ackee blockchain

Gnosis SafeDAO

May 27, 2022

by Ackee Blockchain

Contents

1. Document Revisions

1.0	Final report	May 27, 2022
1.1	Fix review	Jun 23, 2022

2. Overview

This report has been prepared for Gnosis to discover issues, vulnerabilities, and gas optimizations in the source code of the SafeDAO's VestingPool contract.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specialized in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run a free certification course Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, Rockaway Blockchain Fund.

2.2. Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Slither is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzzy testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests.

2.3. Review team

Member's Name	Position
Jan Šmolík	Lead Auditor
Štěpán Šonský	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.4. Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Gnosis engaged Ackee Blockchain to conduct a security review of the SafeDAO's VestingPool contract with a total time donation of two engineering days. The review took place between May 23 and 27, 2022.

Even though the scope was only the `VestingPool` contract, we looked at some other contracts (`SafeToken` and `TokenRescuer`) to give us a better picture.

The commit we worked on is the following:

- `a50728c28dd510ceae1b65bb526db98148a76f31`

In general, we can state that

- the code quality is very high & the code is well commented,
- the documentation is sufficient,
- the client's test coverage is nearly `100%`, and
- the security review did not result in any serious findings.

The review resulted in four findings.

We would like to hear your feedback regarding the medium severity issue, the rest is just informational.

Update June 23, 2022: Gnosis provided an updated codebase that addresses the issues from this report. See Appendix B for a detailed discussion of the exact status of each issue.

4. System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

4.1. Overview

Gnosis Safe is a multi-signature wallet platform to store digital assets on Ethereum for companies, holders, developers, funds, DAOs, and investors. It is the backbone of many big Ethereum Defi, DAO, and NFT projects.

Recently, they are establishing SafeDAO and launching a SAFE Token to give the community the governance over the Safe ecosystem.

The VestingPool contract will be used to distribute SAFE tokens over time to various accounts.

4.2. Contracts

Contracts we find important for better understanding are described in the following section.

SafeToken.sol

The Safe Token is an ownable ERC20 token with the initial supply of one billion tokens that will be minted to the owner's address.

- Ownership can be transferred and revoked by the owner.
- The token is initially paused (not transferrable by anyone except for the owner). To make it transferrable, the owner has to call the `unpause` function. Once unpaused, there is no way to pause it again.
- The initial supply is one billion tokens and there are no `mint` and `burn`

functions. Therefore, the total supply will always stay at one billion.

- It is also possible to recover other ERC20 tokens that have been sent to the token contract's address by mistake using `rescueTokens` of the TokenRescuer contract.

TokenRescuer.sol

The TokenRescuer is a base contract for the SafeToken. The owner of the TokenRescuer can call the `rescueTokens` function to transfer ERC20 tokens from the contract's address to a specified address. Therefore, if some other ERC20 tokens are mistakenly sent to the SafeToken contract's address, it is possible to rescue them.

VestingPool.sol

The VestingPool is used to distribute SAFE tokens over time to different accounts. The Pool Manager can use the pool to create vestings (i.e., dedicate a specific amount of tokens to a specified address to be distributed over specified time) and pause, unpause and cancel vestings that are controlled by the manager (where `managed == true`).

When creating a vesting, the manager has to specify

- the address that owns the vesting and can claim vested tokens,
- the curve to calculate the vested tokens,
- whether or not the vesting is managed (if managed, pause, unpause and cancel can be called),
- the duration of the vesting in weeks,
- the start date of the vesting, and
- the amount of tokens that will be vested.

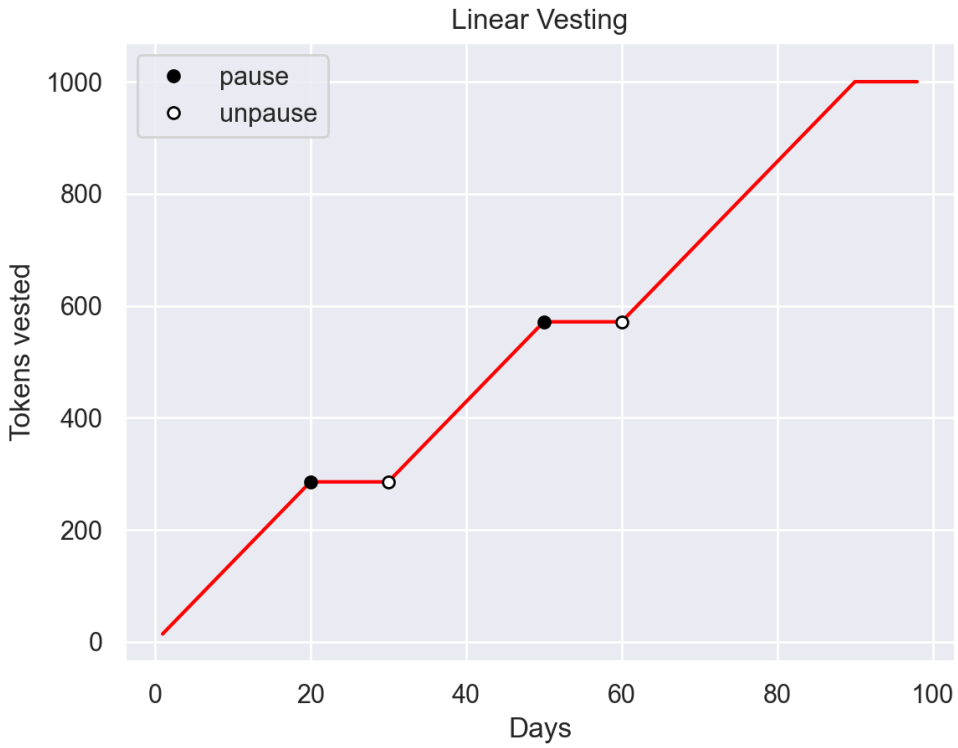
The curve type can be either **linear** or **exponential**:

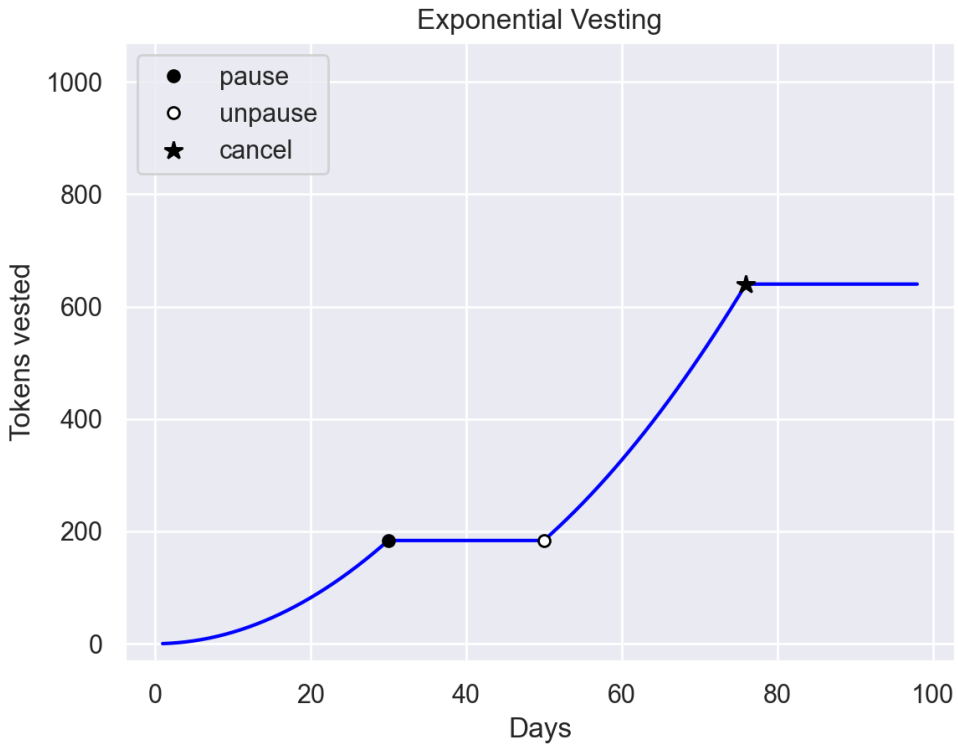
- For linear vesting the following formula is used: `tokens_vested = tokens_total * duration_elapsed / duration_total`, and
- for exponential vesting the following formula is used: `tokens_vested = tokens_total * duration_elapsed^2 / duration_total^2`.

A managed vesting can be **paused** or **cancelled** by the manager.

- When a vesting is paused no additional tokens are vested during that time. Users can claim vested tokens that were vested until it was paused. A paused vesting can be unpaused.
- When a vesting is cancelled it is also marked as paused, but it is not possible to unpause the vesting. Users can claim vested tokens that were vested until it was cancelled, and the unused tokens in the vesting are made available for new vestings.

The following figures show how the vesting behaves and tokens are vested. We obtained the data shown from the actual blockchain. In both cases, 1000 tokens are vested for a period of ten weeks. Sometimes, the manager pauses the vesting for some time, and in the second figure he cancels the vesting.


Linear Vesting
pause
unpause
Tokens vested
0
200
400
600
800
1000
0
20
40
60
80
100
Days


Exponential Vesting
pause
unpause
cancel
Tokens vested
0
200
400
600
800
1000
0
20
40
60
80
100
Days

4.3. Actors

This part describes actors of the system, their roles, and permissions.

Pool Manager

Pool Manager is set in the constructor of the VestingPool and the role can not be transferred nor renounced. Pool Manager

- can add vestings via `addVesting`,
- can pause and unpause `managed` vestings via `pauseVesting` and `unpauseVesting`, and
- can cancel `managed` vestings via `cancelVesting`.

Safe Token Owner

Safe Token owner is the owner of the SafeToken contract. He

- receives the initial supply of one billion,
- can transfer and renounce the ownership,
- can unpause the contract to make it transferrable, and
- can rescue tokens via TokenRescuer.

4.4. Trust model

There are no trust issues from the security point of view.

Users have to trust the Safe token deployer to deploy the token flawlessly and the Pool Manager to set up the vestings correctly (according to this).

5. Vulnerabilities risk methodology

Each finding contains an *Impact* and *Likelihood* ratings.

If we have found a scenario in which the issue is exploitable, it will be assigned an impact of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Informational*.

Low to *High* impact issues also have a *Likelihood* which measures the probability of exploitability during runtime.

5.1. Finding classification

The full definitions are as follows:

Impact

High

Code that activates the issue will lead to undefined or catastrophic consequences for the system.

Medium

Code that activates the issue will result in consequences of serious substance.

Low

Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.

Warning

The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as "Warning" or higher, based on our best estimate of whether it is currently exploitable.

Informational

The issue is on the border-line between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

High

The issue is exploitable by virtually anyone under virtually any circumstance.

Medium

Exploiting the issue currently requires non-trivial preconditions.

Low

Exploiting the issue requires strict preconditions.

6. Findings

This section contains the list of discovered findings. Unless overriden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*, and
- a *Recommendation*

Many times, there might be multiple ways to solve or alleviate the issue, with varying requirements in terms of the necessary changes to the codebase. In that case, we will try to enumerate them all, making clear which solve the underlying issue better (albeit possibly only with architectural changes) than others.

Summary of Findings

	Type	Impact	Likelihood
M1: Pool Manager role	Access controls	High	Low
I1: Public functions	Gas optimization	Info	N/A
I2: Typos in the comments	Code readability	Info	N/A
I3: Possible gas optimization in `claimVestedTokens()`	Gas optimization	Info	N/A

Table 1. Table of Findings

M1: Pool Manager role

Impact:	High	Likelihood:	Low
Target:	VestingPool.sol	Type:	Access controls

Description

In the VestingPool, the role Pool Manager has the authority over the listed functions.

- `addVesting`
- `cancelVesting`
- `pauseVesting`
- `unpauseVesting`

Any compromise to the `Pool Manager` account may allow the attacker to take advantage of it and steal the funds.

Exploit scenario

The vestings have been set for a long time. However, a hacker gains access to the `Pool Manager` key.

He cancels all `managed` vestings, creates himself a vesting with zero `durationWeeks` and claims the tokens.

Recommendation

We advise the client to carefully manage the role account's private key to avoid any potential risks of being hacked. Consider using a multi-signature wallet.

Go back to Findings Summary

I1: Public functions

Impact:	Informational	Likelihood:	N/A
Target:	VestingPool.sol	Type:	Gas optimization

Description

The following functions are declared public even though they are not called internally anywhere.

- `addVesting`
- `claimVestedTokens`
- `cancelVesting`
- `pauseVesting`
- `unpauseVesting`

Recommendation

If functions are not called internally, they should be declared external. It helps gas optimization because function arguments do not have to be copied into memory.

Go back to Findings Summary

I2: Typos in the comments

Impact:	Informational	Likelihood:	N/A
Target:	VestingPool.sol	Type:	Code readability

Description

There are a few errors in the comments.

On #L15-L16:

```
// amountClaimed should always be equal or less than amount
// pausingDate should always be equal or greater than startDate
```

`equal` → `equal to`

On #L132:

```
/// @param tokensToClaim Amount of tokens to claim in atoms or max uint256
to claim all available
```

`max uint256` → `max uint128`

On #L196:

```
/// @dev Only vestings that have not been cancelled can beunpaused
```

`beunpaused` → `be unpaused`

On #L272:

```
/// @param elapsedTime Time that has ellapsed for the vesting
```

`ellapsed` → `elapsed`

Recommendation

Fix the typos.

Go back to Findings Summary

I3: Possible gas optimization in `claimVestedTokens()`

Impact:	Informational	Likelihood:	N/A
Target:	VestingPool.sol	Type:	Gas optimization

Description

`claimVestedTokens` is a function that will be called by the users (owners of the vestings), therefore it makes sense to have this function consume as little gas as possible.

We have found an adjustment that reduces gas consumption by approximately 9000 gas per `claimVestedTokens` transaction, however, it also reduces consistency with other functions.

Listing 1. This is the function as it is in the `VestingPool.sol`*:*

```
function claimVestedTokens(
        bytes32 vestingId,
        address beneficiary,
        uint128 tokensToClaim
    ) public {
        require(beneficiary != address(0), "Cannot claim to 0-address");
        Vesting memory vesting = vestings[vestingId];
        require(vesting.account == msg.sender, "Can only be claimed by
vesting owner");
        // Calculate how many tokens can be claimed
        uint128 availableClaim = _calculateVestedAmount(vesting) - vesting
.amountClaimed;
        // If max uint128 is used, claim all available tokens.
        uint128 claimAmount = tokensToClaim == type(uint128).max ?
availableClaim : tokensToClaim;
        require(claimAmount <= availableClaim, "Trying to claim too many
tokens");
        // Adjust how many tokens are locked in vesting
        totalTokensInVesting -= claimAmount;
        vesting.amountClaimed += claimAmount;
        vestings[vestingId] = vesting;
        require(IERC20(token).transfer(beneficiary, claimAmount), "Token
transfer failed");
        emit ClaimedVesting(vestingId, vesting.account, beneficiary);
    }
```

Recommendation

Consider the following adjustment for gas optimization. The adjustment is

- changing `Vesting memory vesting = vestings[vestingId]` to `Vesting storage vesting = vestings[vestingId]`
- line `vestings[vestingId] = vesting` removed

Go back to Findings Summary

7. Appendix A

7.1. How to cite

Please cite this document as:

Ackee Blockchain, Gnosis SafeDAO, May 27, 2022.

If an individual issue is referenced, please use the following identifier:

`ABCH-{project_identifer}-{finding_id}`,

where `{project_identifier}` for this project is `GNOSIS-SAFEDAO` and `{finding_id}` is the id which can be found in Summary of Findings. For example, to cite M1 issue, we would use `ABCH-GNOSIS-SAFEDAO-M1`.

8. Appendix B: Fix Review

On June 23, 2022, Ackee Blockchain reviewed Gnosis Safe's fixes for the issues identified in this report. The following table summarizes the fix review.

Fix log

Id		Type	Impact	Likelihood	Status
M1F	M1F: Pool Manager role	Access controls	High	Low	**Not an issue**
I1F	I1F: Public functions	Gas optimization	Info	N/A	**Not an issue**
I2F	I2F: Typos in the comments	Code readability	Info	N/A	**Fixed**
I3F	I3F: Possible gas optimization in `claimVestedTokens()`	Gas optimization	Info	N/A	**Fixed**

Table 2. Table of fixes

Refactoring

Gnosis also did a minor refactoring change. The code logic without the `transfer()` from `claimVestedTokens()` was extracted into a new internal method `updateClaimedTokens()`. We have found no issues here.

M1F: Pool Manager role

Impact:	High	Likelihood:	Low
Target:	M1: Pool Manager role	Type:	Access controls

Description

Gnosis informed us that they will be using a multi-signature wallet for the management of the Pool Manager account.

Therefore, the issue, as described in M1, is not an issue at all.

Go back to Fix log

I1F: Public functions

Impact:	Informational	Likelihood:	N/A
Target:	I1: Public functions	Type:	Gas optimization

Description

Gnosis gave the following response, which is absolutely valid:

”The methods have not been changed to `external` from `public` as recommended in the audit report, as no gas improvement was observed after the storage changes. Therefore to keep the flexibility when implementing the Aidrop contract the methods will be kept `public`.”

Go back to Fix log

I2F: Typos in the comments

Impact:	Informational	Likelihood:	N/A
Target:	I2: Typos in the comments	Type:	Code readability

Description

Every mistake in the comments that we have found and discussed in I2 was correctly fixed.

Go back to Fix log

I3F: Possible gas optimization in `claimVestedTokens()`

Impact:	Informational	Likelihood:	N/A
Target:	I3: Possible gas optimization in `claimVestedTokens()`	Type:	Gas optimization

Description

We recommended adjusting the storage access in `claimVestedTokens()` to lower the gas consumption.

Firstly, the logic from `claimVestedTokens()` was moved to a new function `updateVestedTokens()` (see this).

Secondly, the storage access was adjusted in all methods, not just in `updateVestedTokens()`, as it also improved the gas consumption.

The following functions now work directly with the vesting in the storage:

- `updateVestedTokens()`
- `cancelVesting()`
- `pauseVesting()`
- `unpauseVesting()`
- `calculateVestedAmount()`

Go back to Fix log

ackee blockchain

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/wpM77gR7en