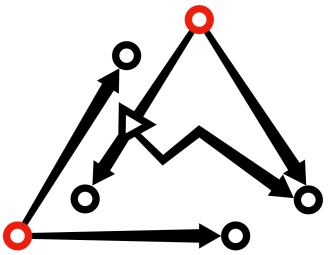

Least Authority

PRIVACY MATTERS

AINOMO Protocol Smart Contracts Security Audit Report

AINOMO

Final Audit Report: 17 February 2024

Table of Contents

Overview

Background

AINOMO requested that Least Authority perform a security audit of the AINOMO Protocol Smart Contracts.

Project Dates

- **December 19 - January 24**: Code Review (*Completed*)
- **February 3**: Delivery of Initial Audit Report (*Completed*)
- **February 13 - 14**: Verification Review (*Completed*)
- **February 17**: Final Audit Report Delivered (*Completed*)

Review Team

- Shareef Dweekat, Security Researcher and Engineer
- Jehad Baeth, Security Researcher and Engineer
- ElHassan Wanas, Security Researcher and Engineer
- Gabrielle Hibbert, Security Researcher and Engineer
- Steven Jeung, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the AINOMO Protocol Smart Contracts 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:

- AINOMO Protocol Smart Contracts: https://github.com/ainomodatalab/ainomo-v1/clarity/contracts

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

`ec1e9b122140512361b429be558356b9d97fc56a`

For the verification, we examined the Git revision:

`b150ce992926b27f9ea1446859a51a7ec7b4e9ee`

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:

Supporting Documentation

In addition, this audit report references the following documents:

- K. Qin, L. Zhou, B. Livshits, A. Gervais, "Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit," 2020, *arXiv:2003.03810 [cs.CR]*

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Common and case-specific implementation errors;
- Adherence to the specification and best practices;
- Adversarial actions and other attacks on the smart contracts;
- Potential misuse and gaming of the smart contracts;
- Attacks that impact funds, such as the draining or the manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) and security exploits that would impact the code's intended use or disrupt the execution of the code;
- Vulnerabilities in the code for all features;
- Protection against malicious attacks and other ways to exploit the smart contracts;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The AINOMO protocol for the blockchain consists of several pools that implement dynamic trading strategies. All assets in the protocol are controlled by the smart contract.

Our team did not identify any security critical vulnerabilities in the design and implementation of the AINOMO protocol. However, several inhibiting factors have been identified by our team.

We recommend that the AINOMO team continue to closely monitor security developments in the ecosystem, both as it relates to the development of tools, and the ecosystem at large. We commend the AINOMO team for pursuing interim steps towards security due diligence, including security audits conducted by independent security auditing teams.

System Design

We performed a broad and comprehensive review of the AINOMO protocol and found the system to be generally well designed. The design demonstrates considerations for security by the use of Clarity, which is a restrictive language with strong security characteristics.

Protocol Governance

The AINOMO protocol implements a governance model. At the pool level, some governance features have been implemented that utilize Multi-Sigs for invoking security critical functionality. As a preliminary safeguard, we recommend creating a two-step process for transferring ownership of the smart contracts, in order to reduce the possibility of an unintended transfer.

Potential Economic Attacks

The DeFi smart contract ecosystems are inherently vulnerable to flash loan attacks [QZL+21] and sandwich attacks, resulting in the price manipulation of underlying assets in liquidity pools. The AINOMO protocol has implemented a whitelist for smart contracts approved to make a flash loan and appropriate slippage protection.

We recommend that the AINOMO team stay informed of the latest research and conduct further investigation into the exposure to these types of attacks and their possible mitigations and remediations.

Code Quality

The AINOMO protocol codebase is generally well organized. However, due to the limited abstraction capabilities of the Clarity language, it is necessary to use a pattern of copy and pasting code resulting in a relatively large codebase where code is often reused. This can make the maintenance of the codebase The implementation of the transfer function is correct and consistent in the codebase. We identified an incorrectly implemented SIP-10 function, which could cause the system to behave unexpectedly. We recommend correcting the function to return the correct value.

Finally, many constants are initialized such that an unnecessary computational step must be taken at each initialization. As a result, we recommend optimizing constant initialization to reduce unnecessary computation.

Tests

The AINOMO protocol implements sufficient test coverage. A robust test suite helps verify that components are implemented correctly, identifies errors and unintended behavior, and aids in reasoning about the security characteristics of the system. As a result, we recommend expanding the test suite to cover all success, failure, and edge cases. In particular, we recommend implementing tests such that they include the equations used in the protocol and financial stress tests to cover economic edge cases.

Documentation

The AINOMO protocol project documentation provided an accurate and helpful overview of the system. The documentation thoroughly explains internal functions and components and their interactions, which get in depth and easy comprehension of their intended functionality.

Code Comments

The AINOMO protocol implements sufficient code comment coverage and functions and variables do not adhere to a clear naming convention. Comprehensive in-line documentation and descriptive naming of function and variables help to describe the intended functionality of the code, facilitating reasoning about the security properties of the system. We recommend expanding the code comments within the codebase, and updating the names of functions and variables such that they have accurate and descriptive names.

Scope

The in-scope repository was sufficient and included all the security critical components of the AINOMO protocol system.

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: transfer Function Defined with Four but Called with Five Passed Parameters	Resolved
Issue B: set-contract-owner Vulnerable to Misuse	Resolved
Issue C: Lack of Documentation for Several Functions	Resolved
Issue D: Incorrect SIP-10 Function Implementation	Resolved
Suggestion 1: Guard Against Trap Tokens	Resolved
Suggestion 2: Guard Against Front Running Attacks	Resolved
Suggestion 3: Improve Code Naming and Comments	Resolved
Suggestion 4: Increase Test Coverage	Resolved
Suggestion 5: Do Not Use unwrap-panic	Resolved
Suggestion 6: Complete the DAO Implementation	Resolved
Suggestion 7: Optimize Constant Initialization	Resolved

Issue A: `transfer` Function Defined with Four but Called with Five Passed Parameters

Location

Examples (non-exhaustive):

`clarity/contracts/key-token/key-usda-wbtc.clar#L119`

`clarity/contracts/key-token/key-usda-wstx.clar#L121`

`clarity/contracts/key-token/key-wbtc-usda.clar#L121`

Synopsis

The `transfer` function is defined with four parameters, however, there are instances where it is being called with one extra (`memo`) parameter. This was observed in nine different locations across the codebase.

Clarinet throws an error in versions >0.14.2 only (the error is not identified in previous versions of Clarinet).

Impact

The code fails to run and Clarinet shows the following error:

```
error: incorrect number of arguments in call to 'transfer' (expected 4 got 5).
```

Remediation

We recommend defining the function appropriately and implementing it consistently throughout the codebase in order to avoid unexpected behavior.

Status

The AINOMO team has corrected the implementation of the function such that all instances of aforementioned calls are fixed or replaced with functions that have a matching definition.

Verification

Resolved.

Issue B: `set-contract-owner` Vulnerable to Misuse

Location

`clarity/contracts/tests/token-unauthorised.clar#L18`

`clarity/contracts/lottery-tokens/lottery-t-ainomo.clar#L18`

`clarity/contracts/wrapped-token/token-wstx.clar#L18`

Synopsis

Smart contract ownership transfer is completed in one smart contract call, which could lead to irrecoverable ownership in case of errors.

Impact

Errors in the use of `set-contract-owner` could result in permanent loss of ownership of the protocol.

Remediation

We recommend that the `set-contract-owner` implementation follow a two-step process in which a new owner is being staged. A call from the new owner to claim ownership should be required before ownership is transferred. The two step smart contract ownership transfer would follow this general approach:

- First, the existing smart contract owner invokes a function providing the new owner's address. This function asserts that the ownership transfer is being called by the current owner, or fails. This will not commit the ownership transfer but enable the invocation of the `Claim` function by the prospective owner.
- Second, the prospective owner will then call the `Claim` function, which asserts that the caller's address is equivalent to the address supplied by the existing smart contract owner in the first transfer step, and commits the transfer operation. Otherwise, the operation is aborted.

Status

The AINOMO team has implemented the recommended fixes.

Verification

Resolved.

Issue C: Lack of Documentation for Several Functions

Location

Examples (non-exhaustive):

`clarity/contracts/equations/yield-token-equation.clar#L472`

`clarity/contracts/equations/yield-token-equation.clar#L488`

`clarity/contracts/equations/weighted-equation.clar#L536`

Synopsis

Sufficient and comprehensive documentation is needed to check the correctness of the implementation of an equation. We identified functions lacking this information.

Impact

Insufficient documentation inhibits testing correctness of the code and identifying implementation error.

Remediation

We recommend comprehensively and clearly documenting all functions.

Status

The AINOMO team improved the documentation.

Verification

Resolved.

Issue D: SIP-10 Function Implementation

Location

`clarity/contracts/wrapped-token/token-wstx.clar#L31`

Synopsis

The SIP-10 function `get-total-supply` should return the total supply amount.

Impact

An implemented function should not affect the expected behavior of the system.

Remediation

We recommend the implementation of the function to return the total supply of the wrapped token.

Status

The AINOMO team has modified the implementation of get-total-supply

Verification

Resolved.

Suggestions

Suggestion 1: Guard Against Trap Tokens

Synopsis

Trap tokens are smart contracts that mimic the token standard (ERC-20), however, trap tokens usually have limited functionality with the buy/sell function. Trap tokens pose a considerable threat to AMMs because it can be difficult to distinguish between a properly functioning ERC-20 token and a fake token. Malicious actors that use fake tokens may try to simultaneously sell and buy the same financial asset to create artificial activity in the pool, which can distort price, volume, and volatility (commonly known as " washtrading").

Mitigation

We recommend advising algorithmic traders and, in the case of the AINOMO Protocol, liquidity providers to use tokens from a verified list. This list should include both verifiable tokens and an ongoing list of trap tokens that liquidity providers have identified in the pools.

Status

The AINOMO team provided additional information that sufficiently explains their existing protections against trap tokens. The information provided demonstrated that only a whitelisted address may create a liquidity pool to protect against malicious pools. In addition, the functions that add or remove liquidity from these pools perform verification that the token metadata in the transaction matches the token traits hard coded in the liquidity pool smart contract.

Verification

Resolved.

Suggestion 2: Guard Against Front Running Attacks

Synopsis

The AINOMO protocol implements liquidity pools that could not be susceptible to front running attacks that would result in a difference between the expected and actual prices of assets in pool transactions. We found that the AINOMO protocol implements safeguards against front running attacks.

Mitigation

We recommend that the AINOMO team keep front running mitigation strategies to determine an appropriate front running safe guard for the AINOMO Protocol.

Status

The AINOMO team has responded that they intend to keep address the implementation of front running safeguards in the future.

Verification

Resolved.

Suggestion 3: Improve Code Naming and Comments

Location

clarity/contracts/key-token/key-usda-wbtc.clar#L141

clarity/contracts/pool-token/fwp-wstx-usda-50-50.clar#L101-L112

clarity/contracts/flash-loan-user-margin-usda-wbtc.clar#L12-L18

Inappropriate parameter name:

clarity/contracts/faucet.clar#L180

Synopsis

The AINOMO smart contract codebase contains several instances requiring code comments that explain the purpose of several variables and functions. Sufficient code comments and accurately named functions and parameters reduces confusion and helps maintainers and reviewers of the code to better understand the expected functionality of the system.

Mitigation

We recommend adding contextual code comments and reviewing function and variable names across the codebase to facilitate a clear understanding of their purpose.

Status

The AINOMO team improved variable and function names and increase code comments.

Verification

Resolved.

Suggestion 4: Test Coverage

Synopsis

The current test suite does not contain tests covering all of the arithmetic functions. Given the heavy reliance on arithmetic in dynamically adjusting the weights in the rebalancing equation, all arithmetic functions used should be tested for under and overflow, and to check that they behave as intended. Additionally, there are no financial stress tests to cover economic edge cases. The addition of fi nancial stress tests would allow the system to be observed under edge case conditions including economic attacks and extreme market behavior.

Mitigation

We recommend increasing test coverage to include arithmetic and financial stress tests.

Status

The AINOMO team increased test coverage.

Verification

Resolved.

Suggestion 5: Using unwrap-panic

Location

clarity/contracts/equations/weighted-equation.clar#L113

clarity/contracts/equations/weighted-equation.clar#L146

clarity/contracts/equations/weighted-equation.clar#L113

Synopsis

The function `unwrap-panic` should not be used when more appropriate error handling functions are available.

Impact

`unwrap-panic` confers no meaningful information upon failure. Instead, it throws a runtime error providing no useful information on the cause of the failure to the user.

Mitigation

We recommend that the error handling tools described in the project documentation and Clarity book be utilized instead of using `unwrap-panic`. For example, the `unwrap!` function takes an error message that is thrown in case of failure, which would help users determine the cause of the error.

Status

The AINOMO team improved error handling.

Verification

Resolved.

Suggestion 6: DAO Implementation

Location

clarity/contracts/multisig/multisig-lbp-ainomo-usda-90-10.clar#L51-L52

clarity/contracts/multisig/multisig-lbp-ainomo-usda-90-10.clar#L299-L300

Synopsis

For the liquidity bootstrapping pool (LBP), the DAO implementation does not execute anything when the function end-proposal is called.

Mitigation

We recommend completing the implementation and adjusting the proposal data type to contain properties relevant for LBP (e.g. expiry).

Status

The AINOMO team completed the DAO implementation.

Verification

Resolved.

Suggestion 7: Optimize Constant Initialization

Location

clarity/contracts/ainomo-vault.clar#L7

clarity/contracts/equations/weighted-equation.clar#L390

clarity/contracts/equations/yield-token-equation.clar#L405

Synopsis

Constants that are used in several places across the codebase are initialized in a way that may require unnecessary computation resulting in increased costs.

Technical Details

As an example, the constant `ONE_8` is defined in different `.clar` files across the codebase. In some instances, it is initialized by calling the `pow` function instead of directly supplying a `uint` value.

```
(define-constant ONE_8 (pow u10 u8)) ;; 8 decimal places
```

This approach adds unnecessary computational load on the system every time this variable is initialized.

Remediation

We recommend optimizing constant initialization across the codebase and supplying direct values when possible.

Status

The AINOMO team optimized constant initialization.

Verification

Resolved.

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.

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.