// HALBORN

Sienna.Network Lending Rewards & Launchpad (Updated code)

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **July 26th, 2022 - August 4th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/03/2022	Alexis Fabre
0.2	Document Updates	08/04/2022	Alexis Fabre
0.3	Draft version	08/04/2022	Alexis Fabre
0.4	Draft Review	08/04/2022	Gabi Urrutia
1.0	Remediation Plan	08/11/2022	Alexis Fabre
1.1	Remediation Plan Review	08/11/2022	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
Alexis Fabre	Halborn	Alexis.Fabre@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Sienna.Network engaged Halborn to conduct a security audit on their smart contracts beginning on July 26th, 2022 and ending on August 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 a full-time security engineer to audit the security of the smart contracts. 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
- Review codebase changes since last audit

In summary, Halborn identified only minor improvements to reduce the likelihood and impact of the risks, which were acknowledged by Sienna team:

- Ensure the total speed distribution speed is validated.
- Ensure that function's behavior are matching their name.

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

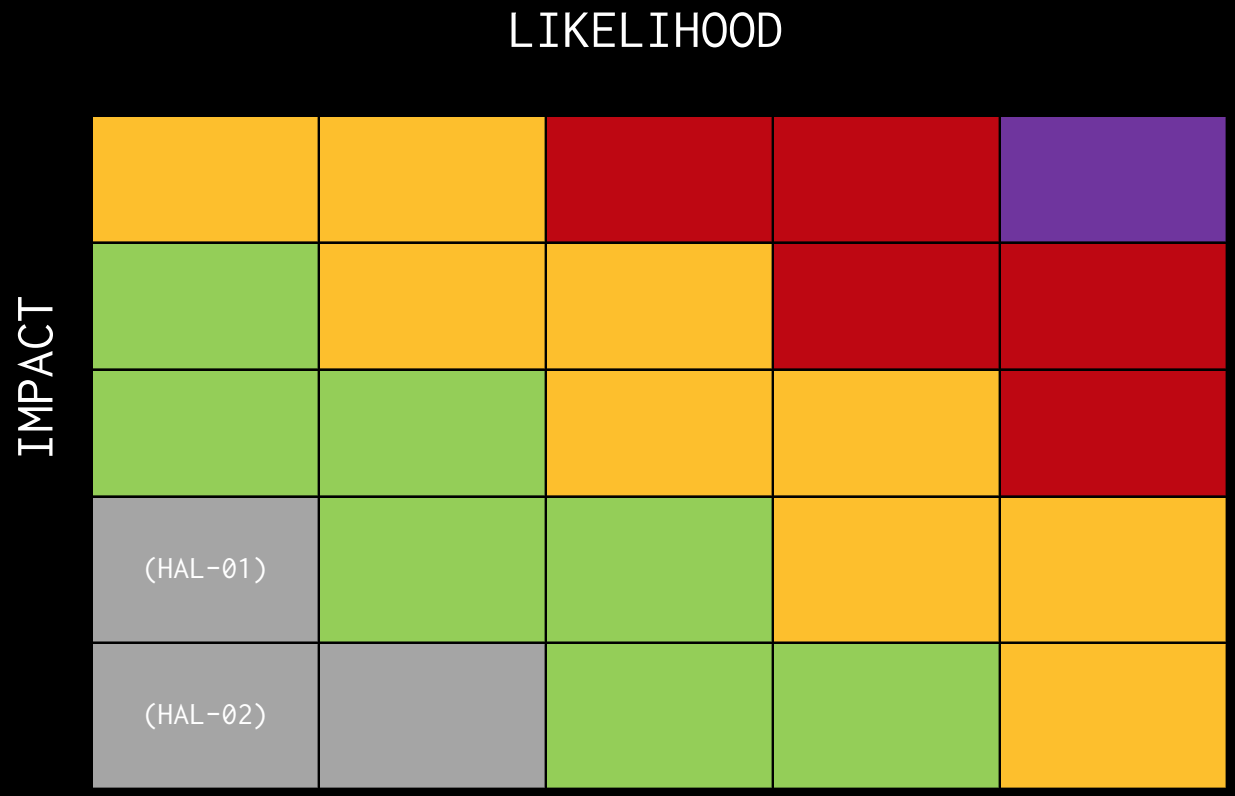
1. CosmWasm Smart Contracts
    (a) Repository: https://git.sienna.network/SiennaNetwork/contracts
    (b) Commit IDs in scope:
        i. Lending Rewards update: b88b365a690a65121eb77523378be70c4ec604f5
        ii. Launchpad update: 3dce7413f4d1678595b288501f8f6fd8110829ff
    (c) Contracts in scope:
        i. lending
        ii. launchpad

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. Lending: dbe3c8688e75dd0b89634e6f22f861e44c849f06
2. Launchpad: ee2a77996c480ce97fbc14322fc9abe612923dae
3. Rewards V3: 20dce5c6a7dfcd983ae2fbc4292b1b58678ae07e

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	2



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) TOTAL SPEED IS NOT VALIDATED	Informational	ACKNOWLEDGED
(HAL-02) CONFUSING FUNCTION NAMES	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) TOTAL SPEED IS NOT VALIDATED - INFORMATIONAL

Description:

Parameters configuration has high impact on the behavior of the protocol. As such, it is considered a good practice to validate the parameters each time they are initialized or changed.

It was found that in the **lend/overseer/contract.rs**, the rewards_rate was not validated, as opposed to premium and close_factor that are validated. However, that parameter is used as a verification for each ongoing rewards distribution, the sum of each should result in rewards_rate, so it doesn't require as much limitation as other parameters. Nevertheless, a reward distribution speed can be updated without bounds as long as the rewards_rate matches the total speed.

Code Location:

- The reward's rate is not validated:

Listing 1: libraries/lend-shared/src/interfaces/overseer.rs,

```
267 pub fn new(
268     premium: Decimal256,
269     close_factor: Decimal256,
270     rewards_rate: Decimal256,
271 ) -> StdResult<Self> {
272     Self::validate_close_factor(&close_factor)?;
273     Self::validate_premium(&premium)?;
274
275     Ok(Self {
276         premium,
277         close_factor,
278         rewards_rate,
279     })
280 }
```

- The rewards rate is not validated:

Listing 2: contracts/lend/overseer/src/contract.rs, (Line 426)

```
425 if let Some(rate) = rewards_rate {
426     config.rewards_rate = rate;
427 }
```

- The distribution speed are updated without limitation:

Listing 3: contracts/lend/overseer/src/contract.rs,

```
430 Markets::update(deps, &market.address, |mut m| {
431     if m.supply_distribution.speed > market.supply_speed {
432         total_speeds =
433             (total_speeds - (m.supply_distribution.speed - market.
 ↳ supply_speed)?)?;
434     } else {
435         total_speeds =
436             (total_speeds + (market.supply_speed - m.
 ↳ supply_distribution.speed)?)?;
437     }
438
439     m.supply_distribution.speed = market.supply_speed;
440
441     if m.borrow_distribution.speed > market.borrow_speed {
442         total_speeds =
443             (total_speeds - (m.borrow_distribution.speed - market.
 ↳ borrow_speed)?)?;
444     } else {
445         total_speeds =
446             (total_speeds + (market.borrow_speed - m.
 ↳ borrow_distribution.speed)?)?;
447     }
448     m.borrow_distribution.speed = market.borrow_speed;
449
450     Ok(m)
451 })?;
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to limit the total speed value within reasonable bounds.

Remediation plan:

ACKNOWLEDGED: The Sienna.Network team acknowledged this finding.

3.2 (HAL-02) CONFUSING FUNCTION NAMES - INFORMATIONAL

Description:

The use of confusing function names regarding their behavior may not have a direct impact on the security of the code, but may be introducing logic bugs in the future for unaware developers.

In the **lend/market/state.rs** file, the total_borrows and total_supply state are implementing increase and decrease functions. These functions effectively add or subtract the adequate amount and stores the new value, but does return the value before the update.

That feature is used a lot through all the market contract, and may be a source of bugs for future updates.

Code Location:

- Source of the behaviour for integer state values:

Listing 4: contracts/lend/market/src/state.rs, (Lines 92,104)

```
83 pub fn increase(
84     storage: &mut impl Storage,
85     amount: $data_type,
86 ) -> StdResult<$data_type> {
87     let current = Self::load(storage)?;
88     let new = (current + amount)?;
89
90     Self::save(storage, &new)?;
91
92     Ok(current)
93 }
94
95 pub fn decrease(
96     storage: &mut impl Storage,
97     amount: $data_type,
98 ) -> StdResult<$data_type> {
```

```
 99     let current = Self::load(storage)?;
100     let new = (current - amount)?;
101
102     Self::save(storage, &new)?;
103
104     Ok(current)
105 }
```

- Source of the behaviour for account balances value:

Listing 5: contracts/lend/market/src/state.rs, (Lines 243,259)

```
232 /// Returns the balance as it was before the increment.
233 pub fn add_balance(&self, storage: &mut impl Storage, amount:
 ↳ Uint256) -> StdResult<Uint256> {
234     let account_balance = self.get_balance(storage)?;
235
236     if let Ok(new_balance) = account_balance + amount {
237         self.set_balance(storage, &new_balance)?;
238     } else {
239         return Err(StdError::generic_err(
240             "This deposit would overflow your balance",
241         ));
242     }
243     return Ok(account_balance)
244 }
245
246 /// Returns the balance as it was befor the decrement.
247 pub fn subtract_balance(&self, storage: &mut impl Storage, amount:
 ↳  Uint256) -> StdResult<Uint256> {
248     let account_balance = self.get_balance(storage)?;
249
250     if let Ok(new_balance) = account_balance - amount {
251         self.set_balance(storage, &new_balance)?;
252     } else {
253         return Err(StdError::generic_err(format!(
254             "insufficient funds: balance={}, required={}",
255             account_balance, amount
256         )));
257     }
258
259     return Ok(account_balance);
260 }
```

- Example of utilisation:

Listing 6: contracts/lend/market/src/contract.rs,

```
249 let old_total_borrows = TotalBorrows::increase(&mut deps.storage,
 ↳ amount)?;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to give a more understandable name for the function.

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

THANK YOU FOR CHOOSING

// HALBORN