HALBORN

Sienna.Network - Rewards Contract - Bonding Update

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 23rd, 2022 - August 26th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/25/2022	Elena Maranon
0.2	Draft Review	08/26/2022	Gabi Urrutia
1.0	Remediation Plan	08/27/2022	Elena Maranon
1.1	Remediation Plan Review	08/29/2022	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
Elena Maranon	Halborn	Elena.Maranon@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Sienna.Network engaged Halborn to conduct a security audit on their smart contracts beginning on August 23rd, 2022 and ending on August 26th, 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 four days for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert 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 has identified some minor security risks:

- Unchecked math / arithmetic overflow.
- Misleading nomenclature.
- Lack of code reuse.

1.3 TEST APPROACH & METHODOLOGY

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: SiennaNetwork

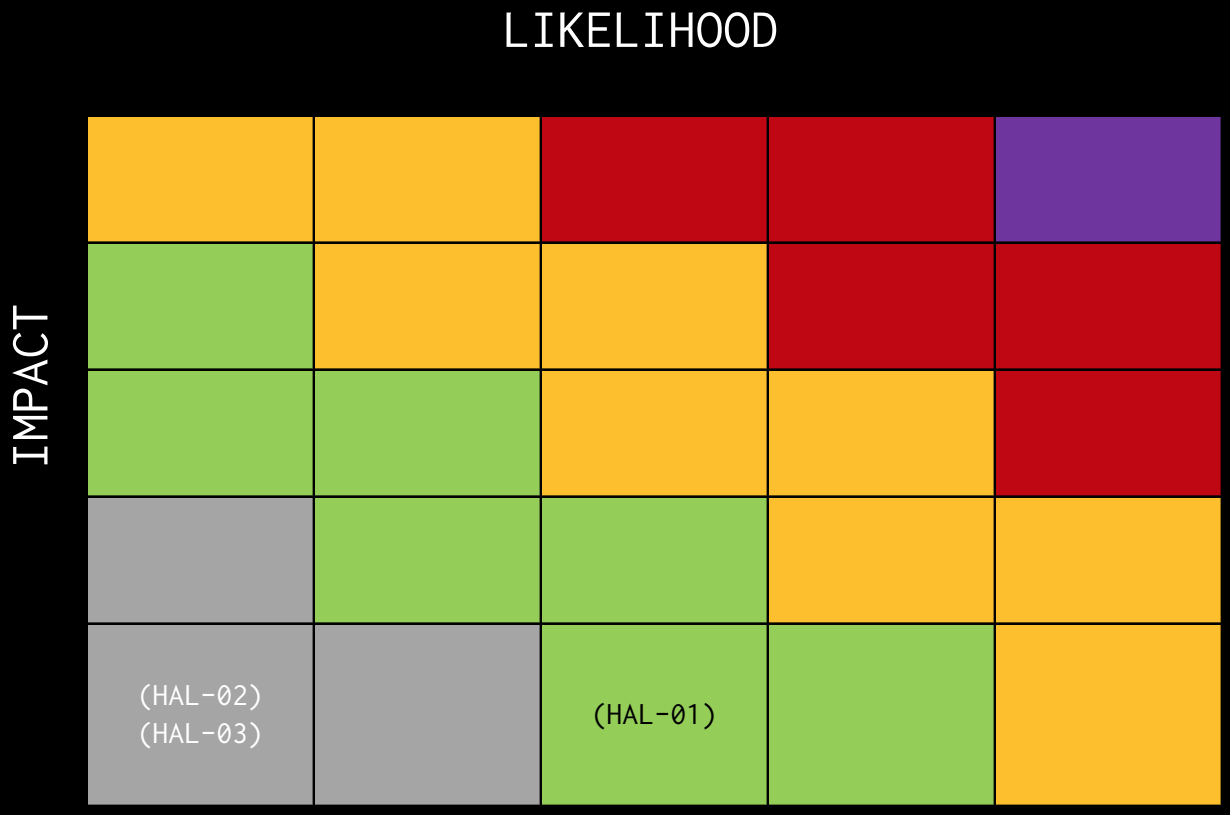
1. Smart Contracts
    (a) Commit ID: d8fc33bc92b6f3252eda5b52c6e4d95380f996cb
    (b) Contracts in scope:
        i. rewards: Bonding related updates

It is worth noting that the results of this audit are a complement to the information provided in previous reports for the security audits performed to the codebase with the following commit IDs:

1. Rewards V3: 20dce5c6a7dfcd983ae2fbc4292b1b58678ae07e
2. Lending Rewards update: b88b365a690a65121eb77523378be70c4ec604f5

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	2



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) CLIENT QUERY COULD CAUSE UNDERFLOW	Low	RISK ACCEPTED
(HAL-02) MISLEADING NOMENCLATURES	Informational	ACKNOWLEDGED
(HAL-03) LACK OF CODE REUSE	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) CLIENT QUERY COULD CAUSE UNDERFLOW - LOW

Description:

The query function user_bonding_balances from file **contracts/amm/rewards/query.rs** allows a user to consult their balances, accepting two input values: at and auth_method.

The at value it is supposed to be a time (seconds since UNIX epoch) but, since this input is not validated, it could be any u64 value. This value will be inherited by BondingHistory.now parameter during BondingHistory::load() function.

The function BondingHistory::bonding from file **contracts/amm/rewards/bonding_history.rs**, which is called by BondingHistory::balances one, uses an unchecked mathematical operation which would produce an underflow if the at value is manipulated to be smaller that bonding_period. The underflow will not panic the execution, but it will lead to a wrong calculation of balances.

It is always recommended to use safe methods on mathematical operations.

Code Location:

bonding function:

Listing 1: contracts/amm/rewards/bonding_history.rs (Line 191)

```
188     fn bonding(&self) -> Uint128 {
189         let is_bonding = |entry: &&HistoryEntry| {
190             entry.bonding_type == BondingType::Bonding
191                 && entry.timestamp > self.now - self.
 ↳ bonding_period
192         };
193         self.history
194             .iter()
195             .filter(is_bonding)
```

```
196             .map(|entry| entry.amount.u128())
197             .sum::<u128>()
198             .into()
199     }
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

In “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/saturating_add methods, the subtraction operator with Rust’s checked_subs/saturating_sub methods and so on.

Remediation Plan:

RISK ACCEPTED: The Sienna.Network team accepted the risk of this finding.

3.2 (HAL-02) MISLEADING NOMENCLATURE - INFORMATIONAL

Description:

The function save_and_expire_bonded, from file **contracts/amm/rewards/bonding_history.rs**, updates the BondingHistory.history vector in order to eliminate the expired bonding entries.

The variable named not_bonding could lead to misunderstood because that closure is designed to search for currently “bonding” entries in order to save them and remove those which expired and get into “bonded” state.

Code Location:

save_and_expire_bonded function:

Listing 2: contracts/amm/rewards/bonding_history.rs (Line 175)

```
171     pub fn save_and_expire_bonded<S: Storage>(
172         &self,
173         core: &mut impl MutableStorageWrapper<S>,
174     ) -> StdResult<()> {
175         let not_bonding = |entry: &&HistoryEntry| match entry.
 ↳ bonding_type {
176             BondingType::Bonding => {
177                 self.bonding_period > 0 && (entry.timestamp > self
 ↳ .now - self.bonding_period)
178             }
179             _ => true,
180         };
181         let filtered = &self.history.iter().filter(not_bonding).
 ↳ collect::<Vec<_>>();
182         ns_save(core.storage_mut(), Self::NS, self.user.as_slice()
 ↳ , filtered)
183     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to rename the closure as `not_bonded` or `is_bonding`.

In addition, as mentioned in previous findings, it is recommended to substitute the unsafe mathematical operations for safe arithmetic methods.

Remediation Plan:

ACKNOWLEDGED: The Sienna.Network team acknowledged this finding.

3.3 (HAL-03) LACK OF CODE REUSE - INFORMATIONAL

Description:

The function transfer_withdraw_and_earned from file **contracts/amm/rewards/accounts.rs** is responsible to generate the **HandleResponse** for the withdrawn amounts. In case the balance of the account is zero and remains some earnings pending to transfer, it also includes them into the transfer response.

This functionality is already implemented on claim function, which also includes additional security checking like if self.total.budget == Amount ::zero(), so it is recommended to reuse the already existing function.

Code Location:

transfer_withdrawn_and_earned function:

Listing 3: contracts/amm/rewards/accounts.rs (Lines 283-289)

```
274     fn transfer_withdrawn_and_earned(
275         &mut self,
276         core: &mut C,
277         withdrawn: Uint128,
278     ) -> StdResult<HandleResponse> {
279         let mut response = HandleResponse::default();
280         // If all tokens were withdrawn
281         if self.balance == Amount::zero() {
282             // And if there is some reward claimable
283             if self.earned > Amount::zero() && self.
 ↳ claim_countdown == 0 {
284                 // Also transfer rewards
285                 self.commit_claim(core)?;
286                 let reward_token = RewardsConfig::reward_token(
 ↳ core)?;
287                 response = response
288                     .msg(reward_token.transfer(&self.address, self
 ↳ .earned)?)?
```

```
289                     .log("reward", &self.earned.to_string())?;
290             } else {
291                 // If bonding is not over yet just reset even if
 ↳ some rewards were earned
292                 self.reset(core)?;
293             }
294         }
295         let msg = RewardsConfig::lp_token(core)?.transfer(&self.
 ↳ address, withdrawn)?;
296         response.msg(msg)
297     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to reuse the already existing function claim() instead of repeat the same functionality inside other function that uses less security conditions.

Remediation Plan:

ACKNOWLEDGED: The Sienna.Network 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-2020-0071	time	Potential segfault in the time crate
RUSTSEC-2018-0015	term	term is looking for a new maintainer
RUSTSEC-2020-0077	memmap	memmap is unmaintained
RUSTSEC-2021-0139	ansi-term	ansi-term is unmaintained

THANK YOU FOR CHOOSING

// HALBORN