Stader Labs Stake+ Contracts

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 21st, 2022 - March 4th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/21/2022	Jose C. Ramirez
0.2	Document Update	03/03/2022	Jose C. Ramirez
0.3	Document Update	03/04/2022	Jakub Heba
0.4	Draft Version	03/04/2022	Jose C. Ramirez
0.5	Draft Review	03/04/2022	Gabi Urrutia
1.0	Remediation Plan	03/07/2022	Jakub Heba
1.1	Remediation Plan Review	03/07/2022	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
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 February 21st, 2022 and ending on March 4th, 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 two weeks for the engagement and assigned two full-time security engineers to audit the security of the smart contract. The security engineers are a 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 mostly addressed by Stader Labs team. The main ones are the following:

- Capitalization normalization of user and manager addresses being used throughout the contracts.
- Consistent application of fee upper limits on the Stake contract.

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.

EXECUTIVE OVERVIEW

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: https://github.com/stader-labs/staking-plus

1. CosmWasm Stake+ Smart Contracts
   (a) Commit ID: c7bcadf8e6ac4bbe157c237bdbb80fbf2f6e3c34
   (b) Contracts in scope:
      i. airdrop-sink
      ii. reward
      iii. staking

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	1	5

LIKELIHOOD

IMPACT

🟨	🟨	🟥	🟥	🟪
🟩	🟨	🟨	🟥 (HAL-01)	🟥
🟩	🟩 (HAL-02)	🟨	🟨	🟥
⬜	🟩	🟩	🟨	🟨
⬜ (HAL-03) (HAL-04) (HAL-05) (HAL-06) (HAL-07)	⬜	🟩	🟩	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) LACK OF ADDRESS LOWERCASE NORMALIZATION	High	SOLVED - 03/07/2022
(HAL-02) FEE CAP NOT ENFORCED UPON INSTANTIATION	Low	SOLVED - 03/07/2022
(HAL-03) LACK OF DENOM LOWERCASE NORMALIZATION	Informational	SOLVED - 03/07/2022
(HAL-04) CONFIGURATION PARAMETER NOT SET UPON INSTANTIATION	Informational	ACKNOWLEDGED
(HAL-05) MULTIPLE INSTANCES OF UNCHECKED MATH	Informational	SOLVED - 03/08/2022
(HAL-06) MISUSE OF HELPER METHODS	Informational	SOLVED - 03/08/2022
(HAL-07) UNUSED CONFIGURATION STATE VARIABLE	Informational	SOLVED - 03/07/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) LACK OF ADDRESS LOWERCASE NORMALIZATION - HIGH

Description:

The multiple functionalities of the in-scope contracts do not consider Terra addresses to be valid in both upper and all lower case. While a 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**) will fail. A clear example of when this can create a security issue is when info.sender is compared to a previously stored address to enforce access controls.

The risk rating for this issue has been raised to high as one of the instances could cause a direct loss of funds to contract users. The stake contract included a deposit_on_behalf_of_user function that takes the address of the user from an external input without lowercase normalization. When a valid uppercase address is used for the deposit of funds through this function, the funds will be locked forever, as the withdrawal process will fail to compare the stored uppercase address with the user's lowercase address.

Code Location:

Listing 1: contracts/staking/src/user.rs (Lines 33,40,41,42)

```rust
33     let user = deps.api.addr_validate(user.as_str())?;
34     let mut state = STATE.load(deps.storage)?;
35 
36     let mut msgs = vec![];
37     let current_er = calculate_exchange_rate(deps.storage.deref(),
           &deps.querier, &env)?;
38     let deposit_breakdown = compute_deposit_breakdown(&config,
         current_er, amount)?;
39 
40     let mut user_info = USER_INFO
41         .may_load(deps.storage, &user)?
42         .unwrap_or_else(|| UserInfo::new());
```

```

    update_user_withdrawable_airdrops(&state, &mut user_info)?;
```

Listing 2: Affected resources

```
contracts/airdrops-sink/src/contracts.rs #27, 73
contracts/rewards/src/contracts.rs #28, 210
contracts/staking/src/contracts.rs #57, 58, 200, 246, 422, 441,
    495, 1206, 1244, 1335
contracts/staking/src/user.rs #33
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

In addition to validation, addresses should be normalized to lowercase before being stored for future usage.

Remediation plan:

SOLVED: The issue was fixed in the following commits:

- 0500fd2b2b51c2dc002f7c712754b190336d750e
- 3ac72933d4524952308b26cead1fb5be0a173d19
- b94db7e106d3db937a3fed68583309fa5f125b88
- 0ce2d7711a5a6c0e09e31e208abeab2ef5553d85

3.2 (HAL-02) FEE CAP NOT ENFORCED UPON INSTANTIATION - LOW

Description:

The instatiate function from the staking contract limit the contracts fees (protocol_deposit_fee, protocol_withdraw_fee, protocol_reward_fee) to values lesser than 1 instead of the maximum expected values as the update_starder_configs function does (0.05, 0.05, 0.1).

This could result in undesired distributions if the instantiate values are incorrectly set.

Code Location:

Listing 3: contracts/staking/src/contracts.rs (Lines 49,50,51)

```
49 if msg.protocol_reward_fee.gt(&Decimal::one())
50     || msg.protocol_deposit_fee.gt(&Decimal::one())
51     || msg.protocol_withdraw_fee.gt(&Decimal::one())
52 {
53     return Err(ContractError::ProtocolFeeAboveLimit {});
54 }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Upper limits for protocol fees should be consistently enforced through all the functions that set those values.

Remediation plan:

SOLVED: The issue was fixed in commit 3ac72933d4524952308b26cead1fb5be0a173d19.

3.3 (HAL-03) LACK OF DENOM LOWERCASE NORMALIZATION - INFORMATIONAL

Description:

The staking contract does not normalize denoms to lowercase. Although not a security issue at the time of the audit, it caused inconvenience for users attempting to use the update_airdrop_pointers and withdraw_airdrops functions that will receive non-descriptive errors if denoms are submitted in uppercase.

Code Location:

Listing 4: contracts/staking/src/contracts.rs (Line 1108)

```
1104 let denoms_to_withdraw = denoms_to_withdraw.unwrap_or_else(|| {
1105     state
1106         .global_airdrop_pointer
1107         .iter()
1108         .map(|x| x.denom.clone())
1109         .collect::<Vec<String>>()
1110 });
```

Listing 5: contracts/staking/src/contracts.rs (Line 1045)

```
1041 for token in tokens {
1042     let contract_response = get_airdrop_contracts(
1043         deps.querier,
1044         airdrop_registry_contract.clone(),
1045         token.clone(),
1046     )?;
1047
1048     ...snip...
```

Listing 6: contracts/staking/src/contracts.rs (Line 969)

```
961 for rate in airdrop_rates {
962     if rate.amount.is_zero() {
```

```
            continue;
        }

        let contract_response: GetAirdropContractsResponse =
            get_airdrop_contracts(
            deps.querier,
            airdrops_registry_contract.clone(),
            rate.denom.clone(),
        )?;

        ...snip...
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Normalize denoms to lowercase when taken from user input before using them.

Remediation plan:

SOLVED: The issue was fixed in commit 07fa22adfb207297ba123a9b93a6818f017a9066.

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

Description:

The instantiate function did not set the staking_contract address, as done with other contract addresses required in the configuration.

Instead, it relied on update_config being called post initialization. This could cause undesirable situations if this address is not set right after deployment, as users will not be able to use the system.

Code Location:

Listing 7: contracts/airdrops-sink/src/contracts.rs (Line 28)

```
26 let config = Config {
27     stader_manager: deps.api.addr_validate(msg.stader_manager.
           as_str())?,
28     staking_contract: Addr::unchecked("0"),
29     airdrop_registry_contract: deps
30         .api
31         .addr_validate(msg.airdrop_registry_contract.as_str())?,
32 };
```

Listing 8: contracts/reward/src/contracts.rs (Line 30)

```
27 let config = Config {
28     stader_manager: deps.api.addr_validate(msg.stader_manager.
           as_str())?,
29     reward_denom: "uluna".to_string(),
30     staking_contract: Addr::unchecked("0"),
31 };
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

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

Remediation plan:

ACKNOWLEDGED: The Stader Labs team acknowledged this finding.

3.5 (HAL-05) 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 9: Affected assets

```
packages/stader-utils/src/coin_utils.rs #171, 179, 187, 199, 204,
    253, 288, 441, 446, 458
```

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.

Remediation plan:

SOLVED: Some of the highlighted instances use data types, such as cosmwasm_bignumber::Decimal256, do not implement checked math, but do panic on an overflow when using the add or mul traits. Therefore, they have also been marked as solved. The rest of the instances were fixed in the following commits:

- d765a43cc445430329d1df4859e675b7b47c7843
- 725b2b8ec248bac407f1020c45385c66fa03f250

3.6 (HAL-06) 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 10: Affected assets

```
contracts/staking/src/contract.rs #1213, 1231, 1257, 1310
```

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

Remediation plan:

SOLVED: Instances related to the result of checked_add operations have not been modified as the risk of overflow in those cases is minimal, since all supplying Terra supply will not cause the value of uint128 overflow. The rest of the instances were fixed in the following commits:

- d14887ea4599c90a23661745d05780637c5bf60e
- 0fc82705038bf64f358c8d516ba4d5bd1c973f82

3.7 (HAL-07) UNUSED CONFIGURATION STATE VARIABLE - INFORMATIONAL

Description:

The stake contract sets the active Boolean as part of its configuration and allowed to update it. However, this variable is not used anywhere in the code.

Code Location:

Listing 11: contracts/staking/src/contracts.rs (Line 62)

```
56 let config = Config {
57     stader_manager: deps.api.addr_validate(msg.stader_manager.
           as_str())?,
58     operating_manager: deps.api.addr_validate(msg.
           operating_manager.as_str())?,
59     vault_denom: "uluna".to_string(),
60     min_deposit: msg.min_deposit,
61     max_deposit: msg.max_deposit,
62     active: true,
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Unused variables should be removed from the codebase.

Remediation plan:

SOLVED: The issue was fixed in commit 3ac72933d4524952308b26cead1fb5be0a173d19.

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