Audit Report

Sigma

v1.0

May 3, 2022

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 Sigma to perform a security audit of Sigma Protocol.

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/Sigma-Protocol/sigma-contracts

Commit hash: `d9d0f840f45723ee851162661ca84458f58816dd`

The governance contract in `contracts/sigma_gov` has been excluded from the audit scope.

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

Sigma is a DeFi protocol that enables the creation and usage of collateralized options on the Terra blockchain. Sigma allows users to mint new options by providing the necessary underlying collateral. In addition, the protocol implements vault infrastructure to solve the problem of option liquidity fragmentation arising from the multiple strike price and expiry dates for a given underlying asset.

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 improve 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 a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Attacker can extract all Sigma tokens from the factory contract	Critical	Resolved
2	Logic error prevents the redemption of aUST	Critical	Resolved
3	`update_config` will overwrite the value of astroport_router	Major	Resolved
4	Settlement of stage 4 might run out of gas if too many depositing addresses exist	Major	Resolved
5	Depositors may receive different amount than original deposit	Major	Resolved
6	Updatable option configuration parameters may impact the exercisability of options that are in the money	Major	Resolved
7	Excess funds sent while exercising via vault contract will be lost	Major	Resolved
8	Missing tax deductions	Minor	Acknowledged
9	`treasury_take`, `percent` and `fee` should be validated to ensure they fall within an expected range of values	Minor	Resolved
10	Misconfigured distribution schedule config can cause panics in LP staking contract	Minor	Resolved
11	Subtraction may cause wrap around underflow	Minor	Resolved
12	Extra funds sent may be lost	Minor	Resolved
13	`distribute` function allows collector to send any asset	Minor	Acknowledged
14	`CREATOR_TO_UNLOCK` may cause an out of gas error	Minor	Resolved
15	Iterations over several maps may cause out of gas errors	Minor	Acknowledged
16	Oracles for low liquidity assets might be prone to manipulation.	Informational	Acknowledged

17	Canonical address transformations are inefficient	Informational	Acknowledged
18	Unnecessary use of less than or equal to	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium-High	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium	-
Test coverage	Medium-High	-

Detailed Findings

1. Attacker can extract all Sigma tokens from the factory contract

Severity: Critical

The `claim_deposit` function in `contracts/sigma_factory/src/contract.rs:363` allows an attacker to extract all Sigma tokens from the factory contract. Currently, any address with an unlocked deposit can perform this attack. There is no step in the function where entries are removed from `CREATOR_TO_UNLOCK` and the deposit redemption is not recorded anywhere.

The following is a potential attack scenario:

1. The attacker creates an option.
2. The attacker waits until the deposit unlock has been reached.
3. The attacker can repeatedly call `claim_deposit` and will continue to receive `config.deposit_amount` each transaction until the factory contract has no Sigma tokens left.

Recommendation

We recommend removing the `CREATOR_TO_UNLOCK` map entries after they are utilized in `claim_deposit`. In addition, we recommend storing the amount the user deposited and only returning it rather than returning `config.deposit_amount`. This amount can be decremented or set to zero after the user has claimed their deposit.

Status: Resolved

2. Logic error prevents the redemption of aUST

Severity: Critical

The `redeem_aust_if_necessary` function will never be able to redeem the option contract's `aUST` because of an error in the logic in `contracts/sigma_option/src/contract.rs:431`. Currently, the function is only sending an Anchor `RedeemStable` message if the option's `aUST` is equal to zero. This logic should be changed to perform the `RedeemStable` message if the `aUST` balance is not equal to zero. The result of this will be that the option will not be able to redeem its `aUST`.

This same logic error also occurs in `contracts/sigma_vault/src/contract.rs:850`. `RedeemStable` is only performed if the contract's `aUST` balance is zero,

Recommendation

We recommend fixing the condition in `contracts/sigma_option/src/contract.rs:431` and `contracts/sigma_vault/src/contract.rs:850` to perform `RedeemStable` if the contract has a non-zero balance of `aUST`.

Status: Resolved

3. `update_config` will overwrite the value of `astroport_router`

Severity: Major

The `update_config` function in `contracts/sigma_collector/src/contract.rs:179-182` will overwrite the value of `config.astroport_router`. This will have two effects, it will update the value of `config.astroport_router` to the wrong value and it also blocks the ability to update the value of `config.sig_token`.

Recommendation

We recommend fixing the `update_config` function in `contracts/sigma_collector/src/contract.rs:181` to update the value of `config.sig_token` rather than `config.astroport_router`.

Status: Resolved

4. Settlement of stage 4 might run out of gas if too many depositing addresses exist

Severity: Major

In `contracts/sigma_vault/src/contract.rs:996-1017`, iterations over the `deposits` may run out of gas if the vector has many entries. Many entries might occur if an attacker would create many addresses to prevent the settlement of puts. This would result in the vault no longer being able to settle puts. There is currently no way to recover from this issue.

Recommendation

We recommend requiring a minimum deposit amount as a partial fix and monitoring the number of depositor addresses per vault. A minimum amount would increase the costs for an attacker to deliberately block the settlement of puts. Nevertheless, a rapid growth in user accounts could still disable the settlement functionality. As a more permanent solution, we recommend creating a depositor limit to ensure this issue is never encountered.

Status: Resolved

5. Depositors may receive different amount than original deposit

Severity: Major

The `claim_deposit` function in `contracts/sigma_factory/src/contract.rs:383` allows the depositor to claim their deposit after it has reached its `unlock_timestamp`. The deposit is sent back in a `CW20` transfer message in lines `379-386`. This message amount sends the current `config.deposit_amount` (an updatable value) to the depositor. This is problematic if the `deposit_amount` at the time that the option was created is different from the `deposit_amount` at the time of the claim.

The value of `deposit_amount` is updatable in line `323` by the gov contract or the owner. If this value is changed in the period of time after the option has been created but before the deposit is claimed it will cause the user to receive a different amount than they originally deposited when they created the option.

Recommendation

We recommend recording the amount that is deposited when an option is created in `create_option`, this can be accomplished in a similar fashion to how the option creator info is stored in `CREATOR_TO_UNLOCK`. Then when a deposit is being claimed, check that the deposited amount is equal to the `config.deposit_amount`, or just simply return the saved deposit amount rather than `config.deposit_amount`.

Status: Resolved

6. Updatable option configuration parameters may impact the exercisability of options that are in the money

Severity: Major

The `update_config` function in `contracts/sigma_option/src/contract.rs:122` allows `config.owner` to update the option's parameters. There is currently no safeguard to ensure that these parameters cannot be updated on live options.

Allowing for these parameters to be updated is very problematic. For example, if the owner were to change `config.call_or_put`, it would affect the exercisability of previously minted shares. This could be done accidentally or intentionally. In a situation where the owner is compromised, the attacker could change parameters to ensure that no users can exercise their options.

There does not appear to be a reason why the option parameters would need to be updated once they are initially set, and removing this functionality completely would eliminate this possibility completely.

The following parameters should not be updatable after they have been originally set:

- `underlying_asset`
- `denomination`
- `call_or_put`
- `american_or_european`
- `strike_price`
- `expiration_time`
- `exercisable_token`
- `obligation_token`

Recommendation

We recommend removing the functionality to update the parameters mentioned above once they have been initially configured if there is no valid use case for updating them. This can be accomplished by only allowing updates to the values that are `None`. This will allow for the initial configuration updates but will disallow changes after the values have already been set. If there is a case for updating specific parameters, a validation can be implemented to ensure that the values may only be updated when no options have been minted.

Status: Resolved

7. Excess funds sent while exercising via vault contract will be lost

Severity: Major

In both `contracts/sigma_vault/src/contract.rs:570` and `582`, if the user sends more funds than the exercisable units balance of the vault then they will effectively remain in the contract and not be returned back to the sender. This may result in the caller receiving less value than they have sent to the vault contract.

In lines `570` and `582` there are validations to ensure that the `units_sent` or `funds_sent` are less than `exercisable_units`, and it will return an error if the funds are insufficient. This check does not account for a situation where the caller sends funds in excess of the amount of `exercisable_units`. After this step, the function uses `exercisable_units` directly in all of the remaining messages and calculations and disregards the actual amount sent by the caller. This will result in the excess funds remaining in the vault contract and the caller will receive less value than they sent.

Recommendation

We recommend adding additional validations to ensure that the funds or units sent by the caller are equal to `exercisable_units`. Rather than returning an error if the amount is less than `exercisable_units`, also return an error if the amount is greater than `exercisable_units`. Another approach could be to return the excess amount back to the caller at the end of the transaction.

Status: Resolved

8. Missing tax deductions

Severity: Minor

While Terra's tax rate has been set to zero, the tax mechanism is still implemented and the rate might be increased again in the future. It is still best practice to include functionality to deduct taxes.

A non-zero rate could be reinstated via a governance proposal due to circumstances where the expected income from the tax rewards increases significantly. In this situation, stablecoin transactions on Terra would expand to a state where a meaningful portion of the staking rewards income is derived from tax rewards rather than the vast majority coming from swap fees.

We consider this to only be a minor issue since the contract owner can recover from tax mismatches by simply sending funds back to the contract. Additionally, the likelihood of the Terra team to increase taxes again is low.

See https://agora.terra.money/t/proposal-to-reduce-the-terra-tax-rate-to-zero/3524 for more detail around the tax rate discussion.

Recommendation

We recommend implementing a tax rate query and deducting taxes from native assets to ensure future compatibility.

Status: Acknowledged

9. `treasury_take`, `percent` and `fee` should be validated to ensure they fall within an expected range of values

Severity: Minor

In multiple occurrences values are set or updated without validations to ensure that they fall within an expected range.

These are:

1. `config.treasury_take` in the `update_config` function and the `instantiate` function in `contracts/sigma_collector/src/contract.rs:155`.

2. `percent` in the `ExecuteMsg::Convert` handling in `contracts/sigma_collector/src/contract.rs:84`.

3. `config.fee` in the `update_config` function and the `instantiate` function in `contracts/sigma_factory/src/contract.rs:338` and `contracts/sigma_factory/src/contract.rs:49`, a misconfiguration could cause an error in `contracts/sigma_option/src/contract.rs:574`.

Recommendation

We recommend validating `config.treasury_take` in both the `update_config` function and in the `instantiate` function; validating `percent` in the `ExecuteMsg::Convert` handling; and validating `config.fee` in both the `update_config` function and in the `instantiate` function to ensure that all values fall within their expected ranges. At a minimum, it is recommended to ensure that the values are greater or equal to zero and less than or equal to one.

Status: Resolved

10. Misconfigured distribution schedule config can cause panics in LP staking contract

Severity: Minor

In `contracts/sigma_lp_staking/src/contract.rs:35` a `distribution_schedule` may be stored with a start time that is greater than the end time. This would cause `contracts/sigma_lp_staking/src/contract.rs:331` to panic.

For informational purposes, we also highlight that it is possible to supply a schedule with gaps and overlaps. Such gaps and overlaps do not cause errors, but might not be intended.

Recommendation

We recommend adding validation to ensure the end of a distribution schedule is greater than the start.

Status: Resolved

11. Subtraction may cause wrap around underflow

Severity: Minor

There are multiple instances of subtractions in `contracts/sigma_xsig_reward/src/xsig.rs` that may cause an underflow if not handled properly. It is best practice to perform checked subtraction to return `None` or to use saturating subtraction to keep the result at the numeric bounds, rather than wrapping around during an underflow.

This is important because Rust behaves differently in debug mode and release mode. In debug mode, Rust adds built-in checks for overflow/underflow and panics when an overflow/underflow occurs at runtime. However, in release or optimization mode, Rust silently ignores this behavior by default and computes two's complement wrapping.

The following are occurrences of unchecked subtraction:

- `contracts/sigma_xsig_reward/src/xsig.rs:30`
- `contracts/sigma_xsig_reward/src/xsig.rs:31`
- `contracts/sigma_xsig_reward/src/xsig.rs:32`
- `contracts/sigma_xsig_reward/src/xsig.rs:68`
- `contracts/sigma_xsig_reward/src/xsig.rs:69`

Recommendation

We recommend enabling overflow-checks for the release profile by adding `overflow-checks = true` under the `[profile.release]` of the workspace's `Cargo.toml`. We also recommend updating the previously mentioned occurrences of subtraction to either be checked subtraction or saturating subtraction.

Status: Resolved

12. Extra funds sent may be lost

Severity: Minor

There are multiple instances in the contracts where native funds are improperly handled which may lead to a loss of funds. The current implementation does not account for a situation where the caller sends multiple funds. `info.funds` is a vector of `Coin`, but the current implementation only accounts for the specific `denom` used by the contract. It is best practice to provide an error if funds are sent that the contract does not handle.

The following are instances of where funds are handled in this manner:

- `contracts/sigma_option/src/contract.rs:319`
- `contracts/sigma_option/src/contract.rs:358`
- `contracts/sigma_vault/src/contract.rs:563`
- `contracts/sigma_vault/src/contract.rs:576`
- `contracts/sigma_vault/src/contract.rs:1309`
- `contracts/sigma_vault/src/contract.rs:1379`
- `contracts/sigma_vault/src/contract.rs:1386`

Recommendation

We recommend updating the previously mentioned occurrences to ensure that the length of `info.funds` is equal to one and returning an error if this condition is not true.

Status: Resolved

13. `distribute` function allows collector to send any asset

Severity: Minor

The `distribute` function in `contracts/sigma_collector/src/contract.rs:205` allows the caller to specify any asset while this function appears to be intended to only handle Sig distributions. This can be problematic if the caller specifies an asset that is not Sig.

Recommendation

We recommend using `config.sig_token` value rather than allowing the caller to specify the asset to distribute.

Status: Acknowledged

14. `CREATOR_TO_UNLOCK` may cause an out of gas error

Severity: Minor

The `CREATOR_TO_UNLOCK` map's elements are never removed in the factory contract. This may cause an out of gas error in `query_deposits`. It is best practice to delete map entries once they have been used and are no longer necessary and also to limit the maximum number of entries to a map.

Recommendation

We recommend deleting the `CREATOR_TO_UNLOCK` map entries after they are utilized in claim_deposit, see finding 1 for more information and also limiting the number of entries to this map.

Status: Resolved

15. Iterations over several maps may cause out of gas errors

Severity: Minor

Iterations over several maps that are used in the factory contract might run out of gas and cause errors. These are:

1. `tickers_by_option_info` in `contracts/sigma_factory/src/contract.rs:447`
2. `expiries_by_option_info` in `contracts/sigma_factory/src/contract.rs:465`
3. `strikes_by_option_info` in `contracts/sigma_factory/src/contract.rs:484`

Out of gas errors can happen because those maps are unbounded. Since the initial amount of options is limited by the maximum expiry of three years it will not happen at launch and therefore this is only a minor issue. However, as options accumulate over the years this might become a serious problem for the protocol.

Recommendation

We recommend deleting expired options after every iteration over the maps.

Status: Acknowledged

16. Oracles for low liquidity assets might be prone to manipulation

Severity: Informational

Traditional options with physical delivery do not require to be in the money to be exercised. In some cases, settling an option out of the money or near the money can be economically beneficial – e.g. if an equivalent order would have a large price impact on the markets.

The use of oracles makes sure that options are not executed at a loss. However, if the oracle price is manipulated the oracle can stop users from executing options in the money.

Recommendation

We recommend considering options that do not use oracles and allow for execution at a loss for less liquid assets in the future.

Status: Acknowledged

17. Canonical address transformations are inefficient

Severity: Informational

Usage of canonical addresses for storage is no longer recommended as a best practice. The reason is that canonical addresses are no longer stored in a canonical format, so the transformation just adds overhead without much benefit. Additionally, the codebase is more complicated with address transformations.

Recommendation

We recommend using the new `Addr` type instead of `CanonicalAddr`.

Status: Acknowledged

18. Unnecessary use of less than or equal to

Severity: Informational

The `receive_cw20` function in `contracts/sigma_xsig_reward/src/contract.rs:126` checks if the amount sent is less than or equal to zero. This can be changed to check only if the amount is equal to zero before returning the error.

Recommendation

We recommend changing the less than or equal to condition to equal to in line `126`.

Status: Resolved