// HALBORN

SiennaNetwork - Launchpad IDO

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **June 27th, 2022 - July 8th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/11/2022	Elena Maranon
0.2	Document Updated	07/12/2022	Jakub Heba
0.3	Draft Review	07/13/2022	Gabi Urrutia
1.0	Remediation Plan	07/19/2022	Elena Maranon
1.1	Remediation Plan Review	07/19/2022	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
Jakub Heba	Halborn	Jakub.Heba@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

SiennaNetwork engaged Halborn to conduct a security audit on their smart contracts beginning on June 27th, 2022 and ending on July 8th, 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 should be addressed by SiennaNetwork. The main improvements highlighted in this report are:

- Fix the change_config function to properly save the modified data.
- Add extra conditions to avoid useless operations, also adding more security.
- Avoid to use unsafe methods in the code.

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.
- 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

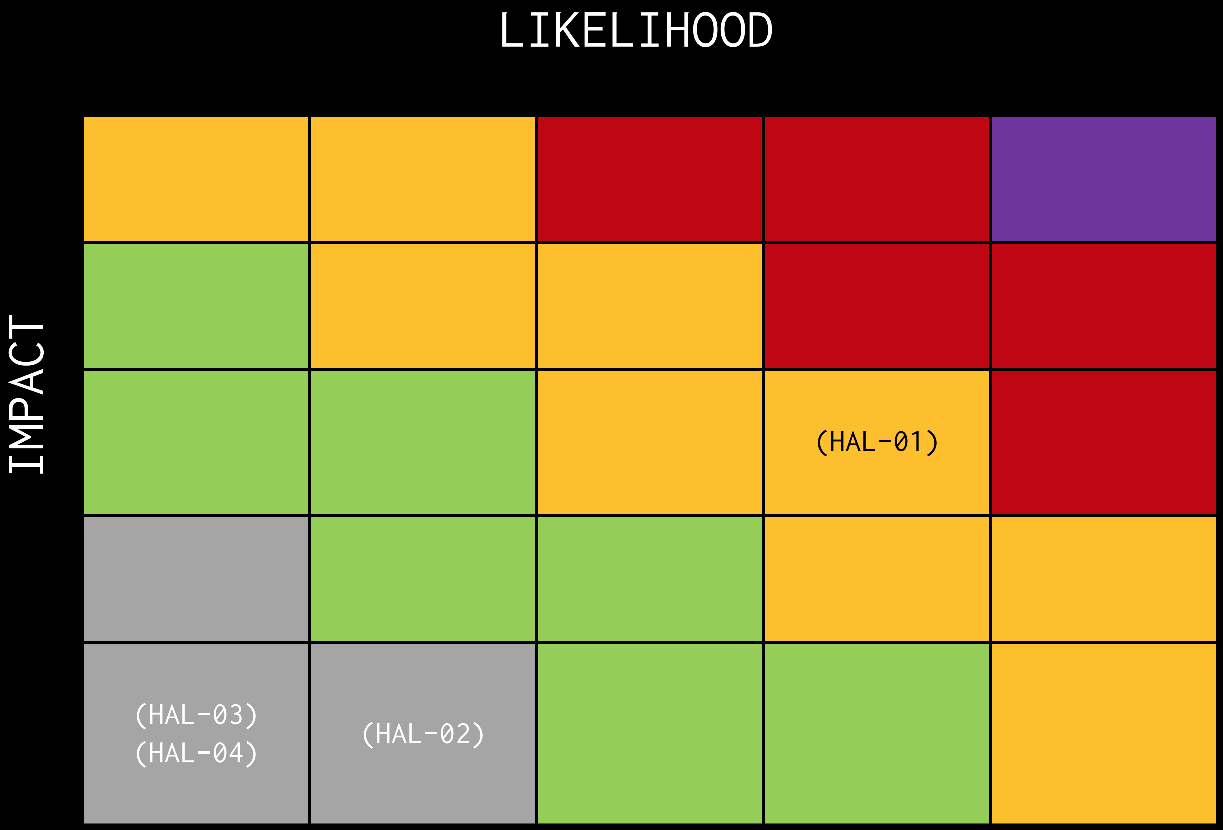
Code repository: SiennaNetwork

1. Smart Contracts
    (a) Commit ID: ee2a77996c480ce97fbc14322fc9abe612923dae
    (b) Contracts in scope:
        i. launchpad/ido
        ii. launchpad/launchpad
        iii. libraries/lpd-shared

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	0	3



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01 NEW CONFIGURATION SETTINGS ARE NOT SAVED)	Medium	SOLVED - 07/15/2022
(HAL-02 UNCHECKED MATH)	Informational	ACKNOWLEDGED
(HAL-03 LACK OF INPUT VALIDATION)	Informational	ACKNOWLEDGED
(HAL-04 EXTRA CONDITIONS TO AVOID USELESS OPERATIONS)	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) NEW CONFIGURATION SETTINGS ARE NOT SAVED - MEDIUM

Description:

The function change_config from file **contracts/launchpad/src/contract.rs** allows the administrator to change some configuration values of the contract after init.

However, when some constraints are modified, the loaded values are not saved after the modification, so the change will not have any effect.

Code Location:

Fragment of the change_config function:

Listing 1: contracts/launchpad/launchpad/src/contract.rs (Lines 134,145)

```rust
133     if min_pre_lock_duration.is_some() || min_sale_duration.is_some() {
134         let mut constraints = Constants::load_constraints(&deps.storage)?;
135
136         match (min_pre_lock_duration, min_sale_duration) {
137             (Some(min_pre_lock_duration), Some(min_sale_duration)) => {
138                 constraints.min_pre_lock_duration = min_pre_lock_duration;
139                 constraints.min_sale_duration = min_sale_duration;
140             }
141             (Some(min_pre_lock_duration), None) => constraints.min_pre_lock_duration = min_pre_lock_duration,
142             (None, Some(min_sale_duration)) => constraints.min_sale_duration = min_sale_duration,
143             (None, None) => unreachable!()
144         }
145         //MISSING Constants::save_constraints
146     }
```

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

It is recommended to apply a save method at the end of the value modification as `constraints.save(&mut deps.storage)?`.

Remediation Plan:

SOLVED: The Sienna.Network team solved the issue in the new commit d3c32520828971c1e39b3af70bcc12f3fbc3630a

3.2 (HAL-02) 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.

The function swap from contract **contracts/launchpad/ido/src/sale.rs** implements a mathematical adding operation which could raise an overflow. This issue has been categorized as informational only, as the exploitation scenario is very specific (PoC Test below). However, as these are potentially risky patterns, they have been highlighted as such.

In addition, the implementation of the macro impl_number_storage from **contracts/launchpad/ido/src/state.rs** also contains an unsafe adding operation. Because this is a macro that can be reused in the future, it is recommended to use checked math, although if the current uses are not vulnerable to overflow.

PoC Test:

Listing 2

```
#[test]
fn testing_overflow() {
    let mut launchpad = Launchpad::new();

    let input_token = launchpad.init_token("Test1", "test3", 18);
    let sold_token = launchpad.init_token("Test2", "test2", 18);

    let max_u128 = 340282366920938463463374607431768211455;
    let min_alloc = 1000000000000000000000000000000000000;

    launchpad.whitelist_project_owner(USERS[0]);

    let ido = launchpad
        .create_new_ido(
            input_token.clone(),
            lpd_shared::interfaces::ido::TokenRelay::Existing(
↳ sold_token.clone()),
            vec![USERS[1].into()],
            Uint128(10000000000000000000),
            lpd_shared::interfaces::ido::SaleConfig {
                max_allocation: Uint128(max_u128),
                min_allocation: Uint128(min_alloc),
                sale_type: lpd_shared::interfaces::ido::SaleType::
↳ PreLockAndSwap,
                vesting_config: None,
            },
            USERS[0],
        )
        .unwrap();

    launchpad.mint_balance(sold_token.clone(), USERS[0], Uint128(
↳ max_u128));
    launchpad.mint_balance(input_token.clone(), USERS[1], Uint128(
↳ max_u128));

    launchpad
        .launch(
            ido.address.clone(),
            Uint128(max_u128),
            lpd_shared::interfaces::ido::LaunchOptions {
                sale_duration: DAY * 14,
                sale_start: None,
                pre_lock_duration: Some(DAY * 4),
            },
        )
        .unwrap();

    launchpad
        .pre_lock(ido.address.clone(),USERS[1],
            Uint128(300000000000000000000000000000000000000),
        )
        .unwrap();

    let time = launchpad.ensemble.block().time;
    launchpad.ensemble.block_mut().time = time + DAY * 8;

    //Overflow should be raised
    launchpad
        .swap(ido.address.clone(),USERS[1],
            Uint128(340282366920938463463374607431768211455),
            None,
        )
        .unwrap_err();
}
```

Code Location:

Fragment of swap function:

Listing 3: contracts/launchpad/launchpad/src/contract.rs (Line 63)

```
51     let profit = if amount.is_zero() {
52         account.pre_lock_amount
53     } else {
54         let output_amount: Uint128 = convert_token(
55             amount.u128(),
56             token_config.constants.rate.u128(),
57             token_config.constants.input_token_decimals,
58             token_config.constants.sold_token_decimals,
59         )?
60         .clamp_u128()?
61         .into();
62 
63         output_amount + account.pre_lock_amount
64     };
65 
66     if profit.is_zero() {
67         return error::nothing_to_claim();
68     } else if account.total_bought.is_zero() && profit <
 ↳ min_allocation {
69         return error::min_allocation_not_reached();
70     }
71 
```

impl_number_storage macro implementation:

Listing 4: contracts/launchpad/ido/src/state.rs (Line 244)

```rust
macro_rules! impl_number_storage {
    ($save_name:ident, $load_name:ident, $key:literal) => {
        pub fn $save_name(storage: &mut impl Storage, amount:
 ↳ Uint128) -> StdResult<()> {
            let total = Self::$load_name(storage)?;

            save(storage, $key, &(total + amount))
        }

        pub fn $load_name(storage: &impl Storage) -> StdResult<
 ↳ Uint128> {
            let result = load(storage, $key)?.unwrap_or_default();

            Ok(result)
        }
    };
}
```

Risk Level:

Likelihood - 2
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 method, the subtraction operator with Rust's checked_subs method, and so on.

Remediation Plan:

ACKNOWLEDGED: The Sienna.Network team acknowledged this finding.

3.3 (HAL-03) LACK OF INPUT VALIDATION - INFORMATIONAL

Description:

The function change_config from **contracts/launchpad/src/contract.rs** allows changing some configuration values of the contract after instantiation, but the inputs used are not validated before assigning them. It is recommended to perform address validation/not-empty or check that constraints values are not zero.

In addition, on function receive from **contracts/launchpad/launchpad/src/lib.rs** the input amount should not be zero in any of the three possible scenarios. Only swap operation admits zero amount but, in this case, it is called through another function (claim_tokens). Although each scenario performs the corresponding checks later on, due to this is a common constraint for the three cases, it is recommended to check this condition at the beginning of the function.

Code Location:

Fragment of change_config method:

Listing 5: contracts/launchpad/launchpad/src/contract.rs (Lines 120,138,139,152,153)

```
112 if tiers.is_some() || rewards_pool.is_some() {
113     let mut config = WhitelistingConfig::load(deps)?;
114
115     match (tiers, rewards_pool) {
116         (Some(mut tiers), Some(rewards_pool)) => {
117             tiers.validate();
118             config.tiers = tiers;
119
120             config.rewards_pool = rewards_pool;
121         }
122         (Some(mut tiers), None) => {
123             tiers.validate();
```

```
            config.tiers = tiers;
        }
        (None, Some(rewards_pool)) => config.rewards_pool = rewards_pool,
        (None, None) => unreachable!()
    }

    config.save(deps)?;
}

if min_pre_lock_duration.is_some() || min_sale_duration.is_some() {
    let mut constraints = Constants::load_constraints(&deps.storage)?;

    match (min_pre_lock_duration, min_sale_duration) {
        (Some(min_pre_lock_duration), Some(min_sale_duration)) => {
            constraints.min_pre_lock_duration = min_pre_lock_duration;
            constraints.min_sale_duration = min_sale_duration;
        }
        (Some(min_pre_lock_duration), None) => constraints.min_pre_lock_duration = min_pre_lock_duration,
        (None, Some(min_sale_duration)) => constraints.min_sale_duration = min_sale_duration,
        (None, None) => unreachable!()
    }
}

if token_contract.is_some() || ido_contract.is_some() {
    let mut contracts = Contracts::load(&deps.storage)?;

    match (token_contract, ido_contract) {
        (Some(token_contract), Some(ido_contract)) => {
            contracts.ido = ido_contract;
            contracts.token = token_contract;
        }
        (Some(token_contract), None) => contracts.token = token_contract,
        (None, Some(ido_contract)) => contracts.ido = ido_contract,
        (None, None) => unreachable!()
    }
```

```
159
160     contracts.save(&mut deps.storage)?;
161 }
```

Fragment of receive method:

Listing 6: contracts/launchpad/launchpad/src/contract.rs (Line 193)

```
183 fn receive(
184         from: HumanAddr,
185         amount: Uint128,
186         msg: Option<Binary>,
187         _sender: HumanAddr,
188     ) -> StdResult<HandleResponse> {
189         let msg = msg.ok_or_else(|| StdError::generic_err("Message
  ↳  field can't be empty."))?;
190         let msg = from_binary::<ReceiverCallbackMsg>(&msg)?;
191         let token_config = Config::load_token_config(&deps.storage
  ↳ , &deps.api)?;
192
193     //Add input check here
194
195         match msg {
196             ReceiverCallbackMsg::Launch { options } => {
197                 if token_config.sold.link().address != env.message
  ↳ .sender {
198                     return Err(StdError::unauthorized());
199                 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to validate the input parameters of the functions mentioned.

Remediation Plan:

ACKNOWLEDGED: The Sienna.Network team acknowledged this finding.

3.4 (HAL-04) EXTRA CONDITIONS TO AVOID USELESS OPERATIONS - INFORMATIONAL

Description:

The functions receive_tokens and refund_tokens from **contracts/launch-pad/src/lib.rs** could avoid performing useless operations by adding some simple checks, in addition to make the code more secure.

The receive_tokens function allows users to claim for their vested tokens, and it could be called by external users at anytime. Since there is an option for periodic refund, the user could call this function several times, receiving a portion of tokens on each call.

A simple checking at the beginning of the function would avoid executing the get_portion code.

Listing 7

```
if account.total_claimed.u128() >= account.total_bought.u128() {
    return error::nothing_to_claim;
}
```

The refund_tokens function allows the admin to refund the non-bought tokens from the contract once the sale has finished. This operation should be done just once, and it must avoid to refund by mistake the user tokens that are waiting on vesting period. The functions **Global::load_-total_bought** and **Global::load_total_claimed** are used for that control.

However, although these functions avoid admin to refund users tokens, it does not make a basic and simple check: if (balance-remaining)== 0 there is nothing to claim for the admin, so is not necessary to make the transfer call with zero amount.

Code Location:

Fragment of receive_tokens function:

Listing 8: contracts/launchpad/launchpad/src/contract.rs (Line 102)

```
98  #[handle]
99      fn receive_tokens(recipient: Option<HumanAddr>) -> StdResult<
 ↳ HandleResponse> {
100         let mut account = Account::load(deps, &env.message.sender)
 ↳ ?;
101
102         //Add checking here
103
104         let (unlocked, claimable) = get_portion(deps, &account,
 ↳ env.block.time)?;
105
106         if claimable == 0 {
107             return error::nothing_to_claim();
108         }
109
110         account.total_claimed += claimable.into();
111         account.save(deps)?;
112
113         Global::increment_total_claimed(&mut deps.storage,
 ↳ claimable.into())?;
```

Fragment of refund_tokens function:

Listing 9: contracts/launchpad/launchpad/src/contract.rs (Lines 163,170)

```
155 // Don't refund any unclaimed amount when vesting.
156 if let ReturnTokenType::Refund {} = return_type {
157     if Config::load_vesting_config(&deps.storage)?.is_some() {
158         let bought = Global::load_total_bought(&deps.storage)?;
159         let claimed = Global::load_total_claimed(&deps.storage)?;
160
161         let remaining = (bought - claimed)?;
162
163         balance = (balance - remaining)?;
164     }
165 }
166
167 Ok(HandleResponse {
    messages: vec![snip20::transfer_msg(
        recipient.unwrap_or(env.message.sender),
        balance,
        None,
        None,
        BLOCK_SIZE,
        token.code_hash,
        token.address,
    )?],
    data: None,
    log: vec![],
})
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to add some extra checks to add security and avoid useless operations.

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/crate	Short Description
RUSTSEC-2020-0071	time	Potential segfault in the time crate, upgrade to >=0.2.23

THANK YOU FOR CHOOSING

HALBORN