// HALBORN

DRAFT

Stader Labs LunaX Contracts

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 17th, 2022 - February 7th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	01/17/2022	Jose C. Ramirez
0.2	Document Updates	02/04/2022	Jose C. Ramirez
0.3	Document Updates	02/07/2022	Connor Taylor
0.4	Draft Version	02/07/2022	Jose C. Ramirez
0.5	Draft Review	02/07/2022	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
Jose C. Ramirez	Halborn	jose.ramirez@halborn.com
Connor Taylor	Halborn	connor.taylor@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Stader Labs engaged Halborn to conduct a security audit on their smart contracts beginning on January 17th, 2022 and ending on February 7th, 2022. The security assessment was scoped to the LunaX related smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided three weeks for the engagement and assigned two 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 security risks, many of these risks were addressed by the Stader Labs team during the testing window, a single outstanding medium risk vulnerability related to the lack of functionality to transfer the manager privileged address in case of need.

In addition, some improvements were identified to reduce the likelihood and impact of risks, which should be addressed by Stader Labs team. The main ones are the following:

- Add a ťmanagerť transfer functionality to allow changes on the privileged account of the contracts.
- Check the “Protocol Inactive” state on all the publicly accessible functionalities related to fund updates, stake tracking or swap contract interaction.

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:

- Research into architecture, purpose, and use of the platform.
- Manual code read and walkthrough.
- Manual assessment of use and safety for the critical Rust variables and functions in scope to identify any contracts logic related vulnerability.
- Fuzz testing (Halborn custom fuzzing tool)
- Checking the test coverage (cargo tarpaulin)
- Scanning of Rust files for vulnerabilities (cargo audit)

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.

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

1.4 SCOPE

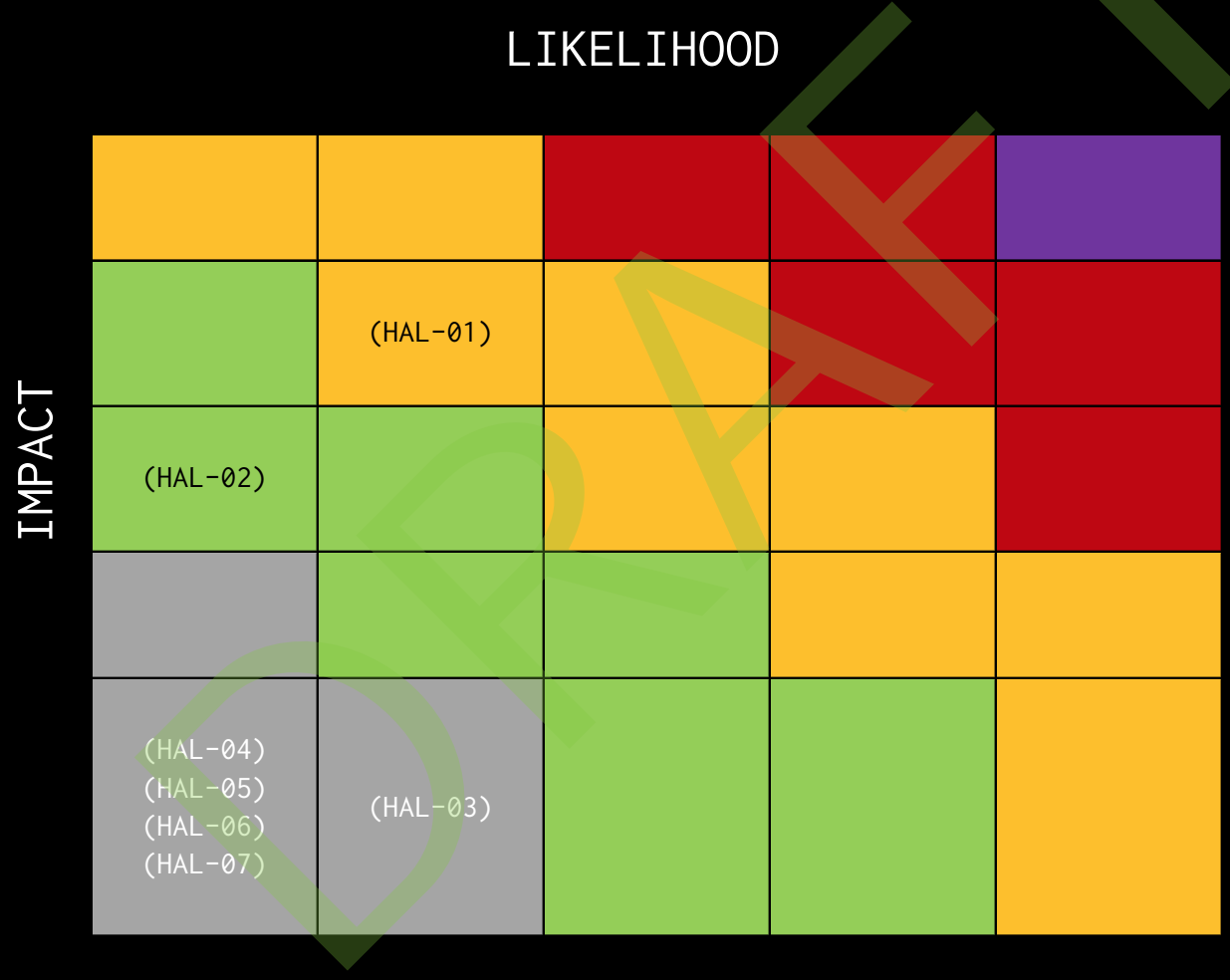
Code repository: https://github.com/stader-labs/stader-liquid-token

1. CosmWasm Smart Contracts - LunaX
    (a) Commit ID: e2561633859cc846bea024854b61248dd6f28665
    (b) Contracts in scope:
        i. Airdrop registry
        ii. Reward contract
        iii. Staking contract

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	1	5



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MANAGER ADDRESS CANNOT BE TRANSFERRED	Medium	-
(HAL-02) UNDERUSED PROTOCOL INACTIVE STATE	Low	-
(HAL-03) CONFIGURATION PARAMETER COULD NOT BE UPDATED	Informational	-
(HAL-04) OUTDATED INFORMATION DISPLAYED TO USERS	Informational	-
(HAL-05) MISUSE OF HELPER METHODS	Informational	-
(HAL-06) MULTIPLE INSTANCES OF UNCHECKED MATH	Informational	-
(HAL-07) UNFINISHED DEVELOPMENT COMMENTS	Informational	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) MANAGER ADDRESS CANNOT BE TRANSFERRED - MEDIUM

Description:

The functions to update the configuration in the contracts within scope lacked the option of setting a new `manager` as a privileged address.
If the keys of the `manager` account were suspected to be compromised, or the development team needed to change the address for an operational reason, a sizable portion of the contract's functionality will be rendered unusable.

Code Location:

Listing 1: Affected functions

```
1 contracts/airdrops-registry/src/contract.rs:59:pub fn
      update_airdrop_registry(
2 contracts/reward/src/contract.rs:204:pub fn update_config(
3 contracts/staking/src/contract.rs:143:pub fn update_config(
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendations:

It is recommended to add **manager transfer** capabilities to the contracts, split into two different functions: `set_manager` and `accept_manager`. The latter function allows the transfer to be completed by the recipient, protecting the contract against potential typing errors compared to single step ownership change features.

3.2 (HAL-02) UNDERUSED PROTOCOL INACTIVE STATE - LOW

Description:

The Staking contract implemented a switch to go into a pause-like state called “Protocol Inactive” by using the config.active setting. This check was not implemented consistently across all the functions accessible for unprivileged users, but only on the deposit one.

This state could be desirable for a number of reasons, where external users’ interference would like to be kept to the minimum among all features of the contract.

Code Location:

Listing 2: Affected assets

```
contracts/staking/src/contract.rs:486 redeem_rewards function
contracts/staking/src/contract.rs:523 swap_rewards function
contracts/staking/src/contract.rs:486 reinvest function
contracts/staking/src/contract.rs:615 reimburse_slashing function
contracts/staking/src/contract.rs:715 undelegate_stake function
contracts/staking/src/contract.rs:810 reconcile_funds function
contracts/staking/src/contract.rs:888 withdraw_funds_to_wallet
    function
contracts/staking/src/contract.rs:976 claim_airdrops function
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to extend the usage of the “Protocol Inactive” check to the rest of publicly accessible functionalities related to fund updates, stake tracking or swap contract interaction.

3.3 (HAL-03) OUTDATED INFORMATION DISPLAYED TO USERS - INFORMATIONAL

Description:

The reimburse_slashing function of the Staking contract did not update the total_staked and exchange_rate state variables. As detailed in the comment of line 635, most functionalities perform a check_slashing at the beginning or update these values. However, this is not the case of query_user_info and query_compute_deposit_breakdown.

Although not a security issue itself, if a user queries those functions after a reimburse_slashing has been performed, they would receive erroneous information due to the lack of update.

Code Location:

Listing 3: Affected assets

```
1 contracts/staking/src/contract.rs:615 reimburse_slashing function
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Update the total_staked and exchange_rate state variables consistently so every function returns or work with up-to-date information.

3.4 (HAL-04) CONFIGURATION PARAMETER COULD NOT BE UPDATED - INFORMATIONAL

Description:

The instantiate function did not set the cw20_token_contract address, as done with other contract addresses required in the configuration. Instead, it relied on update_config being called post initialization, which could cause undesirable situations if this address is not set right after deployment.

The update_config function only allowed to set the CW20 address if it contained the initial value Addr::unchecked("0"). This effectively forbade any future change after the first update. Although desired to lock the token used on the Staking contract, it could potentially render the protocol unusable if any kind of error were made on the first update or if the address were required to change.

Code Location:

Listing 4: contracts/staking/src/contract.rs (Lines 62)

```
55 airdrop_registry_contract: deps
56     .api
57     .addr_validate(msg.airdrops_registry_contract.as_str())?,
58 airdrop_withdrawal_contract: deps
59     .api
60     .addr_validate(msg.airdrop_withdrawal_contract.as_str())?,
61 reward_contract: deps.api.addr_validate(msg.reward_contract.as_str
       ())?,
62 cw20_token_contract: Addr::unchecked("0"),
63
64 protocol_fee_contract: deps.api.addr_validate(msg.
       protocol_fee_contract.as_str())?,
```

Listing 5: contracts/staking/src/contract.rs (Lines 153)

```
143 pub fn update_config(
144     deps: DepsMut,
145     info: MessageInfo,
146     env: Env,
147     update_config: ConfigUpdateRequest,
148 ) -> Result<Response, ContractError> {
149     let mut config = CONFIG.load(deps.storage)?;
150     validate(&config, &info, &env, vec![Verify::SenderManager])?;
151
152     if let Some(cw20_contract) = update_config.cw20_token_contract
            {
153         if config.cw20_token_contract == Addr::unchecked("0") {
154             config.cw20_token_contract = deps.api.addr_validate(
                    cw20_contract.as_str())?;
155         }
156     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

The cw20_token_contract variable should be set upon instantiate, as with the other contract addresses. In addition, a functional privileged method for updating this address should be implemented.

3.5 (HAL-05) MISUSE OF HELPER METHODS - INFORMATIONAL

Description:

The use of the unwrap and expect function is very useful for testing environments because a value is forcibly demanded to get an error (aka panic!) if the “Option” does not have “Some” value or “Result”. Nevertheless, leaving unwrap or expect functions in production environments is a bad practice because not only will this cause the program to crash out, or panic!, but also (in case of unwrap) no helpful messages are shown to help the user solve, or understand the reason of the error.

Code Location:

Listing 6: Affected assets

```
contracts/reward/src/contract.rs: #L93, 168, 171, 217
contracts/staking/src/contract.rs: #L266, 361, 371, 400, 433, 579,
    632, 696, 701, 743, 773, 785, 830, 836, 867, 880, 946, 956,
   1020, 1108, 1121, 1137
contracts/staking/src/helpers.rs: #L90, 98, 122, 163
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to not use the unwrap or expect functions in a production environment because this use provokes panic! and may crash the Spectrum contracts without error messages. Some alternatives are possible, such as propagating the error by putting a "?", using unwrap_or / unwrap_or_else / unwrap_or_default functions, or using error-chain crate for errors.

Reference: https://crates.io/crates/error-chain

3.6 (HAL-06) MULTIPLE INSTANCES OF UNCHECKED MATH - INFORMATIONAL

Description:

Some mathematical operations that could cause unexpected behavior under specific circumstances were found on the codebase. Although no effective arithmetic over/underflow were found and the `overflow-checks = true` flag was set on `Cargo.toml`, it is still recommended to avoid unchecked math as much as possible to follow best-practices and limit the risk of future updates introducing an actual vulnerability.

Code Location:

Listing 7: Affected assets

```
packages/stader-utils/src/coin_utils.rs:173:    let c_u256: Decimal = (b_u256 * a_u256).into();
packages/stader-utils/src/coin_utils.rs:181:    let c_u256: Decimal = (b_u256 + a_u256).into();
packages/stader-utils/src/coin_utils.rs:189:    let c_u256: Decimal = (a_u256 - b_u256).into();
packages/stader-utils/src/coin_utils.rs:243:    Uint128::new(coin.amount.u128() + existing_coin.u128()),
packages/stader-utils/src/coin_utils.rs:278:    Uint128::new(existing_coin.u128() - coin.amount.u128()),
packages/stader-utils/src/coin_utils.rs:432:        coin.amount.u128() * ratio.numerator() / ratio.denominator(),
packages/stader-utils/src/coin_utils.rs:437:    (num * dec.numerator() / dec.denominator()) as u128
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

In the “release” mode, Rust does not panic on overflows and overflown values just “wrap” without any explicit feedback to the user. It is recommended then to use vetted safe math libraries for arithmetic operations consistently throughout the smart contract system. Consider replacing the addition operator with Rust’s `checked_add` method, the multiplication with `checked_mul` and so on.

3.7 (HAL-07) UNFINISHED DEVELOPMENT COMMENTS - INFORMATIONAL

Description:

Multiple “ToDo” comments and commented code instances were found on the codebase. Although incomplete code does not directly cause a security vulnerability or affect the audit’s outcome as far as it is functional, having development comments could simplify the process for an attacker to find a valid attack surface within the contract.

In addition, it shows that the audited code will be different from the released one, which could cause that new vulnerabilities were to be introduced in the code after the audit.

Code Location:

Listing 8: Affected assets

```
staking/src/helpers.rs:29:// TODO: bchain99 - write unit-tests to validate.
staking/src/contract.rs:1092:// TODO - GM. Test this
staking/src/contract.rs:1101:    // TODO - GM. Will converting u64 to string for batch id start work?
staking/src/contract.rs:521:// TODO - GM. Does swap have a fixed cost or a linear cost?
reward/src/contract.rs:77:// TODO - GM. Does swap have a fixed cost or a linear cost? Useful to make this permissionless.
reward/src/contract.rs:94:    // let denoms: Vec<String> = total_rewards
reward/src/contract.rs:95:    //     .iter()
reward/src/contract.rs:96:    //     .map(|item| item.denom.clone())
reward/src/contract.rs:97:    //     .collect();

```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Remove all the instances development related comments, reviewing if some modifications are still pending to be made.

AUTOMATED TESTING

4.1 AUTOMATED ANALYSIS

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

THANK YOU FOR CHOOSING

// HALBORN