.addr_validate(&msg.mirr_lock_contract_address)?,
        mirr_staking_contract_address: deps
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Unused is_test_contract variable should be removed.

Remediation plan:

ACKNOWLEDGED: The Brokkr team acknowledged this finding.

3.5 (HAL-05) NO CONVENTION IN VARIABLE TYPES - INFORMATIONAL

Description:

During the analysis of the contract code, it was noticed that in many places the variables storing addresses are assigned the CanonicalAddr type; however, there are still single cases of using the standard Addr type.

Considering the fact that the contract is largely adapted to support canonical addresses, it is a good practice to keep the convention, which will increase the readability of the code and make the impact of potential future changes to a smaller number of types.

Code Location:

Listing 7: Samples usage of standard Addr types:

```
packages/services/src/delta_neutral.rs: #261, 262, 299, 302
contracts/delta_neutral/src/state.rs: #254, 262, 286, 294
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

To keep the convention, we suggest that you consider adapting the contract to handle one type of variable holding addresses.

Remediation plan:

ACKNOWLEDGED: The Brokkr team acknowledged this finding.

3.6 (HAL-06) UNMANTAINED DEPENDENCY - INFORMATIONAL

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was cargo audit, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in https://crates.io are stored in a repository named The RustSec Advisory Database. cargo audit is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. To better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

ID	package	Short Description
RUSTSEC-2020-0025	bigint	biginit is unmaintained, use uint instead

Code Location:

Listing 8: Dependency tree

```
bigint 4.4.3
 cosmwasm-bignumber 2.2.0
     moneymarket 0.3.0
         brotocol-delta-neutral 1.0.0
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Beware of using dependencies and packages that are no longer supported by the developers or have publicly known security flaws, even when not exploitable at the moment.

Remediation plan:

ACKNOWLEDGED: The Brokkr team acknowledged this finding.

THANK YOU FOR CHOOSING

HALBORN