Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Decenteralized Exchange

Veridise Inc.

January 18, 2024

► **Prepared For:**

AlphaSwap
https://alphaswap.pro/

► **Prepared By:**

Jon Stephens
Sorawee Porncharoenwase
Yanju Chen

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

► **Version History:**

Jan. 10, 2024	Initial Draft

Contents

1 Executive Summary

From Jan. 3, 2024 to Jan. 10, 2024, AlphaSwap engaged Veridise to review the security of their Decenteralized Exchange. The review covered the core AlphaSwap exchange protocol, their ARC20 token implementation, a token faucet to distribute AlphaSwap tokens and several wrappers that allow external tokens, including Aleo credits, to interact with the protocol. Veridise conducted the assessment over 3 person-weeks, with 3 engineers reviewing code over 1 week on commit `e8f2056`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The AlphaSwap developers provided the source code of the Decenteralized Exchange for review. The codebase consists of five Aleo programs: a decentralized exchange, an ARC20 token implementation, a wrapper for Aleo credits, a wrapper for ARC20 tokens and a token faucet to distribute tokens used by the exchange. Significant portions of the AlphaSwap exchange's behavior were inherited from Uniswap V2, but AlphaSwap also provides the ability to swap tokens privately. To facilitate the on-boarding of existing tokens such as Aleo credits or custom ARC20 tokens, the developers provide token wrapper implementations that manage internal AlphaSwap exchange tokens. Additionally, since the Aleo ecosystem has not yet converged on an official ARC20 token standard, the AlphaSwap developers also provide an implementation of one of the token drafts, allowing custom tokens to be easily created. Finally, a token faucet was also provided that will distribute tokens to users on the Aleo testnet.

The AlphaSwap decentralized exchange contains documentation in the form of READMEs as well as some in-line documetation in the source code. The Veridise auditors also referred to the documentation for Uniswap V2 as portions of the exchange were based on this protocol. Finally, the Veridise auditors also referred to the draft ARC20 specification located at the following link as the AlphaSwap documentation indicated their ARC20 token implementation was based on this draft: `https://github.com/AleoHQ/ARCs/pull/41`.

Summary of issues detected. The audit uncovered 10 issues, one of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, V-ASP-VUL-001 identifies missing adjustments to the total supply of swap tokens when allocating fees which could prevent the removal of liquidity from the protocol. The Veridise auditors also identified a medium-severity issue and 6 low-severity issues, including the potential for a pair to be rendered inoperable if the recipient of protocol fees does not exist in the balances mapping, as well as 2 warnings and 0 informational findings.

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
Decenteralized Exchange	`e8f2056`	Leo	Aleo

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Jan. 3 - Jan. 10, 2024	Manual & Tools	3	3 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	0	0	0
High-Severity Issues	1	1	1
Medium-Severity Issues	1	1	1
Low-Severity Issues	6	3	5
Warning-Severity Issues	2	2	2
Informational-Severity Issues	0	0	0
TOTAL	10	7	9

Table 2.4: Category Breakdown.

Name	Number
Usability Issue	3
Locked Funds	1
Denial of Service	1
Centralization	1
Maintainability	1
Frontrunning	1
Data Validation	1
Information Leakage	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Decenteralized Exchange's smart contracts. In our audit, we sought to answer questions such as:

- Are behaviors inherited from Uniswap V2 consistent with that protocol?
- Can information be leaked when users interact with the protocol privately?
- Is state appropriately updated when a user interacts with the protocol privately?
- Can a user improperly remove liquidity or prevent others from removing their liquidity from the protocol?
- Do the Aleo and ARC20 token wrappers appropriately wrap tokens for the AlphaSwap exchange?
- Does the ARC20 token implementation adhere to one of the draft specifications?

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 Leo code defined in the `aleo_wrapper`, `arc20_token`, `arc20_wrapper`, `swap` and `token_faucet` directories of AlphaSwap's private repository.

Methodology. Veridise auditors reviewed the provided AlphaSwap Decenteralized Exchange documentation, the Uniswap V2 Documentation and the ARC20 specification draft used by the project. They then conducted a manual audit of the code 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.

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

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

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

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

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

Vulnerability Report 4

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-ASP-VUL-001	Missing Adjustment of Total Supply	High	Fixed
V-ASP-VUL-002	DoS possibility when there's no balance	Medium	Fixed
V-ASP-VUL-003	Divergence from ARC20 specification	Low	Fixed
V-ASP-VUL-004	Centralization Risk	Low	Acknowledged
V-ASP-VUL-005	Unapproved Spending of Private Tokens	Low	Intended Behavior
V-ASP-VUL-006	Duplicate constants throughout the codebase	Low	Fixed
V-ASP-VUL-007	Frontrunning in approve()	Low	Fixed
V-ASP-VUL-008	Monopolization of Token Claiming	Low	Acknowledged
V-ASP-VUL-009	Unnecessary Input	Warning	Fixed
V-ASP-VUL-010	Potential Private Value Leak	Warning	Fixed

4.1 Detailed Description of Issues

4.1.1 V-ASP-VUL-001: Missing Adjustment of Total Supply

Severity	High	**Commit**	e8f2056
Type	Locked Funds	**Status**	Fixed
File(s)	swap/src/main.leo		
Location(s)	add_liquidity(), remove_liquidity(), ...		
Confirmed Fix At	7179a52		

Similar to Uniswap V2, AlphaSwap allows a pair to define a fee recipient who will receive a percentage of the collected fees. When liquidity is added or removed from the pair, their fee should then be distributed by creating new shares in the pair for them. Currently, however, pair tokens are given to the fee recipient by adjusting their balance, but the total supply of the pair's token is never increased accordingly as shown below.

```
finalize add_liquidity(
    ...
) {
    let pid: field = get_pair_id(token_a, token_b);
    let pair: Pair = pairs.get(pid);
    let ti_pair: TokenInfo = tokens.get(pid);
    let gs: GlobalState = global_state.get(true);

    // calculate and update fee
    let root_k: u128 = gs.fee_to ==
    aleo1qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq3ljyzc ? 0u128 :
    sqrt_u128(pair.reserve_a * pair.reserve_b);
    let last_root_k: u128 = sqrt_u128(pair.last_k);
    let fee: u128 = pair.last_k == 0u128 || root_k <= last_root_k ? 0u128 : ti_pair.
    total_supply * (root_k - last_root_k) / (root_k * 5u128 + last_root_k);
    let bid_fee: field = get_balance_id(pid, gs.fee_to);
    balances.set(bid_fee, balances.get(bid_fee) + fee);

    ...

    // mint liquidity
    let bid_p: field = get_balance_id(pid, to);
    balances.set(bid_p, balances.get_or_use(bid_p, 0u128) + liquidity);
    tokens.set(pid, TokenInfo {
        name: ti_pair.name,
        symbol: ti_pair.symbol,
        decimals: ti_pair.decimals,
        total_supply: ti_pair.total_supply + liquidity,
        admin: ti_pair.admin,
        mintable: ti_pair.mintable,
        burnable: ti_pair.burnable,
    });

    ...
}
```

Snippet 4.1: Location where shares are given to the fee recipient without increasing total supply

Impact Since the total supply is not updated appropriately, the sum of the pair token's balances will be greater than the total supply. Therefore, if the fee recipient claims greater than the locked 1000 tokens allocated when the pair is created, users may be unable to remove their liquidity.

Recommendation In functions where fees are minted to the fee recipient, ensure the total supply is updated appropriately. Additionally note that in Uniswap V2, the total supply is updated before calculating liquidity.

4.1.2 V-ASP-VUL-002: DoS possibility when there's no balance

Severity	Medium	**Commit**	e8f2056
Type	Denial of Service	**Status**	Fixed
File(s)	`swap/src/main.leo`		
Location(s)	add_liquidity(), remove_liquidity(), ...		
Confirmed Fix At	7179a52		

The `add_liquidity`, `add_liquidity_privately`, `remove_liquidity`, and `remove_liquidity_privately` functions distribute fees to a fee recipient set by the admin. When sending these funds, however, these functions use `get` rather than `get_or_use` to fetch the fee recipient's current balance as shown below. It is possible that an admin may have set `fee_to` to someone without an entry in the `balances` mapping, though, which would cause `get` to revert the transition.

```
let bid_fee: field = get_balance_id(pid, gs.fee_to);
balances.set(bid_fee, balances.get(bid_fee) + fee);
```

Snippet 4.2: A snippet from the mentioned functions

Impact The pair would be out of service in the described situation.

Recommendation Change `.get` to `.get_or_use`.

4.1.3 V-ASP-VUL-003: Divergence from ARC20 specification

Severity	Low	**Commit**	e8f2056
Type	Usability Issue	**Status**	Fixed
File(s)	`arc20_token/src/main.leo`		
Location(s)	approve()		
Confirmed Fix At	7179a52		

While Aleo has not converged on an official ARC20 specification, based on a comment at the top of their ARC20 token source code, the developers appear to be following this ARC20 specification. According to that specification, there should be functions `approve_public` and `unapprove_public` which are missing.

Impact Clients may expect that the functions in the specification are defined. In addition, the source code currently implements an `approve` function which has a frontrunning risk as discussed in V-ASP-VUL-007. The risk would be reduced by switching to the functions in the specification.

Recommendation Implement the functions specified in the specification.

Developer Response Since the ARC20 specification is still a draft and there is no consensus yet, we have chosen not to be strictly compliant with the specification.

4.1.4 V-ASP-VUL-004: Centralization Risk

Severity	Low	**Commit**	e8f2056
Type	Centralization	**Status**	Acknowledged
File(s)	`arc20_token/src/main.leo`, `swap/src/main.leo`, `token_faucet/src/main.leo`		
Location(s)	N/A		
Confirmed Fix At			

Similar to many projects, AlphaSwap's ARC20 token, swap program and ARC20 faucet declare an administrator role that is given special permissions. In particular, these administrators are given the ability to mint tokens, set the swap program's fee receiver and withdraw all tokens from the ARC20 faucet.

Impact If a private key were stolen, a hacker would have access to sensitive functionality that could compromise the project. For example, a malicious minter could mint a large number of tokens for themselves.

Recommendation As these are all particularly sensitive operations, we would encourage the developers to utilize a decentralized governance or multi-sig contract as opposed to an Aleo account, which introduces a single point of failure.

Developer Response We plan to use a DAO or multisig approach to control admin privileges in the future.

4.1.5 V-ASP-VUL-005: Unapproved Spending of Private Tokens

Severity	Low	**Commit**	e8f2056
Type	Usability Issue	**Status**	Intended Behavior
File(s)	`arc20_token/src/main.leo`, `swap/src/main.leo`		
Location(s)	N/A		
Confirmed Fix At			

The ARC20 token allows users to spend tokens privately with the `PrivateToken` record. Rather than spending a private token that is too large, which could allow a transition to spend more funds than the user intends, users should be able to create a private token of the exact size they intend to spend. The ARC20 token makes this difficult, however, as the functionality to split private tokens has been commented out. Currently, to get an exact number of private tokens one must first make the the tokens public and then create the appropriately sized private token which could leak information if not done properly.

Recommendation Consider adding the split functionality to both the ARC20 token and the swap tokens.

Developer Response Adding separate split functionality is unnecessary because `transfer_private` can already be used to split tokens.

4.1.6 V-ASP-VUL-006: Duplicate constants throughout the codebase

Severity	Low	**Commit**	e8f2056
Type	Maintainability	**Status**	Fixed
File(s)	`token_faucet/src/main.leo`, `aleo_wrapper/src/main.leo`, `arc20_wrapper/src/main.leo`		
Location(s)			
Confirmed Fix At	7179a52		

There are many uses of duplicate constants throughout the codebase that should have been named and reused for maintainability. As an example, the following function hard-codes the addresses for the AlphaSwap program and the aleo wrapper program.

```
inline get_token_id() -> field {
  return BHP256::hash_to_field(TokenIdData {
    base: aleo1crhcpeqfa6pl3gpd5352am8agrzduyafeslsrxh8l65nk5gv4q9q5mzkxt,
    creator: aleo1pnn0j04vvgj6xqjxsp9swt5p9qn29tyqh755lx3w96hety64ly8shwwu3c,
    salt: 1u128,
  });
}
```

Snippet 4.3: The definition of the Aleo wrapper program's `get_token_id` function which uses hard-coded literals instead of constants

Impact Hard-coding literals can introduce errors since developers must ensure that whenever a literal is updated, all duplicates are updated as well.

Recommendation Use a language construct to name the constants and mention them by name.

4.1.7 V-ASP-VUL-007: Frontrunning in approve()

Severity	Low	**Commit**	e8f2056
Type	Frontrunning	**Status**	Fixed
File(s)	`arc20_token/src/main.leo`		
Location(s)	approve()		
Confirmed Fix At	7179a52		

Setting allowance carries a frontrunning risk similar to with ERC20 tokens. Specifically consider the following scenario:

1. Party A approves party B to transfer with the limit of 1000.
2. Party B transfers with the amount of 1000.
3. Without seeing (2), party A approves party B to transfer with the limit of 900, with the intention of lowering the limit by 100 (i.e., un-approves by 100).
4. Party B can now transfers with the amount 900, in addition to the earlier 1000.

Impact Such a frontrunning attack could allow an approved user to inappropriately spend funds.

Recommendation Separate `approve` into `approve_public` and `unapproved_public` as specified in ARC20 specification to reduce the frontrunning risk.

4.1.8 V-ASP-VUL-008: Monopolization of Token Claiming

Severity	Low	**Commit**	e8f2056
Type	Data Validation	**Status**	Acknowledged
File(s)	`token_faucet/src/main.leo`		
Location(s)	claim()		
Confirmed Fix At			

The token faucet program distributes AlphaSwap tokens to users by allowing a small number of tokens to be claimed from a larger pool. As shown below, however, it is possible for a user to repeatedly call `claim` even in the same transaction. This could allow a small number of users to claim the bulk of the tokens.

```
transition claim(public token_id: field, public amount: u128) {
    aleoswap.leo/transfer_public(token_id, self.caller, amount);
    return then finalize(token_id, amount);
}

finalize claim(public token_id: field, public amount: u128) {
    // Can't claim more than the faucet setting.
    assert(amount <= faucets.get(token_id));
}
```

Snippet 4.4: The definition of the claim function which can be called multiple times

Impact The tokens may not be available for a wide range of users to claim due to the monopolization.

Recommendation Implement a limiting mechanism such as only allowing an address to claim once or adding a cooldown period to claims.

Developer Response The token faucet contract is only intended for use on testnet and therefore restrictions are unnecessary as the funds will not have value.

4.1.9 V-ASP-VUL-009: Unnecessary Input

Severity	Warning	**Commit**	e8f2056
Type	Usability Issue	**Status**	Fixed
File(s)	`swap/src/main.leo`		
Location(s)	create_pair_privately		
Confirmed Fix At	7179a52		

The `create_pair_privately` function allows a user to create a pair using private tokens and will privately return the resulting swap pair tokens. In this function, rather than calculating the number of swap pair tokens to send the user, it instead requires the user to provide this amount as an input. It later validates that the input amount is correct as shown below, but this calculation could be performed in the transition.

```
transition create_pair_privately(
    ...
    public liquidity: u128,
 ) -> (PrivateToken, PrivateToken, PrivateToken) {
    ...

    let pid: field = get_pair_id(token_a, token_b);
    let pt_lp: PrivateToken = PrivateToken {
        owner: to,
        token: pid,
        amount: liquidity - 1000u128,
    };
    return (pt_lp, change_a, change_b) then finalize(token_a, token_b, amount_a,
    amount_b, pid, liquidity);
}

finalize create_pair_privately(
    ...
    public liquidity_in: u128,
) {
    assert(!pairs.contains(pid));
    assert(!tokens.contains(pid));

    let liquidity: u128 = sqrt_u128(amount_a * amount_b);
    assert(liquidity_in == liquidity);

    ...
}
```

Snippet 4.5: Definition of the `create_pair_privately` function

Impact Users may be unaware of how to specify the amount of swap tokens, which could make it difficult to use the function.

Recommendation Since `sqrt_u128` is an inline function, consider calling it in the transition instead. Additionally, if `sqrt_u128` is moved to the transition function, consider changing the if statements in `log2` into ternary operations as they are more efficient.

4.1.10 V-ASP-VUL-010: Potential Private Value Leak

Severity	Warning	Commit	e8f2056
Type	Information Leakage	Status	Fixed
File(s)	token_faucet/src/main.leo		
Location(s)	init()		
Confirmed Fix At	7179a52		

In the Leo language, parameters to transitions default to the private visibility, so developers must specify if a value should be public. Since no visibility is specified for the `admin` parameter of the `init` function shown below, it is therefore private. When the variable is passed to the finalize function, however, it will be publicly exposed.

```
transition init(admin: address) {
    assert_eq(self.caller,
    aleo1tdszx3hcgnyp2jw3y3fzvw27vremxcs24u4pys6vptg9y2jfsvps8e8ffz);
    return then finalize(admin);
}
```

Snippet 4.6: Definition of the `init` function

Recommendation Consider making this parameter public.