// HALBORN

DRAFT

Sienna.Network Rewards V3

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **December 6th, 2021 - December 22nd, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/06/2021	Luis Quispe Gonzales
0.2	Document Updates	12/21/2021	Luis Quispe Gonzales
0.3	Draft Version	12/22/2021	Luis Quispe Gonzales
0.4	Draft Review	12/27/2021	Gabi Urrutia
1.0	Remediation Plan	02/21/2022	Luis Quispe Gonzales
1.1	Remediation Plan Review	02/21/2022	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
Luis Quispe Gonzales	Halborn	Luis.QuispeGonzales@halborn.com

EXECUTIVE OVERVIEW

1.1 AUDIT SUMMARY

Sienna.Network engaged Halborn to conduct a security assessment on CosmWasm smart contracts beginning on December 6th, 2021 and ending December 22nd, 2021.

The security engineers involved on the audit are blockchain and smart contract security experts with advanced penetration testing, smart contract hacking, and in-depth knowledge of multiple blockchain protocols.

The purpose of this audit is to achieve the following:

- Ensure that smart contract functions work 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 were mostly addressed by Sienna.Network team. The main ones are the following:

- Update the logic of handle_receive_function to verify the sender.
- Enable the transfer of remaining rewards when pools are closed.
- Remove drain function in contract to avoid rug-pull related attacks. Otherwise, enable additional security measures for this functionality.
- Harden access control for migration functions.
- Split admin address transfer functionality to allow transfer to be completed by recipient.

External threats, such as financial related attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 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

1.3 SCOPE

1. CosmWasm Smart Contracts
    (a) Repository: rewards
    (b) Commit ID: 649dec6fe6ae4af2f8235e5abdab6ebc5eb3a851
    (c) Contracts in scope:
        i. rewards
2. Retest of updated code
    (a) Repository: amm-rewards
    (b) Commit ID: 39e87e425f3ebabfbdc4e45d41026185e8ac2858
    (c) Contracts in scope:
        i. rewards

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	3	1	2	2

LIKELIHOOD

IMPACT

		(HAL-03)	(HAL-02)	(HAL-01)
	(HAL-05)			
(HAL-06) (HAL-07)				(HAL-04)
(HAL-08)				
	(HAL-09)			

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) USERS CAN INCREASE THEIR STAKED TOKENS WITHOUT DEPOSITING	Critical	SOLVED - 12/22/2021
(HAL-02) REWARDS CANNOT BE CLAIMED WHEN REWARD POOLS ARE CLOSED	High	SOLVED - 12/23/2021
(HAL-03) POSSIBILITY TO TRANSFER AN ARBITRARY AMOUNT OF TOKENS OUT OF REWARD POOLS	High	SOLVED - 12/30/2021
(HAL-04) INSUFFICIENT ACCESS CONTROL IN MIGRATION FUNCTIONS	High	SOLVED - 12/22/2021
(HAL-05) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION	Medium	SOLVED - 12/28/2021
(HAL-06) MIGRATION REQUEST FUNCTION DOES NOT VERIFY THAT OLD REWARD POOL IS ENABLED TO MIGRATE	Low	RISK ACCEPTED
(HAL-07) REWARDS UPDATE IS NOT ENFORCED WHEN CLAIMING	Low	NOT APPLICABLE
(HAL-08) BONDING PERIOD COULD UNWRAP TO AN INADEQUATE DEFAULT VALUE	Informational	SOLVED - 12/28/2021
(HAL-09) FUNCTION TO QUERY LP TOKEN INFO DOES NOT WORK PROPERLY	Informational	SOLVED - 12/29/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) USERS CAN INCREASE THEIR STAKED TOKENS WITHOUT DEPOSITING - CRITICAL

Description:

handle_receive_migration function in **contracts/rewards/lib.rs** allows users to increase without restrictions their amount of staked LP tokens in reward pools without depositing any token. It is important to mention that there is no need that reward pools are closed because this function can be called at anytime (and many times, too).

As a consequence, malicious users can later completely claim / withdraw all accumulated rewards and staked LP tokens in reward pools.

A proof of concept video showing how to exploit this security issue is included in the report.

Code Location:

The amount of staked LP tokens is increased without verifying the sender:

Listing 1: contracts/rewards/lib.rs (Lines 228,229)

```
215 fn handle_receive_migration (&mut self, env: Env, data: Binary) ->
216     StdResult<HandleResponse>
217 {
218     let (migrant, vk, staked): AccountSnapshot = from_slice(&data.
            as_slice())?;
219     let id = self.canonize(migrant.clone())?;
220     // Set the migrant's viewing key
221     if let Some(vk) = vk {
222         // for some reason it does not see Auth as implemented
223         //Auth::save_vk(&mut core, id.as_slice(), &vk)?;
224         self.set_ns(crate::auth::VIEWING_KEYS, id.as_slice(), &vk)
                ?;
225     }
226     // Add the LP tokens transferred by the migration
```

```
227     // to the migrant's new account
228     Account::from_addr(self, &migrant, env.block.time)?
229         .commit_deposit(self, staked)?;
230     HandleResponse::default()
231         .log("migrated", &staked.to_string())
232 }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Update the logic of handle_receive_migration function to verify that the sender should be in the old reward pool and its address registered in the new reward pool (CAN_MIGRATE_FROM).

Remediation plan:

SOLVED: The issue was fixed in commit 20dce5c6a7dfcd983ae2fbc4292b1b58678ae07e.

3.2 (HAL-02) REWARDS CANNOT BE CLAIMED WHEN REWARD POOLS ARE CLOSED - HIGH

Description:

According to the migration flow defined, once a reward pool is closed, users should be able to withdraw their stake and claim any remaining rewards.

However, when users try to **deposit**, **withdraw** or **claim** in a closed reward pool, the function force_exit from **contracts/rewards/algo.rs** is called automatically. This function transfers staked LP tokens to users, but does not return remaining rewards.

As a consequence, users will never be able to claim their rewards because when a reward pool is closed, it cannot be opened anymore.

Code Location:

force_exit function transfers staked LP tokens to users, but does not return rewards:

Listing 2: contracts/rewards/algo.rs (Lines 648,651)

```
644 fn force_exit (&mut self, core: &mut C) -> StdResult<
      HandleResponse> {
645     if let Some((ref when, ref why)) = self.total.closed {
646         let amount = self.staked;
647         let response = HandleResponse::default()
648             .msg(RewardsConfig::lp_token(core)?.transfer(&self.
                  address, amount)?)?
649             .log("close_time",   &format!("{}", when))?
650             .log("close_reason", &format!("{}", why))?;
651         self.commit_withdrawal(core, amount)?;
652         Ok(response)
653     } else {
654         errors::pool_not_closed()
```

Risk Level:

Likelihood - 4
Impact - 5

Recommendation:

Update the logic of `force_exit` function to transfer remaining rewards to users when a reward pool is closed.

Remediation plan:

SOLVED: The issue was fixed in commit `6bb07ab9a720258fa1e19941397c5bb346186512`.

3.3 (HAL-03) POSSIBILITY TO TRANSFER AN ARBITRARY AMOUNT OF TOKENS OUT OF REWARD POOLS - HIGH

Description:

drain function in **contracts/rewards/drain.rs** allows admin to unrestrictedly increase allowance of a potentially malicious external account and later transfer an arbitrary amount of tokens (LP or SIENNA tokens) out of **reward pools**. The maximum amount of tokens to transfer depends on the amount of staked LP tokens and accumulated rewards in each **reward pool**.

Attack scenario:

1. Malicious (or compromised) admin calls drain function with snip20 = <lp_token_contract> and recipient = <malicious_account>.
2. As a consequence of **Step 1**, the aforementioned function will increase the allowance of the malicious account for LP token.
3. The malicious (or compromised) admin calls transfer_from function in **LP token** contract to transfer all tokens out of **reward pool**.
4. **Step 1** to **Step 3** are repeated using snip20 = <SIENNA_token_contract>.
5. **Step 1** to **Step 4** are repeated for each **reward pool**.

Code Location:

Listing 3: contracts/rewards/drain.rs (Line 27)

```
17 // Update the viewing key if the supplied
18 // token info for is the reward token
19 let reward_token = RewardsConfig::reward_token(self)?;
20 if reward_token.link == snip20 {
21     self.set(RewardsConfig::REWARD_VK, key.clone())?
22 }
```

```
23 let allowance = Uint128(u128::MAX);
24 let duration  = Some(env.block.time + DAY * 10000);
25 let snip20    = ISnip20::attach(snip20);
26 HandleResponse::default()
27     .msg(snip20.increase_allowance(&recipient, allowance, duration
          )?)?
28     .msg(snip20.set_viewing_key(&key)?)
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

If not used, it is recommended to totally remove drain function to avoid rug-pull related attacks. Otherwise, the following security measures should be applied:

- As a prerequisite to enable drain function to admin, reward pool should be closed.
- Closed reward pool should allow users to withdraw their stakes and claim their remaining rewards. Alternatively, users should be able to migrate their stakes / rewards to a new reward pool if available.
- Once a reward pool is closed, drain function could be enabled to admin after a predefined wait time, which should be hardcoded in contract and with a minimum threshold (in case is modifiable).

Remediation plan:

SOLVED: The issue was fixed in the following commits:

- 48f5b768a31e78b93f95e064c7b10f0630d720c5
- 5872d165ab7d6fafaef048c33c19b282ef26b233

3.4 (HAL-04) INSUFFICIENT ACCESS CONTROL IN MIGRATION FUNCTIONS - HIGH

Description:

handle_request_migration function in **contracts/rewards/migration.rs** and handle_export_state function in **contracts/rewards/lib.rs** do not have adequate access controls for migration process, which generates the following consequences:

- In handle_request_migration function, previous reward pool address (prev) is not restricted; thus, users can execute any potentially malicious message on behalf of current reward pool.
- handle_export_state function verifies the value of can_export_state function before continuing with its execution. However, this latter function entirely ignores sender address, which means that does not exist access control for handle_export_state function.

Code Location:

handle_request_migration function does not verify old reward pool address:

Listing 4: contracts/rewards/migration.rs (Line 178)

```
175 fn handle_request_migration (&mut self, env: Env, prev:
      ContractLink<HumanAddr>)
176     -> StdResult<HandleResponse>
177 {
178     HandleResponse::default().msg(CosmosMsg::Wasm(WasmMsg::Execute
          {
179         contract_addr:      prev.address,
180         callback_code_hash: prev.code_hash,
181         send:               vec![],
182         msg: self.wrap_export_msg(EmigrationHandle::ExportState(
              env.message.sender))?,
183     }))
```

handle_export_state function verifies the value of can_export_state function (see below) before continuing with its execution:

Listing 5: contracts/rewards/lib.rs (Line 177)

```
174 fn handle_export_state (&mut self, env: &Env, migrant: &HumanAddr)
175     -> StdResult<HandleResponse>
176 {
177     let receiver    = self.can_export_state(&env, &migrant)?;
178     let mut account = Account::from_addr(self, &migrant, env.block
            .time)?;
179     let staked      = account.staked;
180     let id          = self.canonize(migrant.clone())?;
181
182     let snapshot = to_binary(&((
183         migrant.clone(),
184         Auth::load_vk(self, id.as_slice())?.map(|vk|vk.0),
185         staked
186     ) as AccountSnapshot))?;
```

can_export_state function does not verify that address of receiver_link is equal to sender address:

Listing 6: contracts/rewards/migration.rs (Lines 84,85)

```
71 fn can_export_state (&mut self, env: &Env, migrant: &HumanAddr)
72     -> StdResult<ContractLink<HumanAddr>>
73 {
74     // The ExportState transaction is not meat to be called
           manually by the user;
75     // it must be called by the contract which is receiving the
           migration
76     if &env.message.sender == migrant {
77         return Err(StdError::generic_err("This handler must be
               called as part of a transaction"))
78     }
79     // If migration to the caller contract is enabled,
80     // its code hash should be available in storage
81     let id = self.canonize(env.message.sender.clone())?;
82     let receiver_link: Option<ContractLink<HumanAddr>> =
83         self.get_ns(Self::CAN_MIGRATE_TO, id.as_slice())?;
84     if let Some(receiver_link) = receiver_link {
85         Ok(receiver_link)
```

```
86      } else {
87          Err(StdError::generic_err("Migration to this target is not
                enabled."))
88      }
89  }
```

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

Harden access control for functions mentioned above by verifying the sender address.

Remediation plan:

SOLVED: The issue was fixed in the following commits:

- 20dce5c6a7dfcd983ae2fbc4292b1b58678ae07e
- 6e9511a5721a38690b02c8fbab1239eea96d0a08.

3.5 (HAL-05) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION - MEDIUM

Description:

An incorrect use of AuthHandle::ChangeAdmin message in **contracts/rewards/auth.rs** can set admin of **rewards** contract to an invalid address and inadvertently lose total control of this contract, which cannot be undone in any way.

Currently, the admin of **rewards** contract can change the **admin address** using the aforementioned message in a single transaction and without confirmation from the new address.

Code Location:

AuthHandle::ChangeAdmin message in **rewards** contract is routed to change_admin function:

Listing 7: contracts/rewards/auth.rs (Lines 37,38)

```
35 fn handle (&mut self, env: Env, msg: AuthHandle) -> StdResult<
       HandleResponse> {
36     match msg {
37         AuthHandle::ChangeAdmin { address } =>
38             self.change_admin(env, address),
39         AuthHandle::CreateViewingKey { entropy, .. } =>
40             self.create_vk(env, entropy),
41         AuthHandle::SetViewingKey { key, .. } =>
42             self.set_vk(env, key)
43     }
44 }
```

change_admin function calls save_admin function:

Listing 8: contracts/rewards/auth.rs (Line 55)

```
53 fn change_admin(&mut self, env: Env, address: HumanAddr) ->
       StdResult<HandleResponse> {
54     self.assert_admin(&env)?;
55     self.save_admin(&address)?;
56     Ok(HandleResponse::default())
57 }
```

save_admin function saves the new admin address in a single transaction:

Listing 9: contracts/rewards/auth.rs (Line 68)

```
66 fn save_admin(&mut self, address: &HumanAddr) -> StdResult<()> {
67     let admin = self.api().canonical_address(address)?;
68     self.set(ADMIN_KEY, Some(&admin))?;
69     Ok(())
70 }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended to split **admin transfer** functionality into set_admin and accept_admin functions. The latter function allows the transfer to be completed by the recipient.

Remediation plan:

SOLVED: The issue was fixed in commit 52a04b7261d5cdfe79d98d76c50c0b63103a0bd4.

3.6 (HAL-06) MIGRATION REQUEST FUNCTION DOES NOT VERIFY THAT OLD REWARD POOL IS ENABLED TO MIGRATE - LOW

Description:

handle_request_migration function in **contracts/rewards/migration.rs** does not verify that old reward pool is closed (if applicable) and enabled to migrate to a new one (CAN_MIGRATE_TO), so it can be called at anytime and not just during migration process as expected.

Code Location:

Listing 10: contracts/rewards/migration.rs

```
175 fn handle_request_migration (&mut self, env: Env, prev:
      ContractLink<HumanAddr>)
176     -> StdResult<HandleResponse>
177 {
178     HandleResponse::default().msg(CosmosMsg::Wasm(WasmMsg::Execute
          {
179         contract_addr:      prev.address,
180         callback_code_hash: prev.code_hash,
181         send:               vec![],
182         msg: self.wrap_export_msg(EmigrationHandle::ExportState(
              env.message.sender))?,
183     }))
184 }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Verify in handle_request_migration function that the old reward pool is closed (if applicable) and enabled to migrate to a new reward pool (CAN_MIGRATE_TO).

Remediation plan::

RISK ACCEPTED: The Sienna.Network team accepted the risk for this finding in favor of being able to run several versions of the contract at the same time.

3.7 (HAL-07) REWARDS UPDATE IS NOT ENFORCED WHEN CLAIMING - LOW

Description:

According to epoch flow defined, SIENNA tokens are transferred to reward pools in a daily basis. When users call claim function in **contracts/rewards/algo.rs**, it does not update rewards for calling user before transferring SIENNA tokens.

As a consequence, in the worst scenario, users could claim rewards that are not updated since the previous day. Furthermore, this difference cannot be claimed later because users' state, which is used to calculate rewards, is reset after claiming: starting_pool_volume, starting_pool_rewards and volume.

Code Location:

claim function calls commit_claim function and transfer SIENNA tokens without updating rewards for calling user:

Listing 11: contracts/rewards/algo.rs (Lines 638,640)

```
628 fn claim (&mut self, core: &mut C) -> StdResult<HandleResponse> {
629     if self.total.closed.is_some() {
630             self.force_exit(core)
631     } else if self.bonding > 0 {
632         errors::claim_bonding(self.bonding)
633     } else if self.total.budget == Amount::zero() {
634         errors::claim_pool_empty()
635     } else if self.earned == Amount::zero() {
636         errors::claim_zero_claimable()
637     } else {
638         self.commit_claim(core)?;
639         HandleResponse::default()
640             .msg(RewardsConfig::reward_token(core)?.transfer(&self
                    .address, self.earned)?)?
641             .log("reward", &self.earned.to_string())
642     }
```

commit_claim function updates the amount claimed without updating rewards for calling user:

Listing 12: contracts/rewards/algo.rs (Line 689)

```
684 fn commit_claim (&mut self, core: &mut C) -> StdResult<()> {
685     if self.earned > Amount::zero() {
686         self.reset(core)?;
687         self.total.commit_claim(core, self.earned)?;
688     }
689     Ok(())
690 }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to automatically update rewards of calling users before transferring them SIENNA tokens.

Remediation plan:

NOT APPLICABLE: Sienna.Network team claimed that **rpt** contract **[out of scope for this security audit]** distributes a fixed amount of tokens among all reward pools once per 24 hours. So, if someone calls vest function in the middle of that period, it won't do anything.

3.8 (HAL-08) BONDING PERIOD COULD UNWRAP TO AN INADEQUATE DEFAULT VALUE - INFORMATIONAL

Description:

get function in **ITotal** trait from **contracts/rewards/algo.rs** unwraps **bonding period** (if it is **None**) to a value of 0, which could affect severely rewards claiming speed. However, this is an unlikely scenario because total.bonding is set to a value of Some(DAY) during initialization of **rewards** contract.

Code Location:

Listing 13: contracts/rewards/algo.rs (Line 396)

```
392 // # 4. Throttles
393 // * Bonding period: user must wait this much before each claim.
394 // * Closing the pool stops its time and makes it
395 //   return all funds upon any user action.
396 total.bonding      = core.get(RewardsConfig::BONDING)?.unwrap_or(0
        u64);
397 total.closed       = core.get(RewardsConfig::CLOSED)?;
398 Ok(total)
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Update the logic of get function to unwrap **bonding period** (if it is **None**) to a default value that does not affect negatively claiming speed, e.g.: Some(DAY).

Remediation plan:

SOLVED: The issue was fixed in commit b144914a5d3cc538f42f2358ef6344193835340c.

3.9 (HAL-09) FUNCTION TO QUERY LP TOKEN INFO DOES NOT WORK PROPERLY - INFORMATIONAL

Description:

token_info function in **contracts/rewards/lib.rs** should show information about staked LP token in **rewards** contract. However, the values for the following fields are not real, but static ones: symbol, decimals and total_supply.

Code Location:

Listing 14: contracts/rewards/lib.rs (Line 162)

```
156 fn token_info (&self) -> StdResult<Response> {
157     let info = RewardsConfig::lp_token(self)?.query_token_info(
        self.querier())?;
158     Ok(Response::TokenInfo {
159         name:         format!("Sienna Rewards: {}", info.name),
160         symbol:       "SRW".into(),
161         decimals:     1,
162         total_supply: None
163     })
164 }
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Update the logic of token_info function to show the real values of symbol, decimals and total_supply for LP token.

Remediation plan:

SOLVED: The issue was fixed in commit 57c9d10bc240032c4d23632ee68aa5cd5a7d9ca1.

THANK YOU FOR CHOOSING

// HALBORN