Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Arcane Finance

RFQ Contract

Veridise Inc.

February 16, 2024

► **Prepared For:**

Arcane Finance
https://www.arcane.finance/

► **Prepared By:**

Bryan Tan
Andreea Buțerchi
Yanju Chen

► **Contact Us:** contact@veridise.com

► **Version History:**

Feb. 16, 2024	V1.1 - Updated issue statuses in response to developer fixes
Jan. 26, 2024	V1

Contents

1 Executive Summary

From Jan. 17, 2024 to Jan. 19, 2024, Arcane Finance engaged Veridise to review the security of the RFQ Contract component of their Arcane Finance protocol. Veridise conducted the assessment over 9 person-days, with 3 engineers reviewing code over 3 days on commit `d17b0ff`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Project summary. The security assessment covered the request-for-quote (RFQ) smart contract used in the Arcane Finance protocol, which is a decentralized exchange protocol that uses the Aleo blockchain. In Arcane's RFQ workflow, users make an off-chain request to a market maker for the price of a token swap that they want to perform. The maker may then respond with a cryptographically signed quote, which the user can then submit on-chain to the RFQ contract to perform the swap. The RFQ contract supports both Aleo Credits as well as "Arcane Tokens" specific to the Arcane Finance protocol.

Code assessment. The RFQ Contract developers provided the source code of the RFQ Contract for review. The source code consists of a single Leo program that implements the behavior of the RFQ system, and it appears to be mostly original code written by the RFQ Contract developers. It contains some documentation in the form of READMEs and documentation comments on functions and state variables. To improve the auditors' understanding of the code, the RFQ Contract developers shared a design document describing the overall goals of the protocol. The source code does not contain an automated test suite, but the README describes several test cases that must be executed manually.

Summary of issues detected. The audit uncovered 2 issues, comprising 1 medium-severity issue and 1 informational issue. Specifically, V-ARF-VUL-001 notes that due to missing checks for ArcaneTokens with ID 0, makers may be able to steal Aleo Credits from the contract. The Veridise auditors also identified an issue V-ARF-VUL-002 where private token amounts are published but the code does not clearly indicate whether this is intended. Of the 2 issues, Arcane Finance has fixed 1 issue. Arcane Finance has acknowledged the 1 remaining issue but deemed it too minor to fix.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
RFQ Contract	`d17b0ff`	Leo	Aleo

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Jan. 17 - Jan. 19, 2024	Manual & Tools	3	9 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	1	1	1
Low-Severity Issues	0	0	0
Warning-Severity Issues	0	0	0
Informational-Severity Issues	1	0	1
TOTAL	2	1	2

Table 2.4: Category Breakdown.

Name	Number
Theft	1
Information Leakage	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of the RFQ Contract. In our audit, we sought to answer questions such as:

- Does the RFQ Contract implementation adhere to the specifications described by the developers?
- Is it possible for users to steal funds from the RFQ Contract?
- Can information be leaked when users interact with the protocol privately?
- Are type casts and range checks appropriately performed when converting between Arcane Tokens and Aleo Credits?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our prototype Aleo static analysis tool. This tool is designed to find instances of common smart contract vulnerabilities, such as privacy leakages and mathematical errors.

Scope. The scope of this audit is limited to the `src/main.leo` file of the source code provided by the RFQ Contract developers, which contains the smart contract implementation of the RFQ Contract.

Methodology. Veridise auditors reviewed the documentation of the RFQ Contract and met with the RFQ Contract developers to understand the intended behavior of the code. They then conducted a manual code review assisted by the Aleo static analyzer.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-ARF-VUL-001	Missing checks for ArcaneToken ID of 0 may allo. .	Medium	Fixed
V-ARF-VUL-002	Private amounts are publicized	Info	Acknowledged

4.1 Detailed Description of Issues

4.1.1 V-ARF-VUL-001: Missing checks for ArcaneToken ID of 0 may allow theft of Aleo Credits

Severity	Medium	**Commit**	d17b0ff
Type	Theft	**Status**	Fixed
File(s)	`main.leo`		
Location(s)	See description		
Confirmed Fix At	129a1d0		

The transitions `add_rfq_liquidity` and `remove_rfq_liquidity` allow market makers to deposit and remove (respectively) ArcaneTokens from the RFQ smart contract. These maker balances will be stored in a mapping where each key is the hash of the pair (maker address, token ID), where the token ID is the ID of the corresponding ArcaneToken. The RFQ smart contract also allows makers to provide Aleo Credits as liquidity using the `add_credit_liquidity` and `remove_credit_liquidity` transitions, in which case the token ID is assumed to be 0. However, `add_rfq_liquidity` and `remove_rfq_liquidity` do not prevent ArcaneTokens with ID 0 from being used.

```
1  transition add_rfq_liquidity(t: arcanetoken.leo/ArcaneToken, maker_address: address,
       amount: u128) -> ArcaneToken {
2          let deposit_id: field = get_deposit_id(maker_address, t.token_id);
3          // ...
4  }
5
6  transition remove_rfq_liquidity(token_id: u64, amount: u128) -> ArcaneToken {
7          let deposit_id: field = get_deposit_id(self.caller, token_id);
8      // ...
9  }
10
11 transition add_credit_liquidity(t: credits.leo/credits, maker_address: address,
       amount: u64) -> credits {
12     let deposit_id: field = get_deposit_id(maker_address, 0u64);
13     // ...
14 }
15
16 transition remove_credit_liquidity(amount: u64) -> credits.leo/credits.record {
17     let deposit_id: field = get_deposit_id(self.caller, 0u64);
18     // ...
19 }
```

Snippet 4.1: The relevant transition functions

Similarly, the transition `quote_swap` does not validate that both `token_in` and `token_out` are non-zero.

Consequently, for each maker, Aleo Credits and ArcaneToken with ID 0 will share the same balance.

```
transition quote_swap(
        t: arcanetoken.leo/ArcaneToken,
        q: Quote,
        s: signature
) -> (ArcaneToken, ArcaneToken) {
    assert(s.verify(q.maker_address, q));

        assert(t.token_id == q.token_in);
    assert(t.amount >= q.amount_in);
    assert(q.token_in != q.token_out);
        ...
}
```

Snippet 4.2: Relevant lines in `quote_swap`

Impact Due to the lack of validation for ArcaneToken of ID 0 in transitions `add_rfq_liquidity`, `remove_rfq_liquidity`, and `quote_swap`, this will allow malicious makers to corrupt the bookkeeping for Aleo Credits. The consequences of such corruption may allow makers, for example, to add ArcaneToken with ID 0 as liquidity (or swap for such tokens) and then remove the same amount as Aleo Credits (`remove_credits_liquidity`) without ever having provided Aleo Credits as liquidity.

As a more specific example, consider the following scenario:

1. A maker provides quotes that allow exchanging some ArcaneTokens with ID N for ArcaneTokens with ID 0.
2. A taker uses `quote_swap` with a valid quote to trade ArcaneTokens with ID N for X amount of ArcaneTokens with ID 0.
3. Because the maker's balance for token ID 0 has increased by X, the maker can now use `remove_credit_liquidity` to extract X amount of Aleo Credits from the smart contract. Note that the maker will profit from this if Aleo Credits are worth more than ArcaneTokens of ID 0.
4. Other makers may then attempt to use `remove_credit_liquidity`, which may revert if the smart contract has run out of Aleo Credits.

Recommendation The root cause of the problem is that the same balance is used for ArcaneTokens of ID 0 and Aleo Credits. Some potential solutions include:

- Prevent ArcaneTokens of ID 0 from being used. Specifically:
  - `add_rfq_liquidity` and `remove_rfq_liquidity` should assert that the `token_id` is nonzero.
  - `quote_swap` should assert that both `token_in` and `token_out` are nonzero.
  - The `arcanetoken.leo` contract should prevent token ID 0 from being created or minted.
- If it is intended for ArcaneTokens of ID 0 to be supported, then the Aleo Credits balances should be stored in a separate mapping.
- Remove Aleo Credits functionality from the RFQ smart contract, and modify `arcanetoken.leo` to support a special ID (e.g., 0) that can be freely exchanged with Aleo Credits at a fixed 1:1 ratio.

Developer Response The developers added a new mapping `credit_balances` that is used to track the Aleo Credits.

4.1.2 V-ARF-VUL-002: Private amounts are publicized

Severity	Info	**Commit**	d17b0ff
Type	Information Leakage	**Status**	Acknowledged
File(s)	`main.leo`		
Location(s)	See description		
Confirmed Fix At	N/A		

The following transitions allow a maker to transfer Aleo Credits or Arcane Tokens to/from the RFQ smart contract:

- `add_rfq_liquidity`
- `remove_rfq_liquidity`
- `add_credit_liquidity`
- `remove_credit_liquidity`

Each of these transitions has a private `amount` parameter indicating the amount of token that will be transferred in/out. However, the `amount` will be passed as an argument to the finalize part of the transition, which publishes the `amount` on-chain.

Similarly, the quote swapping-related transitions will also pass the the private `amount_out`, `amount_in` fields of the quote to the finalize part and publish them on-chain:

- `quote_swap`
- `quote_swap_credits_in`
- `quote_swap_credits_out`

Impact The above transitions may publicize information about the involved amounts. Specifically, by comparing a maker's token balance before and after a transaction is finalized, it will be possible to recover the amount involved in the liquidity or swap operation.

It was not immediately clear to the auditors whether the `amounts` involved in the transitions are intended to be published. However, since the `maker_balance` mapping is public anyways, it appears that the amounts (of tokens to add or remove) *are* intended to be published.

Recommendation To improve the clarity of the code, consider explicitly marking the transition parameters as `private` and adding comments indicating that the amounts are intended to be published.

Developer Response The developers noted that the RFQ system is intended to keep the identity of the token owners private. It was not clear to the auditors whether amounts are meant to have a similar guarantee.

