Stader Labs SD Token Staking

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 9th, 2022 - March 15th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 3
CONTACTS 4
1 EXECUTIVE OVERVIEW 5
1.1 INTRODUCTION 6
1.2 AUDIT SUMMARY 6
1.3 TEST APPROACH & METHODOLOGY 7
RISK METHODOLOGY 7
1.4 SCOPE 9
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 10
3 FINDINGS & TECH DETAILS 11
3.1 (HAL-01) LACK OF MANAGER TRANSFER FUNCTIONALITY - LOW 13
Description 13
Code Location 13
Risk Level 14
Recommendation 15
Remediation plan 15
3.2 (HAL-02) MISSING BOUNDS ON CONFIGURATION VARIABLES - LOW 16
Description 16
Code Location 16
Risk Level 18
Recommendation 18
Remediation plan 19
3.3 (HAL-03) LACK OF ADDRESS NORMALIZATION - INFORMATIONAL 20
Description 20

Code Location 20
Risk Level 21
Recommendation 21
Remediation plan 21
3.4 (HAL-04) CONFIGURATION PARAMETER NOT SET UPON INSTANTIATION - INFORMATIONAL 22
Description 22
Code Location 22
Risk Level 23
Recommendation 23
Remediation plan 23
4 AUTOMATED TESTING 24
4.1 AUTOMATED ANALYSIS 25
Description 25

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/09/2022	Jose C. Ramirez
0.2	Document Update	03/15/2022	Jakub Heba
0.3	Document Update	03/18/2022	Jose C. Ramirez
0.4	Draft Version	03/18/2022	Jakub Heba
0.5	Draft Review	03/18/2022	Gabi Urrutia
1.0	Remediation Plan	03/25/2022	Jakub Heba
1.1	Remediation Plan Review	03/27/2022	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
Jakub Heba	Halborn	jakub.heba@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Stader Labs engaged Halborn to conduct a security audit on their smart contracts beginning on March 9th, 2022 and ending on March 15th, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week 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 were partially addressed by the Stader Labs team. The main ones are the following:

- Implement manager transfer functionalities for lock and rewards-pool contracts.
- 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:

- 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.

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

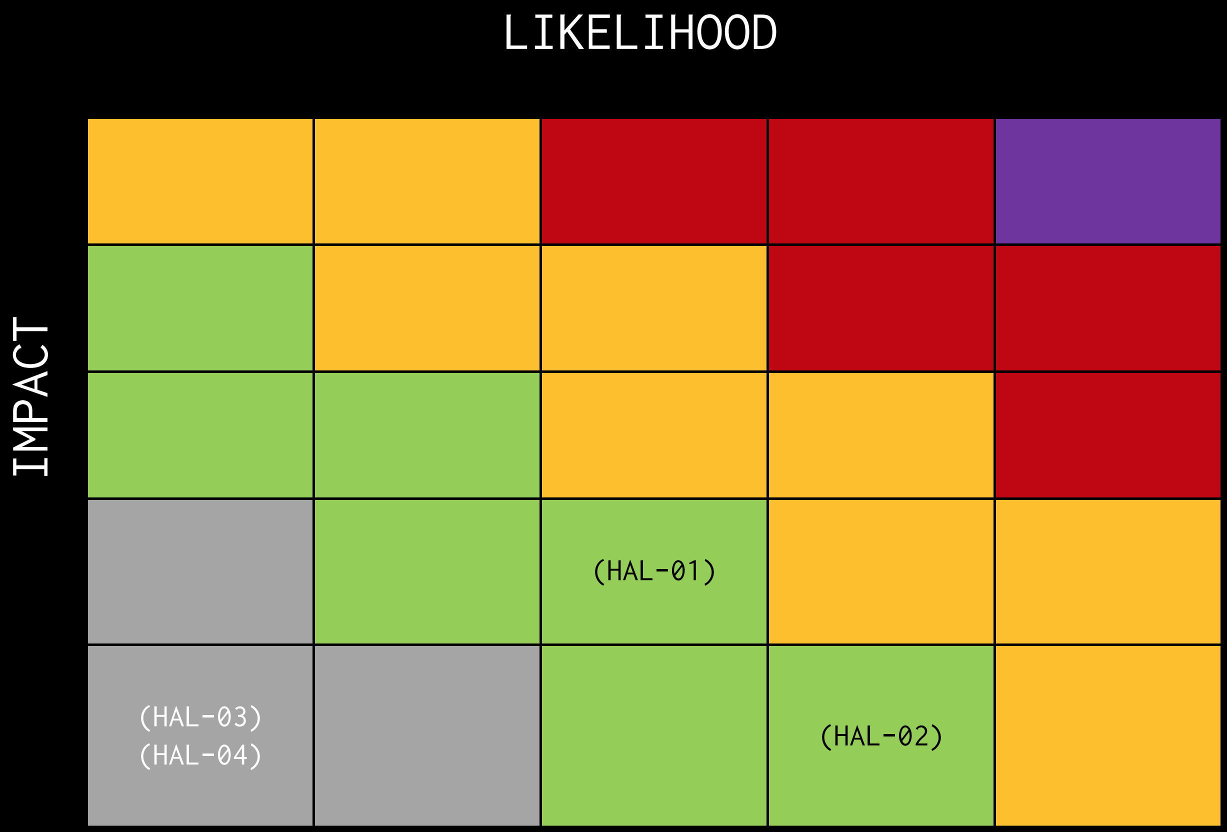
Code repository: terra-tokenomics

1. CosmWasm Smart Contracts
   (a) Commit ID: e5fb44cecaedad4ad78bed93351ac697e94adc7c
   (b) Contracts in scope:
      i. lock
      ii. rewards-pool
      iii. staking

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	2



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) LACK OF MANAGER TRANSFER FUNCTIONALITY	Low	RISK ACCEPTED
(HAL-02) MISSING BOUNDS ON CONFIGURATION VARIABLES	Low	SOLVED - 03/21/2022
(HAL-03) LACK OF ADDRESS NORMALIZATION	Informational	SOLVED - 03/21/2022
(HAL-04) CONFIGURATION PARAMETER NOT SET UPON INSTANTIATION	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) LACK OF MANAGER TRANSFER FUNCTIONALITY - LOW

Description:

The **rewards-pool** and **lock** contracts do not implement any functionality to change the manager address, which is set at contract instantiation.

If the manager account keys are suspected to be compromised or if the development team needs to change the address for an operational reason, some functionality of the contract could be rendered unusable.

It is worth mentioning that this issue has a very limited impact, as there is only one manager-only functionality that can be used once.

Code Location:

Listing 1: contracts/rewards-pool/src/contract.rs (Line 63)

```
54 pub fn update_configs(
55     deps: DepsMut,
56     info: MessageInfo,
57     _env: Env,
58     staking_contract: String,
59 ) -> Result<Response, ContractError> {
60     let mut config = CONFIG.load(deps.storage)?;
61
62     let sending_manager = info.sender.to_string();
63     if sending_manager.ne(&config.manager) {
64         return Err(ContractError::Unauthorized {});
65     }
66
67     if config.staking_contract.eq(&Addr::unchecked("0")) {
68         config.staking_contract = deps
69             .api
70             .addr_validate(staking_contract.to_lowercase().as_str
 ↳ ())?;
71     }
72
73     CONFIG.save(deps.storage, &config)?;
74
75     Ok(Response::new()
76         .add_attribute("method", "update_configs")
77         .add_attribute("staking_contract", staking_contract))
78 }
```

Listing 2: contracts/lock/src/contract.rs (Line 65)

```
57 pub fn update_configs(
58     deps: DepsMut,
59     info: MessageInfo,
60     _env: Env,
61     staking_contract: String,
62 ) -> Result<Response, ContractError> {
63     let mut config = CONFIG.load(deps.storage)?;
64     let sending_manager = info.sender;
65     if sending_manager.ne(&config.manager) {
66         return Err(ContractError::Unauthorized {});
67     }
68
69     if config.staking_contract.eq(&Addr::unchecked("0")) {
70         config.staking_contract = deps
71             .api
72             .addr_validate(staking_contract.to_lowercase().as_str
 ↳ ())?;
73     }
74
75     CONFIG.save(deps.storage, &config)?;
76
77     Ok(Response::new().add_attribute("method", "update_configs"))
78 }
```

Risk Level:

Likelihood - 3

Impact - 2

Recommendation:

Manager transfer capabilities should be added to contracts as a two-step process split into a set_manager and a accept_manager functions. The latter requires the recipient to complete the transfer, effectively protecting the contract against potential typographical errors, compared to one-step manager transfer mechanisms.

Remediation plan:

RISK ACCEPTED: The risk for this issue was accepted by the Stader Labs team. They also mentioned that there is no use case for changing the manager in the future.

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

Description:

The **staking** contract has missing bounds on the emission_rate and unbonding_period variables. This can lead to unexpected contract behavior.

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: contracts/staking/src/contract.rs (Lines 44,48,64,66)

```
35 pub fn instantiate(
36     deps: DepsMut,
37     env: Env,
38     _info: MessageInfo,
39     msg: InstantiateMsg,
40 ) -> Result<Response, ContractError> {
41     set_contract_version(deps.storage, CONTRACT_NAME,
 ↳ CONTRACT_VERSION)?;
42
43     let state = State {
44         emission_rate: msg.emission_rate,
45         last_exchange_rate_updated_block: env.block.height,
46         next_undelegate_id: 0,
47     };
48     STATE.save(deps.storage, &state)?;
49
50     let config = Config {
51         manager: deps
52             .api
53             .addr_validate(msg.manager.to_lowercase().as_str())?,
54         xsd_cw20_contract: Addr::unchecked("0"),
55         sd_cw20_contract: deps
56             .api
57             .addr_validate(msg.sd_cw20_contract.to_lowercase().
↳ as_str())?,
        rewards_pool_contract: deps
            .api
            .addr_validate(msg.rewards_pool_contract.to_lowercase
↳ ().as_str())?,
        lock_contract: deps
            .api
            .addr_validate(msg.lock_contract.to_lowercase().as_str
↳ ())?,
        unbonding_period: msg.unbonding_period,
    };
    CONFIG.save(deps.storage, &config)?;

    Ok(Response::new().add_attribute("method", "instantiate"))
}
```

Listing 4: contracts/staking/src/contract.rs (Lines 369,370,373,374)

```
pub fn update_config(
    deps: DepsMut,
    info: MessageInfo,
    _env: Env,
    update_config_request: UpdateConfigRequest,
) -> Result<Response, ContractError> {
    let mut config = CONFIG.load(deps.storage)?;
    let mut state = STATE.load(deps.storage)?;
    let sending_manager = info.sender.to_string().to_lowercase();

    if sending_manager.ne(&config.manager) {
        return Err(ContractError::Unauthorized {});
    }

    if let Some(sd_cw20) = update_config_request.sd_cw20_contract
↳ {
        config.sd_cw20_contract = deps.api.addr_validate(sd_cw20.
↳ to_lowercase().as_str())?;
    }

    if let Some(xsd_cw20) = update_config_request.
↳ xsd_cw20_contract {
        if config.xsd_cw20_contract.eq(&Addr::unchecked("0")) {
            config.xsd_cw20_contract = deps.api.addr_validate(
↳ xsd_cw20.to_lowercase().as_str())?;
        }
    }

    if let Some(lock) = update_config_request.lock_contract {
        config.lock_contract = deps.api.addr_validate(lock.to_lowercase().as_str())?;
    }

    if let Some(rpc) = update_config_request.rewards_pool_contract {
        config.rewards_pool_contract = deps.api.addr_validate(rpc.to_lowercase().as_str())?;
    }

    if let Some(er) = update_config_request.emission_rate {
        state.emission_rate = er;
    }

    if let Some(up) = update_config_request.unbonding_period {
        config.unbonding_period = up;
    }

    CONFIG.save(deps.storage, &config)?;
    STATE.save(deps.storage, &state)?;

    Ok(Response::new().add_attribute("method", "update_config"))
}
```

Risk Level:

Likelihood - 4
Impact - 1

Recommendation:

Upper and lower bounds for the emission_rate and unbonding_period variables should be applied when they are updated during the execution of the update_config function and at instantiation.

Remediation plan:

SOLVED: The issue was fixed with the above recommendation in commits:

- 2f1aa9ae1db519d452fa3456fa912a562041b666
- ea46b4313fc5ce14d9b8adedb290a1a4b4b28829

3.3 (HAL-03) LACK OF ADDRESS NORMALIZATION - INFORMATIONAL

Description:

The set_manager function of the staking contract takes user input directly as an address without performing any validation. This could lead to failed messages or the storage of incorrect information.

However, the impact in this particular case is very limited since the address submitted in the set_manager function must execute the accept_manager function to finish the process, protecting against potential mistakes such as those mentioned.

Code Location:

Listing 5: contracts/staking/src/contract.rs (Lines 93,103)

```
89  pub fn set_manager(
90      deps: DepsMut,
91      info: MessageInfo,
92      _env: Env,
93      manager: String,
94  ) -> Result<Response, ContractError> {
95      let config = CONFIG.load(deps.storage)?;
96      if info.sender.ne(&config.manager) {
97          return Err(ContractError::Unauthorized {});
98      }
99
100     TMP_MANAGER_STORE.save(
101         deps.storage,
102         &TmpManagerStore {
103             manager: manager.clone(),
104         },
105     )?;
106
107     Ok(Response::new()
108         .add_attribute("method", "set_manager")
109         .add_attribute("new_manager", manager))
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

One of the two approaches detailed below should be used:

- In addition to validation, addresses should be normalized to lowercase before being stored for future use.
- Instead of just validation, addresses should be stored in canonical format 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 also be canonicalized beforehand.
- To send funds to a canonicalized address or include it in a message to a different contract, it 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 with the above recommendation in commit 0f3fbb9237c43f381987e113fa418d4cd1afbb57.

3.4 (HAL-04) CONFIGURATION PARAMETER NOT SET UPON INSTANTIATION - INFORMATIONAL

Description:

The instantiate function did not set the staking_contract address, as it did for other required contract addresses in the configuration. Instead, it relied on update_config being called post initialization. If the stacking address is not set by calling update_config before executing transfer_rewards, an attempt to transfer to address “0” will end the panic! macro and unnecessary consumption of gas fee per transaction.

It is worth noting that the update_config function only allowed setting the staking address if it contained the initial value Addr::unchecked("0"). This effectively forbids any future change after the first update.

Code Location:

Listing 6: contracts/rewards-pool/src/contract.rs (Line 25)

```
18 pub fn instantiate(
19     deps: DepsMut,
20     _env: Env,
21     _info: MessageInfo,
22     msg: InstantiateMsg,
23 ) -> Result<Response, ContractError> {
24     let config = Config {
25         staking_contract: Addr::unchecked("0"),
26         sd_cw20_contract: deps
27             .api
28             .addr_validate(msg.sd_cw20_contract.to_lowercase().
   ↳ as_str())?,
29         manager: deps
30             .api
31             .addr_validate(msg.manager.to_lowercase().as_str())?,
32     };
```

Listing 7: contracts/lock/src/contract.rs (Line 31)

```
18 pub fn instantiate(
19     deps: DepsMut,
20     _env: Env,
21     _info: MessageInfo,
22     msg: InstantiateMsg,
23 ) -> Result<Response, ContractError> {
24     let config = Config {
25         manager: deps
26             .api
27             .addr_validate(msg.manager.to_lowercase().as_str())?,
28         sd_cw20_contract: deps
29             .api
30             .addr_validate(msg.sd_cw20_contract.to_lowercase().as_str())?,
31         staking_contract: Addr::unchecked("0"),
32     };
33     set_contract_version(deps.storage, CONTRACT_NAME, CONTRACT_VERSION)?;
34     CONFIG.save(deps.storage, &config)?;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

The staking_contract variable should be set on instantiate, as with the other contract addresses.

Remediation plan:

ACKNOWLEDGED: The Stader Labs team acknowledged this finding.

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. All vulnerabilities shown here were already disclosed in the above report. However, 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