// HALBORN

Archway - Vesting

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **December 19th, 2022 - December 27th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/19/2022	Alexis Fabre
0.2	Document Updates	12/23/2022	Alexis Fabre
0.3	Draft Version	12/27/2022	Alexis Fabre
0.4	Draft Review	12/28/2022	Luis Quispe Gonzales
0.5	Draft Review	12/28/2022	Gabi Urrutia
0.6	Document Updates	01/04/2023	Alexis Fabre
0.7	Document Updates Review	01/04/2023	Gabi Urrutia
1.0	Remediation Plan	01/23/2023	Alexis Fabre
1.1	Remediation Plan Review	01/24/2023	Luis Quispe Gonzales

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
Alexis Fabre	Halborn	Alexis.Fabre@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com
Lukasz Mikula	Halborn	Lukasz.Mikula@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Archway engaged Halborn to conduct a security audit on their smart contract beginning on December 19th, 2022 and ending on December 27th, 2022. The security assessment was scoped to the smart contract provided in the GitHub repository vesting-contracts. Commit hashes and further details can be found in the Scope section of this report.

The audited vesting smart contract enables beneficiaries to withdraw vested coins after a predetermined vesting period, while also delegating the locked funds to validators and participating in Archway's governance. The smart contract also includes a clawback feature, which allows the contract's funder to cancel vesting, delegations, and claim all available funds.

1.2 AUDIT SUMMARY

The team at Halborn assigned two full-time security engineers to audit the security of the smart contract. The security engineers are 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 were mostly addressed by the Archway team. The main ones were the following:

- Ensure that the beneficiary cannot access and redelegate funds after the funder has initiated a clawback event.
- Change the clawback feature in the vesting contract to withdraw the entire contract balance.

- Modify the clawback feature to properly handle a large number of delegations.
- Include proper validation of instantiation messages in the vesting contract.
- Avoid redundant bonded denom verification by performing the comparison at contract initialization.

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 the architecture, purpose, and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could led to arithmetic vulnerabilities.
- Finding unsafe Rust code usage (cargo-geiger).
- Test coverage review (cargo tarpaulin).
- Scanning of Rust files for common vulnerabilities (cargo audit).
- Local deployment (Archway).

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

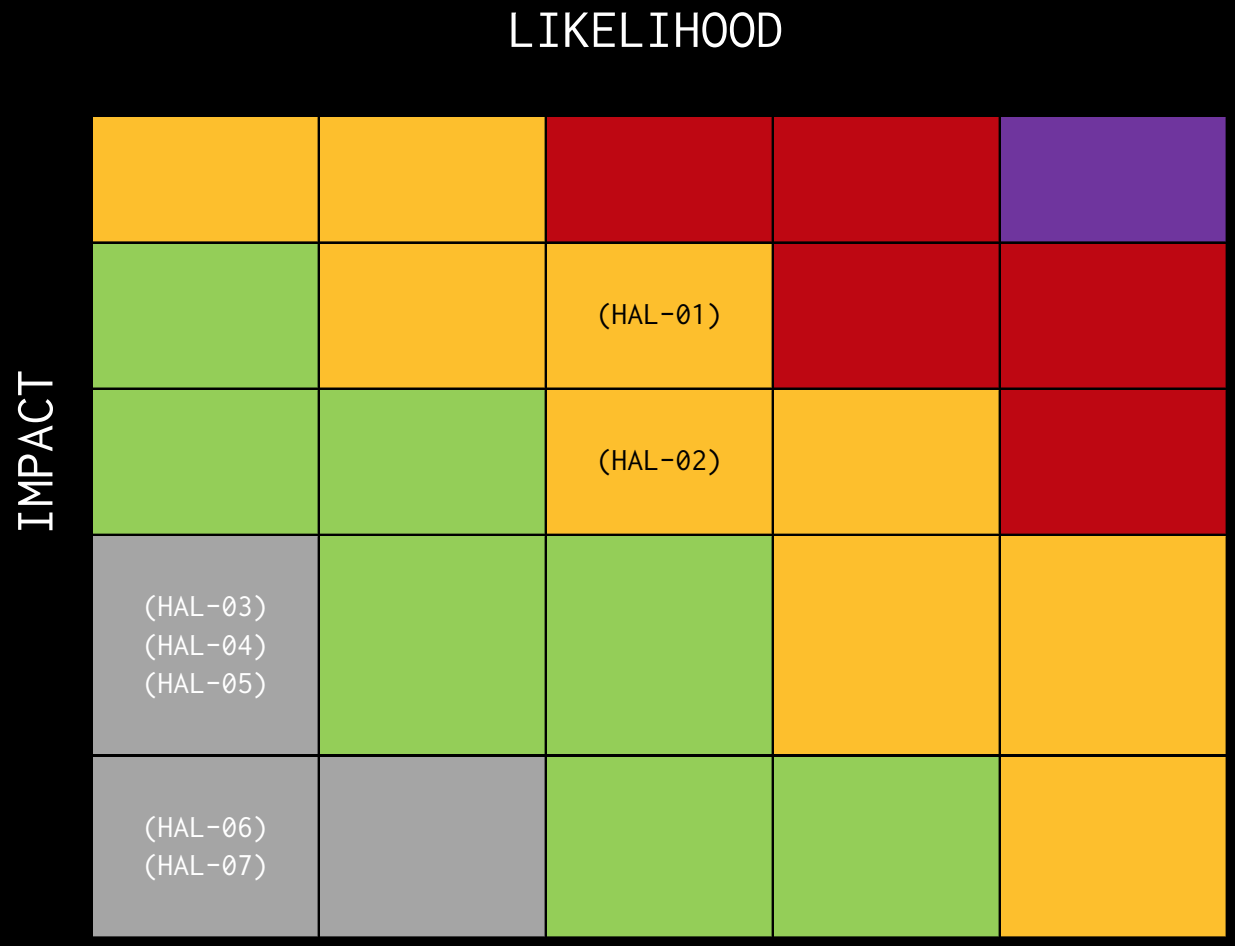
Code repository: vesting-contracts

1. CosmWasm Vesting Smart Contract
   (a) Commit ID: 58c6c29
   (b) Contracts in scope:
      - vesting

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	0	5



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) BENEFICIARY ALLOWED TO LOCK AND REDELEGATE FUNDS DESPITE CLAWBACK	Medium	SOLVED - 01/23/23
(HAL-02) THE CLAWBACK FEATURE DOES NOT WITHDRAW THE MAXIMAL AMOUNT OF FUNDS	Medium	NOT APPLICABLE
(HAL-03) POSSIBILITY TO PROTECT AGAINST CLAWBACK BY EXCEEDING GAS LIMIT	Informational	ACKNOWLEDGED
(HAL-04) MISSING VALIDATION OF THE INSTANTIATION MESSAGE	Informational	SOLVED - 01/23/23
(HAL-05) REDUNDANT VERIFICATION CAUSES UNNECESSARY GAS SPENDING	Informational	SOLVED - 01/23/23
(HAL-06) IMPROPER ERROR HANDLING	Informational	ACKNOWLEDGED
(HAL-07) MISSING ENFORCEMENT OF OVERFLOW CHECKS SETTING IN CONTRACTS WORKSPACE	Informational	SOLVED - 01/23/23

FINDINGS & TECH DETAILS

3.1 (HAL-01) BENEFICIARY ALLOWED TO LOCK AND REDELEGATE FUNDS DESPITE CLAWBACK - MEDIUM

Description:

The smart contract implements a clawback feature, which allows the contract funder to cancel vesting, delegations, and claim all available funds. Because undelegating takes 21 days to release the funds, the funder can withdraw them later.

The vesting contract's redelegate entry point does not verify whether the funder has clawed back the contract. As a result, even if the funder has clawed back the contract, the beneficiary can still redelegate funds and move the stake from one validator to another, keeping the coins in an unbonding state during the transition periods, during which the funder cannot interact with the coins.

Being able to repeat this operation indefinitely, this behavior allows the beneficiary to retain some control over funds that they should not have access to, which could result in financial loss for the funder.

Code Location:

Listing 1: contract.rs (Line 185)

```
175 let vesting = VESTING.load(deps.storage).unwrap();
176 if vesting.beneficiary != info.sender {
177     return Err(ContractError::Unauthorized(info.sender.to_string()
 ↳ ));
178 }
179 if vesting.denom != deps.querier.query_bonded_denom()? {
180     return Err(ContractError::Unavailable(
181         "vested coins denom is different from staking bond denom".
 ↳ to_string(),
182     ));
183 }
```

```
184
185 Ok(Response::new().add_message(StakingMsg::Redelegate {
186     src_validator,
187     dst_validator,
188     amount: Coin {
189         denom: vesting.denom,
190         amount,
191     },
192 }))
```

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

It is recommended that all beneficiary entry points be restricted to ensure that the contract has not yet been clawed back.

Remediation Plan:

SOLVED: The issue has been remediated in commit 997a767. In the execute_redelegate function, the contract now verifies that vesting .balance.clawed_back is **false** before continuing the execution. If clawed_back were set to **true**, the execute_redelegate function would revert.

3.2 (HAL-02) THE CLAWBACK FEATURE DOES NOT WITHDRAW THE MAXIMAL AMOUNT OF FUNDS - MEDIUM

Description:

The vesting contract contains a clawback feature that enables the contract's funder to cancel vesting, delegations, and claim all available funds. During clawback events, the contract calculates the number of locked coins still in the contract and returns that amount to the funder.

This approach may not allow the funder to claim the maximum amount of funds. The funder, for example, would not have access to the unlocked coins even if the beneficiary had not yet withdrawn or spent them. As a result, the funder may be unable to recover the entire amount of funds to which they are entitled. Furthermore, the beneficiary could still withdraw their unlocked coins, and no checks are in place to prevent this.

Code Location:

Listing 2: balance.rs (Line 99)

```
94 pub fn clawback(&mut self, time: Timestamp) -> Uint128 {
95     if self.clawed_back {
96         panic!("clawback twice")
97     }
98     self.clawed_back = true;
99     self.funder_clawback = self.initial_amount - self.unlocked(
↳ time);
100     self.funder_withdraw()
101 }
```

Risk Level:

Likelihood - 3

Impact - 3

Recommendation:

It is recommended that the clawback feature in the vesting contract be changed to withdraw the entire contract balance, rather than just the amount of locked coins that are still in the contract. This ensures that the funder receives the maximum amount of funds during a clawback event. Additional safeguards may also be necessary to prevent the beneficiary from withdrawing their unlocked funds during a clawback event.

Remediation Plan:

NOT APPLICABLE: The behavior mentioned above is expected by the Archway team. They provided the following reason:

> It is expected from the funder to not be able to withdraw unlocked coins: unlocked coins already belong to the beneficiary.

3.3 (HAL-03) POSSIBILITY TO PROTECT AGAINST CLAWBACK BY EXCEEDING GAS LIMIT - INFORMATIONAL

Description:

The vesting contract contains a clawback feature that enables the contract's funder to cancel vesting, delegations, and claim all available funds. The contract queries all delegations and attempts to undelegate funds during a clawback event.

If the beneficiary has delegated to many validators, the clawback execution may run out of gas and fail. As a result, the funder may be unable to recover the entire amount of funds to which they are entitled.

Code Location:

Listing 3: contract.rs (Lines 326,330)

```
324 for delegation in deps
325     .querier
326     .query_all_delegations(env.contract.address)?
327     .into_iter()
328 {
329     UNDELEGATIONS_TMP.save(deps.storage, id, &delegation.amount.
 ↳ amount)?;
330     undelegations.push(SubMsg::reply_on_success(
331         StakingMsg::Undelegate {
332             validator: delegation.validator,
333             amount: delegation.amount,
334         },
335         id,
336     ));
337     id = id + 1;
338 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended that the clawback feature be modified to ensure that it can handle many delegations properly. This could imply specifying as a parameter a set of validators from which to undelegate.

Remediation Plan:

ACKNOWLEDGED: The issue has been acknowledged by the Archway team, stating the following:

> Forcing a limit on active delegation and undelegation might make the staking behavior a lot less flexible.

3.4 (HAL-04) MISSING VALIDATION OF THE INSTANTIATION MESSAGE - INFORMATIONAL

Description:

The cliff period and cliff ratio are not properly validated during the vesting contract's initialization. The cliff period does not have to be longer than the vesting period, nor does the cliff ratio have to be between 0 and 1.

If the funder provides invalid values for the cliff period and cliff ratio, the contract will accept them, potentially leading to unexpected and harmful behavior. If the cliff ratio is greater than one, the contract may try to send more funds than expected.

However, the beneficiary withdraw function's output validation ensures that only available funds can be withdrawn, preventing the execution from reverting.

Code Location:

Listing 4: contract.rs (Lines 48-50)

```
41 VESTING.save(
42     deps.storage,
43     &Vesting::new(
44         info.sender.to_string(),
45         msg.beneficiary.clone(),
46         env.block.time,
47         vesting_coins.amount,
48         msg.vesting_duration,
49         msg.cliff_duration,
50         msg.cliff_unlock_ratio,
51         vesting_coins.denom,
52     ),
53 )?;
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended that the vesting contract include proper validation of the instantiation messages. This may entail adding assertions to ensure that the cliff period is greater than the vesting period and that the cliff ratio is between 0 and 1.

Remediation Plan:

SOLVED: The issue has been remediated in commit 1d47ebb. The InstantitateMsg::validate function carries additional verifications for the cliff duration, vesting duration and cliff unlock ratio parameters.

3.5 (HAL-05) REDUNDANT VERIFICATION CAUSES UNNECESSARY GAS SPENDING - INFORMATIONAL

Description:

The vesting contract contains several instances where the chain bonded denom is compared to the vested denom. In this comparison, a StakingDenom query to the chain is used, which is a more expensive operation in terms of gas fees.

Performing this comparison at multiple points throughout the contract wastes gas. This behavior may result in higher gas fees for contract users, potentially affecting the contract's overall cost and efficiency.

Code Location:

The verification can be found in:

- delegate
- undelegate
- redelegate

Listing 5: contract.rs (Lines 111-112)

```
111 let staking_denom = deps.querier.query_bonded_denom()?;
112 if staking_denom != vesting.denom {
113     return Err(ContractError::Unavailable(format!(
114         "vested coins denom is different from staking bond denom:
 ↳ {} <-> {}",
115         staking_denom, vesting.denom
116     )));
117 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to avoid redundant bonded denom verification by performing the comparison at contract initialization rather than at multiple points throughout its execution. This would reduce the amount of gas required to use the contract, improving its efficiency and lowering overall user costs.

Remediation Plan:

SOLVED: The issue has been remediated in commit ba02627. A can_stake variable has been added and keeps track of whether the initialization parameter vesting denom did match the bonding denom during the contract's instantiation. Instead of calling query_bonded_denom function in execute_delegate, execute_redelegate and execute_undelegate, the contract now only verifies that can_stake is true which is faster and costs less gas.

3.6 (HAL-06) IMPROPER ERROR HANDLING - INFORMATIONAL

Description:

The vesting contract includes a clawback function to handle situations where a clawback is called multiple times. However, the current implementation handles this scenario with the macro “panic!”, which is generally discouraged as an error handling logic because it is a “last resort” instruction for unpredictable situations.

While using the “panic!” macro is generally considered bad practice, the clawback function includes a check to prevent multiple clawbacks. To ensure the contract is as robust and user-friendly as possible, it is still recommended to implement a more robust error handling strategy.

Code Location:

Listing 6: balance.rs (Line 96)

```
94      pub fn clawback(&mut self, time: Timestamp) -> Uint128 {
95          if self.clawed_back {
96              panic!("clawback twice")
97          }
98          self.clawed_back = true;
99          self.funder_clawback = self.initial_amount - self.unlocked
↳ (time);
100         self.funder_withdraw()
101     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended that a more robust error handling strategy be implemented in the vesting contract's clawback function. This could entail taking a different approach when a clawback is called multiple times, such as gracefully reverting the transaction or returning a clear and helpful error message to the caller.

Remediation Plan:

ACKNOWLEDGED: The issue has been acknowledged by the Archway team, stating the following:

> The panic! macro is in a path that cannot be reached.

3.7 (HAL-07) MISSING ENFORCEMENT OF OVERFLOW CHECKS SETTING IN CONTRACTS WORKSPACE - INFORMATIONAL

Description:

The `overflow-checks` setting in the Cargo.toml file controls the `-C overflow-checks` flag, which determines the behavior of runtime integer overflow in a contract. When overflow-checks are enabled, an overflow panic occurs, protecting the contract from this attack vector and ensuring a safer outcome than unexpected numeric values from arithmetic operations.

While the vesting contract uses default CosmWasm types (such as Uint128) that automatically check for and prevent overflows, it is still recommended to enforce the `overflow-checks` setting at the workspace level. This ensures that the contract is as robust and secure as possible, and it aids in preventing potential vulnerabilities from being introduced via code changes or updates.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended that the overflow-checks setting in the vesting contract be enforced at the workspace level. That can be achieved by setting `overflow-checks` to `true` in the `[profile.release]` section of the workspace's `Cargo.toml`.

Remediation plan:

SOLVED: The issue has been remediated in commit `0fbfbb9`. The `overflow-checks = true` directive has been added to the `Cargo.toml`.

THANK YOU FOR CHOOSING

// HALBORN