// HALBORN

Sienna.Network Lending Protocol (Updated code)

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 26th, 2022 - April 29th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/26/2022	Alexis Fabre
0.2	Document Updates	04/29/2022	Alexis Fabre
0.3	Draft version	04/29/2022	Alexis Fabre
0.4	Draft Review	04/29/2022	Gabi Urrutia
1.0	Remediation Plan	04/29/2022	Alexis Fabre
1.1	Remediation Plan Review	04/29/2022	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
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 April 25th, 2022 and ending on April 29th, 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 an improvement to reduce the likelihood and impact of the risks, which was accepted by Sienna team:

- Ensure that all numeric operation uses checked methods.

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

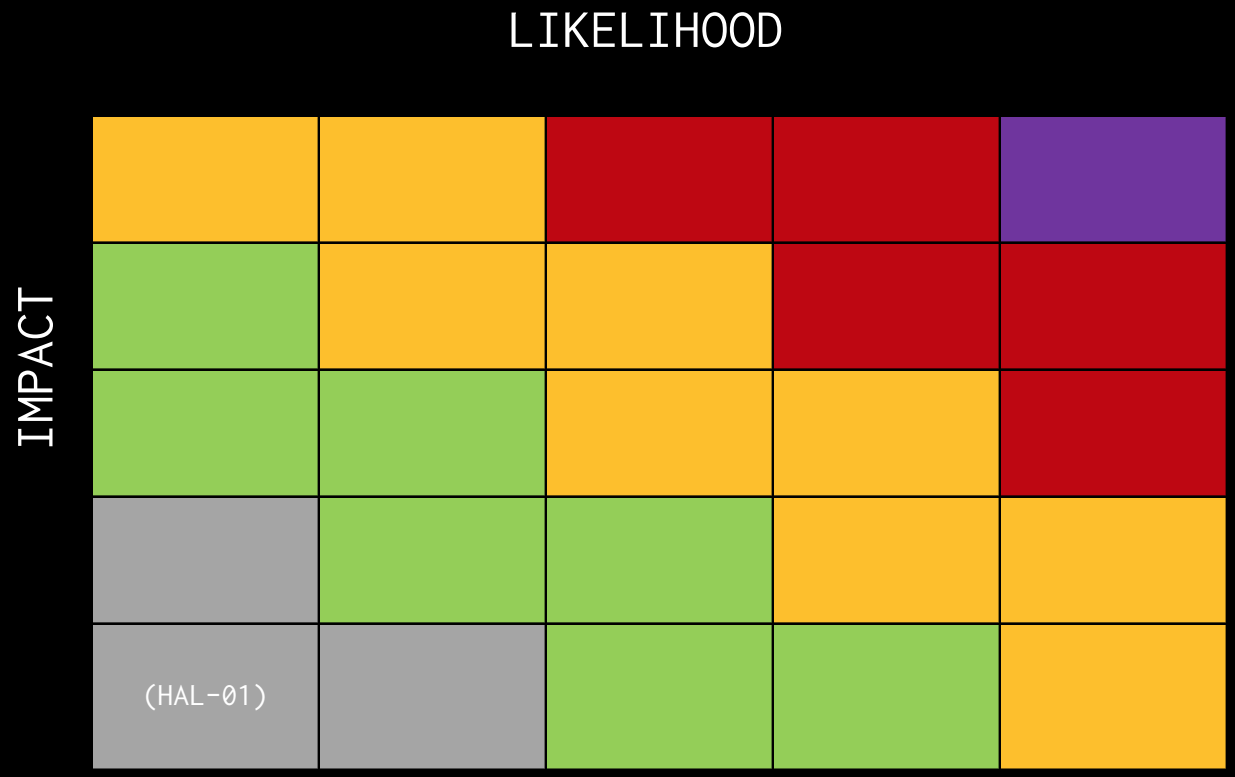
1.4 SCOPE

1. CosmWasm Smart Contracts
   (a) Repository: https://git.sienna.network/SiennaNetwork/contracts
   (b) Commit IDs in scope:
      i. Oracle decimal scaling: 014a376eee9162218f0c52e8d26a24303b1f7dc3
      ii. Market Send/Receive: 915aaf9e6535a8d012371063dad09b96e2287d46
      iii. Refund overpaid amount: 6dc6ad1fde26d8b5dc6fff81514fb0164cf67c07
      iv. Simulate liquidation: 2fa2d8066753828953bc554266ac84d8e872fbb6
   (c) Contracts in scope:
      i. oracle
      ii. market
      iii. overseer

It is worth noting that the results of this audit are a complement to the information provided in a previous report for the security audit performed to the codebase with commit id dbe3c8688e75dd0b89634e6f22f861e44c849f06.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	1



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) UNCHECKED MATH	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) UNCHECKED MATH - INFORMATIONAL

Description:

In computer programming, an overflow occurs when an arithmetic operation attempts to create a numeric value that is outside the range that can be represented with a given number of bits -- either larger than the maximum or lower than the minimum representable value.

In that particular case, a user sends funds to repay his debt, and a remainder is calculated to refund the extra amount. That remainder is then casted down from 256 bits to 128 bits unsigned integer without checking that its value is lower than the maximal value carried by a 128 bits unsigned integer.

This issue has been raised as informational only, as it was not possible to define a clear exploitation scenario for the affected cases, mainly because all SNIP20 amounts and transfers are 128 bits unsigned integers and cannot exceed that limit. However, it seemed like a potentially risky pattern and therefore has been highlighted as such.

Code Location:

Listing 1: contracts/lend/market/src/contract.rs, (Line 797)

```
785 let remainder = snapshot.subtract_balance(interest.borrow_index(&
 ↳ deps.storage)?, amount)?;
786 borrower.save_borrow_snapshot(&mut deps.storage, snapshot)?;
787
788 let amount = (amount.0 - remainder.0).into();
789 TotalBorrows::decrease(&mut deps.storage, amount)?;
790
791 if remainder > Uint256::zero() {
792     let underlying = Contracts::load_underlying(deps)?;
793
794     Ok(HandleResponse {
795         messages: vec![snip20::transfer_msg(
```

```
796             sender,
797             remainder.low_u128().into(),
798             None,
799             None,
800             BLOCK_SIZE,
801             underlying.code_hash,
802             underlying.address
803         )?],
804         log: vec![],
805         data: None
806     })
807 } else {
808     Ok(HandleResponse::default())
809 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use the clamp_u128 method since the casting operation would fail on overflow, which the low_u128 method does not.

Remediation Plan:

ACKNOWLEDGED: The Sienna.Network acknowledged this finding. They also mentioned that low_u128 cannot overflow and thus cannot panic. They use this method in places where they know that in practice the number can never exceed 128::MAX because SNIP-20 tokens themselves use 128-bit numbers and are constrained by this because of the underlying token being used.

THANK YOU FOR CHOOSING

// HALBORN