Audit Report

DAO DAO Vesting and Payroll Factory

v1.0

March 16, 2023

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by In With The New to perform a security audit of the DAO DAO Vesting and Payroll Factory CosmWasm smart contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repository:

https://github.com/DA0-DA0/dao-contracts

Commit hash: `0b5cae57fecbbadb1045f3dc2bb4ad4fe5a98ee8`

The scope of this audit was limited to:

- `contracts/external/cw-vesting`
- `contracts/external/cw-payroll-factory`
- `packages/cw-wormhole`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The submitted code implements the following functionality:

- The `cw-payroll-factory` contract serves as a factory that instantiates the `cw-vesting` contract.
- The `cw-vesting` contract enables the creation of native and CW20 token streams, allowing payments to be vested continuously over time.
- The `cw-wormhole` package implements a CosmWasm KV store that allows setting values from the past.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium-High	The `cw-wormhole` package contains particularly complex code.
Code readability and clarity	High	Most functions are well documented with clear and concise comments.
Level of documentation	High	Detailed documentation was provided at https://gist.github.com/0xekez/15fab6436ed593cbd59f0bdf7ecf1f61, https://github.com/DA0-DA0/dao-contracts/blob/main/contracts/external/cw-vesting/SECURITY.md and in README files in the repository.
Test coverage	Medium-High	`cw-wormhole`: 82.3% `cw-vesting`: 92.2% `cw-payroll-factory`: 87.6%

Summary of Findings

No	Description	Severity	Status
1	Undelegations will fail when redelegating to a new validator	Critical	Resolved
2	Misconfiguring the total vested amount to be lower than the sent CW20 amount would cause a loss of funds	Major	Resolved
3	Misconfiguring start time to be in the past might cause undesired payouts	Minor	Resolved
4	Vesting duration parameter is not validated	Minor	Resolved
5	Undelegation does not emit custom events	Informational	Resolved
6	Values held in `StakeTracker` cannot be queried	Informational	Resolved
7	Potential misleading error in `ReceiveMsg::Fund`	Informational	Acknowledged

Detailed Findings

1. Undelegations will fail when redelegating to a new validator

Severity: Critical

In `contracts/external/cw-vesting/src/stake_tracker.rs:71-74`, the cardinality is not increased when redelegating to a new validator. Consequently, undelegations will fail in line `100` as the cardinality tries to decrement a zero-value.

Although the vesting recipient can prevent this issue by redelegating back to the source validator, it is irrecoverable if the vesting is canceled. In that case, the owner can no longer withdraw the delegated funds.

Please see the `test_redelegate_should_increase_cardinality` test case to reproduce the issue.

Recommendation

We recommend increasing the cardinality if the destination validator is a new validator during the redelegation process.

Status: Resolved

The issue is resolved in `29b9340ee0c0ce5a4ec249151245d1bea698d021`.

2. Misconfiguring the total vested amount to be lower than the sent CW20 amount would cause a loss of funds

Severity: Major

In `contracts/external/cw-vesting/src/contract.rs:38`, the total amount of tokens to be vested depends on the `PayrollInstantiateMsg.total` amount (see `contracts/external/cw-payroll-factory/src/contract.rs:87`). If the total amount to vest is lower than the sent CW20 amount, the excess tokens will be left in the `cw-payroll-factory` contract.

Suppose a scenario where the `cw-payroll-factory` does not have any owner. An attacker can steal the excess tokens by calling `ReceiveMsg::InstantiatePayrollContract` with a higher `PayrollInstantiateMsg.total` amount, causing the `cw-payroll-factory` contract to send the excess funds to the newly instantiated `cw-vesting` contract.

Please see the `test_inconsistent_cw20_amount` test case to reproduce the issue.

Recommendation

We recommend validating the value of `instantiate_msg.total` to be equal to `receive_msg.amount` in `contracts/external/cw-payroll-factory/src/contract.rs:66`.

Status: Resolved

The issue is resolved in fadf2e4a2bbcc6363ca06c1e6e7bb0c745f00f99.

3. Misconfiguring start time to be in the past might cause undesired payouts

Severity: Minor

In `contracts/external/cw-vesting/src/contract.rs:34`, the `start_time` of the vesting can be set to a past value, which might lead to undesired payouts.

We classify this issue as minor since only the owner can cause it.

Recommendation

We recommend validating that the vesting starts at the current or a future timestamp.

Status: Resolved

The issue is resolved in 5ba5c7634e9dd1d6849449b385c23569e82dccc3.

4. Vesting duration parameter is not validated

Severity: Minor

In `cw-vesting/src/contract.rs:41`, the `duration_seconds` parameter is initialized with a non-validated value. This results in the possibility of instant vesting, releasing funds immediately upon reaching `start_time`, which may not be intended.

The value of `duration_seconds` is handled using the `SaturatingLinear` variant of the `Schedule` type. Using it with a `start_time` in the future and `duration_seconds` equal to `0` renders it equivalent to the `PiecewiseLinear` schedule consisting of a single step releasing the whole amount.

We classify this issue as minor since only the contract instantiator can cause it.

Recommendation

We recommend implementing minimum and maximum vesting duration as configuration values of the payroll factory.

Status: Resolved

The issue is resolved in 5ba5c7634e9dd1d6849449b385c23569e82dccc3.

5. Undelegation does not emit custom events

Severity: Informational

During undelegation in the `execute_undelegate` function in `contracts/external/cw-vesting/src/contract.rs:366`, no events or attributes are emitted upon successful execution. This is inconsistent with `execute_delegate` and `execute_redelegate`, where attributes are emitted in lines `259` and `324`, respectively.

Recommendation

We recommend emitting relevant attributes in the `execute_undelegate` function.

Status: Resolved

The issue is resolved in d0f60b4e0dbea06f05d9e7ed8b4749ec8ecb9d39.

6. Values held in `StakeTracker` cannot be queried

Severity: Informational

There is a certain amount of code implementing tracking of delegations in the `cw-vesting` smart contract. The structure `StakeTracker`, defined in `stake_tracker.rs`, contains the fields `total_staked`, `validators`, and `cardinality`. They all reflect the chronology of staked funds by the validator and account unbonding times.

However, call graph analysis reveals that only `total_staked` affects the outcome of transaction processing. In contrast, `validators` and `cardinality` are never read outside tests in `stake_tracker_tests.rs`. `StakeTracker` does, however, update them, e.g. in the `on_bonded_slash` function in `stake_tracker.rs:113` with a clever technique of reconstructing proper delegated amounts after slashing events.

The concern is that fields `validators` and `cardinality` require computation and on-chain space while not being utilized in any way except testing.

Recommendation

We recommend implementing queries for the `cardinality` and `validators` fields (`stake_tracker.rs:18` and `21`), which would benefit the user experience. Alternatively, these values could be computed and stored off-chain, which would be more efficient.

Status: Resolved

The issue is resolved in [8d7ad103c9c4bc3a2477a4d4f50532ef573f4d83](8d7ad103c9c4bc3a2477a4d4f50532ef573f4d83).

7. Potential misleading error in `ReceiveMsg::Fund`

Severity: Informational

In `cw-vesting/src/contract.rs:153`, the error `ContractError::Funded` is thrown if the status of the vesting is not `Unfunded`. However, the contract also can be in the `Canceled` state.

Recommendation

We recommend returning a more accurate or more generic error, for example, `ContractError::NotFundable`.

Status: Acknowledged

Appendix A: Test Cases

1. **Test case for "Undelegations will fail when redelegating to a new validator"**

The test case should pass if the issue is patched.

```
#[test]
fn test_redelegate_should_increase_cardinality() {
    let storage = &mut mock_dependencies().storage;
    let time = Timestamp::default();

    let init = VestInit {
        total: Uint128::new(100),
        schedule: Schedule::SaturatingLinear,
        start_time: time,
        duration_seconds: 100,
        denom: CheckedDenom::Native("ujuno".to_string()),
        recipient: Addr::unchecked("recv"),
        title: "t".to_string(),
        description: Some("d".to_string()),
    };
    let payment = Payment::new("vesting", "staked", "validator", "cardinality");

    payment.initialize(storage, init).unwrap();
    payment.set_funded(storage).unwrap();

    let src = String::from("validator1");
    let dst = String::from("validator2");
    let amount = Uint128::new(10);
    let ubs : u64 = 25;

    // delegate twice amount to validator 1
    payment
        .on_delegate(storage, time, src.clone(), amount + amount)
        .unwrap();

    // relegate to validator 2
    payment.on_redelegate(storage, time, src.clone(), dst.clone(),
amount).unwrap();

    // undelegate for validator 1
    payment
        .on_undelegate(storage, time, src.clone(), amount, ubs)
        .unwrap();

    // undelegate for validator 2
    payment
```

```
        .on_undelegate(storage, time, dst.clone(), amount, ubs)
        .unwrap(); // err due to attempt to subtract cardinality by 0-1
}
```

2. Test case for "Misconfiguring the total vested amount to be lower than the sent CW20 amount would cause a loss of funds"

The test case should fail if the issue is patched.

```
#[test]
pub fn test_inconsistent_cw20_amount() {
    let mut app = App::default();
    let code_id = app.store_code(factory_contract());
    let cw20_code_id = app.store_code(cw20_contract());
    let cw_vesting_code_id = app.store_code(cw_vesting_contract());

    // Instantiate cw20 contract with balances for Alice
    let cw20_addr = app
        .instantiate_contract(
            cw20_code_id,
            Addr::unchecked(ALICE),
            &cw20_base::msg::InstantiateMsg {
                name: "cw20 token".to_string(),
                symbol: "cwtwenty".to_string(),
                decimals: 6,
                initial_balances: vec![Cw20Coin {
                    address: ALICE.to_string(),
                    amount: Uint128::new(INITIAL_BALANCE),
                }],
                mint: None,
                marketing: None,
            },
            &[],
            "cw20-base",
            None,
        )
        .unwrap();

    let instantiate = InstantiateMsg {
        owner: Some(ALICE.to_string()),
        vesting_code_id: cw_vesting_code_id,
    };
    let factory_addr = app
        .instantiate_contract(
            code_id,
            Addr::unchecked("CREATOR"),
            &instantiate,
            &[],
            "cw-admin-factory",
            None,
        )
        .unwrap();

    // Mint alice native tokens
```

```
    app.sudo(SudoMsg::Bank({
        BankSudo::Mint {
            to_address: ALICE.to_string(),
            amount: coins(INITIAL_BALANCE, NATIVE_DENOM),
        }
    }))
    .unwrap();

    let amount = Uint128::new(1000000);
    let unchecked_denom = UncheckedDenom::Cw20(cw20_addr.to_string());

    let instantiate_payroll_msg = PayrollInstantiateMsg {
        owner: Some(ALICE.to_string()),
        recipient: BOB.to_string(),
        title: "title".to_string(),
        description: Some("desc".to_string()),
        total: amount - Uint128::new(1), // lesser amount than sent
        denom: unchecked_denom,
        schedule: Schedule::SaturatingLinear,
        vesting_duration_seconds: 200,
        unbonding_duration_seconds: 2592000, // 30 days
        start_time: None,
    };

    let res = app
        .execute_contract(
            Addr::unchecked(ALICE),
            cw20_addr,
            &Cw20ExecuteMsg::Send {
                contract: factory_addr.to_string(),
                amount: amount,
                msg: to_binary(&ReceiveMsg::InstantiatePayrollContract {
                    instantiate_msg: instantiate_payroll_msg,
                    label: "Payroll".to_string(),
                })
                .unwrap(),
            },
            &coins(amount.into(), NATIVE_DENOM),
        )
        .unwrap();

    // Get the payroll address from the instantiate event
    let instantiate_event = &res.events[4];
    assert_eq!(instantiate_event.ty, "instantiate");
    let cw_vesting_addr = instantiate_event.attributes[0].value.clone();

    // Check that the vesting payment contract is active
    let vp: Vest = app
        .wrap()
        .query_wasm_smart(cw_vesting_addr, &PayrollQueryMsg::Info {})
```

```
        .unwrap();
    assert_eq!(vp.status, Status::Funded);
}
```