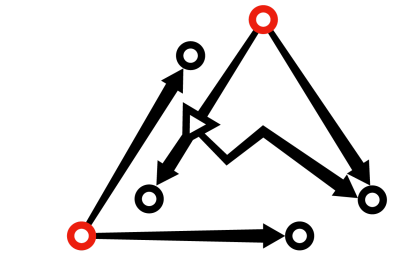

Least Authority

PRIVACY MATTERS

Staking and Claims Registry Smart Contracts

Security Audit Summary

Fractal

Final Audit Summary Report: 8 May 2021

Table of Contents

Overview

Background

Fractal has requested that Least Authority perform a security audit of the Staking and Claims Registry smart contracts.

Fractal is an open source protocol that aims to define a basic standard to exchange user information in a fair and open way while ensuring a high-quality version of the free internet.

Project Dates

- **April 28 - 30**: Code review *(Completed)*
- **May 3**: Delivery of Initial Audit Summary Report *(Completed)*
- **May 7:** Verification *(Completed)*
- **May 8:** Delivery of Final Audit Summary Report *(Completed)*

Review Team

- Nathan Ginnever, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Staking and Claims Registry Smart Contract followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code repositories are considered in-scope for the review:

- Staking smart contract: https://github.com/trustfractal/fractal-contracts/blob/master/contracts/Staking.sol
- Claims Registry smart contract: https://github.com/trustfractal/fractal-contracts/blob/master/contracts/ClaimsRegistry.sol

Specifically, we examined the Git revisions for our initial review:

`cf88a2f82673e7fc9bf08b4937f48ad651fe24ea`

The updates made after our initial review can be found in the following commit:

`8e1b797ef86f9b99952bedca641ed7024a69221d`

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third party code, unless specifically mentioned as in-scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- ERC-780: https://github.com/ethereum/EIPs/issues/780
- ERC-20: https://eips.ethereum.org/EIPS/eip-20
- Fractal - Medium: https://medium.com/frctls

Areas of Concern

Our investigation focused on the following areas:

- Any attack that impacts funds, such as draining or manipulating of funds and/or results in unbalanced trading
- Other ways to exploit contracts
- Interactions with 3rd party contracts
- Anything else as identified during the initial analysis phase

Findings

General Comments

Our team has conducted a manual code review of the Fractal code base, and found it well written and follows the latest and standard Solidity patterns. The code is generally well commented, we recommend additional code comments and documentation that describe the reward calculation functionality.

We concluded that security considerations have been made in the design of this project. We found extensive tests that provide confidence in the correctness of the implementation of the Staking and Claims Registry. Additionally, the contracts are using the latest compiler features including new EVM instructions to prevent over and underflow of integers, removing the necessity of the standard safe math libraries.

We examined the possibility of using tokens that have a missing return value bug and found that these tokens will not be used, making a safe transfer wrapper unnecessary. Our team was not able to identify any severe security issues.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Use Check-Effects-Interactions Pattern	Resolved
Issue B: APY Calculation Fix (Found By Fractal Team)	Resolved
Suggestion 1: Add Additional Documentation	Resolved
Suggestion 2: Consider Using A Domain Separator	Resolved
Suggestion 3: Remove Hardhat Logging Imports	Resolved

Issue A: Use Check-Effects-Interactions Pattern

Location

`master/contracts/Staking.sol#L169-L180`

Synopsis

In the staking contract, the state that updates the balances of removed tokens from staking happens after the transfer function call. If a malicious token is used for staking, it may re-enter and drain the tokens held by the staking contract before the state updates. This state is intended to ensure that the reward calculation of the staker does not over-withdraw.

Impact

If a malicious token is used in the staking protocol, the token funds stored on the staking contract may be drained.

Preconditions

A staking token must be used that has a malicious transfer function designed to specifically drain the stake from this specific staking contract.

Feasibility

This is most certainly not feasible as the tokens that are planned to be used will be audited or known to not contain malicious transfer code.

Remediation

Use the recommended check-effects-interactions pattern from the Solidity documentation on preventing re-entrance attacks.

Status

Resolved. The withdraw function has now been updated and additionally the team applied the check-effects-interactions pattern to the staking function for extra caution.

Issue B: APY Calculation Fix (Found By Fractal Team)

Location

`master/contracts/CappedRewardCalculator.sol`

Synopsis

There was a problem with the `Staking.currentAPY` calculation.

The function currently returns the APY for the following period of time: `(now, now + 1 day)`. It was unclear if this was the intended time frame or if `(now - 1 day, now + 1 day)` was.

Before the start date it should either `return 0` or `revert`. Or it should just return the first day's APY. But it's always starting from `uint currentReward = calculateReward(startDate, current, amount);` which gives you an unpredicted result. Because you expect a 1-day reward, but `startDate - current` will be lower than 1 day and always changing.

Impact

Since this function was added at last minute, and is purely for informative purposes only (to display an estimated APY on the staking UI, with no actual consequences to the staking calculations themselves), its

full behaviour was not properly thought out. Given that this is meant for UI purposes, the scope of this change is outside of the audit itself.

Remediation

After the team's discussing, the desired output is:

Before `startDate`, it should have the same result as if it were called exactly at `startDate`. So rather than returning `0`, we must show what the estimated APY will be once it starts

The range (now, now + 1 day) is correct, as we want to provide an answer to "what will be my APY should I stake right now?"

Status

Resolved. Ensuring that, if using the view before the start, that the current date is moved forward to the start date will give this view function the consistency that they desire.

Suggestions

Suggestion 1: Add Additional Documentation

Location

`master/contracts/Staking/CappedRewardCalculator.sol#L54`

Synopsis

Generally there are good comments throughout the code, however the signature verification library does not contain any code comments. This is a common library and comments are not entirely necessary here and we only suggest adding them for consistency if desired.

The reward calculation however is particularly complicated and does not have many accompanying comments. There is a spreadsheet of expected values provided.

Mitigation

We suggest that there be additional documentation provided that describes the staking reward calculation in better detail beyond the spreadsheet of expected values.

Status

Resolved. Additional documentation has been provided to the reward calculation contract.

Suggestion 2: Consider Domain Separation

Location

`master/contracts/ClaimsRegistry.sol#L49`

Synopsis

While EIP-712 is not recommended in this case, given that the signed data is typed and human readable off-chain, and contains the standard prefix that prevents signed messages from executing transactions, It is possible however for another organization to use this code to create a registry whereby the signatures that are valid for fractal will also be valid for their project. This is only possible if the claim hashes and signatures on those hashes are able to be stored in the new registry, which may not be the case. This

domain separation does not directly affect this project unless there is a migration to a new version where it is desired to not allow old claims.

Mitigation

Ensuring that the address of the “verifying contract”, in this case the claims registry, is present in the signature will remove the possibility of using signed data from one platform on another platform. Using a new claim issuer could potentially work as a form of domain separation, but this is not a standard way of preventing signature reuse.

Status

Resolved. The team internally discussed the use case of the registry and determined that domain separation is not necessary.

Suggestion 3: Remove Hardhat Logging Imports

Location

All

Synopsis

Hardhat logging is still present on the files meant for better visibility while testing. While this is not a security concern, it can be removed easily with tooling.

Mitigation

There are no console logs present in the code, but Hardhat provides a tool for removing logging before deployment and may be useful to help cut out the unnecessary imports. It would likely be easier to simply remove the unnecessary importing of Hardhat logging on each contract.

Status

Resolved. All imports of hardhat logging have been removed from contracts meant to be deployed.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create

an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.